PhD Research Proposal: Semantic Modelling of Network Traffic for Anomaly Detection

Henry Clausen

February 2, 2019

1 Research description **2**
1.1 Introduction and motivation . . . 2
1.1.1 Anomaly detection . . . 3
1.1.2 Gaps . . . 4

2 Proposal **6**
2.1 Project aim . . . 6
2.2 Work achieved so far . . . 7
2.2.1 Data analysis, strategy, and gathering useful data sources . . 7
2.2.2 Ground truth data generation . . . 8
2.2.3 Testing initial approaches on realistic data . . . 10

3 Project specific objectives **12**
3.1 Learning representations of connection/protocol behaviour . . . 12
3.1.1 Probabilistic Real-Time Automata . . . 13
3.1.2 Autoencoder-based . . . 15
3.1.3 Data and result validation . . . 16
3.2 Software evolution and drift . . . 17
3.2.1 Evaluation . . . 17
3.2.2 Adaptive learning . . . 18
3.3 Improving flow-level based methods . . . 18
3.4 Traffic generation and data fusion . . . 19

4 Time-span and potential risks **21**
4.1 Risks . . . 21

A **22**
A.1 Literature Review . . . 22

1. Research description

1.1 Introduction and motivation

Sophisticated data breaches affect hundreds of million customers and inflicts tremendous financial, reputational, and logistic damage. One reason for the recent rise of cyber crime is the increased use of sophisticated techniques for the attack of specific targets. Attackers use customised social engineering and custom-build malware that penetrate defensive barriers and potentially stay undetected in an infected system for an extended period of time.

Cyber-Security relies on a range of defensive techniques, including sophisticated intrusion detection systems and firewalls that try to detect and prevent attacks against software subsystems. Malicious software still remains the biggest threat to computer users, and its detection is of utmost importance.

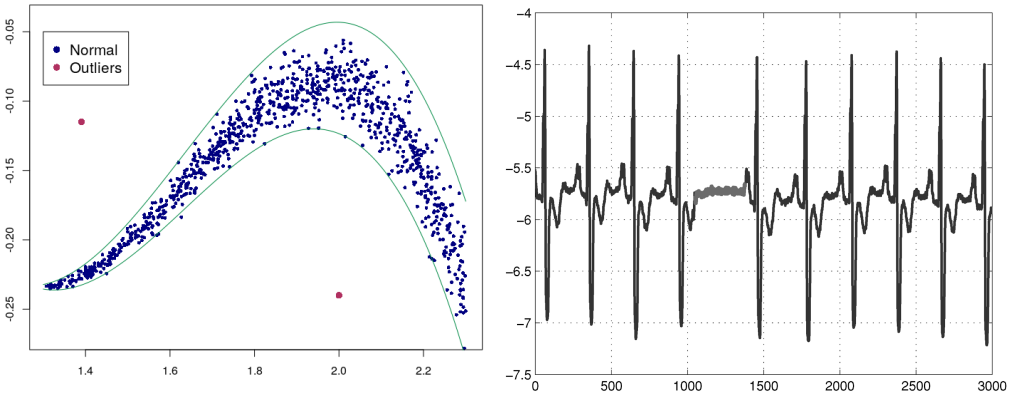
ueberarbeiten****** Program analysis methods are used widely to automatically test software for particular characteristics and behaviour, and thus identify malicious instances. Close attention has to be paid at the type of features and models that quantify the behaviour of software, as a lack of general program representation leads to low robustness against new or polymorphic malware, and consequently a poor classification performance. Chen et al. (2016) [4, 3] demonstrated convincingly that semantic features are suitable to reflect the nature of software, benign or malware, in an accurate manner. ******ueberarbeiten

In modern computer networks, it is often impossible to use software analysis on a large-scale basis to prevent network intrusions through malicious software. Here, network intrusion detection systems play a vital role in protecting computer networks from malicious access. The field of intrusion detection is concerned with the development of methods and tools that identify and locate possible intrusions in a computer network. An *intrusion detection system* (IDS) is typically a device or software application that detects malicious activity or policy violations in an automated way by scanning incoming data collected from one or more sources for patterns that a model or a set of rules classifies as malicious.

Intrusion detection is a well researched area, with the first IDSs emerging in the late 1980's. Intrusion detection today comprises a variety of research areas in terms of different types of data sources, system architectures, detection sope, and so forth. Figure **??** provides a broad yet uncomplete overview of these different areas.

Current detection methods are predominantly based on the analysis of previously identified attack signatures, which provides great reliability and low false alert rates. However, these methods are dependent on an updated attack signature database and provide no protection against previously unseen attacks.

Another approach that has recently gained traction in commercial deployment



(a) (b)

Figure 1.1: The left plot (a) depicts simple anomalies that deviate in distance from regular data. The right plot (b) shows a contextual anomaly where a group of data instances do not follow a repeating timely behaviour with respect to all other data points (corresponding to an *Atrial Premature Contraction*, taken from [2])

is based on detecting malware and other undesired activity as anomalous behaviour when compared to benign computer activity. In this approach, known as **traffic anomaly detection**, models that quantify the behaviour of normal network are trained on attack-free traffic data. Observed behaviour that significantly deviates from the trained model is then denoted as anomalous, and the intrusion detection system is taking further steps to investigate a possible intrusion.

1.1.1 Anomaly detection

Anomaly detection refers to the problem of identifying data instances that have significantly different properties than previously observed *"normal"* data instances. Anomaly detection has its origins in statistical research where normal data instances are assumed to be generated as random variables by a probability distribution $P_N(X)$. New data is then identified as anomalous if its properties correspond to regions with vanishing likelihood, i.e. this particular data instance is highly unlikely to be generated by $P_N(X)$. The hard part in anomaly detection is normally to use observed data efficiently to build an estimated distribution $\hat{P}_N(X)$ that resembles $P_N(X)$ closely in order to identify anomalous events while asigning normal instances a non-vanishing likelihood. A variety of techniques exist to achieve this assuming comparibly simple generating distributions. However, this may not always be the case as distributions generating for many interesting types of normal data can be complex and changing over time, and individual data points can have intricate interdependencies.

Anomaly detection has found wide application in areas where stability of critical systems or detection of their abuse is crucial. Examples of these include engine-fault detection in wind-turbines, fraud detection for credit cards. The assumption here is that the modelled system, such as the sensor-data stream of an engine or the buying behaviour of a customer, remains stable and generates consistent data. Detected anomalies in new observations then indicate potential fault or abuse in this system. Obviously, it is not inherently clear that every abuse or fault generates

data that differs from normal data. Therefore, it is important to choose data sources are able to reflect the unique nature of a particular system.

Anomaly detection and NIDS

Similar to the above described examples, anomaly detection has been applied to network security since the late 90's in order to assist IDSs, with the behaviour of network computers being quantified in the form of events logs capturing network traffic, system calls, etc. The basic approach is to use collected training logs to infer a model of trustworthy activity, and then compare new behaviour against this model. Again, the assumption is that unauthorized and malicious activity will correspond to behaviour that differs from trustworthy one. As an example, an anomalous network traffic pattern in a involving unusual connection pairs could signify that a hacked computer is extracting or sending out sensitive data to an unauthorized destination.

Insert stuff about NIDS and anomaly detection, drill down on network traffic, maybe talk about technical structure, how training data is gathered, then trained, collection of the traffic, possible anomalies in training data etc, scarcity of datasets. Talk about how anomaly detection should be able to find new attacks compared to signature based approaches.

Literature Summary and identified gaps

Network traffic can be processed into data features of very different nature. Consequently, approaches to apply anomaly detection in network security vary significantly in their scope and methodology. Individual

Although commercial intrusion detection systems still relies predominantly on *Signature-based* methods, anomaly detection is more and more used examples

I provide a detailed analysis of the most important techniques and approaches in my literature review, which can be found in Section A.1.

1.1.2 Gaps

As pointed out above, anomaly detection techniques currently perform best in the detection of attacks of larger volume, such as *DoS attacks* or *Port Scans*, and currently are the most common application in commercial IDS systems reference or example of IDS. In contrast, research in this area has been less convincing in the context of more targeted attacks that consist of only one or a few connections, with *R2L* and *U2R* attacks currently being the least detected attack classes [10].

This can in part be explained by a lack of understanding of how these smaller, more point-like attacks differentiate themselves from normal traffic. Attacks using a larger number of connections will inevitably disturb the distribution of one or multiple measures of the network, such as the byte-throughput or the port entropy [8]. However, an attacking connection inserting malicious code to gain control over a computer does not necessarily have different external properties[1] such as length, size, or port number, than the diverse range of normal connections. Yet, most attempts to detect these types of attacks via anomaly detection rely exclusively on

[1] also caled *connection summaries*

sumary properties of individual connections or packets to draw a border between normal and malicious traffic. Only very few approaches try to detect malicious connections as contextual anomalies, i.e. as traffic events that are not necessarily unusual on their own, but deviate from normal traffic in the context of their immediate temporal neighbourhood. More information about this can be found in my literature review in section A.1 exact chapter reference. Considering both the lack of literature in this area, this project will lok more at the intrinsic nature of network traffic which is manifested in the *semantic structure* of packet and connection sequences.

Talk maybe a bit about robustness? but maybe just mention it in the probject proposal

Cite Sommer and Paxson [15].

2. Proposal

2.1 Project aim

Chen et al. [4, 3] recently demonstrated the effectiveness of state-based software models in the identification of malicious sequences actions in a streamof permission and API calls of Android applications. This usage of semantic features allowed the discovery of new malware instances and improved the overall robustness of the classifier drastically.

This project will use the idea of semantic software models and apply it to network traffic data. It combine methods from machine learning and state-based software models, to automatically learn precise semantic models of network traffic generated by one or more machines. The semantic features of network traffic are here understood in terms of sequences of packet or connection metadata that correspond to fixed protocols of information exchange. These models together form a collection of normal semantic network behaviour of a network of machines, with malicious traffic being detected in an anomaly detection manner. Furthermore, a main aspect of this project will be to guarantee adaptivity and robustness of learned models to the evolution of software programs and their corresponding traffic.

The motivation for the use of semantic models in network traffic stems from the fact that current anomaly detection methods perform poorly in the identification of temporally isolated malicious connections, as described above, as their external properties such as size, duration, or direction, do not necessarily deviate from normal traffic. Attacks with traffic that falls into this category often break into a machine by exploiting loopholes in the processing of information of programs, and in one way or the other breaking the insert. Traffic with malicious intent will therefore likely deviate in its intrinsic structure from benign one, and should be detectable when modelling this structure using semantic models.

argue what that means, why it is promising, and that is has been done before on system calls and promises better robustness

- Goal 1
- Goal 2
- ...

2.2 Work achieved so far

2.2.1 Data analysis, strategy, and gathering useful data sources

Although this project is motivated by previous successful applications of ML-driven semantic models for anomaly detection, the differences in the type of data used in this project means that different tools and modelling strategies are necessary. Network traffic usually contains more intrinsic variation and noise, with packets potentially being dropped, and data distribution being afffected by the level of network congestion or by varying input parameters. Therefore, significant amount of time has been spent on initial data analysis and reflection on good modelling strategies.

Key assumption in this project is the following:

Hypothesis 1 *Application A running on a computer can be seen as a probability distribution $P_A(X)$ generating sequences X of network events, with X behaving as a random variable.*

Hypothesis 2 *Two applications A and B usually have different $P_A(X)$ and $P_B(X)$, which is applies especially to malicious applications. A computer running a set N of applicationsgenerates sequences X of network events with a distribution $P_N(X) = \sum_{n \in N} P_n(X)$. Consequently, a sophisticated and well enough trained anomaly model of a machine's traffic distribution $P_N(X)$ should be able to identify the existence of a new (and potentially malicious) application as traffic that did not originate from the set of existing applications N.*

An additional assumption about systems we are looking at is that the set of installed software is relatively constant throughout the network, with new software installations being rather rare and controlled.

Semantic structures and corresponding variations

Semantic behaviour of a program denotes how it transmits and reacts to information. This is can be observed both on a packet and on a flow level: Programs follow specific protocols on key-exchange etc. when a connection is established or how and with what frequency new bulks of data-packets are pushed, but also react to data retrieved in one connections with new connection requests such as the establishing of a HTTP-connection to an address received by an earlier DNS-request. Variation and noise is introduced in several ways:

- Varying sizes of data means a varying number and sizes of packets to be transmitted, which can in turn influence the way the receiver responds. Similarly, requested content can be made up of several files of different sizes which are transmitted in a varying number of connections with differing sizes.

- Encryption can in a simtoilar way introduce variation in packet or connection sizes and numbers.

- Variation in the computational load or on the network on a computer can translate into varying response times to retreived information or much data is buffered and pushed at once. Similarly, packet loss or deviations from the established protocol by one side usually means that information has to be transmitted again.

Alan Turing Data Study Group

2.2.2 Ground truth data generation

Building semantic models of network traffic means to build an understanding how different network interactions can be distinguished via their traffic trace. We want to use ML techniques to automatically extract meaningful sets of sequences that represent these different interactions. In order to ensure that this is actually true, that our extracted models are indeed representing real distinguishable interactions and not just nonsense, we need validation from *ground truth data*.

Network traffic datasets are already h7ard to obtain due to privacy concerns. However, as the correspondance between individual network traffic events and their particular purpose are virtually never recorded on a computer, their exist close to zero datasets containing ground truth about the contained events. For that reason, a particular aspect of this project is to generate useful ground truth data with appropriate content myself.

Over the course of the last year, Nikola Pavlov and I developed a framework that generates controlled and isolated network traffic from a variety of applications and tasks. For this, a virtual network was created using the virtualisation programm *Docker*. In this network, two or more parties can communicate as containers, which are sandboxes containing programs inside a minimal virtualised operating system. The benefit of this design is that individual containers can only communicate with each other via the virtualised network while the host is in complete control of the parallel execution of tasks in multiple containers. In order to capture the traffic, every container in the network was complemented with a *tcpdump*[1]-container hooked onto the network interface. The captured traffic can then be labelled according to the particular scenario it was generated by.

We implemented a variety of network service scenarios to capture a diverse set of network traffic. Most, but not all, are set up in a server-client manner. Implemented services so far include:

- A simple icmp ping service.
- A *nginx* server hosting a html-webpage with a client accessing it, both encrypted and unencrypted. Different client requests are implemented (*Wget* and *Siege*).
- An *Apache* server hosting the same html-webpage with a client accessing it, also both encrypted and unencrypted. Again, different client requests are implemented (*Wget* and *Siege*).
- A *Wordpress* server with a corresponding webpage and a client accessing it.

[1] A common packet capture utility

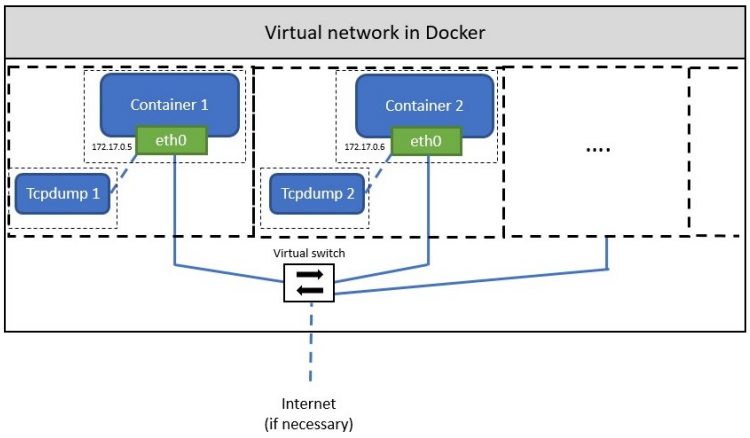


Figure 2.1: Visualisation of the virtual network in Docker

- Different versions of an ftp server and client.
- A mail transfer using *mailx*.
- An ssh-server and client.
- An *IRC* chat server and two clients.
- A file synchronisation service with multiple clients.
- A web-crawler gathering a larger volume of traffic from the internet.

For each service, a number of different scenarios are implemented. For example, the ftp client could pull, push, remove, or a combination of all one or multiple files[2], or make directory. Furthermore, randomisation is introduced on parameters like the file-sizes, request times, etc. in order to explore the dimensional variation of the traffic from individual actions.

As ..., the data generation should obey two basic principle:

1. The execution of individual scenarios should provide consistent data.
2. The data should be completely free from external influence, i.e. properties of or activities on the host machine should in no way disturb the data generation.

Both properties were tested for selected scenarios with positive results. For the first, randomisation of the input was avoided as much as possible and the results of several scenario executions were compared. The second was tested by executing scenarios under differing workload and on different machines, and summaries over the different packet properties were tested using a Kolmogorov-Smirnov test.

write about the application of the traffic, how to use it for validation.

[2]that might not exist

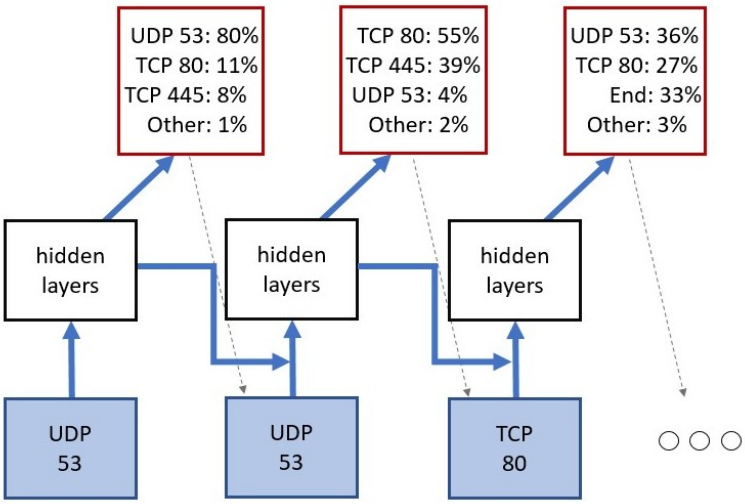


Figure 2.2: Visualisation of the RNN-method used

2.2.3 Testing initial approaches on realistic data

In the context of the overall research goal of this project, some exploratory work on semantic anomaly detection for network events has been conducted by Marc Sabate, Gudmund Grov, Wei Chen, and David Aspinall. This work uses three different machine learning algorithms[3] from the area of *representation learning* to capture meaningful sequences of *NetFlows* and reflect reoccurring patterns in a model. For that, recorded NetFlows are grouped according to the generating host. Furthermore, in order to filter out sequences of flows that are unrelated to each other, a squence of flows that are close in time is grouped into what is called a *sessions* as an approximation "true relation". Each session then serves as a training or test sequence for a behavioural model. Learned semantic behaviour is reflected through the capability of the model to predict features of the whole session and of other flows in the session from a smaller subset of flows, with more accurate predictions being rewarded in the training process. These features include the network protocol and port and Sessions which deviate from previously observed behaviour are then predicted poorly by the model and flagged as potentially malicious.

maybe write about how these methods are directly relevant for this PhD and could be extended, improved or incorporated.

The described methods were tested on the CTU-13 dataset [6], a laboratory generated dataset available to study botnet detection. The great benefit of this data is the inclusion of both benign background traffic, on which the models were trained, and malicious traffic used to test their detection capabilities, and the availability of corresponding labels. Furthermore, it consists of more than ten million flows on multiple machines, thus containing reasonably much variation for testing purposes.

Anomaly-based methods in intrusion detection are often criticised for working only in controlled environments while failing in the real world. To counter this criticism, it is crucial to provide evidence of success on sufficient data gathered under realistic conditions. For the CTU-13 dataset, the generation of both types of traffic was done in a controlled laboratory environment, making it a so called

[3] A Markov Chain model, Finite-State Automata, and a Recurrent Neural Network

"synthetic" dataset. This does not necessarily mean that it does not reflect realistic behaviour, however there is no guarantee that it contains all variations encountered in an operational environment.

As the developed methods are a good foundation to build this project upon, David Aspinall and I agreed that I should apply the same methods on a real-world datasets that I already worked on during my Master's degree to bolster their results and another justification that this is relevent to my project because of the closeness of the methods.

write about LANL and UGR when you are done

3. Project specific objectives

3.1 Learning representations of connection/protocol behaviour

The distinct semantic behaviour of an applications manifests itself most clearly in the first few packages exchanged in a connection. This is the stage in a connection during which a specific client or server request is transmitted, encryption keys are exchanged, users authenticate themselves with passwords, ports for successive connections are defined, etc. This is distinction goes far beyond individual traffic protocols, as even a single application usually can excert different actions which translate to different semantic behaviours in this initial negotiation phase.

The significance of the initial negitiation phase is bolstered by two publications on the area of traffic classification that share the same conclusion and train clusters and classifiers on the metadata of the first five to ten packets to distinguish different applications[1, 5].

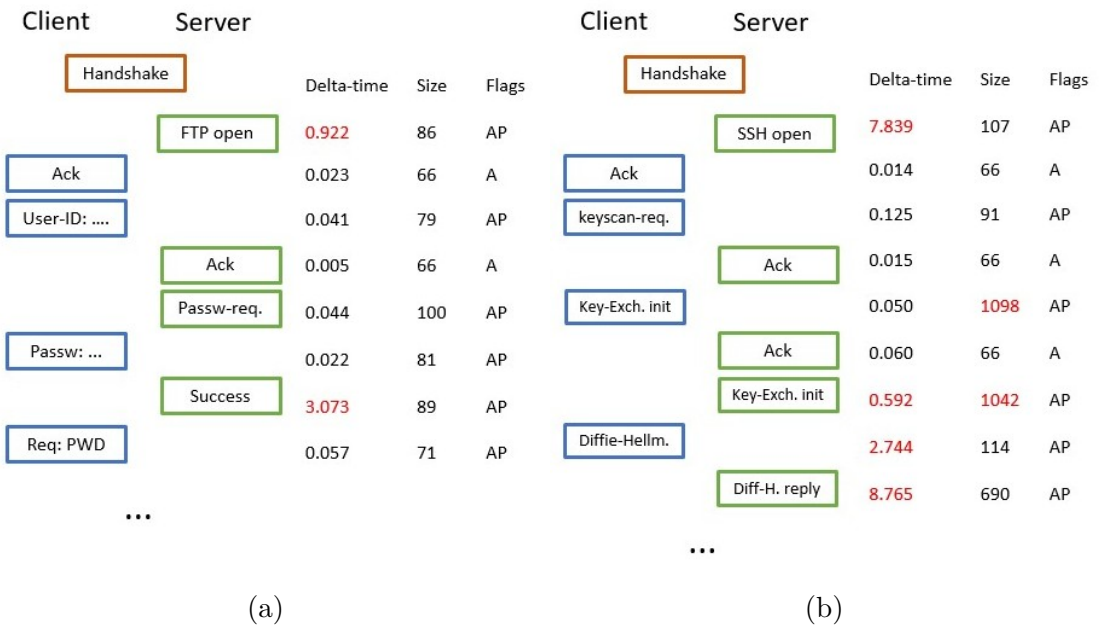
Learning precise semantic representations of this initial negotiation phase from applications running on a computer has several main benefits for this project:

- Several types of attacks exploit loopholes accessible at the negotiation stage. A precise semantic representation model could detect anomalous deviations in particular packet parameters from the learned behaviour. A simple example of such a deviation from expected behaviour would occur during a *buffer overflow attack*, which overflows an input buffer (such as for a password) to directly modify memory locations. This should be observable as a significant deviation in the corresponding packet size.

- Newly installed malware will most likely exhibit different than normal behaviour in this stage, from very different negotiation procedures to just slight deviations in the packet interarrivals.

- The translation of semantic negotiation structures into different behaviour groups could be used to associate the corresponding flow with particular actions, i.e. to give the flow one or more labels. To be meaningful, the identified groups have to correspond to actual semantic behaviours and should be consistent along similar actions. Such a flow labelling could be very helpful for understanding semantic structures of flow traffic, and consequently for anomaly detection on a higher level, such as the described work by Chen et al.[3]. I will describe this more in section 3.3.

- As applications are updated, their particular traffic generation distribution $P(X)$ can also gradually change, causing corresponding traffic to be poten-



(a) (b)

Figure 3.1: Depiction of user-ID and password exchange in an FTP-connection (a) and an SSH-connection (b) during the negotiation phase.

tially flagged as anomalous by current anomaly detection methods. A representational model could identify common semantic substructures between traffic instances and thus indicate that they originate from an updated application. More detail will be given in section 3.2.

Focusing exclusively on this initial phase, i.e. a fixed number of the first packets in a connection, has several benefits: We are dealing with a fixed and comparibly small input size, which lower computational cost and is beneficial for modelling techniques with a higher complexity. Our model is tainted by large data exchanges, which usually do not contain as much useful information but are spread across many more packets, which could create a high imbalance in the traffic representation. And we do not have to wait until a connection is finished for its evaluation.

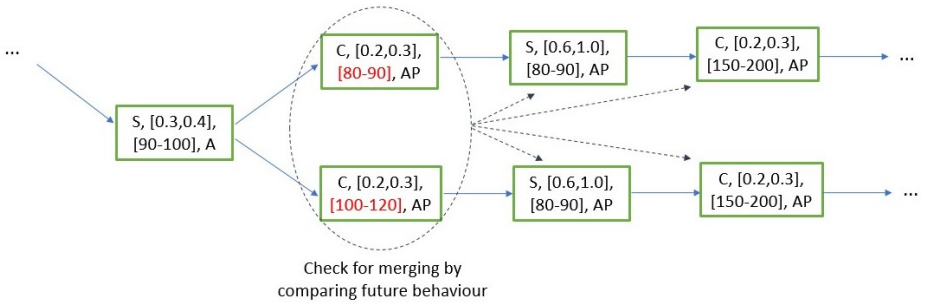
A model that offers enough representation of semantic structure has to go further than the above mentioned work, which was intended for traffic classification instead of cyber-security. Therefore, I propose two unsupervised modelling approaches that are intended to encapsulate significantly more information.

3.1.1 Probabilistic Real-Time Automata

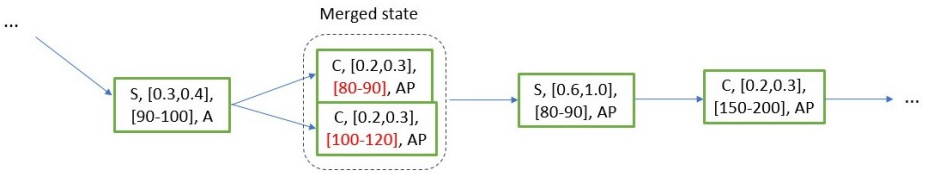
The *non-deterministic finite-state automaton (NFA* is a widely used model for discrete event systems and can be seen as a mathematical model of computation. It is a type of Markov chain with a finite number of states and a set of Markovian transition probabilities from each state to the others. A recent variant of the of the NFA is the *probabilistic real-time automaton (PRTA)*, which models the event interarrival times as a discrete distribution that is dependent on the current state.

Automata are learned from a set of input sequences via state-merging. In this procedure, a trie with all observed event sequences is built, with each path in the trie representing one input sequence. Pairs of paths that are close to each other are

then merged, thus generalising over the input data and learning the structure of the target machine. Closeness is determined by heuristic measures, with a merge considered good if the future behaviour after reaching state q in one path is similar to that after reaching $q\prime$. Figure 3.2 depicts this process.



(a)



(b)

Figure 3.2: Depiction of the state-merging process for two input sequences

As the exchange of network packets is a symbolic sequences of a computational process, they are a very suitable candidate to be modelled by probabilistic automata: The tree-branching in an automata is well suited to represent the different avenues a negotiation can take while the probabilistic state-merging ensures that potential variation within one avenue are accounted for. With enough traning data, automata can therefore provide a much more refined representation of a given set of packet sequences than previously explored techniques without overfitting. Anomaly detection to identify potentially malicious traffic can then be done by computing the likelihood of a input sequence trajectory and flagging it if it false a likelihood-ratio test.

Using probabilistic automata has further benefits in the context of this project. In contrast to many methods, they are interpretable by domain experts when checking, which is a priority for many researchers in order to process the inevitably[1] high occurrencies of false alerts. Furthermore, the branching structure in automata provides a natural grouping of sequences, which can be used to engineer labels for similar sequences in a similar manner to the *hierarchichal clustering method*. Such labels can be very useful in future ambitions discussed in section 3.3.

PRTAs have been applied to network traffic before by Pellegrino et al. [12], however in a different context and scale. To apply automata to this project, network packets would have to be transformed to discrete, finite states. Interesting properties to be included are

1. the direction of the packet,

[1]due to the inbalance between benign but irregular traffic compared to actual attacks

2. TCP-flags,

3. the interarrival time,

4. the size of the payload,

5. and potentially the offered window size.

To create states, continouos variables have to be transformed into a set of finite non-overlapping intervals. For a better resolution, a rescaling onto a logarithmic scale is very sensible. The combination of all attributes into one set of states is then straightforward.

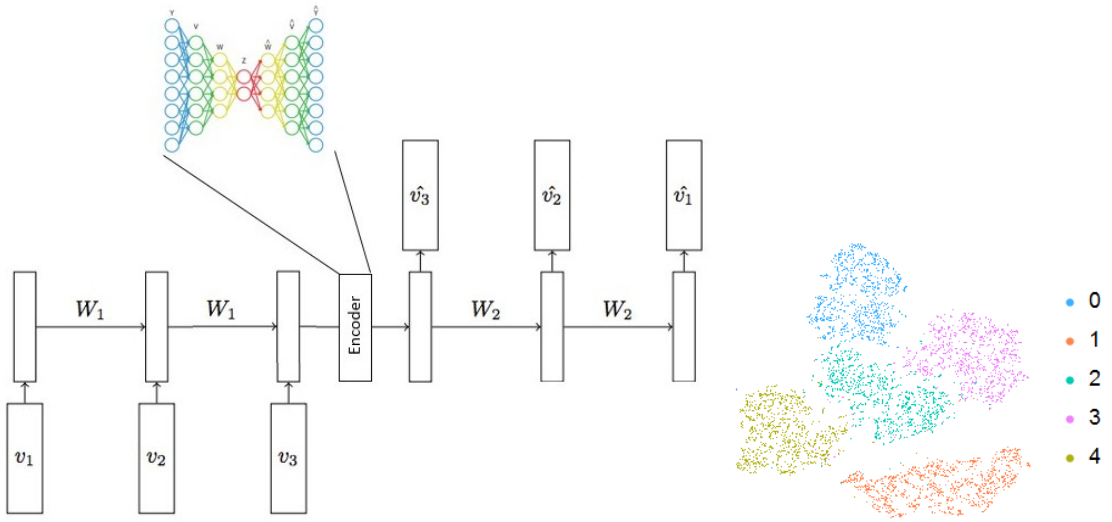
One challenge in this project is the large amount of training data needed to represent the variability of network traffic. Schmidt and Kramer [13] have recently proposed an online learning approach for PRTAs that enables the effective processing of massive datasets, which could be very useful for this project. A further thing to consider is that the chosen similarity measure for state-merging should be able to account for additional *ack*-packets inserted in the connection or potential packet loss. A possible solution is to introduce non-vanishing probabilities for both scenarios, i.e. it should be possible from every state to jump to an empty *ack*-packet and back, or to skip a packet. Another solution which could also reduce the potential computational burden is to learn multiple automatas from the input using rolling windows over, with each automata learning a specific interval of the negotiation phase, in a similar manner as in [12]. This could increase the stability of individual automata and decrease the size of the trie. Furthermore, as each automata would correspond to sub-behaviours in the connection, it would be possible to identify common subsequences in traffic from updated software and thus create a relation to previous traffic.

3.1.2 Autoencoder-based

Autoencoders are the most popular deep learning-based method for unsupervised representation learning. They can learn a compressed representation of input data by self-supervised training using the reconstruction error of the input data. Although autoencoders themselves do not produce a defined model of the input data, they translate complex structures into a more simple space in which they can be described by conventional density estimation methods.

Long short-term memory (LSTM) autoencoders are a recent version of autoencoders that use the structure of RNNs to capture the sequential nature of the input data before the encoding-process, first introduced by Srivastava et al. [16]. Since then, they have been successfully used for artificially creation facial videos, music generation, or text compression. LSTM autoencoders can be trained using backpropagation, and the application of an LSTM autoencoder to packet data is straightforward, however again time and size variables should be logarithmically rescaled. Anomaly detection can then either be performed by thresholding the reconstruction error of new input sequences, as has been demonstrated by Malhotra et al. [9].

LSTM autoencoders are an excellent alternative to PRTAs for the modelling of packet data due to their superb ability to discover non-linearities in sequential data, and the fact that they can be trained on massive datasets. They have been



(a) (b)

Figure 3.3: Visualisation of the design of an LSTM Autoencoder (a), taken from https://machinelearningmastery.com/lstm-autoencoders/, and the embedding of different types of handwritten digits by an autoencoder.

used to discover semantic structures before both for natural languages [17, 14] and computational sequences [18]. Furthermore, the embedding of input data into a lower dimensional continuous space means that clustering of similar traffic instances to create higher level flow labels can be performed, see Figure 3.3b.

3.1.3 Data and result validation

The above described methods should be tested for three properties:

1. Goodness of fit,
2. detection capabilities of malicious behaviour,
3. correspondance of engineered high-level labels to actual semantic behaviour.

Testing of the first two properties is straight-forward given a suitable dataset. Goodness of fit could be tested with any packet data collected from a suitable environment and could even be generated for the experiment. To test detection capabilities, a packet dataset containing labelled malware traffic is necessary. Two widely used datasets, the CICIDS 2017 and the UNSW-NB 2015 dataset are suitable candidates.

The third property is not as straight-forward to test, as hard ground truth about the purpose of individual connections is required to analyse the labelling consistency. Fortunately, such ground truth data is now available to us for a variety of services by our data generation tool, described in Section 2.2.2. To test the labelling consistency appropriately, a given dataset could be superimposed with labelled ground truth traffic, and split into training and test data. If individual scenarios in the test set are both identified consistently while associated with a different label than significantly different scenarios, the labelling can be seen as meaningful.

3.2 Software evolution and drift

Networks and their corresponding traffic are not static, but gradually change over time. Consequently, making intrusion detection systems adaptive to such changes in normal behaviour is an increasingly important issue in the research community [15, 11].

In our context, such changes in normal behaviour would appear in the observed semantic structures and could be induced by new software being installed on the observed machine, or by installed software changing through updates or similar modifications. As new software could equally be of malicious nature, there is no way to make our system robust against such changes without introducing additional notions of overall malicious software[2].

Software updates however should correspond to more gradual changes as only small parts of the software are modified. Hence, it is reasonable to assume that in the process only parts of the semantic traffic structure will change, while larger parts surrounding it stay the same. give an explaining example of this A logical approach to distinguish this gradual software evolution from more abrupt changes would therefore rely on the comparison of semantic sub-features to detect an overall closeness between traffic instances. However, more analysis of traffic evolution under software updates has to be conducted to make reliable statements about precise modelling methods. Yet, a few ideas are presented here.

Chen et al. [3, 4] introduced the concept of common sub-automata between similar malware instances in order to improve the robustness of detection methods. This notion could be used in this project in a similar way: Traffic instances deviating from the learned traffic automata are examined for sub-automata that are within the learned range of normal behaviour, depicted in Figure 3.4. If enough common sub-automata exist, the traffic is marked a potential software update and either authorised by an administrator or directly incorporated into the existing model.

Depending on the exact implementation of methods proposed in Section 3.1.1 and 3.1.2, there are several ways to design such a comparison. In case a rolling window based approach to learn traffic automata is chosen, a comparison between automata from different time-intervals is straightforward, with heuristic measure to decide on an overall closeness to be chosen. If instead complete traffic automata are learned, Chen et al. proposition for a machine-learning-centred algorithm to efficiently choose salient sub-automata could be used. For an autoencoder-based approach, successive traffic sub-intervals can be mapped into lower-dimensions, with their distance to the normal region indicating if each interval corresponds to normal behaviour or not.

3.2.1 Evaluation

In order to gather data and to test developed methods, I will again make use of our ground-truth traffic generation framework. For that, containers with different versions of the same software will be installed and the corresponding traffic will be gathered. Next to initial data analysis, this traffic can then be injected into real-world datasets in a similar fashion as described in Section 3.1.3, with traffic

[2] which is why we are assuming that new software is not installed on a regular basis

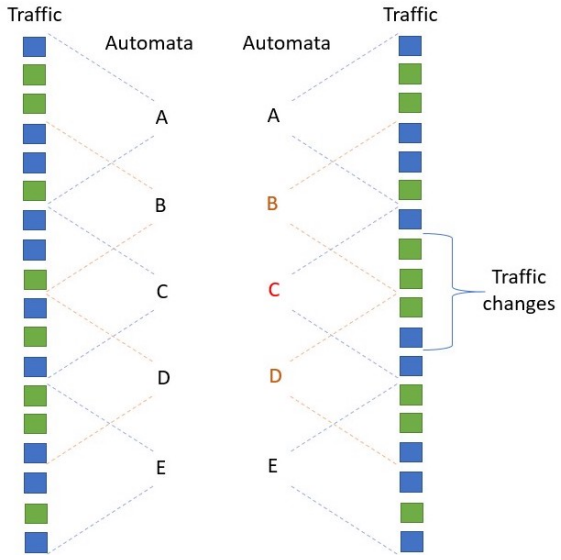


Figure 3.4: Depiction the comparison of sub-structures between two traffic instances.

from older versions being injected into the training data while that from newer versions into the test data. Such a dataset can then be used as a benchmark to test the ability of the above described methods to identify traffic from newer software version as such.

The data produced this way does not necessarily represent the influence of software updates on network traffic as a whole. However, it can give good indications of the amount of change introduced and is a good testing possibility for our methods in a very novel area of cyber-security until more extensive datasets become available.

3.2.2 Adaptive learning

Despite the identification of similar traffic instances, their incorporation into the existing model without introducing concept drifts or catastrophic interference is an important task. Online or incremental learning methods exist for both autoencoders and PRTA and could be used with new traffic gathered over time. However, special attention has to be paid to the fact that once traffic from updated software is identified, it should be included into the model immediately to avoid a flood of traffic alerts from the same software. I cannot propose solutions for this specific problem yet, but am optimistic that they will be found.

3.3 Improving flow-level based methods

Discovering meaningful semantic

because little information, and parameters like size etc. vary a lot more in flows than for individual packets.

Furthermore traffic overlay

Using more features

Using extracted features from the connection

Let above mentioned methods provide semantic understanding of connections to form groups, and then give the detected group as an input feature to the methods. That way, specific tasks performed in a connection can be identified and given as input to methods looking at connection sequences

3.4 Traffic generation and data fusion

To build meaningful semantic models, it is essential to compare discovered traffic features with the corresponding actions the traffic represents, for which ground truth traffic is necessary. Our data generation framework described in Section 2.2.2 is to our knowledge the first of its kind in the way that every generated packet or connection can be linked to a specific application and the specific task that application completed. This framework might not only be beneficial fo us, but also for other researchers interested in data with a high level of explanatory labelling.

In order to harness the unique opportunities this framework is offering on ground truth data generation, I hope to extend it in the following ways:

Simulate real network traffic distributions

As many network intrusion detection methods are designed for application to enterprise network traffic from personal computers, it is reasonable to evaluate them on traffic that has similar characteristics. As network traffic[3] consists of traffic originating from a set of applications performing different tasks, we can try to simulate network traffic on a machine by generating sequentially traffic instances from each scenario from a probability distribution specific to that scenario, and pooling it into one traffic stream. For that, the following questions are interesting to us:

1. What kind of applications and services are typically responsible for the generation of typical enterprise network traffic on individual machines?
2. Into which generalised subtasks can these services be broken down?
3. How can we quantify the frequency of each application generating traffic?
4. What other underlying traffic properties such as service correlations or network congestion did we not capture?

Data fusion

Apart from network traffic, applications often produce additional events captured in log files. These are usually intended to control whether a service is working properly, but are also increasingly used for cyber-security purposes. The fusion of multiple events streams such as log files and network traffic is a very promising direction in current intrusion detection as the combination of multiple indicators for malicious activity has the potential to significantly decrease the rate false positives, and some effort has been made to provide synchronised data sources, notably from the Los Alamos National Laboratory [7]. However, it is traditionally very hard to

[3]at least on an IP protocol level

link specific log events with corresponding network traffic events due to the lack of ground truth traffic. Due to the separation of traffic from individual applications and programmable execution of different scenarios, our framework can provide the unique opportunity to examine and model this relationship.

4. Time-span and potential risks

4.1 Risks

Scalability of methods

Attacks might against our expectations not deviate semantically from normal traffic Lack of appropriate attack data (DoS and port scans are not that interesting, R2L are often not very representative in a dataset)

A.

A.1 Literature Review

Bibliography

[1] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian. Traffic classification on the fly. *ACM SIGCOMM Computer Communication Review*, 36(2):23–26, 2006.

[2] V. Chandola, A. Banerjee, and V. Kumar. Anomaly Detection: A Survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, July 2009. 05458.

[3] W. Chen, D. Aspinall, A. D. Gordon, C. Sutton, and I. Muttik. More semantics more robust: Improving android malware classifiers. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, pages 147–158. ACM, 2016.

[4] W. Chen, D. Aspinall, A. D. Gordon, C. Sutton, and I. Muttik. On Robust Malware Classifiers by Verifying Unwanted Behaviours. In E. Ábrahám and M. Huisman, editors, *Integrated Formal Methods*, volume 9681, pages 326–341. Springer International Publishing, Cham, 2016.

[5] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli. Traffic classification through simple statistical fingerprinting. *ACM SIGCOMM Computer Communication Review*, 37(1):5–16, 2007.

[6] S. Garcia, M. Grill, J. Stiborek, and A. Zunino. An empirical comparison of botnet detection methods. *computers & security*, 45:100–123, 2014.

[7] A. D. Kent. Comprehensive, Multi-Source Cyber-Security Events. Los Alamos National Laboratory, 2015.

[8] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. In *ACM SIGCOMM Computer Communication Review*, volume 35, pages 217–228. ACM, 2005.

[9] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff. Lstm-based encoder-decoder for multi-sensor anomaly detection. *arXiv preprint arXiv:1607.00148*, 2016.

[10] A. Nisioti, A. Mylonas, P. D. Yoo, and V. Katos. From intrusion detection to attacker attribution: A comprehensive survey of unsupervised methods. *IEEE Communications Surveys & Tutorials*, 2018.

[11] J. Noble and N. Adams. Real-Time Dynamic Network Anomaly Detection. *IEEE Intelligent Systems*, 33(2):5–18, Mar. 2018. 00000.

[12] G. Pellegrino, Q. Lin, C. Hammerschmidt, and S. Verwer. Learning behavioral fingerprints from netflows using timed automata. In *Integrated Network and Service Management (IM), 2017 IFIP/IEEE Symposium on*, pages 308–316. IEEE, 2017.

[13] J. Schmidt and S. Kramer. Online induction of probabilistic real-time automata. *Journal of Computer Science and Technology*, 29(3):345–360, 2014.

[14] C. Silberer and M. Lapata. Learning grounded meaning representations with autoencoders. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 721–732, 2014.

[15] R. Sommer and V. Paxson. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *2010 IEEE Symposium on Security and Privacy*, pages 305–316, May 2010. 00654.

[16] N. Srivastava, E. Mansimov, and R. Salakhudinov. Unsupervised learning of video representations using lstms. In *International conference on machine learning*, pages 843–852, 2015.

[17] Z. Yang, Z. Hu, R. Salakhutdinov, and T. Berg-Kirkpatrick. Improved variational autoencoders for text modeling using dilated convolutions. *arXiv preprint arXiv:1702.08139*, 2017.

[18] M. Yousefi-Azar, V. Varadharajan, L. Hamey, and U. Tupakula. Autoencoder-based feature learning for cyber security applications. In *Neural Networks (IJCNN), 2017 International Joint Conference on*, pages 3854–3861. IEEE, 2017.