Securing Microsoft Azure

Automatic Access Control List Rule Generation to Secure Microsoft Azure Cloud Networks

---

Carl Philip Matsson

Faculty: Faculty of Health, Science and Technology

Subject: Computer Science

Supervisor: Javid Taheri

Examiner: Leonardo Martucci

Date: 170607

Securing Microsoft Azure

Automatic Access Control List Rule Generation to Secure Microsoft Azure Cloud Networks

Carl Philip Matsson

Abstract

Cloud computing has been one of the largest growing trends in computing in recent years. This thesis describes the design and implementation of a new application for deploying cloud resources in Microsoft Azure, one of the largest cloud computing providers on the market. One of the main focuses for this application is to make the creation of Azure systems faster and more secure. By aiding the user in securing the intra-network firewall using an algorithm described in the thesis, with the aim to detect and mend security holes in the firewall, the application provides an automatic process of securing the network by generating new firewall rules. The application implements graphical interfaces allowing the user to get an overview of the network and providing functionality that lets the user observe address space utilization and firewall coverage, decreasing the need for manual inspection of the access control lists within the firewalls. The algorithm and designs in this paper are evaluated in the Azure ecosystem but would also be applicable to most other services or networks implementing a firewall with an access control list.

Acknowledgements

First of all I would like to thank the company Sogeti for allowing me to work with them and providing the resources required to produce this application. I'd especially like to thank Mats and Jörgen at Sogeti for their positive and very helpful support with the technical aspects of the application.

Furthermore, I would like to thank my supervisor Javid Taheri for his excellent work as a supervisor. Your guidance, knowledge and positive attitude was inspiring and had a major positive contribution to the project; it would not have been the same without your help.

Contents

List of Figures

List of Tables

1 Introduction

Cloud computing is arguably one of the largest advances in computing in recent years. The surge in demand for software as a service (SaaS), infrastructure as a service (IaaS) and platform as a service (PaaS) have created incentives for large corporations such as Google, Amazon and Microsoft to invest heavily in cloud infrastructure and the market is growing at a high rate. For example, Microsoft's cloud service Azure grew by 128% for the quarter ending September 2014 compared to the same time the previous year[12]. As the market grows the need for security grows with it, but it is not always easy to detect and mend security vulnerabilities in large cloud networks manually.

1.1 Project Goals

The goal with this project is to aid in securely deploying systems to Azure. This project aims to build a prototype able to create resources and deploy them to Azure but also detect intra-network security flaws and correct them before deployment. The main focus will be on the access control list (ACL) within the Network Security Group resource in the Azure ecosystem. To secure the ACLs, the prototype must implement an algorithm capable of detecting security holes within the ACL and securing these holes without causing too much overhead. Large ACLs are infamous of slowing down networks and being incomprehensible when manually inspected due to the sheer amount of data and the complexity of the ACL content [16]. Having large ACLs is therefore counterproductive to this projects goal of creating a better and more secure experience designing and deploying Azure resources. The application should be able to provide a graphical interface of the network utilization and firewall rules reducing the need of manual inspection of the ACLs and allowing the user to get a quick overview of the security status of the network. By displaying notifications to the user when the network is suboptimally configured it allows the user to fix these problems before deploying the project. After deploying to Azure it can be hard to modify

the configuration of resources and even impossible to change certain basic configurations such as the resource name, it can therefore be a great benefit to be able to see errors in configuration before the resource is deployed. The application will also assist the user to follow the naming conventions for Azure, as defined by Microsoft.

The prototype itself shall allow the user to create, modify and connect resources in a graph interface with a focus on being able to quickly design a system. The prototype should treat each graph as a file and the user should be able to save and read these files allowing the application to serve as a document manager.

1.2 Dissertation Layout

The dissertation follows the following layout;

- Chapter 2 describes related work trying to solve issues regarding firewall optimization and security.
- Chapter 3 gives an overview of Azure, its deployment alternatives and the Azure resources implemented in this project. The resources purpose as well as template structure is discussed.
- Chapter 4 gives relevant background on Access Control Lists, specifically Azure ACLs. The structure of the ACL is described and how the CIDR notation may help decrease the amount of rules required to target an address range.
- Chapter 5 defines the research question, the partial motivation behind the project and why it is important.
- Chapter 6 displays the theory and implementation of how the firewall rules are generated.
- Chapter 7 describes the design behind the prototype.

- Chapter 8 describes the implementation of the prototype and its functionality.
- Chapter 9 will go through the conclusions and the result of this project with a short discussion of possible future work.

1.3 Contributions

Currently we have designed a system that allows the user to quickly design a system in Azure implementing a few of the resources available. The application allows the user to deploy the system to Azure and provides error detection for invalid network configurations and shadowing firewall rules. Two separate graphical interfaces, one to observe the network utilization and another that display the security status of the ACLs is provided, reducing the need for manual inspection of the ACL. We propose an algorithm implemented in the application allowing the user to let the application auto generate firewall rules assisting the user in securing the intra-network. In the future, additional resources could be added to the application and the firewall generation could be expanded to resolve shadowing rules automatically.

2 Related work

The main focus of this work is to detect vulnerabilities and generate firewall rules blocking an address range that may or may not already be targeted by a firewall rule. There does not appear to be many works that correlates directly to this particular type of firewall rule generation. Many papers only have the scope of the firewall itself due to this it is seldom that the entire network is at disposal when designing the firewall and also because intra-network firewalls is not as commonly discussed. These papers often discuss improvements and algorithms in improving, reducing or combining the current rules confined within the firewall but not generating completely new rules on address spaces previously not targeted by any rule. In this project the full network is visible at design time and it is therefore possible to take advantage of the possibilities this can bring.

There are multiple papers discussing different solutions to the problem of detecting firewall misconfiguration. For example, in [13] the authors define a terms called shadowing and redundant rules.

Definition 1.1 *Let R be a set of filtering rules. Then R has shadowing iff there exists at least one filtering rule, Ri in R, which never applies because all the packets that Ri may match, are previously matched by another rule, or combination of rules, with higher priority in order.*

Definition 1.2 *Let R be a set of filtering rules. Then R has redundancy iff there exists at least one filtering rule, Ri in R, such that the following conditions hold: (1) Ri is not shadowed by any other rule; (2) when removing Ri from R, the filtering result does not change.*

The authors present an algorithm for detecting and removing shadowing rules and redundant rules, which share some similarities to the algorithm applied in this project. However, the work defined in [13] diverges from our work in that they process the rules and generate new rules from overlapping ones instead of simply warning the user about the potential issue and the algorithm in our work does not handle redundant rules. Another

divergence is that instead of only resolving shadowing or redundant rules; in this case it is also possible to block addresses that are not in use by any resource within the private network allowing the algorithm to further secure the network. This benefit also increases the range of considerations to take into account, instead of resolving conflicts of pre-existing firewall rules the entire address space of the resource implementing the firewall has to be taken into consideration when deciding the rules relevancy, and even blocking addresses that previously was not targeted in the firewall configuration.

Another work that shares some similarities to our work is [10]. In this work the authors produce a framework for representing both the security policy and the network topology. The policies defined can be completely separate from the network topology allowing administrators to keep a consistent policy even as the network topology changes. They present a entity-relationship model that has global knowledge about both the network topology and security policies that allow their model compiler to generate this knowledge into firewall specific configuration files. The combination of topology and security policies are grouped into entities referred to as *roles*. These roles can be implemented by different hosts in the network and have specific capabilities such as acting as a mail service, i.e accepting SMTP traffic at port 25 or a web-enabled role typically applied to intranet hosts allowing users in the intranet to browse the web. This solution is much more complex and flexible than our solution, but requires more work to configure. In our work there is no need for such a complex solution since we have the relationship between hosts defined in the model and can base the firewall configuration on a predefined set of policies.

In the work defined in [15] the authors discuss and analyze the consistency problem in firewall rule sets. They describe inconsistency as "There is a local inconsistency when two or more rules of the same rule set which have different actions overlap, since a packet can be matched with any of the overlapping rules.". They propose a new inconsistency detection algorithm that prevents rule updates that can cause this inconsistency when inserting, removing or updating rules. Similarly to this project we also implement methods

of detecting these inconsistencies, but we do not take decisions based on the findings, we simply notify the user. However, in [15] the consistency of the rule set is contained by not allowing the insertion of an inconsistent group of new rules and only allowing the insertion of rules that are consistent with the rule set.

A tool that implements a similar functionality of the application we built is the ArmViz tool [4]. This tool is browser based and allows the users to visualize their Azure templates in a similar fashion to our application. The tool uses an Azure template to visualize the contents in a graph allowing the user to view their template in a more user friendly way. By editing the template, the graph updates to represent the new modifications allowing the user to observe the effect of the changes in the template right away. The graph visualizes the Azure resources and their name but almost no other properties are exposed in the graph. One of the great features with ArmViz is that it in addition to displaying the resources in the graph it also displays the parameters of the resources, making it clear to the user which parameters are required to be manually assigned during deployment.

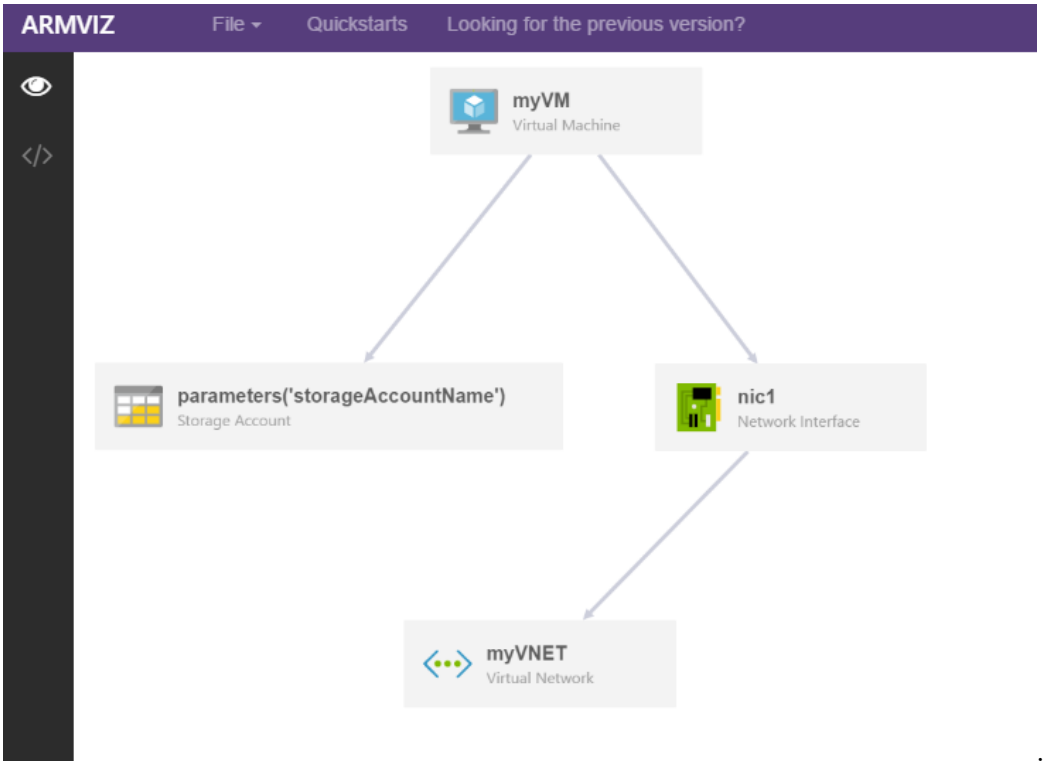

Figure 2.1: Azure Resource Manager Template Visualizer (ArmViz)

Some of the key differences between ArmViz and our application are that, while the graph displays the resources in a similar manner to our application, the contents of these resources still has to be managed by manual changes in the template. In our application simply selecting the resource displays its properties for easy configuration with the potential to validate the user input and allow for a faster and more user friendly experience. Another key feature this tool is missing is validation of network resources and firewall rules. Since ArmViz merely try to visualize the resources is has no form of validation of the contents in the template and does not solve the potential errors that may arise when constructing the template. The older version of the tool did implement a toolbox allowing the user to create resources via the GUI, although configuration of the resource still had to be done via the template. The older version of the tool did also seem to have some form of deployment option available but both of these features seem to be missing from the latest version.

3 Microsoft Azure

Microsoft Azure is a cloud service, announced in 2008 and released in 2010, owned and run by Microsoft. Azure is most recognized for its IaaS offerings with resources such as virtual machines and virtual networks.[12].

3.1 SaaS, IaaS and PaaS

When considering cloud services it is important to know the difference between the services that is offered. The cloud as a concept is very broad and hosts many different types of services that all serve a different purpose. The main service models that exist today are Software as a Service (SaaS), Infrastructure as a Service (IaaS) and Platform as a Service (PaaS).

3.1.1 Software as a Service

Software as a Service stands for software that is hosted on servers in the cloud. Unlike traditional software where you have to buy the software before using it, SaaS is provided as a service that users can subscribe to [17], usually by 'renting' or 'borrowing' the software. Most of us use some primitive form of SaaS daily, for example when using e-mail or conference calls we are actually utilizing SaaS. SaaS is very flexible and usually does not require any installation to run over the web. The users simply access the software via a web browser and use the software without downloading or installing the application [17]. Maintenance and updates of the software does only need to be done once on the server and the updated application will be available to all the users right away [14].

3.1.2 Infrastructure as a Service

Infrastructure as a service refers to when the service offered is the actual hardware and the associated software. For example IaaS can include servers, storage or network and all

associated software such as operating system and file system. The revolutionary benefit with IaaS over simply purchasing the hardware is that IaaS does not require any long term commitment due to that most IaaS providers use a 'pay-as-you-go' model [11, 17, 18]. When purchasing hardware the buyer must commit to using that specific hardware; this often means purchasing more expensive hardware to compensate for unexpected popularity or requirements. Scaling the hardware up or down is also difficult without purchasing new components that could potentially be very costly. With IaaS all those issues are a problem of the past. Scaling the hardware up or down is usually as easy as with a press of a button, and if the hardware is no longer needed the user can simply remove the hardware from their subscription and no longer have to pay for it [17]. Most users of IaaS think that the flexibility of the pricing is the key benefit as it allows them to only pay for the infrastructure that the application delivery requires [11]. The IaaS provider is also responsible for housing, running and maintaining the infrastructure which can save the user a lot of time and money.

3.1.3 Platform as a Service

Platform as a Service provide a combination of Software as a Service and Infrastructure as a Service. PaaS provide a platform on the cloud with all the infrastructure and the applications the client requires [17]. The client then does not need to go through the struggle with buying each software license and installing all the software. With this service developers can get access to all systems and environments required for the entire development cycle, everything from developing, testing and deploying the software [17].

3.2 Templates

When resources are combined into a big JSON file, these are referred to as templates. Microsoft encourages the use and distribution of templates and are hosting hundreds of templates on the Azure Marketplace and the official Azure Github website [2]. Templates

reduce the time it takes to create and manage resources and can give the user get a head start when choosing an existing template with many resources predefined within the template. When a template is done, the user can deploy it to Azure and the resource will be allocated in the user's subscription on the Azure platform. As can be seen in Section 3.3, each resource has a unique JSON structure that must be followed for the template to be valid.

3.2.1 JSON structure

The structure of the JSON template is not very complex and follows a similar pattern for most resources. Many resources rely on other resources to work, to display this dependency the structure defined in Listing 1 can be implemented as a part of the resource. The dependencies are treated as references to resources within the same resource group.

```
{
  "dependsOn": [
    "resource1",
    "resource2",
    "...",
    "resourceN"
  ]
}
```

Listing 1: Dependencies

Although most resources implement a unique set of properties some basic properties tend to exist in most resources. These shared properties are essential to identify and create most resources and are therefore implemented in almost every resource. Some of these shared properties include:

apiVersion - Version of the REST API to use when deploying the resource.

type - Type of resource.

name - Name to identify the resource.

location - Geographical location to deploy the resource in.

properties - Configurations specific to the resource

3.3 Resources

An Azure resource is "A manageable item that is available through Azure" [3] and includes items such as virtual machines and virtual networks. There are a multitude more resources available in Azure than mentioned in this chapter, however the focus in this chapter lie on some of the most relevant resources to the work in this project.

Resources are deployed and managed using JSON to represent the resources, combined in a large file referred to as a template. In this chapter, each resource will be described using their JSON structure as a reference point, and at the end of the chapter the process of deploying the JSON templates and how the full template is structured will be discussed.

3.3.1 Resource group

A Resource group is a simple way to group multiple resources that belong in the same context. The resource group may contain every single resource in the entire solution or just a smaller portion of resources that should be treated as a group and deployed together. The resource group is crucial for this project since it allows the deployment of multiple resources with a single repeatable operation [6].

```
{
    "$schema": "[schema url]",
    "contentVersion": "",
    "parameters": {  },
    "variables": {  },
    "resources": [  ],
    "outputs": {  }
}
```

Listing 2: Resource group

The resource group can contain all the properties defined in Listing 2, however to streamline the implementation of this project, variables, parameters and outputs are not going to be implemented. The purpose of 'Variables' is to make it easier to reuse content on multiple properties in or between the resources when manually adding resources. Since we in this case let the application do the implementation of properties, instead of doing it manually, and can easily propagate a value to other properties, little to no value is lost for the end user by not implementing the variables property in our template. Parameters specify what values the user can input manually when deploying the resources through the command-line user interface (CLI), this could potentially make switching between environments such as development and production go smoother, however when deploying through the application implementing this property is not really feasible due to the lack of direct exposure to a CLI. The major drawback of our approach of not implementing these properties is that the project is not compatible with many of the existing templates that do in fact use these properties. For the future, exploring the option of adding support for older templates is high on the agenda.

Out of the set of properties defined in Listing 2 the properties that is going to be implemented are Schema, which describes the location of the JSON file defining version of the template language to use, Content Version which is the user defined version of the template and Resources which is the container for all the resources belonging to the

resource group. These properties are all required when deploying to Azure.

3.3.2 Virtual network

A virtual network is, as the name suggest, not a completely physical network, it is partially made up of virtual network links. This setup makes it possible to create logical isolation for resources in IaaS and similar services.

In Azure, the virtual networks (VNets) are used to create an isolated environment for resources in the Azure cloud. These VNets also make it possible to establish a connection to on-premise networks, either through a direct connection or via an encrypted tunnel over the Internet (VPN). As a default, all resources connected to a virtual network have Internet access but each VNet is completely isolated from all other VNets. VNets can however be connected to each other, allowing the resources defined within the VNets to communicate with any resource in the set of connected VNets. Resources in the same VNet can always communicate with each other via their private IP address, even if the resources does not belong to the same subnetwork unless explicitly blocked by the subnets firewall.

There is no need to manually configure routes in the VNet since Azure will create the routes automatically. However the routes can be overridden to configure custom routing if the user is so inclined [5].

The VNet's JSON string mainly consists of a name that identifies the VNet, a location that specifies in which region the VNet is located, the address prefix that defines the IP address range of the VNet and its subnet(s) that can be used to further partition the network. A JSON representation of a virtual network can be observed in Listing 3

```
{
      "apiVersion": "[variables('apiVersion')]",
      "type": "Microsoft.Network/virtualNetworks",
      "name": "[parameters('vnetName')]",
      "location": "[resourceGroup().location]",
      "properties": {
        "addressSpace": {
          "addressPrefixes": [
            "[parameters('vnetAddressPrefix')]"
          ]
        },
        "subnets": [ ]
      }
}
```

Listing 3: Virtual network

The Address Prefixes property define a set of address spaces that belong to the VNet and define the address space that the subnets can draw their IP addresses from. These addresses are IP address followed by the CIDR notation. The CIDR notation defines which of the bits in the address are used to identify the network and which bits are used to identify subsequent resources or subnetworks. For example, an address of *10.1.0.0/16* states that the first 16 bits of the address identifies the network and subsequent bits can be used to identify other resources. In this example the network then could potentially have *65534* other resources in its address space. To add other resources such as virtual machines to a VNet a subnetwork is required. The subnets can be used to further partition the networks address space and serve as isolation between resources of different context.

The Subnets property is a container that stores the definitions of the subnets belonging to the Vnet.

3.3.3 Subnetwork

Although not technically classified as an individual resource, since it is just a property of the VNet, the subnetworks (subnets) still play an important part in the network structure and filtering of traffic in Azure and thus warrants its own section in the resource category. As can be observed from Listing 3 subnets are a part of the virtual network and are used as an extra layer of isolation between different groups of resources within the same virtual network. Although the network routing is handled by Azure, filtering, to a large extent, is not. To make use of filtering, subnets may implement a resource called Network Security Group which contains an access control list (ACL) that make it possible to explicitly block or allow traffic to or from IP addresses passing through the subnet. The only resources capable of implementing ACLs in Azure are the subnet and virtual machine.

```
{
    "name": "[parameters('subnet1Name')]",
    "properties": {
            "addressPrefix": "[parameters('subnet1Prefix')]"
    }
}
```

Listing 4: Subnetwork

Just like the VNet, subnets contain an Address Prefix property defining the address space, the minor difference is that while VNets may implement multiple address prefixes subnets can only implement one.

3.3.4 Network security group

The recommended way of filtering traffic in Azure is with a Network Security Group (NSG) resource. The network security group contains a list of rules restricting or granting access to traffic passing through a subnet by targeting certain IP addresses or address ranges. The

NSG may only be associated with subnets or virtual machines. Naturally, if the NSG is associated with a subnet, all traffic within this subnet has to comply with the rules defined in the NSG. On the other hand, if the NSG is implemented within a virtual machine only the traffic moving to or from the virtual machine are affected by the rules of the NSG.

```
{
  "apiVersion": "2015-06-15",
  "type": "Microsoft.Network/networkSecurityGroups",
  "name": "[parameters('frontEndNSGName')]",
  "location": "[resourceGroup().location]"
  "properties": {
    "securityRules": [
      {
        "name": "rdp-rule",
        "properties": {
          "protocol": "Tcp",
          "sourcePortRange": "*",
          "destinationPortRange": "3389",
          "sourceAddressPrefix": "*",
          "destinationAddressPrefix": "10.0.0.5/24",
          "access": "Allow",
          "priority": 100,
          "direction": "Inbound"
        }
      }
    ]
  }
}
```

Listing 5: Network security group

The rules implemented by the NSG are contained within the Security Rules property. The rules within the Security Rules container implement Protocol, that states what protocol the rule apply for, Source Port Range and Destination Port Range which describes the port range of the sender, Source Address Prefix and Destination Address Prefix which describes the address space of the sender and receiver, Access that states whether to allow

or deny the packets corresponding to this rule, Priority states how the rule is prioritized and Direction that describes the direction of the traffic that this rule applies to.

The addresses and ports can all be defined as a range. This potentially allows the user to target a whole range of addresses with a single rule, thus reducing the number of rules required when a set of addresses should be allowed or denied.

3.3.5 Virtual machine

Virtual machines (VMs) are the foundation of any cloud service and there is no exception for Azure. A VM is primarily used as a server and in Azure this server can also be used as an extension to on-premises networks to help with scaling or to alleviate load during heavy traffic. There are many properties that must be defined within a virtual machine. The properties can generally be divided into four categories, hardware, operating system, storage and network.

```
{
  "hardwareProfile": {
          "vmSize": "[parameters('vmSize')]"
  },
}
```

Listing 6: Hardware Profile

The Hardware Profile property describes the hardware of the virtual machine and only implement one property, Vm Size. The Vm Size determines what predefined components, such as disk size, CPU cores and memory the virtual machine should utilize. The options of available hardware specification are determined by Microsoft and include a wide range of options that scale in price with better hardware.

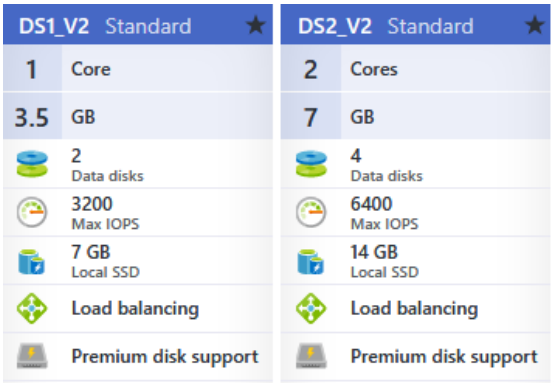

Figure 3.1: VmSize options example

In the JSON template, the Vm Size is defined by the ID of the predefined hardware option available with Microsoft. An example of the IDs can be observed in figure 3.1, where the IDs are *DS1_V2* and *DS2_V2* and also displays their respective hardware specifications.

The next property of the virtual machine is the Os Profile property that describes the user settings of the operating system, such as user name, password and computer name.

```
{
  "osProfile": {
    "computerName": "[variables('vmName')]",
    "adminUsername": "[parameters('adminUsername')]",
    "adminPassword": "[parameters('adminPassword')]"
  }
}
```

Listing 7: OsProfile

The properties in Os Profile are used to identify the virtual machine and as a measure of security that only allows authorized users that know the user name and password to have access to the virtual machine.

The Storage Profile describes the storage options and operating system of the virtual machine. Azure supports many different operating systems with different configurations,

including the most commonly used Windows and Linux.

```
{
  "storageProfile": {
    "osDisk": {
      "name": "[concat(variables('vmName'),'-osDisk')]",
      "osType": "[parameters('osType')]",
      "caching": "ReadWrite",
      "createOption": "FromImage",
      "image": {
        "uri": "[parameters('osDiskVhdUri')]"
      },
      "vhd": {
        "uri": "[vhdUri]"
      }
    }
  }
}
```

Listing 8: StorageProfile

Storage Profile implements the properties Os Type which define the operating system of the virtual machine, Caching that describes the caching requirements (None, ReadOnly, ReadWrite), Create Option that states how the virtual machine should be created (Attatch, FromImage) and Vhd that specifies the URI for the location in storage where the VM should be placed.

3.4 Deployment alternatives

There are a few different alternatives for deploying and creating resources in Azure. In this section a few of the most common ways are discussed with their benefits and drawbacks.

3.4.1 Azure Portal

The Azure Portal is the most accessible method of creating resources in Azure. The Azure Portal comes in the form of a Microsoft website where the user can create resources via a graphical interface directly connected to the Azure platform.

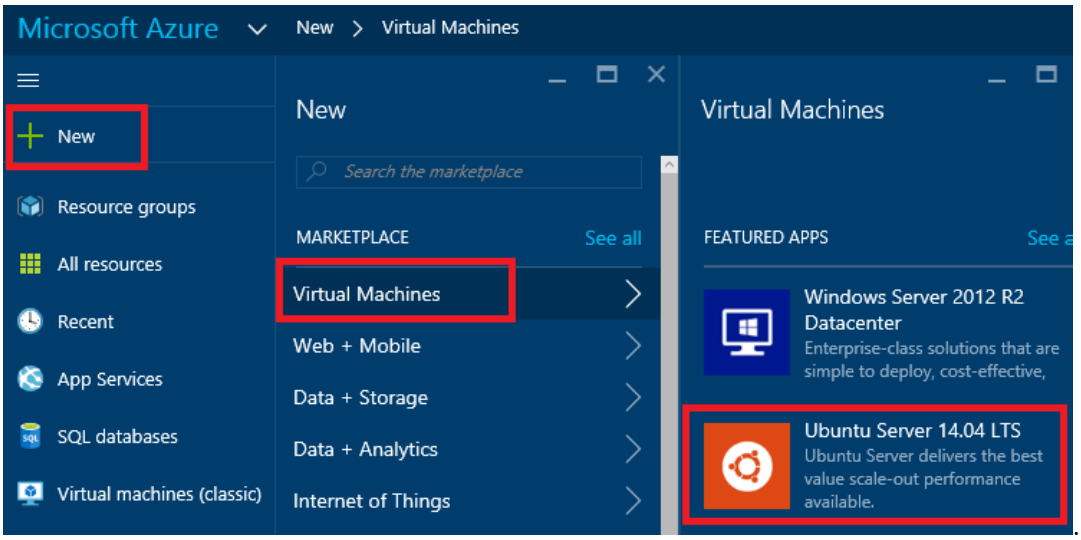

Figure 3.2: Creating a virtual machine in the Azure portal

Credit: *Microsoft,*

https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-template-deploy-portal

The process of creating resources via the Azure portal is easy, but does usually take a long time, especially for larger systems. The Azure portal requires the user to create one resource at a time and it is immediately created within the resource group causing the process to be slow and settings to be specified correctly right away. Modifying properties of the resources after creation can be tricky, renaming resources can even in certain cases be impossible due to the limitations in Azure. The Azure portal does not allow the user to 'sketch' a system and worry about applying correct settings later, which can be a big drawback when the user does not have a complete plan of the network topology from the start. The Azure portal is therefore best used when the user is just getting started with

smaller systems with limited amount of resources and complexity.

3.4.2 Template

One of the preferred methods of creating resources is with the use of templates, described in Section 3.2. There are a few different ways in how the templates are deployed to Azure; the user can utilize PowerShell, Azure CLI or the Azure REST API. Templates are great for fast modification of resources in larger systems and allow the user to experiment with the network topology and resources before fully committing and deploying the template to Azure.

With PowerShell, the full process of deploying a template to a new Resource group can be as simple as shown in Listing 9

```
1 Login-AzureRmAccount
2 Set-AzureRmContext -SubscriptionID {your-subscription-ID}
3 New-AzureRmResourceGroup -Name ExampleGroup -Location "South Central US"
4 New-AzureRmResourceGroupDeployment -Name $deployment
5         -ResourceGroupName $resourcegroup -TemplateFile $template
```

Listing 9: Deploy a template with PowerShell

The user login to Azure via PowerShell, assign the context in which the deployment should be carried out to then simply create a new resource group and deploy the template with just a few lines of PowerShell. If the user is so inclined, it is also possible to deploy individual resources to Azure using PowerShell without a template, this would however have similar drawbacks as with using the Azure Portal.

The other way of deploying the templates is by using the Azure REST API. By using the API users can build applications that communicate directly with Azure. Deploying with the REST API is performed in five steps [7]:

1. Set common parameters and headers, including authentication tokens.

2. Create a new resource group, provide subscription ID, name of the resource group and location needed for the solution.

3. Validate the deployment

4. Create a deployment

5. Get the status of the template deployment

Deploying with the API requires a PUT request with the required parameters, an example can be observed in Listing 10.

```
PUT https://management.azure.com/subscriptions/<YourSubscriptionId>/resourcegroups
            /<YourResourceGroupName>/providers/Microsoft.Resources/deployments
            /<YourDeploymentName>?api-version=2015-01-01
  <common headers>
  {
    "properties": {
      "templateLink": {
        "uri": "http://mystorageaccount.blob.core.windows.net/templates/templ.json",
        "contentVersion": "1.0.0.0"
      },
      "mode": "Incremental",
      "parametersLink": {
        "uri": "http://mystorageaccount.blob.core.windows.net/templates/param.json",
        "contentVersion": "1.0.0.0"
      }
    }
  }
```

Listing 10: Deploy a template with the REST API

4 Azure Access Control List

Access control lists (ACL) are in the center of the research in this project. This chapter attempts to give the reader a basic understanding of what an ACL is, it defines the purpose of an ACL and how they can help to improve network and cloud security but it also explains the drawbacks with many ACLs that we seek to address. Most ACLs follow the same structure, so even though the main focus is on the ACL that can be found in the Azure network security group the same information, with potentially minor modifications, apply to other ACL implementations as well.

4.1 Introduction

ACL's has for a long time been used to administrate file, program or process privileges in many operating systems. The ACL entries, often referred to as Access control entries (ACEs) can define a user or group's right to access any of these objects. The ACE defines specific rights such as if the user or group can read, write to launch the object in question.

In networking, ACLs are instead used to manage the access to networks, services or other resources based on mainly IP addresses and port numbers. In an age where more and more businesses are migrating their sensitive data to the cloud it is crucial that the access to this data is limited to the absolute minimum. One way to secure the network and minimize unauthorized access to network resources is through a well-defined ACL. The main purpose of an ACL is to keep control of who can access specific objects and seek to prevent breaches of security.

The way that ACLs are utilized is by analyzing incoming or outgoing traffic on a packet to packet basis to determine if the traffic is allowed or not. When a packet is received, parameters such as port, address and direction of the traffic are taken into consideration and matched against the list of entries defined in the ACL. The entries are ordered based on priority and the first entry in the ACL that match the traffic is the one that is executed.

If the matched entry does not allow a certain type of traffic, the packets of that traffic type will be dropped; otherwise the packets will be forwarded to its destination.

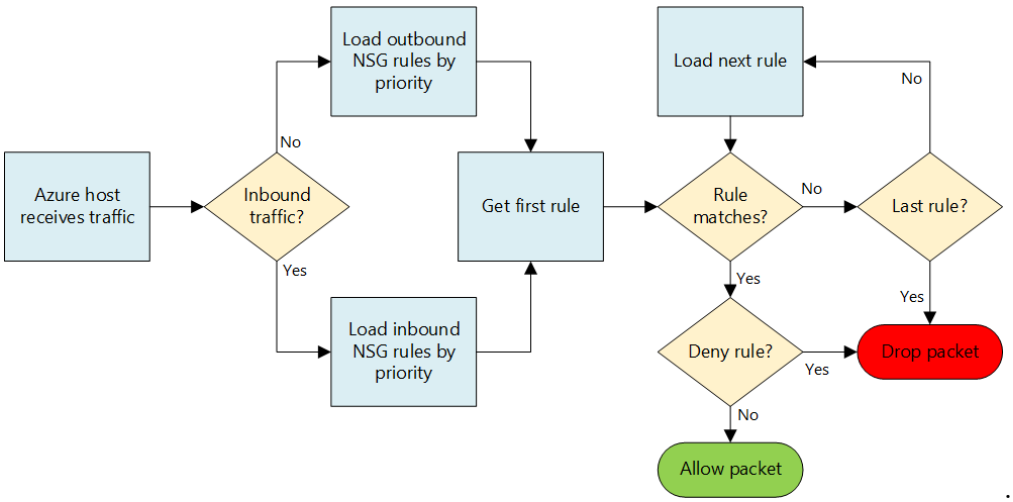

Figure 4.1: Network security group - Traffic filtering

Credit: *Microsoft, https://docs.microsoft.com/en-us/azure/virtual-network/virtual-networks-nsg*

A flow chart of the decision making and how the ACL is applied can be observed in Figure 4.1. In Azure, each entry, also referred to as rule interchangeably, in the ACL is evaluated against the incoming traffic. If an entry matches the traffic, a decision based on the given policy to allow or deny is executed and either drops the packet or allow the packet to travel to its destination. On the other hand, if the packet does not match the current entry, the next entry in the ACL is retrieved and matched against the traffic. This process is repeated until a matching entry can be found or there is no more entries to match against in the ACL. If the traffic does not match any entries there is usually a policy in place to either deny or allow all traffic that the ACL failed to cover. This also infers that the ACL can contain conflicting entries about the same type of traffic, for example an entry can state that traffic of type X is allowed, but further down the list another entry can state that traffic of type X should be denied. Only the first rule will ever be executed since the ACL only need one matching rule to make a decision about the traffic, and the

first entry will be the one discovered first and thus executed every time. This can cause a false sense of security, especially for an ACL containing a big set of entries, where the user has created an entry to block a certain type of traffic, but an entry closer to the beginning of the ACL allow this type of traffic, thus rendering the user's new entry to have no effect, sometimes without the user's knowledge.

Although matching a single entry in the ACL does not take very long, a huge ACL of potentially several thousands of entries has the potential to severely slow down a network when matching entries with low priority. Large ACLs also have a tendency to become redundant, inconsistent, and difficult to understand and thus also difficult to optimize. A manual inspection of an ACL of this size is near impossible due to the sheer amount of data and the complexity of the ACL content [16].

4.2 Structure and characteristics

To better understand how an ACL can identify traffic and how the entries in the ACL can be defined, one must have some knowledge about some basic concepts of IP networking. In this section the structure of the ACL entries will be discussed as well as some well-known general concepts of IP networking such as the CIDR notation.

4.2.1 Address and port ranges

The source address and the source port section of the ACL rule generally describe a range of addresses and ports. This is to reduce the total amount of rules required to handle traffic in a certain range of addresses. A common way of describing the IP address ranges is with the classless inter-domain routing (CIDR) notation. CIDR is used as a postfix addition to ordinary IP addresses and serve to identify the bits of an IP address belonging to a network and subsequently also defines the addresses available in the network.

In table 4.1 we can observe three subnetworks where subnetwork 1 and 2 belong to the same address space of the virtual network. We can identify which network they belong to

Resource	Address space
Virtual Network	10.1.0.0/16
Subnetwork 1	10.1.1.0/24
Subnetwork 2	10.1.2.0/24
Subnetwork 3	10.2.3.0/24

Table 4.1: CIDR example

because that they share the network identifier with the virtual network. An IPv4 address has a total of 32 bits at its disposal to divide IP addresses into subnets, the virtual network has CIDR notation of /16, and thus the first 16 bits of the address are used to identify the virtual network. To see this we can look at the netmask which in this case is 255.255.0.0 or 11111111.11111111.00000000.00000000 in binary. The subnets that begin their address with the virtual network's identifier belong to that virtual network. Since subnetwork 1 and 2 both start their address with 10.1.xx.xx we can assume that they belong to the virtual network. Subnet 3 however, diverges from the pattern of the identifier for the virtual network and does therefore not belong to that particular virtual network. Subsequent networks to the virtual network must also have a CIDR notation greater than the one defined in the virtual network to be able to create a unique identifier for them. The identifier for subnetwork 1 becomes 10.0.1.xx and the identifier for subnetwork 2 becomes 10.0.2.xx with a netmask of 255.255.255.0 and with 8 bits remaining to identify other resources connected to the subnetworks.

There is no CIDR notation when defining port numbers, instead port ranges are often defined in the form of *first port - last port*. So to define a port range from 80 to 200 one would simply enter 80-200.

4.2.2 Priority

The ACL entries are ordered based on priority. Most ACLs orders from lowest to highest, but in some implementations it may be the other way around. If rules overlap, i.e. there are multiple entries targeting the same subgroup of endpoints and source, the entry that

comes first in the list is the rule that is used to decide if the traffic is allowed or not. Rules that never get executed are often referred to as a shadowing rule.

The numeric range of possible priorities varies between implementations, but usually either start from 1 or 100 up to a maximum number defined in the design of the system. In Azure this range is between 100 and 4096.

4.2.3 Protocol and traffic direction

Two other factors most ACLs take into consideration are the protocol and the direction of the traffic. These properties can be used to further filter traffic depending on the protocol the traffic is using and if the traffic is leaving or coming into the network. These properties may further complicate manual inspection of the ACL and gives the user a lot to keep track of when implementing an ACL by hand.

4.2.4 Azure ACL and Network security groups

In Azure, ACLs are used to filter traffic from and to virtual machines. A 'pure' ACL without the need for a network security group can only be directly implemented in the virtual machine resource, but is seldom used. The Network security group, that also contain a similar ACL, can however also be implemented in subnetworks as well as in virtual machines. Microsoft recommends the use of network security groups (NSG) over ACLs whenever possible.

When defining network security groups, Azure allow for some shorthand's when defining source or destination addresses, called 'tags'. For example, 'Internet' and 'VirtualNetwork' are all valid tags used to define an address. The available tags by default are [8]:

- VirtualNetwork
- AzureLoadBalancer
- Internet

These tags are used to automatically retrieve the addresses from the respective resource that the resource implementing the NSG is connected to. The '*', as per usual stands for 'any' and can be used as a shorthand to target every address, port or protocol.

Rule #	**Src. port**	**Dest. port**	**Src. IP**	**Dest. IP**	**Access**
100	80	*	10.0.0.1	*	Allow
101	*	80-1200	*	10.0.0.0/24	Allow
300	*	*	Internet	*	Deny
400	*	*	*	VirtualNetwork	Deny

Table 4.2: Network security group rules

In Table 4.2 some examples of these tags in use can be observed. These tags make it easier to create rules for a network with dynamic IPs or to safeguard in the event of IPs changing during the design of a system and gives the user more options when deciding to change the IPs of resources without having to reconfigure the ACL.

The NSGs contain a set of rules by default implemented my Microsoft. These rules cannot be deleted, but they have the lowest priority possible so they can easily be overridden by custom rules with a higher priority. The default rules, for all protocols, in an NSG can be observed in Table 4.3 and 4.4

Name	**Prio.**	**Source**	**Destination**	**Access**
AllowVNetInBound	65000	VirtualNetwork : *	VirtualNetwork : *	Allow
AllowLoadBalancerInBound	65001	LoadBalancer : *	* : *	Allow
DenyAllInBound	65500	* : *	* : *	Deny

Table 4.3: Inbound default rules

Name	**Prio.**	**Source**	**Destination**	**Access**
AllowVnetOutBound	65000	VirtualNetwork : *	VirtualNetwork : *	Allow
AllowInternetOutBound	65001	* : *	Internet : *	Allow
DenyAllOutBound	65500	* : *	* : *	Deny

Table 4.4: Outbound default rules

5 Improving Azure Network Security

In this section we will discuss the main research in this paper, the motivation behind it and the problems that this project is trying to solve. The research is divided into two sub-topics *Detecting and mending ACL vulnerabilities* and *Reducing the number of ACL entries*.

5.1 Motivation

The motivation behind the research is to create an efficient algorithm to detect intra-network security holes in a firewall implementing an ACL and mend these holes with the least amount of ACL entries without compromising security. This research is important due to the fact that as more and more companies and other entities are migrating their data from on-premise-networks to the cloud, security considerations are not always reflected on adequately nor properly implemented, and can have devastating effects on security and network throughput. One of the most common forms of protection against unauthorized access to data or resources is in the form of ACLs. ACLs tend to scale very poorly and having large ACLs can severely slow down a network if not properly managed. The goal of this work is to have an algorithm that can automatically detect and implement ACL entries with block as default and even overriding the user implemented ACL entries if deemed necessary.

5.2 Detect and mend ACL vulnerabilities

First, we have to define what we mean by ACL vulnerabilities and security holes. There are several subgroups of this definition, but our main definition of ACL vulnerability is if an IP address or IP address range, that should not have access to a resource, is not explicitly blocked by an ACL. There are also other ACL vulnerabilities such as having multiple ACL entries for the same IP address with different priorities, often referred to as

shadowing rules. Shadowing rules can cause a problem when priorities change, especially in large ACLs where it can be hard to see the effect a priority change may have on the overall network and its security.

For private IP addresses, the test environment should be able to produce corrections, errors and warnings for the following scenarios:

Type	**Action**
User defined ACL rule that allows an IP address, but the IP address is not occupied by any network resource	Override the rule with a higher priority rule and deny access
Multiple ACL rules defined for the same IP address or range (shadowing rules)	Generate warning to notify the user
The resource IP address is defined outside of its allocated address space	Generate error to notify the user
No rule defined for an IP address or range of IP addresses	Insert rule to deny access

Table 5.1: ACL private IP vulnerability scenarios

We chose to insert rules to deny addresses with no rule defined because we want to override the default rules added by Azure in the Network Security Group.

During design time it is possible to implement these scenarios due to the fact the test environment is fully aware of connections between the different resources and can take the appropriate action to either notify the user with warnings or by the users request generate the ACL rules required to secure the system. The purpose of these actions is ultimately to remove the hassle and error prone method of manually creating the ACL rules.

5.3 Reduce the number of ACL entries

The second part of the research is focused on how to implement these improvements without causing too much overhead. When generating the rules it is crucial to do so with the least rules possible. The simplest solution would simply be to block each rule one by one, but this would also have the unwanted effect of possibly creating thousands of ACL entries for even some of the midrange address spaces, and could severely slow down the network. We must find a way to detect where the gaps in the ACL are and, with as few rules as possible mend these holes.

6 Auto generation of ACL rules for Azure

This chapter aims to provide the design and display the implementation of the automatic rule generation implemented in the prototype, with the aim of securing the intra-network within Azure.

6.1 Theory

The ACL generation is the applications way of correcting a suboptimal ACL by applying its own entries to correct the mistakes made by the user. When the user requests help in securing the ACL of a subnet the application will analyze the current ACL and the resources connected to the subnet. If the application can see points of improvements to secure the network it attempts to accomplish this with as few additional entries to the ACL as possible.

The main improvements the application is attempting to achieve is if a rule defined by the user is allowing traffic from a private address or address space with no resource connected to it. The application will then override the rule and insert a rule to deny the traffic. If an address does not have any rule defined it will also be blocked by a new rule generated by the application.

Since we want to block these addresses with as few rules as possible, it is crucial that the application determines the largest address space it can block in one entry and make use of the CIDR notation to block a large amount of address with one single rule without interfering with the valid rules defined by the user.

Figure 6.1: Address space representation: 10.0.0.0/26

In Figure 6.1 an address range of 10.0.0.0/26 is defined containing 64 individual IP addresses, ranging from 10.0.0.0 to 10.0.0.63, represented by colored squares. The ACL

contain one rule allowing access to 10.0.0.29/32 visualized by the green square. The rest of the addresses need to be blocked by the application using as few rules as possible. Since we do not want to interfere with the existing rule and prefer not to have shadowing rules, the optimal solution to this problem then become to create multiple rules blocking the remaining addresses in as large chunks per rule as possible.

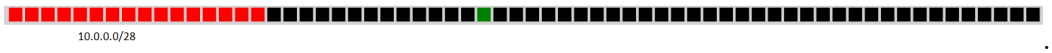

Figure 6.2: Targeting address space: 10.0.0.0/28

To visualize the problem of identifying the correct rules to generate, we start from the first address in the given address space of 10.0.0.0/26. The first span that should be blocked is made up of 28 non-utilized addresses. It is possible to block each address one by one, for example to only block the first address (10.0.0.0) inserting a rule blocking 10.0.0.0/32 could be added to the ACL. However, if each address is blocked in this manner it does not satisfy the condition of blocking the largest number of addresses with as few rules as possible. If we instead incrementally decrease the CIDR we will eventually have a rule targeting 10.0.0.0/28 shown in Figure 6.2. If the CIDR number was to be decreased any further it would be overlapping with the users custom rule at 10.0.0.29/32 thus creating a shadowing rule, we have thus found the largest address span for our first rule covering the address space from 10.0.0.0 to 10.0.0.15.

Figure 6.3: Targeting address space: 10.0.0.16/29

The next step is to repeat this process, starting where the previous rule left off. We now find that we can cover the address space from 10.0.0.16 to 10.0.0.23 with a rule targeting the address space of 10.0.0.16/29, since further decreasing the CIDR would exceed the end of the address range. We repeat this process until the full range of the address space is

covered.

Figure 6.4: Targeting address space: 10.0.0.24/30

Figure 6.5: Targeting address space: 10.0.0.28/32

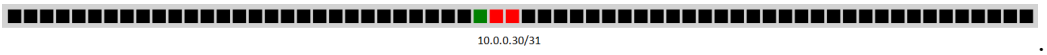

Figure 6.6: Targeting address space: 10.0.0.30/31

Figure 6.7: Targeting address space: 10.0.0.32/27

In total six rules were required to block the full address space without interfering with the previously defined rule in 10.0.0.29/32 and would be considered the optimal solution for this address space, meeting the criteria stated earlier.

There is however other limitations on how large of an address space the rules can target. For example, if we have an address span where the starting address is 10.0.0.16, it is only possible to reduce the CIDR to a minimum 28 no matter if there are user defined rules in the span or not. This is because of how the CIDR netmask work. If the CIDR was to be decreased further it would distort the starting point of our span. The address 10.0.0.16 corresponds to 00001010.00000000.00000000.00010000 in binary octets and the CIDR of 27 corresponds to 11111111.11111111.11111111.11100000. Performing a bitwise AND operation between the address and CIDR would produce 00001010.00000000.00000000.00000000 distorting the start address defined by the CIDR span. Thus, the address of 10.0.0.16/27

would not have 10.0.0.16 as the first address in the span and not correctly represent the span the rule is trying to target.

6.2 Implementation

To identify and generate the ACL entries the first step is to find the continuous spans of addresses that ought to be blocked. To determine if an address should be blocked or not is based upon two parameters, utilization of the address by resources in the network and custom rules defined by the user. The rules for identifying the action for specific addresses are:

1. An address should be blocked if the address is not utilized by a resource.
2. A user defined rule should be overridden by a new deny rule if there is no resource utilizing the address targeted by the rule.
3. If an address is allowed by an existing rule and is utilized, the rule must remain as is without creating a shadowing rule.

The application can never remove user defined rules, but can overrule them with a higher priority and warning the user that a custom rule has been overridden and prompting the user to resolve the issue.

The first step is to extract the addresses targeted by valid rules. The addresses targeted by a valid rule are extracted from the set of addresses by looking if the address is targeted by any rule and the address is also utilized by at least one resource.

```
1 var validRules = addressSecurityStatus.Where(x =>
2                 x.Value != SecurityRuleStatus.None
3                 && ipAllocation[x.Key] > 0);
```

Listing 11: Identifying valid ACL rules

By iterating over the list of valid rules it is possible to identify the address spans. The start of each span is identified by finding the first unprocessed address that is not in the list of valid rules. The end of the span is determined by finding the address previous to the address targeted by the next valid rule. If there are no more valid rules the end of the span is set to the last address in the address space. This process is repeated until all the address spans are found.

```
int lastIndex = -1;
for (int i = 0; i < validRules.Count() + 1; i++)
{
  // Start the span
  var firstIP = addressSecurityStatus.FirstOrDefault(x =>
  ((!validRules.Contains(x)) && addressSecurityStatus.IndexOf(x) > lastIndex)).Key;

  // End the span
  if (i < validRules.Count())
          lastIndex = addressSecurityStatus.IndexOf(validRules.ElementAt(i)) - 1;
  else
          lastIndex = addressSecurityStatus.Count() - 1;

  var lastIP = addressSecurityStatus.ElementAt(lastIndex).Key;
  // Convert the IP spans to the least amount of spans with CIDR
  foreach (var cidrSpan in IPSpanToCidr(firstIP, lastIP))
  {
    // Deny address block
    subnet.NetworkSecurityGroup.Rules.Add( ... );
  }
}
```

Listing 12: Identifying IP address spans

Each span of IP addresses is converted to a new set of addresses of CIDR notation. These addresses are added to the NSG as new rules blocking the given address ranges.

To convert the range of IP addresses to a set of CIDR addresses we convert the first and last address in the span to a bit representation to be able to operate on the addresses

in binary form. To find the largest CIDR address in the given span we create a mask representing the CIDR number, also in bit form. To find out the largest span the current start address can represent we iterate, reducing the CIDR by one each iteration. During each iteration a bitwise AND operation is performed on the start address and the mask comparing its equivalence with the start address.

```
byte cidr = 32;
while (cidr > 0)
{
  long mask = (long)(Math.Pow(2, 32) - Math.Pow(2, (32 - cidr - 1)));
  if ((startAddr & mask) != startAddr)
          break;

  cidr--;
}
```

Listing 13: Finding minimum CIDR for start address

This operation is used to find where the point where the start address is no longer the first address in the address space defined by the CIDR. To give an example, the start address of 10.0.0.4 and a CIDR of 31.

Address	**Operator**	**CIDR**
00001010 00000000 00000000 00000100	AND	11111111 11111111 11111111 11111110

Table 6.1: Allowed binary CIDR address operation

This operation results in the exact same address as the start address. However, when the CIDR is reduced to 29 we get the following

Address	**Operator**	**CIDR**
00001010 00000000 00000000 00000100	AND	11111111 11111111 11111111 11111000

Table 6.2: Faulty binary CIDR address operation

This results in that the last address block, corresponding to 4, is distorted and is no longer representing our start address. It is therefore not possible to reduce the CIDR any further and still have the start address be the first address in the CIDR range.

When the lowest CIDR number for the start address has been found the result must be validated against the end address of the span. The maximum CIDR number is calculated using logarithmic functions and if the current CIDR is lower than the maximum CIDR number between the start address and the end address the maximum CIDR value is used instead. For example, when the start address is 10.0.0.0 the CIDR number calculated first can go as low as 7 but if the end address is also 10.0.0.0 this would be too low so it is corrected to 32 by this second calculation.

```
1  double x = Math.Log(endAddr - startAddr + 1) / Math.Log(2);
2  byte cidrMax = (byte)(32 - Math.Floor(x));
3  if (cidr < cidrMax)
4  {
5          cidr = cidrMax;
6  }
```

Listing 14: Validate start CIDR against CIDR max

Both of these calculations are presented within a loop that iterates until the entire span of addresses is covered, producing multiple CIDR addresses if necessary.

```
while (endAddr >= startAddr)
{
        ... // minimum start address CIDR calculation
        ... // validate start address against max CIDR calculation

        startAddr += (long)Math.Pow(2, (32 - cidr));

        yield return ip + "/" + cidr;
}
```

Listing 15: Converting IP range to CIDR block

To visualize the process with an example, we have an address space of 10.0.0.0/28, utilized by resources shown in Figure 6.8 and the user has defined rules allowing the addresses shown in 6.9.

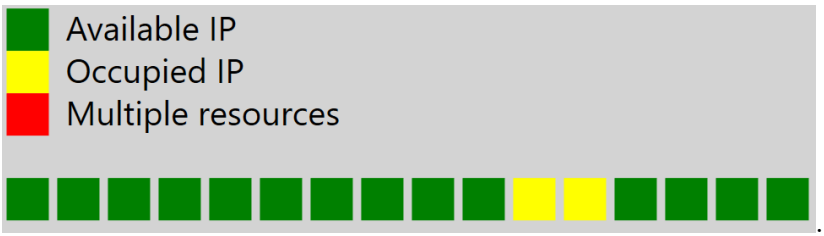

Figure 6.8: Example - Utilization

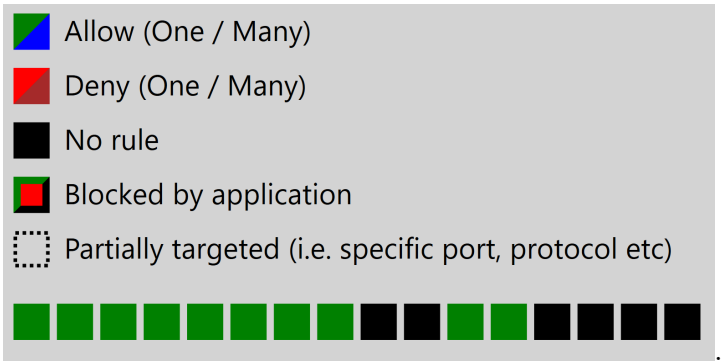

Figure 6.9: Example - User defined ACL rules

The only addresses that should be allowed are the addresses that are utilized by resources in the network. Since the user has an ACL entry allowing other addresses than those that are utilized by resources in the network the application will block those along with the addresses not targeted by any rule. The resulting ACL can be observed in Figure 6.10.

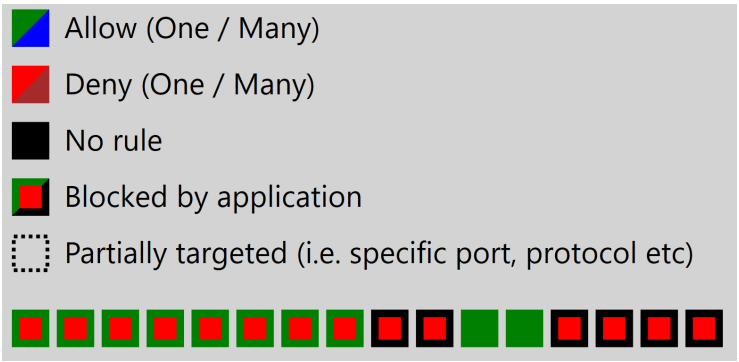

Figure 6.10: Example - After ACL generation

7 Prototype Design

Azure Designer is an application with the goal to assist with the creation and management of Azure resources and to quickly be able to design Azure components and deploy them to Azure. The goal is to help streamline the process of creating resources and provide a graphic interface to observe and generate ACL rules that minimize overlap and the amount of entries in the ACL while maximizing security.

The resources should be visualized in a graph and selected properties of these resources should be exposed for the user to configure. The graphs should be treated as documents and the application should have the possibility to save and load these documents.

The application must be fully aware of all the resources, their configurations and how they are connected to each other. This awareness gives the application the information it requires to make judgments on how the ACL and the network should be configured in return.

The application is divided into two projects, the *Deploy project* and the *Designer project*. The separation of these projects has the benefit of the possibility to reuse one or both of the projects at a later time for other projects. Each project has a separate model that contains the resources and their data. The data must be able to be converted between the models by the application.

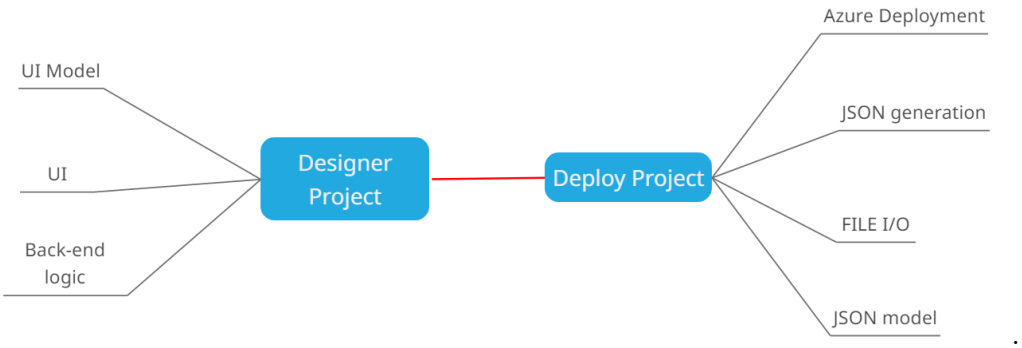

Figure 7.1: Design overview

7.1 Deploy project

The *Deploy project* is responsible for all things regarding exchanging information with Azure, JSON generation and general file management.

7.1.1 Model

The model in the deploy project is very deep in its structure and has many layers of objects by design. The purpose of the model is to represent the Azure JSON template structure in C# code. It acts as a container for all the information that can pass in or out of the application. It also functions as a middle ground between the *Designer project* and the JSON template. To allow for conversion between the application and JSON template, the model must emulate the template structure. By doing this, conversion between JSON and the model can be as simple as using a one line function call to a JSON library.

7.1.2 JSON serialization and deserialization

To generate the JSON string, a JSON library is used to convert the model to its JSON equivalent. Since the model is already structured as a replica of the template we are trying to target the process of serializing the model to JSON string is quite straight forward and it is simply up the library to convert the model. If there are properties that should not exist in the template but might exist in the model, most libraries allow us to mark these properties and for the serializer to ignore them.

However, to deserialize the template file into the model have some additional challenges compared to serialization. The deserializer need some assistance to be able to match the correct resource in the template with the correct resource in the model. Since each resource is identified by type in the pattern of $Microsoft.\{Category\}/\{ResourceType\}$ we can supply the deserializer with this information and guide it to the correct resource in the model.

7.1.3 File management

The deploy project is responsible for reading and saving files. To make the application more versatile, and be able to read previously created templates, the application uses the exact same format as the templates utilized by Azure. The drawback to this approach is that additional information cannot be stored in the file, such as the name of the resource groups or the location of the credentials needed for deployment.

When reading a file the model is populated with the contents of the file. The model is then converted to the model of the Designer project and the user is then able to modify the resources from the interface and vice versa for saving the project.

7.1.4 Azure Deployment

Another responsibility of the deployment project is to deploy the template file to Azure. Deployment takes place in four steps, conversion from the Designer Model to the Deploy model, generation of the JSON string, saving the JSON string to a template file and finally deploying the template to Azure.

To be able to deploy the resources the user must supply their credentials to their Azure subscriptions. These credentials are used to authenticate against Azure.

7.2 Designer project

The *Designer project* controls the graphical interface as well as the logic behind creating new resources and their properties. This project was designed with the model-view-view model (MVVM) architectural pattern in mind.

7.2.1 MVVM

MVVM allows for separation of concerns, i.e. separation of the UI controls and the back-end logic behind them. The MVVM patterns have three components, the Model, the view

and the View Model. These are separated so that the view is unaware of the Model and the View Model is unaware of the view, this allows the model to be developed without much concern for the view. [9]

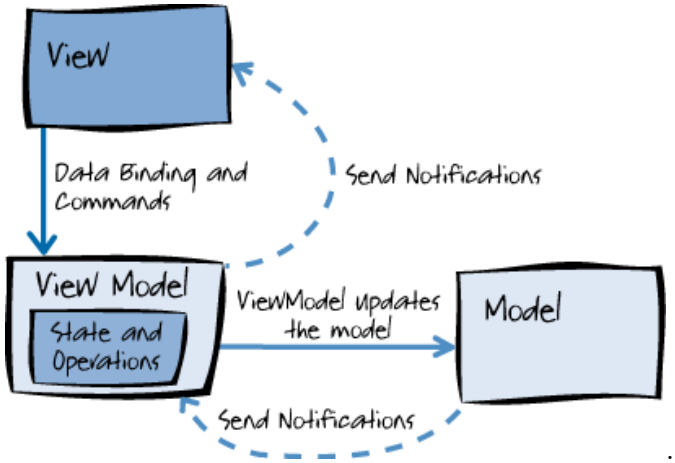

Figure 7.2: MVVM overview

Credit: *Microsoft, https://msdn.microsoft.com/en-us/library/hh848246.aspx*

7.2.2 Model

The model for the Designer project does not have to follow the structure of the JSON template, this makes it flatter to better accommodate the relation between the Model and the View Model. The purpose of the Model is to store data, validate the data and notify the View Model about changes.

7.2.3 View model

The purpose of the view model is to act like an intermediary between the model and the view and it is responsible for updating the model and other operations requested by the view. The view model typically interacts with the model by using methods defined in the model and later provides the data from the model to the view in a simpler way so that the view can interpret and be displayed to the user. The view model also implements the

commands defined in the view, for example, when the user presses a button in the view, this may trigger a command in the view model that implements some logic. [9]

7.2.4 View

The purpose of the view is to display the interface to the user. When using MVVM the view is ideally almost exclusively defined in XAML code with limited 'code-behind' that may not contain business-logic.

7.2.5 Graph

The graph is central in the UI of the application. The Graph represents the resources that are currently active in the document and their connections. The graph displays the resources and selected properties such as name and address, but the majority of the properties should not be modified within the graph itself. The user should instead be directed to another part of the GUI to modify and observe all the properties within a resource.

The user should be able to create connections between resources within the graph interface and visual notifications such as errors and warning with the configuration should also be displayed in the graph.

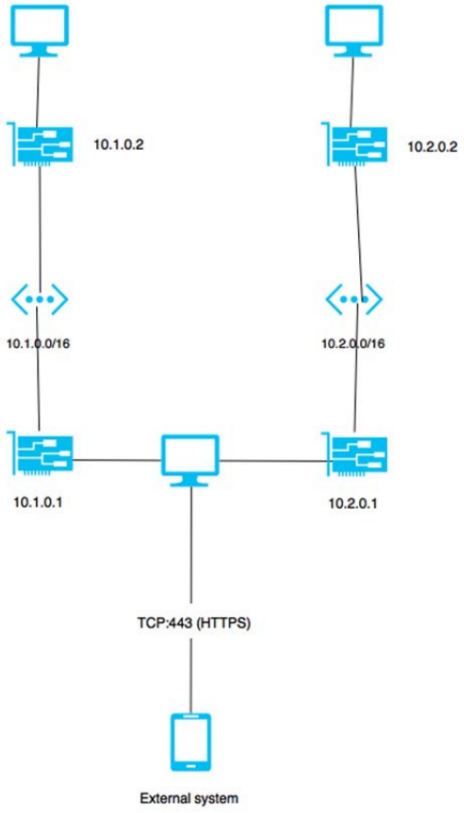

Figure 7.3: Graph design

Figure 7.3 gives a example of how the resources might be presented in the graph. A virtual machine with multiple network cards connects the two networks and acts as a intermediary between the other virtual machines and the external system.

7.2.6 Interface

In addition to having a graph displaying the resources a toolbox should be available, displaying all available resources and allowing the user to drag and drop them into the graph. When a resource is dropped into the graph the corresponding resource is automatically created and stored in the model and the user is able to operate on the resource in the graph. A property grid should be available, displaying detailed information about the currently

selected resource, allowing the user to fully customize the resource and its properties.

7.2.7 Naming conventions

When naming the resources in Azure it can sometimes be hard to keep track of and be consistent with following naming conventions defined by Microsoft. The application aims to help in this process by automatically apply the correct names for the resources based on the project name defined by the user. The project name is propagated to the resources and the view model can update their names accordingly.

Naming convention in Azure is especially important because names can be very difficult to change after the fact and the names must meet the requirements for the specific type of resource [1].

Entity	**Length**	**Casing**	**Pattern**
Resource Group	1-64	Case insensitive	<service short name>-<environment>-rg
Virtual Machine	1-15	Case insensitive	<name>-<role >-vm<number>
Virtual Network	2-64	Case insensitive	<service short name>- [section]-vnet
Subnet	2-80	Case insensitive	<descriptive context>
Network Security Group	1-80	Case insensitive	<service short name>-<context>-nsg

Table 7.1: Naming conventions

In Table 7.1 we can observe some examples of prominent resources and their naming conventions [1]. The application will not define strings like 'name' and 'service short name'

but it can apply the substrings in the correct pattern according to the naming convention and prompt the user to enter the required fields.

7.2.8 Error detection

There is a need for two types of error detection in the application. The main error detection the application caries out is to make sure the defined address space of a resource is valid. There is two types of invalid addresses, one if the parent network does not contain the whole address space defined in the resource and if there are resources in the parent network that have overlapping address spaces.

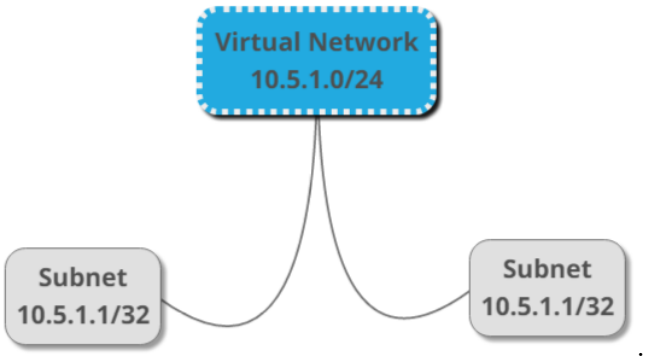

Figure 7.4: Overlapping address spaces

Figure 7.4 shows an example of overlapping address spaces that should produce an error by the application. The address spaces defined by the subnets is technically correct, because their addresses are contained within the virtual network, but both subnets implement the same address space and the traffic directed to the virtual network with a destination address of 10.5.1.1 can go to either subnet.

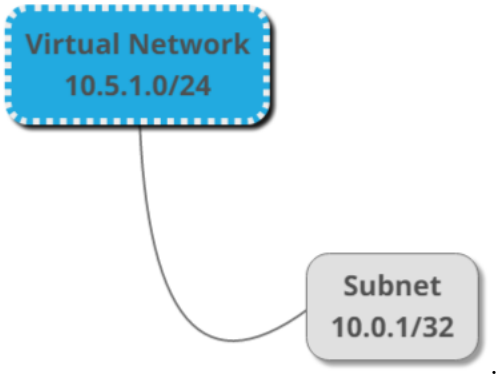

Figure 7.5: Invalid address space

In Figure 7.5 we can observe the other type of error that can occur. In this case there is only one defined subnet, but the subnets address space is not contained in the virtual network hence causing the address to be invalid.

These errors would cause Azure to fail the deployment and by having these error checks done in the application saves time and resources for the user trying to locate the error within the network.

The other form of error detection the application does is to check for overlapping ACL entries. An overlapping ACL entry means that multiple entries target the same addresses, but may differ in whether they allow or deny the traffic. Although overlapping entries are technically allowed by Azure and quite a common practice in many ACLs, the user will receive a warning if a resource has overlapping ACL entries. This is mainly to let the user know that the ACL is probably not optimal and should be modified.

7.2.9 ACL and utilization views

Each subnet will have two additional views available to the user. These views serve the purpose of giving the user a quick overview of which addresses in the address space are utilized by connecting resources and how the ACL entries are affecting the addresses.

In the utilization view, each distinct address in the address space is represented by

a shape and color indicating by if it is utilized or not. The user can then get a quick overview of what addresses are available to add additional resource to, they can even see where multiple resources are connected to the same address. The addresses will be displayed in a grid with 64 addresses on each row.

The ACL view will look similar to the utilization view, but instead of showing the utilization it will display how each distinct address is affected by the entries in the ACL. The user will get a quick overview of how the rules they implement are affecting the network and potential problems can be discovered.

8 Prototype implementation

In this chapter, we showcase the implementation of the prototype application. Snippets of code are provided to observe the implementation of the design.

8.1 Deploy project

In this section, we show the implementation of the model, the serialization process and how the deployment to Azure is implemented.

8.1.1 Model

Since the model is a reflection of the Azure template, the root of the model is the resource group. The resource group contains a collection the resources and also defines the schema and content version.

```
1  public class ResourceGroup
2  {
3    [JsonProperty("$schema", Order = -2)]
4    public string Schema { get; set; }
5
6    [JsonProperty("contentVersion", Order = -2)]
7    public string ContentVersion { get; set; }
8
9    [JsonProperty("resources")]
10   public Collection<Resource> Resources { get; set; }
11 }
```

Listing 16: Resource base class

The Azure resources defined in the model implement a base class, 'resource'. This class contains the common properties shared by all resources, defined by the Azure templates. All resources inherit from this class.

```
public abstract class Resource
{
  /*
  ...
  */

  [JsonProperty("apiVersion")]
  public string ApiVersion { get; set; }

  [JsonProperty("type")]
  public string Type { get; set; }

  [JsonProperty("name")]
  public string Name { get; set; }

  [JsonProperty("location")]
  public string Location { get; set; }

  [JsonIgnore]
  public IEnumerable<Resource> Dependencies
  {
          get { return _dependencies; }
  }

  /*
  ...
  */
}
```

Listing 17: Resource base class

The class defines the Api version used to deploy the resource, the type of resource, the name of the resource and the geographical location of the resource. The attributes defined above the properties tell the JSON serializer how to address these resources during serialization. The 'JsonProperty' attribute declare what the serializer should name the properties and 'JsonIgnore' declare that the serializer should ignore that property and exclude it from the resulting JSON string.

Other than the properties defined in the base class, resources implement properties unique to their resource.

8.1.2 JSON serialization

To generate and read the JSON files we use the *Newtonsoft.Json* library. This library serializes the resources into a JSON string that is later saved into a JSON file automatically, and due to the fact that the resources are modeled after the target JSON structure, this process becomes straight forward.

To observe the process, we can use the following example:

```
Resource resource = new VirtualNetwork();
resource.Name = "myVN1";
resource.Location = "EastAsia";
resource.Subnets = new Subnet[] { new Subnet("mySubnet1") };

// ...

string json = JsonConvert.SerializeObject(resource);
```

Listing 18: Newtonsoft C

The code in Listing 18 produces the following JSON string

```
{
   "Name": "myVN1",
   "Location": "EastAsia",
   "Subnets": [
     {
      "Name": "mySubnet1"
     }
   ]
}
```

Listing 19: Newtonsoft JSON

If a property is not assigned by the user and hence is null, the JSON serializer ignores that property from the resulting JSON string.

The serializer takes a resource group as a parameter and serializes one resource at a time until the template is complete.

8.1.3 JSON deserialization

Deserialization becomes a bit more complicated. The default deserializer cannot tell the difference between the resources in the template, hence it cannot identify the correct resource in the model to convert the resource to. To assist the deserializer in this task we supply a custom deserializer matching the string implementing the type defined in the resource template with the corresponding resource in the model

```
if (FieldExists("type", jObject))
{
        switch (jObject["type"].Value<string>())
    {
            case "Microsoft.Compute/virtualMachines":
                return jObject.ToObject<VirtualMachine>();

        case "Microsoft.Network/networkInterfaces":
            return jObject.ToObject<NetworkInterface>();

        case "Microsoft.Network/publicIPAddresses":
            return jObject.ToObject<PublicIPAddress>();

        case "Microsoft.Storage/storageAccounts":
            return jObject.ToObject<StorageAccount>();

        case "Microsoft.Network/virtualNetworks":
            return jObject.ToObject<VirtualNetwork>();

        case "Microsoft.Network/networkSecurityGroups":
                return jObject.ToObject<NetworkSecurityGroup>();

        }
}
```

Listing 20: Custom deserializer

With the help of the custom deserializer it is possible to convert the template back into the model.

8.1.4 Deployment

The deployment of resources is done with a PowerShell script. Powershell has great support for Azure, it is possible to manage and deploy templates directly from the PowerShell terminal. PowerShell was chosen over the Azure C# API due to time constraints and ease of use, but the application might migrate over to the Azure REST API in the future.

When the user wants to deploy to Azure, the user will be asked to provide credentials to their Azure subscription. The user must provide their user profile, usually saved in JSON format, which contains the authentication token needed for the application to authenticate against, and deploy the resources to Azure. To obtain the user profile the user have to login to Azure via PowerShell or another method provided by Azure. After the user have logged in to Azure, depending on login method, there is a possibility to save the user profile. The profile is valid for a certain period of time, usually around a month and can be used to deploy resources during that period without renewing the profile. The application uses this profile as a parameter passed on to the PowerShell script that is responsible for deploying the resources.

```
1  if (File.Exists(psFile))
2          psScript = File.ReadAllText(psFile);
3  else
4          throw new FileNotFoundException("Could not locate the powershell script.");
5
6
7  PowerShell powerShellInstance = PowerShell.Create();
8  powerShellInstance.AddScript(psScript);
9  powerShellInstance.AddParameter("deployment", deployment);
10 powerShellInstance.AddParameter("resourcegroup", resourceGroup);
11 powerShellInstance.AddParameter("template", templatePath);
12 powerShellInstance.AddParameter("userProfile", userProfile);
13 powerShellInstance.Invoke();
14
15  return powerShellInstance.HadErrors;
```

Listing 21: Creating the powershell instance

The first step of the deployment requires that the PowerShell script is read and added to the PowerShell instance. The parameters such as the name of the deployment, the resource group, the path of the template and the user profile are supplied as parameters to the instance.

```
New-AzureRmResourceGroupDeployment -Name $deployment -
ResourceGroupName $resourcegroup -TemplateFile $template
```

Listing 22: Powershell script

The PowerShell instance will supply the 'HasErrors' property stating if an error occurred during the deployment process.

8.2 Designer project

In this section we describe the implementation of the designer project with its model and graphical interface visible to the user.

8.2.1 Graph

The implementation of the graph is done with the open source version of the GraphX for .NET library. Graphx is a graph layout and visualization library having the capability to build and generate graphs using multiple different layout algorithms. GraphX allow for good customization and have good user controls allowing the user to zoom and drag vertices in the graph area which is ultimately why GraphX was chosen for this project.

8.2.2 Model

All resources implement the 'IResource' interface defining shared properties of the resources such as the 'Name' and 'Location' properties and resources directly connected to the network also implement the 'INetwork' interface. The 'INetwork' interface is used to distinguish resources that have a exposed network address and is useful when filtering resources and retrieving their address space. The classes in the model that are allowed to be vertices in the graph implement the 'Vertex' attribute and inherit from the 'AzureVertex' class.

```
[Vertex]
public class VirtualMachine : AzureVertex, IResource, INetwork
{
...
}
```

Listing 23: Model - Virtual machine

The classes in the model can notify other classes of changes to properties by implementing the event 'PropertyChangedEventHandler'. Other classes can subscribe to this event to get notified when the properties change. By inheriting from the 'BindableBase' class, defined in the 'Prism.Mvvm' namespace, properties can use the 'SetProperty' in the setter field and automatically trigger the event when the property is modified.

```
public string Name
{
  get { return _name; }
  set { SetProperty(ref _name, value); }
}
```

Listing 24: BindableBase

This is particularly useful in the case where the resource group name is updated and all resources must update their name to follow the naming conventions. The view model subscribes to the PropertyChangedEventHandler event that triggers when the resource group name is updated and the view model can take action to update all the other resource names accordingly.

8.2.3 View Model

The view model acts as an intermediary between the model and the view. The main purpose of the view model is to help the view perform operations on the graph.

The view model implements 'RelayCommands' allowing the view to bind actions in the view with commands in the view model. In Listing 25 the command for deleting vertices from the graph is displayed. If the condition is met, in this case if a vertex is selected, the method defined in the relay command is executed.

```
public RelayCommand DeleteVertexCommand
{
  get
  {
    return _deleteVertexCommand
    ?? (_deleteVertexCommand = new RelayCommand(DeleteVertex_impl, ()
    => GraphContainer.SelectedVertex != null));
  }
}
```

Listing 25: RelayCommand

All actions performed on the model from the view must be handled by a 'RelayCommand' and dealt with in the view model.

The view model stores the model in the form of the 'GraphContainer' class, containing the graph and graph related operations.

8.2.4 View

In good MVVM fashion, the view is mostly implemented with XAML. Most of the GUI is produced using the 'AvalonDock' library, allowing for dockable panels making the GUI more intractable and customizable.

The three main components of the view is the tool box, the graph area and the property grid. The tool box displays the available resources that the user can add to the graph. The graph area is displaying the graph defined in the view model and allows the user to view the graph visually and interact with its content. The property grid displays and allows the user to modify the currently selected resource's data.

By implementing the 'Vertex' attribute for classes in the model and using reflection, it is possible to locate the resources that are allowed in the graph and add them to the tool box for the user to drag and drop into the graph. Reflection is functionality in C# allowing for retrieval of attribute information at runtime. This functionality allows us to locate all classes implement the vertex attribute and populate the tool box with these classes.

```
foreach (var vertex in AttributeHelper
.GetVertexAttributeTypes(Assembly.GetExecutingAssembly()))
{
  ToolBox.Items.Add( new ListResourceItem { ... } );
}
...
public static IEnumerable<Type> GetVertexAttributeTypes(Assembly assembly)
{
  foreach (Type type in assembly.GetTypes())
  {
    VertexAttribute[] obj = (VertexAttribute[])type
    .GetCustomAttributes(typeof(VertexAttribute), true);
    if (obj.Length > 0)
    {
            yield return type;
    }
  }
}
```

Listing 26: Retrieving the classes implementing vertex attribute

The types of the classes implementing the vertex attribute are stored in the tool box and when the user drop a new resource into the graph a new instance of that resource can be added.

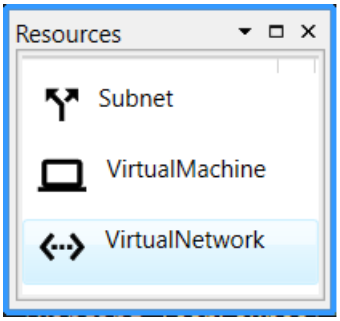

Figure 8.1: Resources added to the tool box

Through a binding to the view model, the selected objects properties are bound to the resource property grid.

```
1 <xctk:PropertyGrid x:Name="ResourcePropertyGrid" NameColumnWidth="110"
2 SelectedObject="{Binding GraphContainer.SelectedVertex}" />
```

Listing 27: Resource property grid binding

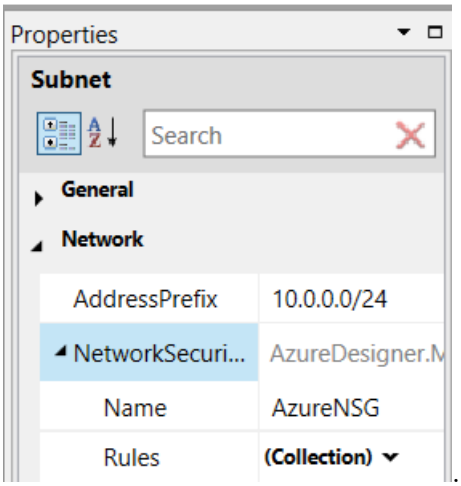

Figure 8.2: Property grid

The graph area is defined within a zoomable canvas, allowing the user to position the camera and to zoom in and out in the graph area. The graph area has a similar binding

to the view model and all put together produces the full interface observed in Figure 8.3

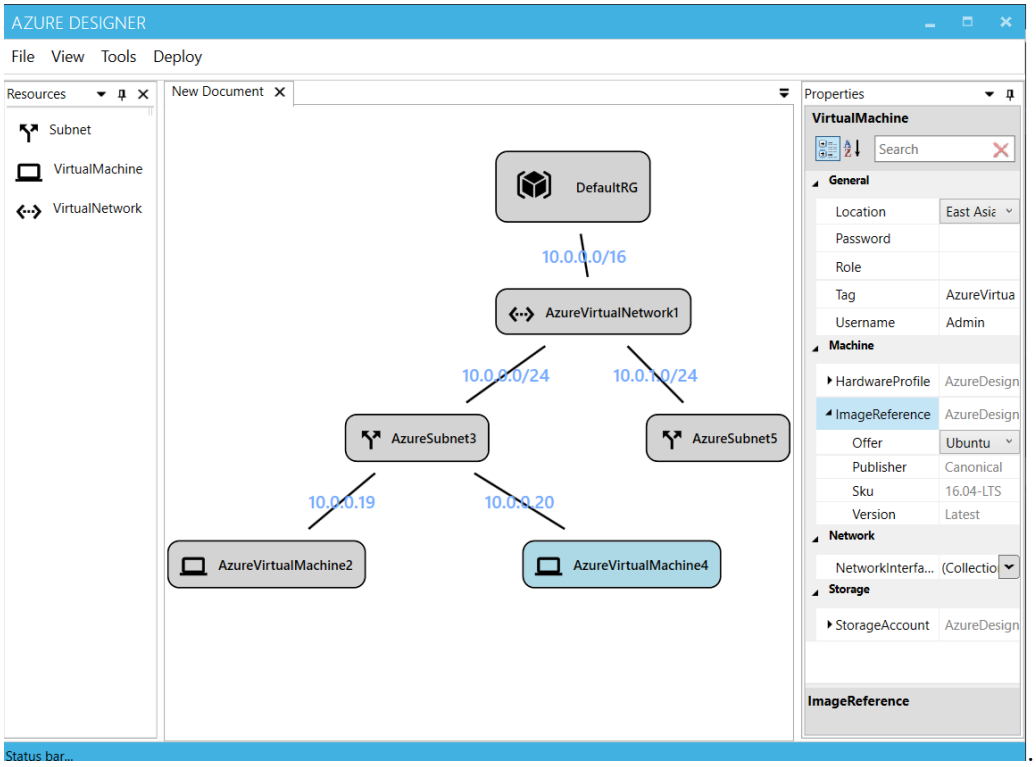

Figure 8.3: Interface

8.2.5 Error detection

Each time the network properties of a resource are updated it triggers a error detection check. When dealing with IPNetworks, addresses and subneting the IPNetwork C# library are of great help. The IPNetwork library contain utility functions for checking for overlapping address spaces, creating subnets and many other helpful tools.

There are mainly two types of error checks produced by the application. The first one checks for invalid addresses in the resources.

```
foreach (var resource in GraphContainer.GetEdgeTargets(subnet))
{
  if(resource is INetwork)
  {
    var networkResource = resource as INetwork;
    var overlapCount = GraphContainer.GetEdgeTargets(subnet)
    .Where(y => y is INetwork)
    .Count(x => ((INetwork)x).Address(subnet) == networkResource.Address(subnet));

    if (overlapCount > 1)
    {
      resource.Error = ErrorTypes.Error;
      GraphContainer.GraphArea.VertexList[resource].ToolTip += "Error: Another "
      + "resource with this IP address is already defined in this subnet. \n";
    }

...
```

Listing 28: Subnet - checking for invalid addresses

The resources the subnet is connected to is retrieved from the graph and one by one evaluated. The first check, shown in Listing 28 looks for duplicate addresses of the connected resources, producing an error if duplicates are found. Each connected resources address is determined and is evaluated against each other. If there is more than 1 of the given address, the resources 'Error' property is set; producing an error in the graph, and the 'ToolTip' for the resource is set explaining the error for the user.

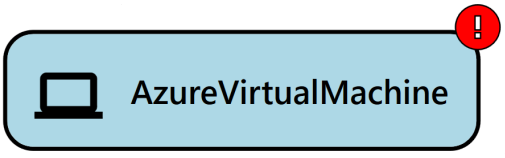

Figure 8.4: Graph error notification

```
...
    IPAddress networkResourceAddr;
    IPAddress.TryParse(networkResource.Address(subnet), out networkResourceAddr);
    if (networkResourceAddr != null && !IPNetwork
    .Contains(IPNetwork.Parse(subnet.Address()), networkResourceAddr))
    {
      resource.Error = ErrorTypes.Error;
      GraphContainer.GraphArea.VertexList[resource].ToolTip += "Error: The defined IP"
      + " address is not a subset of the IP addresses in this subnet. \n";
    }
  }
}
```

Listing 29: Subnet - checking for invalid addresses

The second part of the method checking for invalid addresses, shown in Listing 29, check to see that the address is within the address space defined for the subnet using the helper method 'Contains', defined in the IPNetwork library.

The second check made by the application is looking for overlapping ACL rules in network security groups.

```
var rules = subnet?.NetworkSecurityGroup?.Rules;
for (int i = 0; i < rules.Count - 1; i++)
{
  var currRule = rules.ElementAtOrDefault(i);

  IPNetwork currDest = null;
  var currParsed = IPNetwork.TryParse(currRule.DestinationAddressPrefix, out currDest);
  for (int j = i + 1; j < rules.Count; j++)
  {
    var nextRule = rules.ElementAtOrDefault(j);

    IPNetwork nextDest = null;
    var nextParsed = IPNetwork.TryParse(nextRule.DestinationAddressPrefix,
            out nextDest);

    if(currParsed && nextParsed && IPNetwork.Overlap(currDest, nextDest))
    {
      if (subnet.Error < ErrorTypes.Warning)
              subnet.Error = ErrorTypes.Warning;

      GraphContainer.GraphArea.VertexList[subnet].ToolTip += "Warning: Multiple "
      + "security rules for address:"
      + currRule.DestinationAddressPrefix + "\n";
    }
  }
}
```

Listing 30: Subnet - Overlapping ACL rules

Each rule in the network security group is evaluated against the other rules and produces a warning if overlapping rules are discovered. If the subnets 'Error' property already contain an error type more serious than a warning the error status remain and the warning message is simply added to the 'ToolTip' property, displaying the warning to the user.

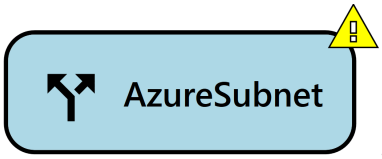

Figure 8.5: Graph warning notification

8.2.6 ACL and utilization view

The ACL and utilization views allow the user to get a visual overview on the distribution of resources and ACL entries in a subnets address space. This is a good way for the users to determine what addresses are available for new resources and how the ACL entries affect the security of the subnet.

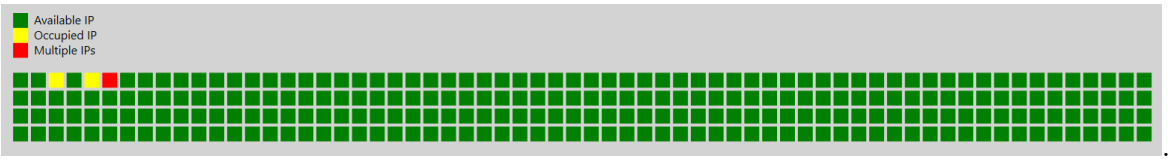

Figure 8.6: Utilization view

The utilization view, displayed in Figure 8.6, indicates that the third and fifth address of the given address space is utilized by one resource and the sixth is occupied by multiple resources causing a conflict. If the user hovers over any of the address representations the full address will be displayed.

The ACL view shown in Figure 8.7 displays, in the same manner as the utilization view, the full address space of the subnet implementing the ACL. Each square represents an address in the address space. The green squares are addresses currently allowing traffic from all sources; red squares display addresses blocked by all sources. If the border of the square is dotted it mean that the address is partially targeted, i.e. there are a certain port range or protocol targeted in that address. There are also indicators whether multiple rules are targeting the address by shifting the color to either blue if it is allowed or brown if it

is denied by the highest priority rule. There is also a special indication if the address has been blocked by an auto generated rule in the form of a red square with a solid colored border.

Figure 8.7: ACL view

9 Conclusion

One of the goals of this thesis was to describe the design and implementation of a new application for deploying cloud resources in Microsoft Azure. The main focus for this application is to make the design of Azure systems faster and more secure. By aiding the user in securing the intra-network firewall using an algorithm described in the thesis, with the goal to detect and mend security holes in the firewall, the application should provide an automatic process of securing the network by generating new firewall rules.

We proposed an algorithm to detect security holes in the firewall and mend them by generating new firewall rules. The algorithm could be implemented fully in the prototype and serves its purpose of further securing the intra-network of the Azure system. Additional views were implemented in the prototype allowing the user to get an overview of the address space utilization as well as firewall coverage, reducing the need for manual inspection in the ACL.

With the help of the IPNetwork C# library, the functionality of detecting faulty network configurations could be implemented in the prototype. The prototype is capable of detecting resources defined outside of its address space and if multiple resources are using the same address space, providing the user with quick feedback and fast iteration time on faulty network configurations. The prototype notifies the user with an error message on the resources in the graph when an error has been detected. The prototype also produces notifications for shadowing firewall rules, prompting the user to manually resolve the conflict.

The implementation of the prototype could be implemented fully. The resources that the application currently supports are Virtual Machines, Network interfaces, Storage accounts, Public IP addresses, Subnets, Network security groups and Virtual networks. The prototype allows users to quickly design Azure systems and deploy them to Azure. It also contains a feature to allow the user to save and load templates, making the prototype function like a file manager. Deployment to Azure were achieved using PowerShell scripts,

in retrospect, using the Azure API would probably be the preferred solution due to being able to communicate directly to Azure, retrieving possible errors during deployment and possibly a simpler authentication process.

9.1 Future work

For future work, more resources can be added to the application expanding its capabilities to handle the full range of resources offered by Azure. The method of deployment will most likely move away from using PowerShell scripts and instead make use of the API made available by Azure since it may provide better feedback on faulty deployments and give the user a better understanding why a deployment has failed. The firewall assistant could be extended to provide more features such as automatically resolve shadowing rules.

References

[1] Azure naming conventions. https://docs.microsoft.com/en-us/azure/architecture/best-practices/naming-conventions. Accessed: 2017-04-26.

[2] Azure quickstart templates. https://github.com/Azure/azure-quickstart-templates. Accessed: 2017-04-18.

[3] Azure resource manager overview. https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-overview. Accessed: 2017-04-04.

[4] Azure resource manager template visualizer (armviz). http://armviz.io/designer. Accessed: 2017-05-09.

[5] Azure virtual network overview. https://docs.microsoft.com/en-us/azure/virtual-network/virtual-networks-overview. Accessed: 2017-04-13.

[6] Creating and deploying azure resource groups through visual studio. https://docs.microsoft.com/en-us/azure/azure-resource-manager/vs-azure-tools-resource-groups-deployment-projects-create-deploy. Accessed: 2017-04-10.

[7] Deploy resources with resource manager templates and resource manager rest api. https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-template-deploy-rest. Accessed: 2017-04-18.

[8] Filter network traffic with network security groups. https://docs.microsoft.com/en-us/azure/virtual-network/virtual-networks-nsg. Accessed: 2017-04-23.

[9] The mvvm pattern. https://msdn.microsoft.com/en-us/library/hh848246.aspx. Accessed: 2017-04-25.

[10] Yair Bartal, Alain Mayer, Kobbi Nissim, and Avishai Wool. Firmato: A novel firewall management toolkit. In *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*. IEEE, 1999.

[11] Sushil Bhardwaj, Leena Jain, and Sandeep Jain. Cloud computing: A study of infrastructure as a service (iaas). *International Journal of engineering and information Technology*, 2(1), 2010.

[12] Marshall Copeland, Julian Soh, Anthony Puca, Mike Manning, and David Gollob. Microsoft azure and cloud computing. In *Microsoft Azure*, pages 3–26. Springer, 2015.

[13] Frédéric Cuppens, Nora Cuppens-Boulahia, and Joaquin Garcia-Alfaro. Detection and removal of firewall misconfiguration. In *Proceedings of the 2005 IASTED International Conference on Communication, Network and Information Security*, volume 1, 2005.

[14] Jeremy Deyo. Software as a service (saas), 2008.

[15] Sergio Pozo, Rafael Ceballos, Rafael M Gasca, and Angel Jesus Varela-Vaca. Fast algorithms for local inconsistency detection in firewall acl updates. In *Emerging Security Information, Systems and Technologies, 2008. SECURWARE'08. Second International Conference on*. IEEE, 2008.

[16] Jiang Qian, Susan Hinrichs, and Klara Nahrstedt. Acla: A framework for access control list (acl) analysis and optimization. In *Communications and Multimedia Security Issues of the New Century*, pages 197–211. Springer, 2001.

[17] Kirit Modi Yashpalsinh Jadeja. Cloud computing-concepts, architecture and challenges. In *Cloud Computing - Concepts, Architecture and Challenges*. IEEE, 2012.

[18] Shucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou. Achieving secure, scalable, and fine-grained data access control in cloud computing. In *Infocom, 2010 proceedings IEEE*, pages 1–9. Ieee, 2010.