

Eemil Aikio

Developing a Virtual Appliance to Simulate Broken Networks

An Open-Source, Infrastructure-As-Code Virtual Appliance

Developing a Virtual Appliance to Simulate Broken Networks

An Open-Source, Infrastructure-As-Code Virtual Appliance

Eemil Aikio
Bachelor's Thesis
Spring 2017
Tietojenkäsittely (Business Information
Technology)
Oulu University of Applied Sciences

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietojenkäsittely, Järjestelmäasiantuntemus

Tekijä: Eemil Aikio

Opinnäytetyön nimi: Rikkinäisten verkkojen simulointi palvelinsovelluksella

Työn ohjaaja: Jukka Kaisto

Työn valmistumislukukausi- ja vuosi: kevät 2017

Sivumäärä: 37

Opinnäytetyön tavoite on kehittää sovellus, joka simuloi rikkinäistä verkkoyhteyttä. Aihe tuli toimeksiantajalta, HowNetWorks Oy:ltä. HowNetWorks on suomalainen startup-yritys, joka on kehittämässä verkon testaustyökaluja ominaisuuksille mitä perinteiset testit eivät testaa.

Sovelluksen nimi on hnwProxy. Kyseessä on virtual appliance, eli käyttövalmis virtuaalipalvelin jonka kuka tahansa saa vapaasti ladattua. Tämä löytyy GitHub:sta osoitteesta github.com/hownetworks/hnwproxy. hnwProxy on kehitetty infrastruktuuri koodina -menetelmillä avoimen lähdekoodin lisenssillä. Infrastruktuuri koodina on uusi ajattelutapa palvelininfrastruktuurin hallinnassa, jossa pyritään käyttämään automaatiotyökaluja ja ohjelmistokehityksestä tuttuja menetelmiä järjestelmän laadun parantamiseksi.

Teoriaosuudessa esitellään verkkojen laatutekijät sekä infrastruktuuri koodina. Tässä ensimmäisessä luvussa pyritään selvittämään mitkä asiat vaikuttavat verkkoyhteyden laatuun, eli mitä hnwProxy tulee simuloimaan. Infrastruktuuri koodina -luvussa on käytössä aihetta laajasti kattava, Kief Morrisin kirjoittama, *Infrastructure as Code* kirja lähteenä. Verkkojen Laatutekijät -luvussa käytetään monta eri lähdettä, näistä kattavin on Kurose & Rossin *Computer Networking: A Top-Down Approach*.

Lopputulos on palvelinjärjestelmä joka täyttää projektin alussa asetetut vaatimukset. hnwProxy pystyy simuloimaan erilaisia rikkinäisiä verkkoja, ja sitä voidaan käyttää useammassa eri virtuaalisointialustassa.

Asiasanat: Infrastruktuuri koodina, tietoverkot, virtualisointi

ABSTRACT

Oulu University of Applied Sciences
Business Information Technology, Option in System Administration

Author(s): Eemil Aikio

Title of Bachelor's thesis: Developing a Virtual Appliance to Simulate Broken Networks

Supervisor(s): Jukka Kaisto

Term and year of completion: Spring 2017

Number of pages: 37

The objective of this thesis is to develop a virtual appliance that will simulate broken networks. A virtual appliance is a ready-to-use server that can be run on a virtualization platform. This thesis was commissioned by HowNetWorks Oy. HowNetWorks is a startup based in Oulu and Helsinki, Finland, developing tools to measure networks in ways that most tests do not. The objective of this appliance is therefore to simulate the network properties that HowNetWorks is testing, to help their development.

The appliance, called hnwProxy, is created using infrastructure-as-code methodologies. This can be downloaded from GitHub at github.com/hownetworks/hnwproxy. Infrastructure-as-code is a new paradigm in infrastructure management, using practices from software engineering along with automation tools to create higher quality, more reliable and higher performing systems.

The theoretical background of this thesis consists of network quality and infrastructure as code. Network quality describes what factors affect the quality of a connection, i.e. what we can simulate with hnwProxy. For infrastructure-as-code, the book *Infrastructure as Code* by Kief Morris is used almost exclusively as reference. Network quality is more dispersed. There is no single book about the topic, the most used reference is Kurose & Ross's *Computer Networking: A Top-Down Approach*. This covers computer networking quite exhaustively, but further references for more niche or loosely related topics are still needed.

The result of this thesis is the virtual appliance, hnwProxy. This can simulate a broken network connection in several different ways and can run on a few different virtualization platforms, so it has met all requirements.

Keywords: Infrastructure as Code, Networks, Virtualization

CONTENTS

1	INTRODUCTION	7
1.1	This Report.....	7
1.2	The Commissioner – HowNetWorks Oy	8
2	NETWORK QUALITY FACTORS	9
2.1	Traditional.....	9
2.2	Esoteric	11
3	INFRASTRUCTURE AS CODE.....	16
3.1	Infrastructure Definition	16
3.2	Server Configuration	18
3.2.1	Change Management Models	18
3.2.2	Scripting.....	19
3.2.3	Ansible	20
3.3	Server Templates	21
3.3.1	Packer.....	22
3.3.2	Atlas.....	22
3.4	Software Engineering Practices	22
3.4.1	Version Control	22
3.4.2	Continuous Integration.....	23
3.4.3	Continuous Delivery.....	24
4	HNWPROXY DEVELOPMENT.....	25
4.1	Requirements	25
4.2	System Architecture	25
4.2.1	Server Templates.....	25
4.2.2	Connecting to hnwProxy	26
4.3	Software Engineering Practices	28
4.4	Proxy CLI.....	29
4.5	Proxy CLI Modules	29
4.5.1	DNS Block.....	30
4.5.2	Transparent Proxy	31
4.5.3	DNS Redirect.....	32
4.6	Maintenance.....	32

5	CONCLUSION.....	34
6	DISCUSSION	35
	REFERENCES	36

1 INTRODUCTION

The topic of this thesis is developing a virtual appliance that can simulate a broken network connection. A virtual appliance is a pre-configured, ready-to-use virtual machine that can be run on a virtualization platform such as Hyper-V. The appliance is named hnwProxy, after the project's commissioner. hnwProxy is open-source, and built using infrastructure-as-code methodologies. hnwProxy simulates a network connection by shaping the network traffic that passes through it, either via VPN connection or SOCKS proxy.

The research questions of this thesis are:

- In what ways can a network be broken?
- How can this be simulated by a server?
- How can we create a virtual appliance using infrastructure-as-code (IAC) methodologies?

hnwProxy is needed because there are no similar tools available. There are numerous network connection simulation programs available, but most of these only simulate a small number of things that could be wrong with a network. One example of this is an application called Comcast. Comcast is a free, open source network connection simulator, that emulates the bandwidth, latency, and packet loss of various types of Internet connections (Comcast, cited 2016-10-25).

1.1 This Report

The theory section of this report describes network quality factors and infrastructure as code. Network quality describes various factors that can affect the quality of a network connection. These are the things that hnwProxy emulates. The second section describes infrastructure as code. It explains what infrastructure-as-code is, how it is applicable to server systems and common principles and practices.

The practical part of this report describes the process of developing hnwProxy. Two important sections include *system architecture* and *software development methods*. System architecture describes the high-level architecture of hnwProxy, how it is in the final product and the reasoning

behind it. Software Development Methods will discuss how software development tools and methodologies were used in the project.

1.2 The Commissioner – HowNetWorks Oy

HowNetWorks is a startup company located in Oulu and Helsinki, Finland. It is one of the first companies being supported by Ääkköset (eng: Scandinavian ABC / ScanABC), a new startup accelerator in Oulu. HowNetWorks is developing technologies to measure network connections in ways that traditional tests do not, with the goal of launching a consumer-facing web app around the beginning of 2017.

There are numerous network measurement tools available for both consumers and IT professionals, but these only measure a small number of properties that factor into the quality of a network connection. A well-known example of this is speedtest.net. Speedtest.net measures your connection's bandwidth and latency. Despite getting a good score, your Internet connection could still work poorly if your router's DNS server is redirecting you to malicious sites. For some of these more esoteric factors, there are individual tests available to IT professionals but no easy way to test everything at once.

2 NETWORK QUALITY FACTORS

2.1 Traditional

The traditional methods of measuring a computer network are bandwidth, latency, and packet loss (Kozierok 2006, 34). These are commonly measured, and simulated by network simulation tools.

Bandwidth is the amount of data that can be transferred over a given time. It is typically expressed in bits (b) per second. A bit is a small amount of data, so SI prefixes are used. A typical local area network (LAN) might have a maximum bandwidth of 1 gigabit per second (Gb/s) whereas an internet connection might be 10% of that, 100 megabits per second (Mb/s). There are many bandwidth measurement tools available, such as speedtest.net. Throughput is sometimes used interchangeably with bandwidth, but these two are not the same. Throughput is an actual measured value, and bandwidth is purely theoretical. (Kozierok 2006, 35.)

Latency is the timing between a cause and effect. In networking, this typically means the time between sending some data and receiving a response (Kozierok 2006, 35). Many Internet connection tests such as speedtest.net also test latency. The ping tool included with most operating systems also tests latency, using ICMP echo request packets (Kozierok 2006, 1464).

Packet loss is the measurement of packets that do not make it from their sender to a recipient. There can be many causes for packet loss, two common ones are network congestion and firewalls. Network congestion is when a router or switch receives more data than it can send out over a short period. This period is determined by the device's packet buffer, a piece of memory where it stores packets upon receiving them, before sending them out. While this memory is full, the device cannot store any more packets and must drop them. (Kurose & Ross, 2013, 41.)

Packet loss can also be caused intentionally, by a network's firewall. Firewalls can be configured to whitelist or blacklist ports, protocols, or other properties depending on how advanced they are. On virtually every network, a firewall is used to ensure some amount of separation between the internal network and Internet. In a simple network, you have one firewall that all traffic passes through. Usually this is configured to only allow traffic in that was initiated by a host in local net-

work. (Kurose & Ross 2013, 731-739.) Due to the mostly binary nature of firewalls (block or allow), if a firewall is causing packet loss, the loss will likely be all or nothing; either blocking all packets with a certain property or letting them through.

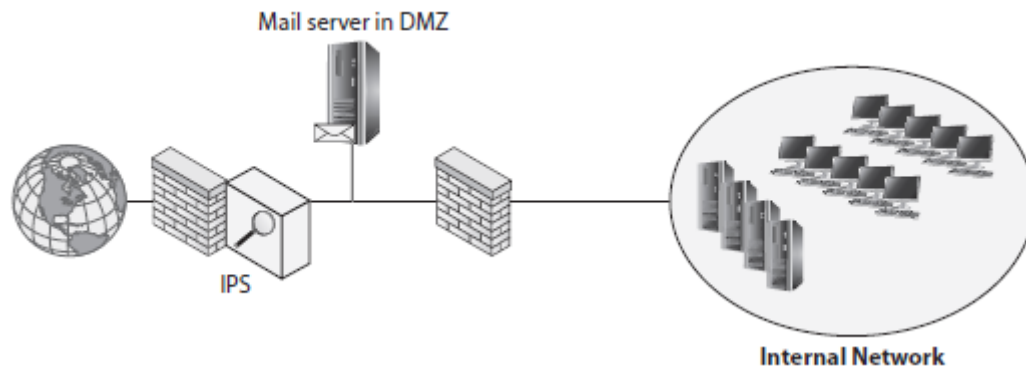


Figure 1. Inline IPS, placed behind a firewall. (Gibson 2016, 280.)

Related to firewalls, Intrusion Prevention Systems (IPS) can also be the cause of intentional packet loss. IPSs are used to detect and stop network based threats, using either signature or anomaly-based detection. Signature-based detection uses known characteristics of different attacks to detect them. An example of this would be a SYN flood attack, in which many SYN packets are sent to a host. Anomaly-based detection works by comparing traffic characteristics to a known baseline. For example, if a host is receiving a larger than normal amount of ICMP packets then it might be under attack. IPSs often use both methods of detecting attacks, since they both have unique weaknesses and strengths. IPSs are not meant to replace firewalls, but to provide additional protection. They are normally placed inline with network traffic, behind a firewall (as in figure 1). This is done because the analysis methods used by IPSs are computationally more expensive than the rules used by firewalls, which can already block a good amount of simple attacks. (Gibson 2016, 275–283.)

2.2 Esoteric

This chapter describes more esoteric factors that affect the quality of a network connection. This might be a property that can be measured, or something binary such as the existence of a technology. These usually aren't measured by network testing tools, nor are they simulated by network simulation tools.

Packet loss was already mentioned in the previous chapter, but it is being featured in this one as well. Typically, packet loss is indicated as a percentage of packets that do not make it to their recipient. If congestion is the cause of packet loss, it can be useful to determine the relation of throughput to packet loss. If the loss percentage does not change with throughput, the packet loss might be described as static. The opposite of this being dynamic packet loss where you lose more packets with increased throughput. This distinction is useful to make, because from it you can infer where in the network there is congestion. If network congestion is occurring in the backbone, packet loss is more likely to be static. This is because a single host makes up a small percentage of total traffic, so they cannot affect how much the network is overloaded by any noticeable amount (in the short term). Alternatively, if the network congestion takes place closer to the host, packet loss is likely to be more dynamic. In this case, a single host makes up a larger portion of all network traffic, so they can cause network to become more or less congested.

Domain Name System (DNS) is a fundamental part of the Internet, and other computer networks. DNS is the protocol used to lookup IP addresses and other information from hostnames (e.g. example.com) and vice versa. DNS servers store DNS records and respond to queries for these records made by clients and other DNS servers. For your Internet connection to work properly, it is necessary to have both *working* and *accurate* name resolution available. Because of the importance of DNS to the Internet, it can be used for censorship or to manipulate access to web sites.

Most computers on a residential Internet connection use DNS servers provided by the connections internet service provider (ISP). In many countries, the government requires ISP's to block access to certain web sites, often via DNS (Savola 2013, 88). On corporate networks, computers will use the company's DNS infrastructure which can also be used for blocking sites.

A much worse consequence of inaccurate name resolution is the ability for a bad actor to hijack your traffic. Because computers use DNS names for virtually every connection they make, an attacker with the ability to modify DNS packets can do a lot of damage. This requires that the attacker either runs their own malicious DNS server and somehow gets their target to use it, or intercepts and modifies DNS packets in-transit. There have been cases where ISP's have modified DNS queries to insert advertisement's in to web pages. (Singel, cited 2016-10-26.) Home routers are a very attractive target for hackers because they typically have weak security. A security researcher in the US estimates that 25–30% of households use a router with known vulnerabilities (Dunn, cited 2016-10-26). This is one way a hacker could get someone to use their malicious DNS server.

Because of the cleartext and unauthenticated nature of DNS, it is not difficult to hijack in-transit. Figure 2 shows a single IPTables rule that redirects DNS queries to the server at 4.2.2.1 (Level3 public DNS). This uses destination NAT in a slightly unorthodox way to rewrite the destination address of packets likely containing DNS queries.

```
iptables -t nat -A OUTPUT -p udp --dport 53 -j DNAT --to-destination 4.2.2.1
```

Figure 2. Firewall rule to redirect DNS queries.

Network Address Translation (NAT) is a common feature of IPv4 networks and can sometimes cause problems. NAT is configured on a network's router, and it translates the IP addresses of packets that the router forwards between networks. The most common usage of NAT is to enable computers to share a single IPv4 address. NAT was originally created for this purpose, to conserve IPv4 addresses. (Kurose & Ross, 2013. 349.)

Figure 3 shows an example of many-to-one NAT, sometimes called port address translation (PAT). This maps any number of internal addresses to a single external address. When a computer sends a packet to the Internet, its source address is the local address of the sending computer (1). This is replaced with the router's public IP address (2) when it is forwarded by the router. When the router receives a reply (3), it then replaces the destination address in the packet with the local IP of the original sending computer (4). The router knows what IP addresses to change because it keeps track of sessions in its NAT translation table. Sessions are distinguished from each other using port numbers (see the table in figure 3).

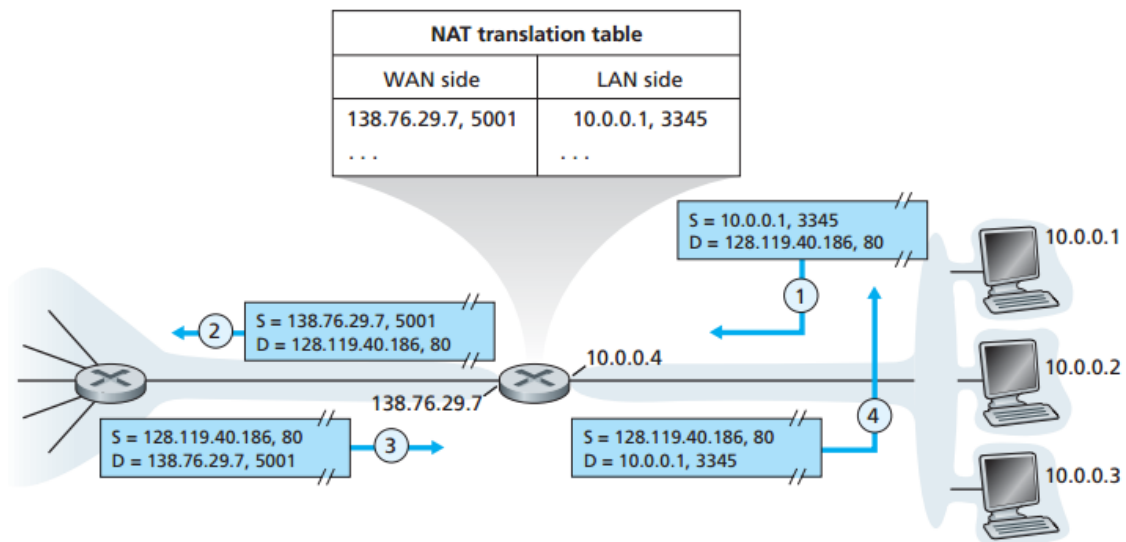


FIGURE 3. NAT example (Kurose & Ross, 2013, 350.)

Static and Dynamic NAT are less commonly used forms of NAT. Both create a one-to-one mapping between external and internal addresses. Most of the time when someone refers to NAT, they mean many-to-one NAT as described in the previous paragraph.

The problem that NAT creates is that it breaks “end-to-end” connectivity. This is the principle that hosts should be able to communicate with each other directly, without anything in between modifying their packets. In a many-to-one NAT, hosts can only initiate connections from within the network, they cannot receive connections. This might seem like a nice security feature, but it is not. You can accomplish the same thing with a modern firewall, with the bonus of being able to whitelist specific port numbers or IP addresses which can receive connections. Another problem that NAT creates is that it breaks some protocols which rely on IP headers, such as IPSEC. (Sriuresh & Holdrege 1999, 2.)

There are many workarounds for problems created by NAT, but the simplest is to stop using it altogether. On an IPv4 network, this can be difficult due to the scarcity of addresses. With IPv6, this is a nonissue. Port forwarding and Universal Plug and Play (UPnP) are workarounds that enable many-to-one NATed devices to receive connections from outside the network. Both accomplish this by creating a mapping on the router which maps a single external port number to an internal host. Any packets sent to that port are translated to the specific host, without the host having to initiate a session. (Kurose & Ross, 2013. 352.)

A **Proxy** is a server acting as an intermediary for requests made by clients. Most of the time proxies are used to handle web requests made by clients on a network. There are many reasons someone might want to use a proxy server:

- Caching
- Web filtering
- Malware scanning
- Statistics gathering
- For some nefarious reasons

Proxies are usually configured on network hosts, either manually or automatically. Web Proxy Auto Discovery Protocol (WPAD) is a mechanism that allows hosts to discover a proxy server using DHCP or DNS (Cisco, accessed 2016-12-03).

Proxies can also be used transparently (transparent proxy). In this situation, the network hosts are unaware of the proxy, and do not have to configure any settings. Instead, web traffic is redirected by a router to the proxy server. (Kozierok 2006, 1386.)

HTTP traffic is trivial to proxy, because it is cleartext. HTTPS is harder, because it is secured using SSL/TLS. Due to this, the proxy server must establish 2 connections for each outbound HTTPS session: one between the proxy and remote server, and one between the proxy and the client. In this case, the proxy server emulates a client to the remote server and emulates the remote server to the local client. This is difficult to do in practice, because it requires the proxy to be able to generate valid certificates for the remote server's hostname. This is not possible unless the client trusts a certificate authority run by the proxying organization. An alternative is to downgrade the client-to-proxy connection to HTTP, while keeping the proxy-to-server connection HTTPS. HTTP downgrading is mostly performed for malicious reasons, if an attacker cannot generate their own trusted certificates. (Marlinspike, M. Accessed 2016-12-03.) Luckily there is a defense for this: HTTP Strict Transport Security (HSTS).

HSTS is a mechanism that allows browsers to keep a list of sites that they should never visit over HTTP. This list is populated either by a web server using the "Strict-Transport-Security" HTTP header field, or can be baked-in to a browser. (Hodges, J., Jackson, C. & Barth, A. 2012. Accessed 2016-12-03.)

The existence of transparent proxy on a network can be a cause for concern. It can mean that your traffic (web or other) is being read or modified. It could also be used just for caching. A hyper-sensitive malware-scanning proxy could cause problems despite the network operator having good intentions.

3 INFRASTRUCTURE AS CODE

Infrastructure is a general term for the hardware/software layer below software applications. This can include networking, storage, compute resources, operating systems, and anything in between. Many organizations use infrastructure automation tools and scripting languages to make the upkeep of this infrastructure more efficient. Infrastructure as code (IAC) is a new approach to infrastructure automation, using practices from software development. The goal of IAC is to improve the quality and efficiency of infrastructure management even more than traditional automation can allow. (Morris 2016, 3.)

Infrastructure as code (IAC) is often used when managing cloud environments because of the huge number of infrastructure components involved. In an IAC environment, your infrastructure is defined using definition files and scripts. These definition files are run through different infrastructure automation tools to produce a resulting server infrastructure. Definition files differ from scripts, in that they are not procedural (e.g. do x, y, then z). Definition files are declarative, meaning that they declare how something should be, and infrastructure automation tools then use their own logic to make this happen. Definition files are typically written using a data serialization language such as YAML or JSON, but scripting languages can also be used. This depends entirely on the tool being used. (Morris 2016, 50.)

3.1 Infrastructure Definition

Infrastructure definition tools are the “meat” of infrastructure as code. They are used to define, implement, and change infrastructure running on an infrastructure platform. Two examples of this are AWS *CloudFormation* and Hashicorp *Terraform*. CloudFormation is used to manage resources in Amazon AWS. Terraform is similar, except that it works with more than one platform.

Figure 4 shows a snippet from a terraform configuration file. This defines a single virtual server running in EC2 as a web server. Lines 1-3 define the properties of the server, and the section after “provisioner” defines what server configuration tool should be used to finish the server’s configuration.


```

resource "aws_instance" "web" {
  instance_type = "t2.micro"
  ami = "ami-87654321"

  provisioner "chef" {
    run_list = [ "role::web_server" ]
    attributes_json = {
      "dns_servers": [
        "192.168.100.2",
        "192.168.101.2"
      ]
    }
  }
}
}
}

```

FIGURE 4. Terraform configuration file. (Morris 2016, 55).

Vagrant is another infrastructure definition tool. Vagrant differs from other tools in that it is not meant for production use, but for creating local virtual server environments. This is useful for developers and other people who can use it to quickly spin up a sandboxed server environment where they can test out their code. Vagrant environments are defined in a Vagrantfile, a declarative configuration file that uses the Ruby scripting language. (Hashimoto 2013, 1.)

```

1  # Vagrant's configuration is done with ruby, so we have the
2  # full language available. For example, we can load an external file.
3  require 'yaml'
4  settings = YAML.load_file('settings.yaml')
5
6  Vagrant.configure("2") do |config|
7    # virtualbox settings
8    config.vm.provider "virtualbox" do |vb, override|
9      vb.memory = 1024
10     vb.cpus = 2
11     ...
12     if settings['vbox_bridged_network'] == true
13       override.vm.network "public_network", use_dhcp_assigned_default_route: true
14     end
15   end
16   # What server template to use.
17   config.vm.box = "bento/ubuntu-14.04"
18   ...
19   # Server settings and provisioners.
20   config.vm.hostname = "servername"
21   config.vm.provision "shell", path: "scripts/provision.sh"
22 end
23 end

```

FIGURE 5. A Vagrantfile

Vagrant uses the term “provider” to describe platforms where virtual servers can run. It supports multiple providers such as VirtualBox, Hyper-V and even some cloud services such as

DigitalOcean and Amazon EC2. All providers require some type of operating system image to create a virtual machine from, and with local providers such as Hyper-V, this is referred to as a “box”. A box is simply a file archive which contains a virtual machine image and some metadata. In a Vagrantfile, you can specify either a box from HashiCorp’s public box repository or a custom box using its download link. (Hashimoto 2013, 76.)

Once a server has been created using a provider, it is configured using a “provisioner”. A provisioner is a component in Vagrant that modifies the guest server in some way. The file provisioner can copy a file, whereas a shell script provisioner will run a shell script. Figure 5 (previous page) shows a shell provisioner which runs the script “provision.sh” from the scripts directory.

3.2 Server Configuration

Server configuration tools are used to configure servers once they have been created in an infrastructure platform. They can also be used when creating server templates as will be described in later chapters. Server configuration tools are also popular outside of infrastructure-as-code environments where they are used for managing server configurations.

3.2.1 Change Management Models

Before looking at server configuration tools, it is important to understand change management models. These models determine how changes are applied to servers, either upon creation or to existing servers.

Ad Hoc change management is the traditional methods of applying changes to servers. Simply put, you apply changes as they are thought to be necessary, either manually or with some form of automation. This is the least automated, most fragile change management option. Ad-hoc change management creates several problems that IAC aims to solve:

- Documentation: everything needs to be manually documented. Even worse, documentation can differ from reality if not kept up-to-date.
- Disaster recovery is difficult.

- Server Reproducibility is poor.

(Morris 2016, 69)

Configuration Synchronization is the process of repeatedly applying configuration definitions to servers. This is done using configuration management tools such as Ansible, Chef, or Puppet. This is the most common approach to infrastructure as code, and is often used in other types of environments. The main weakness of configuration synchronization is that any setting which is not defined is left unmanaged. These settings are prone to configuration drift, require additional documentation effort and make the server less reproducible. (Morris 2016, 69.)

Immutable infrastructure is the idea that servers are not changed at all, and any changes are done by replacing the server with a new one. Changes are made to a server template, which is used to build a new server. This makes servers in production very predictable, and easily replaceable. One difficulty with immutable infrastructure is handling persistent data. (Morris 2016, 70.)

Containerized services are a hybrid of the previous change management methods. Applications and services are packaged in containers which are treated as immutable. Changes are made by creating a new version of the container and then deploying that to production. Containers are a new way of running applications on servers. The premise is that you package your application and its dependencies into a single container image which can then be run on a containerization platform. This means that your application is decoupled from the software and hardware it is running on, and will work on anything capable of running a containerization platform. An example of this is Docker. With containerized services, the host server still needs to be managed with some change management method but changes are much simpler and fewer. (Morris 2016, 70.)

3.2.2 Scripting

Scripting was, and still is, a popular way of automating tasks. Despite the increasing popularity of automation tools, there are still some tasks that can and should be automated using scripting languages. Automation tools cannot do everything, and scripting is often used to fill these gaps.

To work reliably in an infrastructure-as-code environment, scripts should be able to run unattended, without human interaction. This presents a few requirements:

- Idempotence: The script can be run multiple times with no ill effect.
- Pre-checks: The script checks that its starting conditions are appropriate.
- Post-checks: The scripts checks that it has executed successfully.
- Visible failure: Any failures are easily visible.

These requirements increase the complexity of scripts. This may be one of reasons automation tools and declarative configurations are becoming more popular. (Morris 2016, 44.)

3.2.3 Ansible

Ansible is one of many configuration management tools available. It uses YAML (a data serialization language) for its configuration declaration. Ansible uses an agentless architecture, where the servers it manages do not have to have agent software installed. Ansible connects via SSH and does all its configuration using Python scripts. (Hochstein 2015, 1.)

A simple Ansible setup uses 2 configuration files: a playbook and an inventory. A playbook defines what actions will be applied to what servers. For example, on web servers you might require apache to be installed and running. An inventory defines what servers Ansible will manage. This can include servers by IP address or hostname, and can classify servers into groups. The playbook in figure 6 affects servers in the *webservers* group. (Hochstein 2015, 21.)

Once your playbook and inventory are ready, you run them on some machine that has Ansible installed. Ansible will then parse the playbook, and generate some python scripts. It will then connect via SSH to the appropriate servers, copy the python scripts and run them (Hochstein 2015, 4).

```

- name: Configure webserver with nginx
  hosts: webservers
  sudo: True
  tasks:
    - name: install nginx
      apt: name=nginx update_cache=yes

    - name: copy nginx config file
      copy: src=files/nginx.conf dest=/etc/nginx/sites-available/default

    - name: enable configuration
      file: >
        dest=/etc/nginx/sites-enabled/default
        src=/etc/nginx/sites-available/default
        state=link

    - name: copy index.html
      template: src=templates/index.html.j2 dest=/usr/share/nginx/html/index.html
        mode=0644

    - name: restart nginx
      service: name=nginx state=restarted

```

FIGURE 6. Ansible playbook for a web server (Hochstein 2015, 31)

3.3 Server Templates

Server templates are a way of creating new servers in an infrastructure-as-code environment. Simply put, a server template is an image of a server operating system that can be copied to create a new server. Templates are used because they make deploying servers easier and faster, because you do not need to re-install the operating system and make the same configuration changes every time. Typically, the process to create a server template goes like this:

Install Server → Apply Customizations → Capture Image

Depending on the environment in question, you might need to use a highly customized or a minimally customized server template (or somewhere in between these two). A minimal template has a minimal amount of software and configurations installed, so it can be used to create many different types of servers. It takes up little disk space and is flexible, but almost always requires additional configuration after deployment. In this situation, you would apply additional configurations to the server using a server configuration tool as described in the previous chapter. The opposite of this is a highly-customized server template. With a highly-customized template, the template contains all the software and configurations to perform a specific task. This requires very

little configuration after deployment, so deployment is very fast. The downside to this method is the template's lack of flexibility. (Morris 2016, 117.)

3.3.1 Packer

Packer is a tool created by HashiCorp. It is used for creating server templates for different types of platforms such as VMWare, Amazon EC2, and Docker. Packer works by creating a server in the platform in question, customizing it and then capturing an image. Much like Vagrant, Packer also supports the concept of *provisioners*. In each Packer configuration file, you can specify a provisioner (or multiple) that will be used to configure the server before it is captured to create a template. (HashiCorp 2016, cited 2016-11-06.)

3.3.2 Atlas

Atlas is a service run by HashiCorp. Among other things, it can be used to catalog Vagrant boxes and build them using Packer. One popular use of Atlas is storing Vagrant boxes in its public box repository. This lets anyone use the box in their Vagrantfile just by entering the name of the box, such as ubuntu/trusty64 (in this case ubuntu is the user who uploaded the box and trusty64 is the box name).

3.4 Software Engineering Practices

A big part of infrastructure as code is applying practices from software engineering to your environment. This was not possible before but is now with the introduction of declarative configuration and various automation tools.

3.4.1 Version Control

Version control systems are used to store and manage changes done to files. These are usually plaintext files, as used with code and configuration files. Some benefits of version control are:

- **Traceability:** VCS's provide detailed history of changes made.
- **Rollback:** You can easily roll your infrastructure back to a previous version.
- **Actionability:** The VCS can trigger an action when changes are committed, such as running an automated test suite.

(Morris 2013, 15-16.)

As of 2016, Git is the most popular version control system (Version Control System Popularity in 2016, accessed 2017-01-02). Git was created in 2005 for the development of the Linux kernel, by Linux Torvalds and other kernel developers. At the time, there were no free VCS's that could handle large, distributed, and highly parallelized (=branched) software projects. (A Short History of Git, accessed 2017-01-02.)

3.4.2 Continuous Integration

Continuous Integration (CI) is the process of frequently integrating and testing changes to a system. CI has traditionally been used in software development, for testing software builds. A developer would hook some CI system to their version control system, and it would automatically run tests every time a change was made. If any tests failed, the developer would be aware immediately so they could fix the problem. This ensures a very fast feedback loop and stops problems from growing out of control.

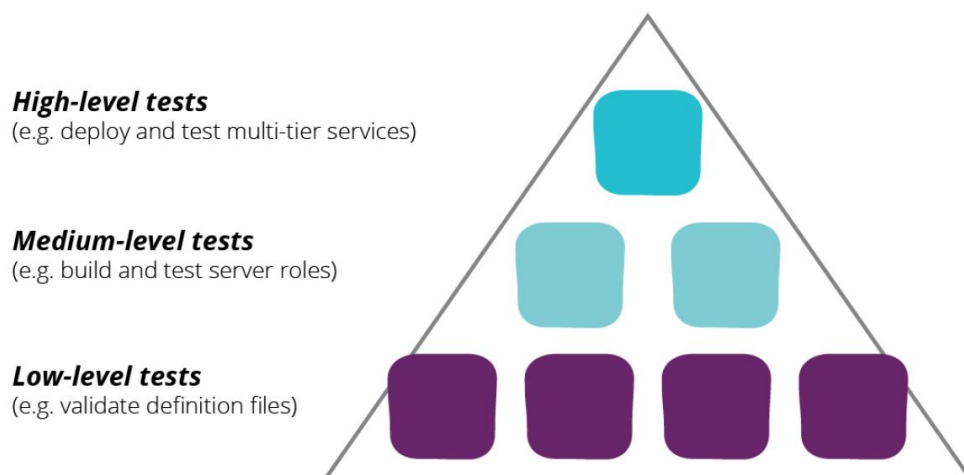


Figure 7. Testing pyramid (Morris 2013, 198.)

With the introduction of infrastructure automation, it is now possible to use CI for infrastructure. Scripts and configuration files can be put in a VCS, and automatically tested using various tools.

Low-level tests are used for testing individual files or small subsections of a system. When testing individual files, they are typically analyzed using a linter; a program that checks the file for syntax correctness, stylistic issues, or bad habits such as inefficient code. This is also called static analysis because it does not run the code/configuration. Unit testing is also low-level, but it works slightly differently. Unit testing attempts to run the code/configuration and checks that it works properly.

There is also mid- and high-level testing. High-level testing aims to test a system as a whole, for example, by deploying it to a test environment. Mid-level testing aims to test a section of a system, such as an individual server. A well-balanced test suite uses all these methods, since they have unique strengths and weaknesses. Low-level tests are quick, easy, and inexpensive to run. Their weakness is that low-level tests usually catch only simple errors. High-level tests are much more expensive, and slower to run, but they can catch more complex problems and verify that the system works. (Morris 2013, 195-210.)

3.4.3 Continuous Delivery

Continuous Delivery (CD) is the next step up from continuous integration. The goal of continuous delivery is to verify that every change is ready to go out to production, by using high-level tests (as described above). This might be done using a test environment, where changes are applied first. Once the changes are applied and tests are passed, the changes can be accepted into production. An extension of this is called Continuous Deployment. With continuous deployment, changes are rolled into production with no human interaction if all tests are successful. (Morris 2013, 187.)

4 HNWPROXY DEVELOPMENT

4.1 Requirements

In an early meeting with HowNetWorks, we discussed various technologies that could be used for realizing hnwProxy. Based on this discussion, some requirements surfaced:

- *100% Codified*: the system exists as a series of plaintext files, stored in a version control system. Anyone can download these and easily spin up their own instance of hnwProxy
- *Support VirtualBox as a local hypervisor*. The main users of hnwProxy use Macs. Because of this, VirtualBox is a good choice since it is cross-platform compatible.
- *Open Source*: there is no reason for the alternative.

4.2 System Architecture

The infrastructure platform used by hnwProxy is Vagrant. As described in earlier chapters, this is a tool used to create virtualized server environments. Using Vagrant, hnwProxy can be brought up either on a local hypervisor or cloud platform with a few simple commands. Properties of the server are configured in Vagrant's Vagrantfile, along with provisioners to configure the server after it is booted. For each supported provider (VirtualBox, Hyper-V and cloud platforms) certain provider-specific settings are also configured.

Ubuntu 14.04 was chosen as the base OS for hnwProxy due to its widespread use and server template availability.

4.2.1 Server Templates

hnwProxy does not use a custom server template. Instead, it uses a freely available template for whichever hypervisor platform it is created on. This is done for two reasons:

1. **Simplicity**. Using a custom server template will increase the complexity of hnwProxy as well as its deployment time. The benefits of a custom server template do not justify the added complexity.

2. **Compatibility.** By creating hnwProxy in a way that it is compatible with most Ubuntu 14.04 pre-made templates, it can be more easily deployed to new virtualization platforms.

4.2.2 Connecting to hnwProxy

Management of hnwProxy is done using SSH. Vagrant automatically manages SSH keys and settings for its machines, so it is not necessary to configure anything manually. By typing “vagrant ssh” when in hnwProxy’s directory, vagrant will connect using a key it created during provisioning. In addition to managing hnwProxy, you also need some way of routing your traffic through it.

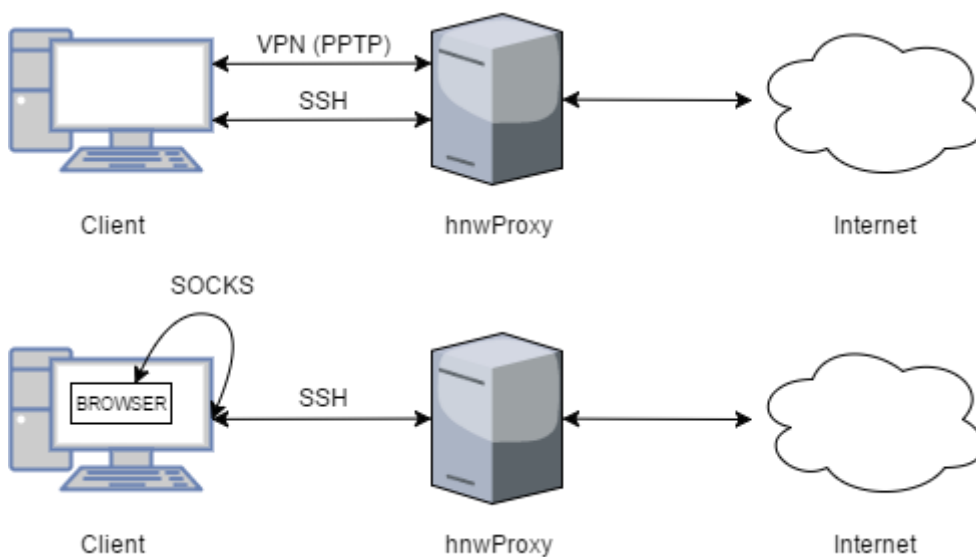


Figure 8. Connecting to hnwProxy, Logical View

There are two ways of connecting to hnwProxy: VPN or SOCKS proxy. Both methods work regardless of whether hnwProxy is running on your local machine or in the cloud, since they connect by IP address. VPN is the simplest: there is a VPN server running on hnwProxy, and you can connect to it using any compatible VPN client. In this situation, all your network traffic is routed through hnwProxy, unless otherwise configured by the VPN client. The PPTP VPN protocol was chosen because of its simple client-side configuration and good compatibility with Windows and MAC OS built-in VPN software. PPTP is not secure, but in this application the benefits of an easy-to-use VPN protocol outweigh the risks of an insecure one.

The second way of connecting to hnwProxy is via SOCKS5 proxy. SOCKS is a proxying protocol that works at a lower level than HTTP proxies. It allows for proxying arbitrary TCP/UDP connec-

tions, though in this case it is used only for HTTP(S). (Leech, Ganis, Lee, Kuris, Koblas & Jones 1996, 2) Most browsers support SOCKS proxies, including Firefox and Chrome. One benefit of SOCKS over HTTP proxies is that it can also proxy DNS queries. It also does not require a SOCKS server to be running, because many SSH clients (including the one Vagrant uses) support dynamically creating SOCKS proxies through their SSH tunnels. Typically, this is done with the “-D” flag. This is the method used by hnwProxy, you can create a SOCKS5 proxy by connecting with the vagrant command “vagrant ssh -- -D 6000” and then configuring your browser or operating system to use localhost:6000 as a SOCKS server.

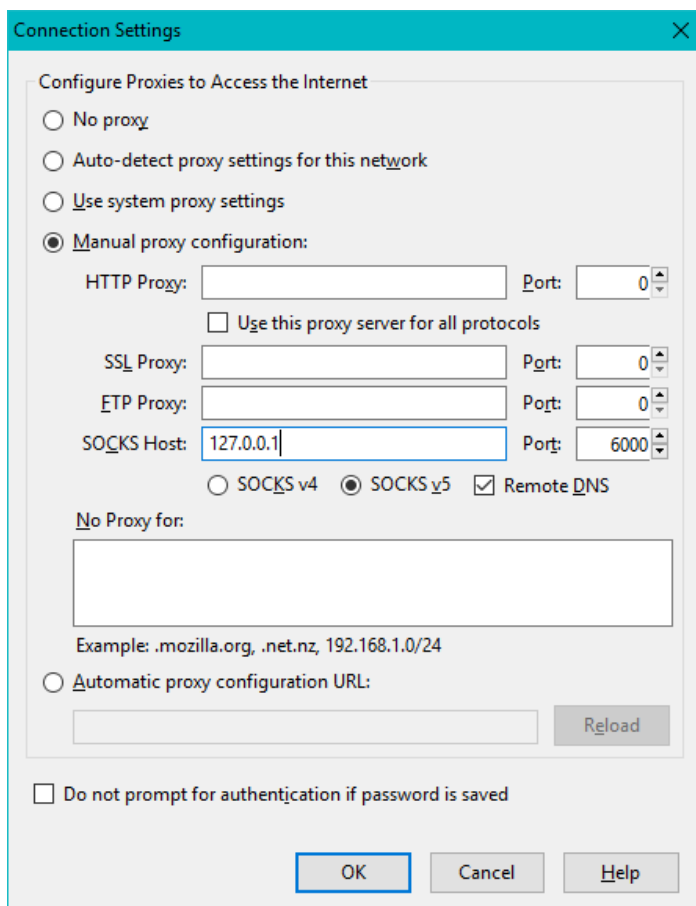


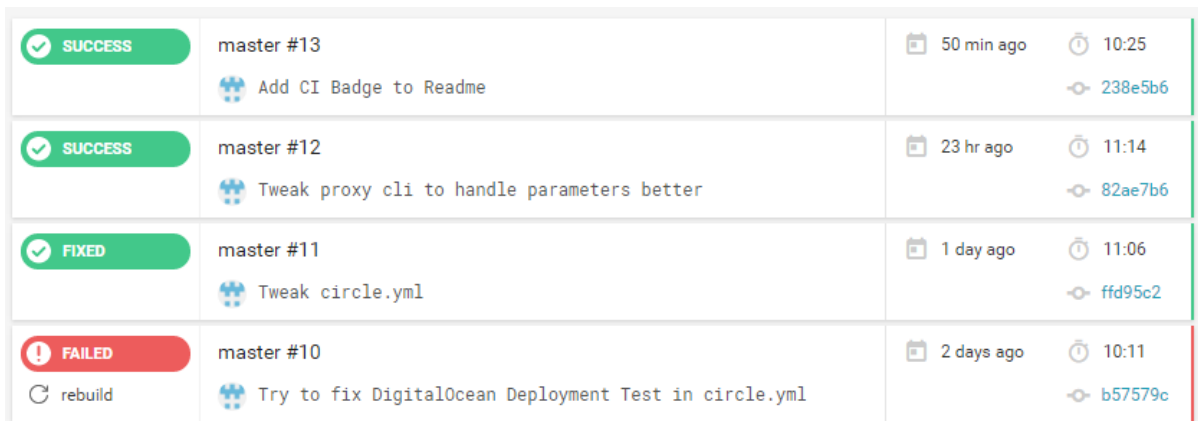
Figure 9. Configuring SOCKS settings in Firefox

4.3 Software Engineering Practices

hnwProxy is an open-source project, hosted on GitHub. It is located at github.com/hownetworks/hnwProxy. hnwProxy uses the MIT License, a permissive free software license. This was chosen for its simplicity and good compatibility with other free software licenses.

A version control system (Git) was used during the development of hnwProxy. Git was chosen because it is currently the de-facto VCS, and is what GitHub supports. This is relevant because HowNetWorks' and ScanABC use GitHub for their own version control repositories.

Continuous integration with CircleCI was used in the development of hnwProxy. Because this is not a traditional software project, normal CI tests could not be used. The way CircleCI "builds" hnwProxy is by deploying, an instance of it to DigitalOcean. If all commands used return an exit code of 0 (no error) then the project is presumed to be working and the build passes. A badge on hnwProxy's readme page shows the state of the last build. This is a high-level test, as it tests how well the different parts of hnwProxy work (Vagrantfile, provision scripts, setup scripts).
























 SUCCESS	master #13  Add CI Badge to README	 50 min ago	 10:25  238e5b6
 SUCCESS	master #12  Tweak proxy cli to handle parameters better	 23 hr ago	 11:14  82ae7b6
 FIXED	master #11  Tweak circle.yml	 1 day ago	 11:06  ffd95c2
 FAILED  rebuild	master #10  Try to fix DigitalOcean Deployment Test in circle.yml	 2 days ago	 10:11  b57579c

Figure 10. hnwProxy Builds in CircleCI

In addition to a deployment test, hnwProxy's shell scripts are also linted using ShellCheck and yamllint. These are low-level tests as described in section 3.4.2. Because both tools will return an error code from even a small, possibly inconsequential error, their output is ignored for the

purpose of passing/failing the CircleCI build. This output is still valuable, and visible in full detail from CircleCI's build page.

```
In ./guestfiles/proxy_cli/proxy_modules/pptp/pptp line 11:
password=$(cat /etc/ppp/chap-secrets | cut -d ' ' -f 3)
      ^-- SC2002: Useless cat. Consider 'cmd < file | ..' or 'cmd file | ..' instead.
```

Figure 11. ShellCheck output

4.4 Proxy CLI

Proxy CLI is a tool used to run and discover modules i.e. collections of scripts from the command line. This is what enables the user to modify hnwProxy to simulate a broken network. Proxy CLI works by running modules located in the proxy_modules folder. For example, running “proxy module1 start” will run “proxy_modules/module1/module1”, with the start parameter. Figure 12. shows the main files used by Proxy CLI. The description file in each module’s directory is used by proxy CLI when listing available modules. The setup file is run once every time hnwProxy is provisioned, to install any dependencies or make changes needed by the module.

```
proxy                                # executable

# required for one module
proxy_modules/module_name            # directory
proxy_modules/module_name/module_name # executable
proxy_modules/module_name/description # plaintext description

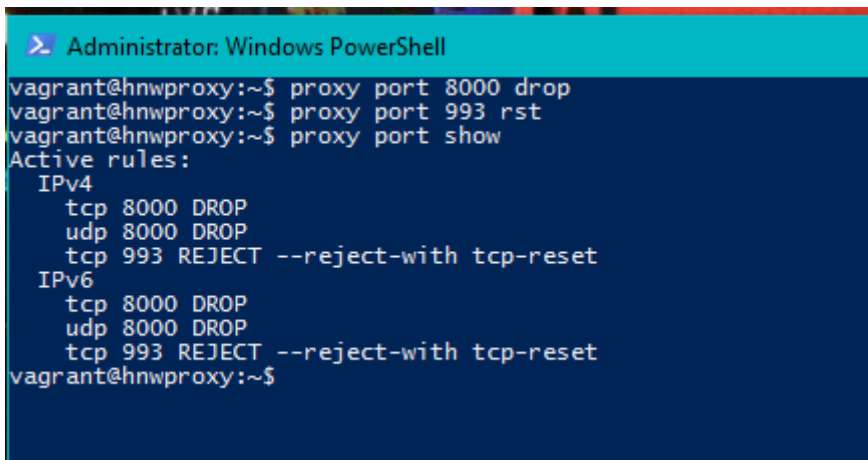
# optional
proxy_modules/module_name/setup      # executable, run during
                                      # server provisioning
```

Figure 12. Proxy CLI Files

4.5 Proxy CLI Modules

Proxy CLIs modules can be any type of executable, but in this case, they’re all bash scripts. For each module, its main executable needs to be located at proxy_modules/module_name/module_name. This is then run by Proxy CLI when the module’s

name is passed as a parameter. Figure 13. shows the port-blocking module being used. This blocks outgoing packets by their destination port number.



```
Administrator: Windows PowerShell
vagrant@hnwproxy:~$ proxy port 8000 drop
vagrant@hnwproxy:~$ proxy port 993 rst
vagrant@hnwproxy:~$ proxy port show
Active rules:
  IPv4
    tcp 8000 DROP
    udp 8000 DROP
    tcp 993 REJECT --reject-with tcp-reset
  IPv6
    tcp 8000 DROP
    udp 8000 DROP
    tcp 993 REJECT --reject-with tcp-reset
vagrant@hnwproxy:~$
```

Figure 13. Using Proxy CLI's port-blocking module

Because the purpose of hnwProxy is to simulate a broken network, most of its modules work via IPTables, a software firewall commonly used by Linux distributions. IPTables was chosen because of its widespread use and documentation availability. Some Linux distributions use an IPTables frontend such as UFW or FirewallD, but these would not have been appropriate as they hide features with the intention of making IPTables easier to use.

4.5.1 DNS Block

The dns-block module blocks queries for specific types of DNS records. For example, A, AAAA or NS records. This works using the IPTables "string" module which matches a sequence of bits in a packet, commonly entered as a string. In this case, we need to match the Type field of a packet, which is 2 bytes and specifies what type of record is being queried. Because the string module searches the whole packet for the sequence of bits, and the type field is only 2 bytes long, using this alone would result in a noticeable number of false positives. One way of reducing false positive is to increase the amount of data that needs to be matched. Luckily, right after the Type field is the Class field, which is almost always the same value of IN, aka. 0x0001 (00000000 00000001 in figure 14). With this we can double the amount of data that needs to be matched, now 4 bytes. Finally, with IPTables protocol and port filtering we can apply the filter only to DNS queries which can be identified by the UDP protocol and destination port 53.

There are several ways this module could have been made, but IPTables-only was chosen because it was simple and did not require any additional software to be installed. Another method would have been to use IPTables' NFQUEUE feature which allows for delegating the decision of passing/dropping a packet to userspace software. A python script could have used the NetfilterQueue library (Kerkhoff Technologies, Inc. 2016, accessed 2017-01-10) to interact with NFQUEUE, and scapy for analyzing received packets.

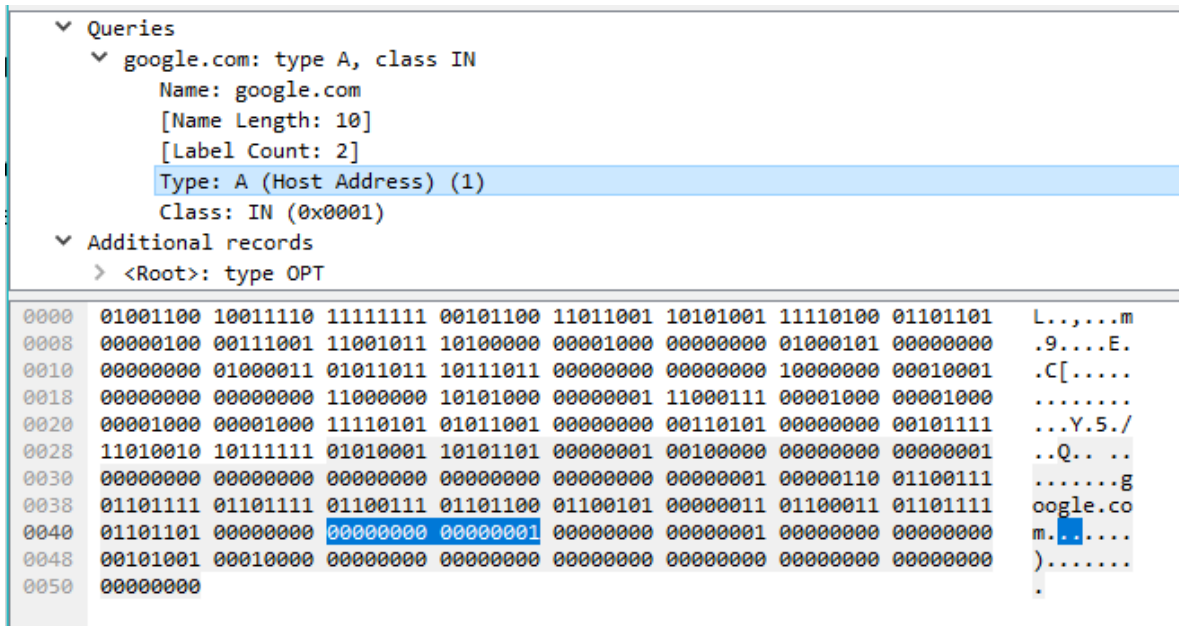


Figure 14. DNS A record query captured with Wireshark

4.5.2 Transparent Proxy

The transparent proxy module transparently proxies http and https traffic. There are two parts to this module: the IPTables rules that redirects network traffic and the transparent proxy server, mitmproxy. The IPTables rule use the REDIRECT target which redirects packets to a specified port on the local machine. This is done by changing the packet's destination IP. Mitmproxy was chosen as the proxy server because it is open source, has a good amount of documentation available, and supports proxying TLS/SSL connections.

4.5.3 DNS Redirect

The `dns-redirect` module redirects DNS packets to a user specified server. This can be useful for simulating a user that is using a malicious DNS server, either because of a hacked device or because their packets are being modified in-transit. This module uses IPTables' destination NAT feature in a slightly unorthodox way to redirect DNS packets (distinguished with the TCP/UDP destination port 53). Normally destination NAT changes the destination address of a packet, for example to direct it to an internal server with a private IP address. This same process is used, except in reverse: the NAT is applied to packets going out from the server instead of coming in. This allows for us to redirect packets to an arbitrary IP address.

4.6 Maintenance

As a software project, development of `hnmProxy` is never over. Even after major development has stopped, it is necessary to maintain the project as its dependencies and other related software continues to change.

The OS used by `hnmProxy` is Ubuntu 14.04. Being an LTS release, it will be supported until 2019. Some of the `hnmProxy`'s base configuration is done with Ansible. This uses declarative configuration, so it should be less fragile than a script that does the same thing. The way `hnmProxy` was developed, it should be compatible with various Linux distributions with a little tweaking. Most Proxy CLI modules work using IPTables, which is configured using Ansible. IPTables is common to many Linux distributions. The greatest maintainability weakness appears to be `hnmProxy`'s use of the `apt` package manager. This is used to install software during provisioning and by some modules in their setup scripts. Not all Linux distributions use `Apt`, and software available — its version, or even existence — varies with each distribution.

Using CircleCI, `hnmProxy` is provisioned to DigitalOcean once for each commit. This should be a good indicator that it still works, as a single provisioning step (such as running a script) failing will cause the test to fail. This does present one concern from the maintainability standpoint: what DigitalOcean account and API key will CircleCI use? Currently it uses the account and API key of this thesis's author along with DigitalOcean credits received as a benefit of being a student. A

single deployment costs almost nothing (0,01 EUR), but this is still something that must be addressed once the thesis is officially ended.

5 CONCLUSION

The purpose of this thesis was to develop a virtual appliance that would simulate a broken network connection. There are several similar projects in existence, but most only simulate the bandwidth, latency, and packet loss of networks. hnwProxy simulates less commonly thought of properties such as packet filtering and transparent proxying. These are not new concepts, but their implementation in a network simulating appliance probably is. This along with this report's *network quality factors* section are the two unique products of this thesis. When completing this thesis, there was no single source that could answer the question: "what makes a network connection good/bad?". Rather, it was necessary to compile information from multiple sources to gain a thorough understanding of the topic.

Infrastructure as code is an exciting and fundamentally different way of managing infrastructure, compared to traditional methods. This had many benefits in developing a virtual appliance, one being the fact that the appliance exists solely as a collection of plaintext files. These are in a public version control system that anyone can use and even contribute to. There is no need to build or host disk images, or for the user to download them, aside from a commonly used server template. The barrier of entry to make changes is low, and changes can be systematically managed using version control principles.

The result of this thesis was a virtual appliance that met its original requirements, and was approved by the project's commissioner. hnwProxy can be deployed to several virtualization platforms (VirtualBox, Hyper-V, and DigitalOcean), and can simulate a broken network in many ways that comparable software solutions do not. Some of these are things measured by HowNetWorks, such as which DNS server someone is using or which TCP/UDP ports are open. Because of hnwProxy's Proxy CLI module system, new features can be added with minimal overhead.

6 DISCUSSION

In many ways, this project is successful. It has met its original requirements and been approved by the commissioner (HowNetWorks). This being said, there are things that could have been improved.

This project was originally started in October 2016 with the intention of finishing in approximately 3 months. This is quite a tight schedule, but I thought it would work. My school schedule was almost empty until January 2017 so a lack of time was not a problem. This chapter was written as the last part of this thesis on January 29th, 2017.

Originally, I met with the HowNetWorks team weekly to discuss the project and get feedback. As the project progressed, these meetings became fewer and fewer. Around December 2016 progress slowed down noticeably and remained that way for the rest of this project. Maybe if I had been proactive and kept these weekly meetings going, that would have prevented this stagnation.

Another failure was starting the practical part of this thesis right in the beginning, alongside the theoretical part. I should have waited for at least 2–3 weeks to research infrastructure-as-code and networking in more detail. A lot of big decisions, such as what infrastructure platform to use and the overall topology of hnwProxy were made right in the beginning and not changed at all during development. If I had waited, I would have been better prepared to make these decisions. This being said, these decisions were discussed with HowNetWorks and seem to have worked out alright.

There are a few reasons why this thesis topic was chosen. The topic itself was not particularly interesting to me. More than that, I was interested in researching something new (infrastructure as code) and applying it to a project. I also wanted to network, to make new professional acquaintances. I could have gotten a topic from my current job in the IT field, and even been paid for my work, but chose not to for the reasons above.

REFERENCES

- 1.2 Getting Started – A Short History of Git. 2014. Accessed 2017-01-02, <https://git-scm.com/book/en/v2/Getting-Started-A-Short-History-of-Git>
- Dunn, J. 2015. Home router security 2015. Accessed 2016-10-26, <http://www.techworld.com/tutorial/security/home-router-security-2015-9-settings-that-will-keep-bad-guys-out-3609122/>
- Gibson, D. 2016. SSCP Systems Security Certified Practitioner All-in-One Exam Guide, Second Edition. USA: McGraw-Hill Education
- Hashimoto, M. 2013. Vagrant: Up and Running. USA: O'Reilly Media Inc.
- Hochstein, L. 2015. Ansible: Up and Running. USA: O'Reilly Media Inc.
- Hodges, J., Jackson, C. & Barth, A. 2012. HTTP Strict Transport Security (HSTS). Accessed 2016-12-03. <https://tools.ietf.org/html/rfc6797>
- Kerkhoff Technologies, Inc. 2016. Accessed 2017-01-10, <https://github.com/kti/python-netfilterqueue>
- Kozierok, C. 2005. The TCP/IP Guide. USA: No Starch Press
- Kurose, J. & Ross, K. 2013. Computer Networking: A Top-Down Approach (6th edition). USA: Pearson
- Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D. & Jones, L. 1996. SOCKS Protocol Version 5. Accessed 2016-12-20. <https://tools.ietf.org/html/rfc1928>
- Marlinspike, M. 2009. New Tricks For Defeating SSL In Practice. Accessed 2016-12-03. <https://blackhat.com/presentations/bh-dc-09/Marlinspike/BlackHat-DC-09-Marlinspike-Defeating-SSL.pdf>
- Morris, K. 2016. Infrastructure as Code. USA: O'Reilly Media Inc.

Packer Documentation, HashiCorp. 2016. Accessed 2016-11-06, <https://www.packer.io/docs/>

Savola, P., 2013, Copyright injunctions against Internet connectivity providers especially with regard to peer-to-peer networking. Aalto University. Communications and Networking. Licentiate Thesis.

Singel, R. 2008. ISPs' Error Page Ads Let Hackers Hijack Entire Web, Researcher Discloses. Accessed 2016-10-26, <https://www.wired.com/2008/04/isps-error-page>

Srisuresh, P. & Holdrege, M. 1999. IP Network Address Translator (NAT) Terminology and Considerations. Accessed 2016-11-06. <https://tools.ietf.org/html/rfc2663>

Treat, T. 2016. Comcast. Accessed 2016-10-25, <https://github.com/tylertreat/comcast>

Version Control System Popularity in 2016. 2016. Accessed 2017-01-02, <https://rhodecode.com/insights/version-control-systems-2016>

Web Proxy Auto Discovery Protocol, Cisco. 2010. Accessed 2016-12-03. https://www.cisco.com/c/en/us/td/docs/security/web_security/connector/connector2972A/WPADA_P.html