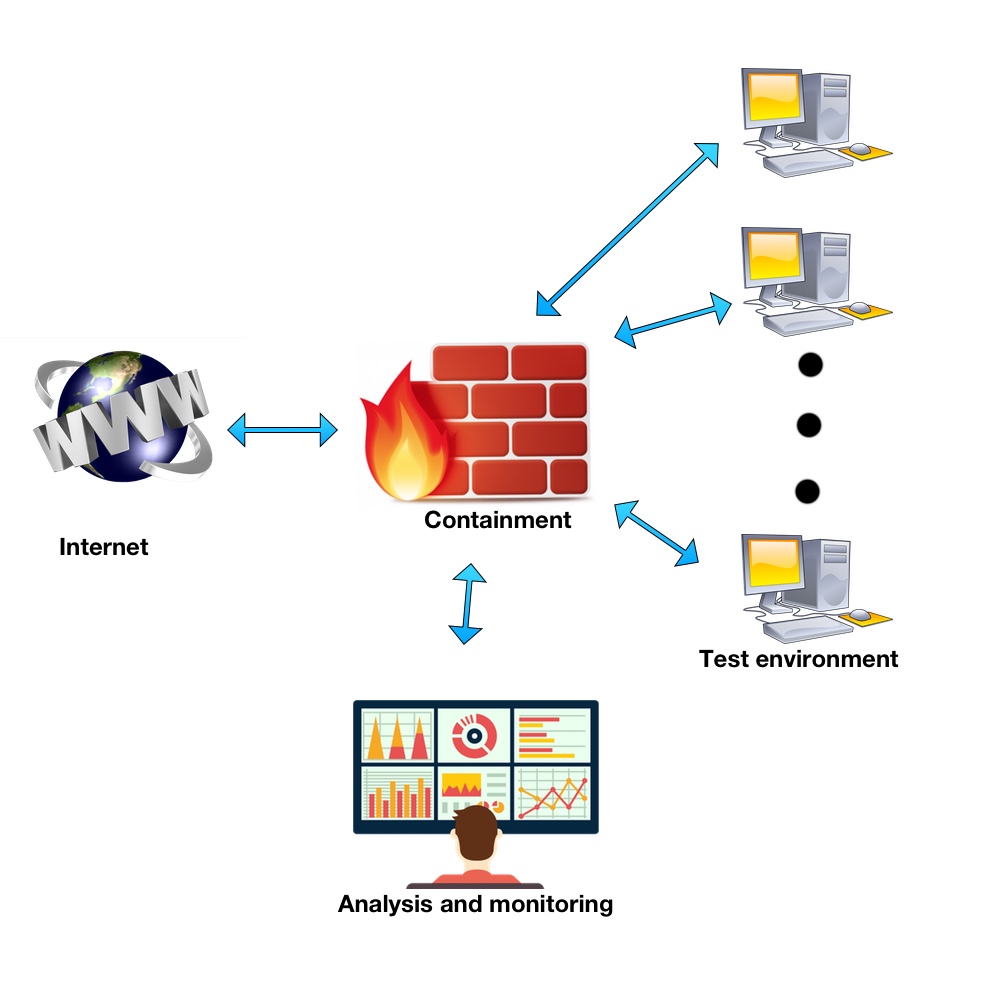
# HoneyPot system – Core features

This document is conducted to describe the existing core features of the HoneyPot system, and is meant for internal use.   
  
The document will take it’s point of departure in this design model of the HoneyPot system:



As shown in this model, the system structure is divided into primarily three sections:

**Test environment**

**Containment**

**Analysis and monitoring**

The final component included in the figure is the internet, illustrating that the system, through the containment filters, is connected to the internet to allow for active communication.

The following section will describe the existing features for each of the three divisions in depth, and discuss the current vision for the implementation of a GUI.

## Test environment

The primary goal of the test environment is to configure and control virtual machines. As the superior goal of a HoneyPot system is to produce data from machines infected by some kind of malware, it’s important to be able to define and control multiple parameters of the virtual machines. To specify, some of the main tasks of the test environment is:

* Able to start/stop virtualization
* Choose which type of malware infection to run.
* Choose which operating system should run on the virtual machines.
* Choose the number of virtual machines to run.

The current test environment is controlled by a python script allowing for the following functionalities:

* Create virtual machines from Jason file
* Duplicate/clone virtual machines
* Start/stop virtualization of machines. For now, this function is manual but it can easily be expanded to be controlled by time intervals or such.
* Generate PCAP files for individual virtual machines.
* Save PCAP files to specific location.
* List active and non-active defined virtual machines.
* Destroy and recreate virtual machines.
* Configure network settings for virtual machines.

Further development of this script will include:

* The ability to easily control what malware type to infect the machine with.
* Effectively emulating user behaviour
* Choice of OS running on virtual machines?

## Containment system

The current containment system consists of three different components:

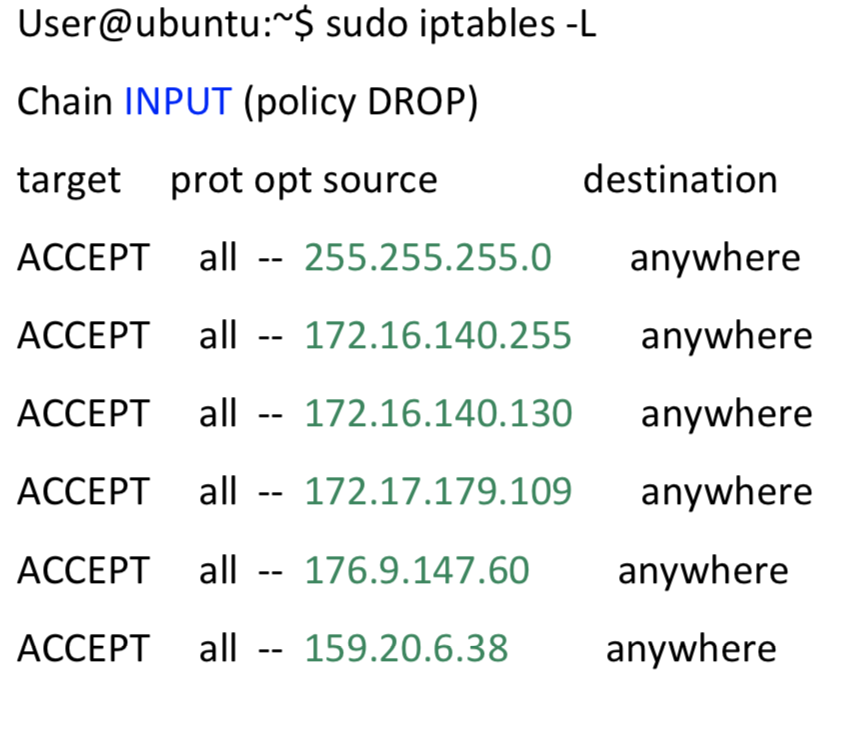
* A bandwidth limiter
* Configurable IPtables firewall
* Dedicated internet connection for the server established to the Danish research internet.

### IPtables

IPtables is a configurable packet-filtering software firewall provided by the Linux Kernel and implemented in all Ubuntu operating systems. It allows for rudimentary control over in- bound, outbound and forwarded traffic in a system. Customization of the firewall is done by defining rulesets. As the firewall is implemented in the Linux Kernel software, it is operated by executing BASH commands in the default Ubuntu terminal. IPtables allows a user with root access to edit pre-existing or define new rules, telling the fire- wall how to treat a packet. These rules are saved in one of the five predefined chains:

* PREROUTING: Packets reach this chain before a decision of routing is made, meaning that it is the initial chain that all inbound traffic will reach.
* INPUT: If told so by the PREROUTING chain, the INPUT chain will handle packets send to be locally delivered.
* FORWARD: Packets that have been routed without a direct local address, traverse this chain to be redirected.
* OUTPUT: Packets being sent from the local system will be directed in this chain.
* POSTROUTING: After routing decision is made, the POSTROUTING chain makes for the last decision of executing the delivery.

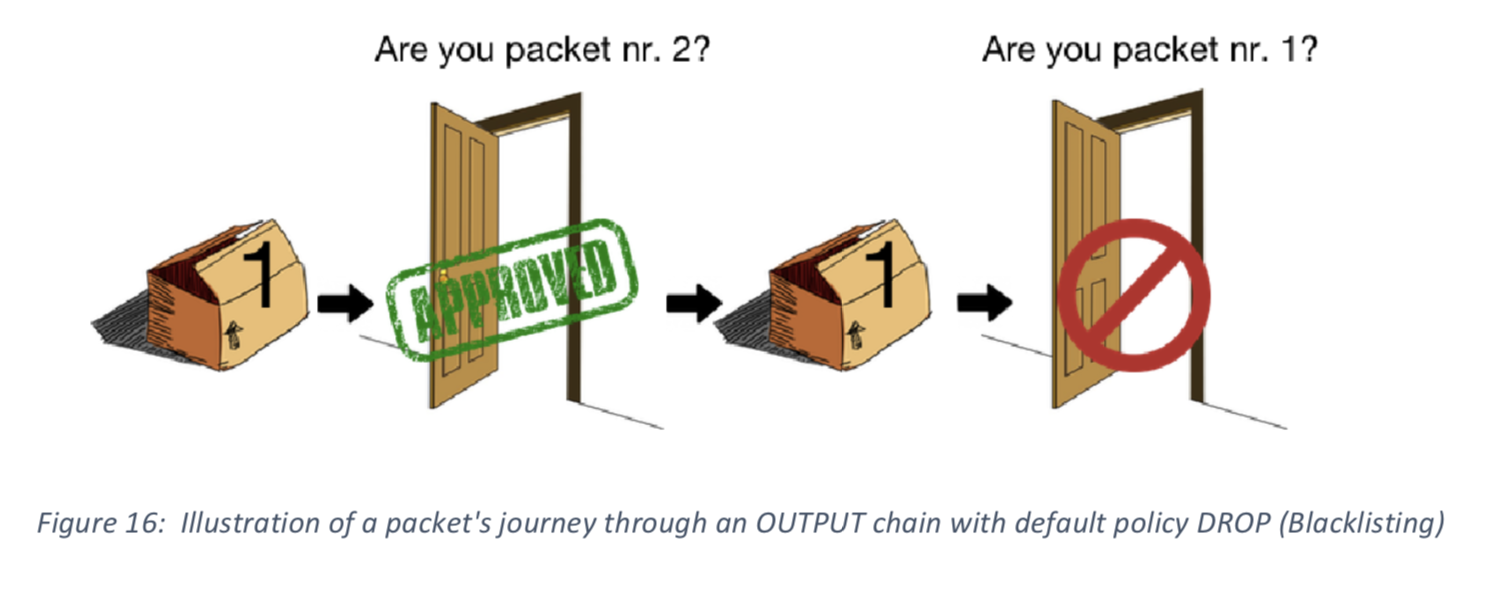
The chains are built after a sequential principle – as the name indicates. Each rule in a chain will traverse in the sequential order that they have been defined. When defining a new rule, it can be added to the end of the specified chain by using the command –A, or inserted as the first position by using –I.  Consider the following example of an INPUT chain containing rules allowing traffic for certain IP- addresses:



If a packet is routed to the INPUT chain, it will systematically test if the sender matches the IP- addresses listed in the rules from top to bottom. If a new rule is added using the –A command, it will be listed below the already existing rules and effectively be examined as the last option. Only packets that didn’t match any of the other IP-addresses will ever reach the last rule of the chain. Opposite, if a new rule is inserted by using the –I command, it will be listed as the top rule and be tested for as the first possible rule.

In conclusion, a packet will continue systematically to traverse through the chain until it meets a matching rule. If a matching rule is met, the target command will be executed. Target commands involve either redirection of the packet, acceptance for the packet to move on or dropping pack- etc. to be discarded. If the last rule of the chain is reached, the default policy of the chain will be executed.

Figure 16 illustrates how a packet traverse through a chain with default policy DROP, as the IPtables firewall will be configured for this project. A packet, labelled number 1, approaches the first rule of the chain, it checks if it is a match. In this case, the rule looks for packet number 2, so the packet continues on to the next rule. If this rule checks for packets with the label 1, the traversing packet will be recognized and the policy of the rule will be carried out.



#### Current IPtables predefined commands

By saving predefined BASH scripts with specific functionalities, and using the terminal to execute these, a more user-friendly interaction with the IPtables firewall is established. Each of these scripts are thought to be connected to a specific function in the GUI controlling the honeypot system.

At the current stage of the development of the IPtables component, the predefined scripts available are:

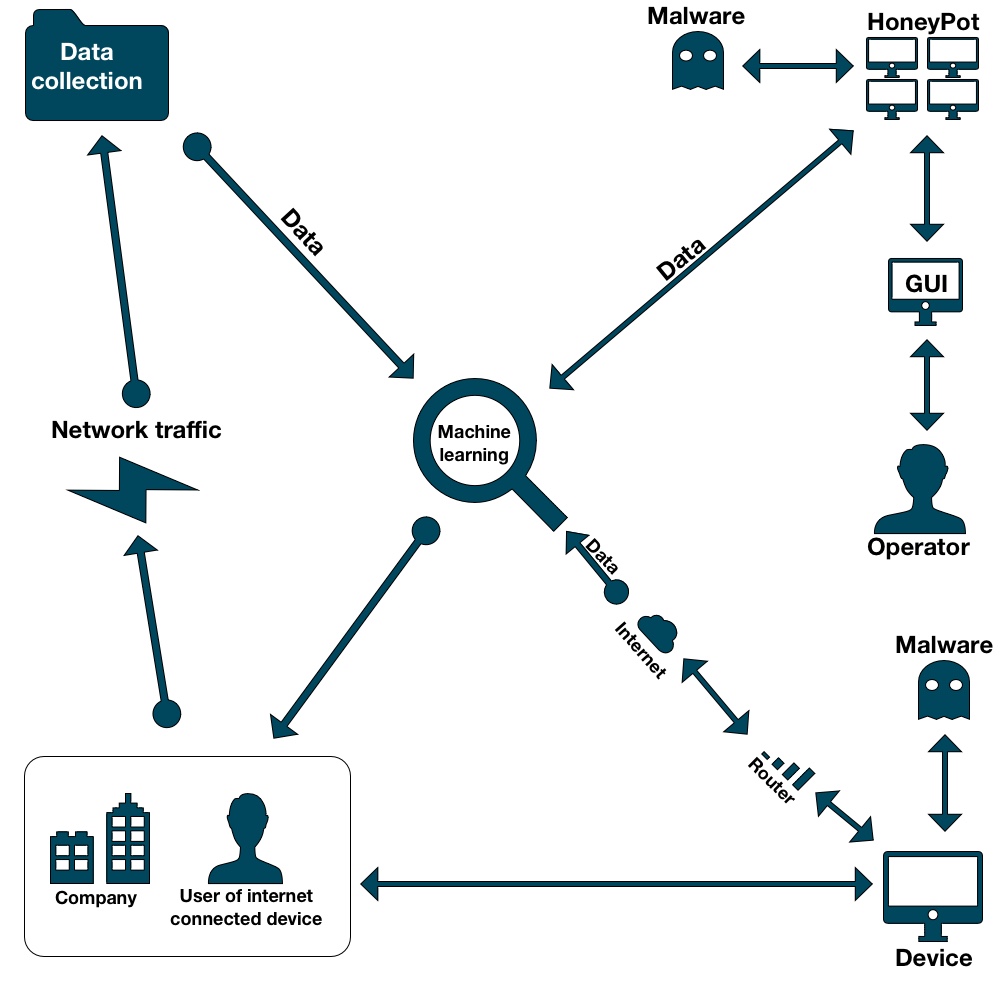
* *commands:* Lists all available commands.
* *clearRules:* Clears all defined rules across all chains.
* *listRules:* Lists all currently defined rules.
* *deleteRule:* Deletes one specific rule from a specific chain.
* *ipv4:* Adds a rule concerning an ipv4 address to a specific chain with chosen policy.
* *ipv4Var:* Lists the variables and their sequence input related to the ipv4 command.
* *port:* Adds a rule concerning a port to a specific chain with chosen policy.
* *portVar:* Lists the variables and their sequence input related to the port command.
* *setChainPolicy:* Changes the default policy of specific chain.
* *showTraffic:* Shows current traffic streams related to defined rules.

## Analysis and monitoring

The analysis and monitoring part of the HoneyPot is where the generated data from the test environment is to be examined. Up until now, there’s no real structured program here or anything. It’s basically just been a platform for doing network analysis of generated PCAP files, looking for parameters on a packet level showing signs of malicious network traffic.

In the HoneyJar project, the analysis and monitoring segment, will have a role of extreme importance. High quality networking analysis on a much deeper level than previously done will be conducted, to establish/extract features in the correct format for a neural network based machine learning algorithm to understand. This data will make the foundation for the detection algorithm as well.

## Explanation of conceptual model for HoneyJar



* **Concepts, metaphors and analogies of usage**

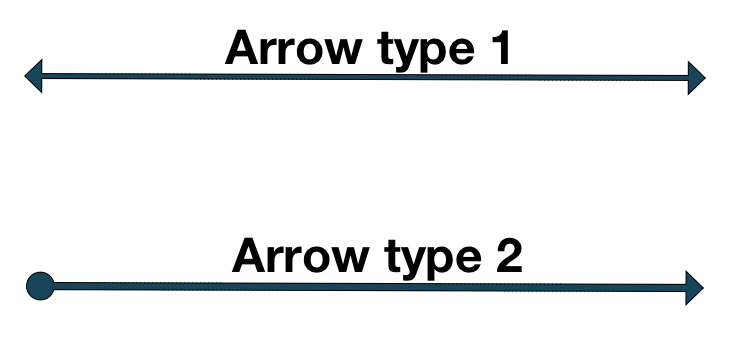
HoneyPot: An automated system running malware tests multiple virtual machines and generating network traffic data in PCAP file format. The virtual machines are emulated closely to actual user behavior, and connected to the internet to attract malware. This process of attracting outside attacking malware, is the logic behind the name HoneyPot, referring to how an actual honeypot will attract bees and other animals interested in the sweetness of the honey.

Malware: Malware is an abbreviation of Malicious software. It’s active code with malicious intents – like stealing data, encrypting data or destructive behavior.

GUI: A graphical user interface Is ideal to control the processes of the HoneyPot. A user-friendly interface to control virtualization, infection type, user behavioral patterns and such, would allow for much more effective use of the system.

* **Objects, attributes, actions and their manipulation**
* *Malware*: Represented by a Ghost, referring to its malicious/evil intents.
* *Network traffic:* Represented by a Lightning, referring to the extremely quick exchange of network traffic between two clients (Internet and device).
* *Operator*: Represented by a human, referring to the operator being an actual human user.
* *Data collection*: Represented by a folder symbol, referring to the data being collected in a file format.
* *Internet*: Represented by a cloud, referring to cloud services.
* *Router:* Router represented by the classic connection bars, referring to a common symbol that most generic computer users know.
* *Machine learning*: Represented by a magnifying glass, referring to the fact that the machine learning algorithm will be searching through data, for specific pre-defined parameters.
* *Device:* Illustrated by a computer symbol, referring to some kind of device connected to the internet. Could be a computer, but could be a smartphone, tablet or such as well.
* *HoneyPot*: Represented by a small cluster of computers, referring to the system running multiple virtual machines.
* **Relationship between concepts**

In the conceptual model, the relations between concepts/components of the system, is represented by two different types of arrows.



Arrow type one has arrowheads in both ends, illustrating that active communication between the two involved components Is established in relations represented by this type of arrow.

Arrow type two has a solid dot in one end and an arrowhead in the other. This is illustrating the relation between the two involved sides of the communication. In relations represented by this type of arrow, the involved component on the arrow side with the dot actively communicates in the direction of the arrowhead, to the component on the side of the arrowhead.