BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

IMPROVEMENT OF ADVERSARIAL CLASSIFICATION IN BEHAVIORAL ANALYSIS OF NETWORK TRAFFIC INTENDED FOR TARGETED ATTACK DETECTION

VYLEPŠENÍ ADVERSARIÁLNÍ KLASIFIKACE V BEHAVIORÁLNÍ ANALÝZE SÍŤOVÉ KOMUNIKACE URČENÉ PRO DETEKCI CÍLENÝCH ÚTOKŮ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **AUTOR PRÁCE**	**Bc. ONDŘEJ SEDLO**
SUPERVISOR **VEDOUCÍ PRÁCE**	**Ing. IVAN HOMOLIAK, Ph.D.**

BRNO 2020

Brno University of Technology
Faculty of Information Technology

Department of Intelligent Systems (DITS) Academic year 2019/2020

Master's Thesis Specification

22643

Student: **Sedlo Ondřej, Bc.**

Programme: Information Technology Field of study: Information Technology Security

Title: **Improvement of Adversarial Classification in Behavioral Analysis of Network Traffic Intended for Targeted Attack Detection**

Category: Security

Assignment:

1. Study the principles of network intrusion detection; focus on behavioral analysis and classification of network flows and utilization of ASNM features.
2. Study the adversarial classification that exploits the shortcomings of network-based behavioral analysis for the evasion of classifiers.
3. Get familiar with the supplied NPBO framework that performs the exploitation of several network vulnerabilities using network-level obfuscations, and test the tool on a few attacks.
4. Extend the NPBO framework to support the exploitation of contemporary network vulnerabilities. Collect a novel dataset of network traces and corresponding ASNM features.
5. Perform the extensive evaluation of the collected dataset, where explicitly focus on evasion and improvements of at least 3 supervised classifiers.
6. Perform cross-dataset evaluation with classifiers trained on older ASNM datasets and validated on the current one (and vice versa).

Recommended literature:

- Homoliak, I., Hanáček, P.: ASNM Datasets: A Collection of Network Traffic Features for Testing of Adversarial Classifiers and Network Intrusion Detectors, arXiv preprint *arXiv:*1910.10528 (2019)
- Homoliak, I., Barabas, M., Chmelař, P., Drozd, M., Hanáček, P.: ASNM: Advanced Security Network Metrics for Attack Vector Description, 2013
- Homoliak, Ivan, et al. "Improving Network Intrusion Detection Classifiers by Non-payload-Based Exploit-Independent Obfuscations: An Adversarial Approach." *arXiv preprint arXiv:1805.02684* (2018).

Requirements for the semestral defence:

- Items 1 to 3.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Homoliak Ivan, Ing., Ph.D.**
Head of Department:	Hanáček Petr, doc. Dr. Ing.
Beginning of work:	November 1, 2019
Submission deadline:	June 3, 2020
Approval date:	October 31, 2019

Abstract

In this work, we study ways to improve the performance of network intrusion detectors. In detail, we focus on behavioral analysis, which uses data extracted from individual network connections. Such data is used by the described framework for obfuscation of targeted network attacks that exploit a set of contemporary vulnerable services. We select vulnerable services by scraping the National Vulnerability Database of NIST while limiting the search for years 2018 and 2019. As a result, we create a novel dataset that consists of direct and obfuscated attacks executed on selected vulnerable services as well as their legitimate traffic counterparts. We evaluate the dataset using a few classification techniques, and we demonstrate the importance of training these classifiers using obfuscated attacks in order to prevent evasion of the classifiers (i.e., false negatives). Finally, we perform the cross dataset evaluation using the state-of-the-art ASNM-NPBO dataset and our dataset. The results indicate the importance of retraining the classifiers with the novel vulnerabilities while still preserving a high detection performance of attacks on older vulnerabilities.

Abstrakt

V této práci se zabýváme vylepšením systémů pro odhalení síťových průniků. Konkrétně se zaměřujeme na behaviorální analýzu, která využívá data extrahovaná z jednotlivých síťových spojení. Tyto informace využívá popsaný framework k obfuskaci cílených síťových útoků, které zneužívají zranitelností v sadě soudobých zranitelných služeb. Z Národní databáze zranitelností od NIST vybíráme zranitelné služby, přičemž se omezujeme jen na roky 2018 a 2019. Ve výsledku vytváříme nový dataset, který sestává z přímých a obfuskovaných útoků, provedených proti vybraným zranitelným službám, a také z jejich protějšků ve formě legitimního provozu. Nový dataset vyhodnocujeme za použití několika klasifikačních technik, a demonstrujeme, jak důležité je trénovat tyto klasifikátory na obfuskovaných útocích, aby se zabránilo jejich průniku bez povšimnutí. Nakonec provádíme křížové vyhodnocení datasetů pomocí nejmodernějšího datasetu ASNM-NPBO a našeho datasetu. Výsledky ukazují důležitost opětovného trénování klasifikátorů na nových zranitelnostech při zachování dobrých schopností detekovat útoky na staré zranitelnosti.

Keywords

IDS, adversarial classification, behavioral network traffic analysis, classification intrusion detection system, NPBO, ASNM

Klíčová slova

IDS, adversariální klasifikace, síťová analýza na základě chování, klasifikační systémy pro odhalení průniku, NPBO, ASNM

Reference

SEDLO, Ondřej. *Improvement of Adversarial Classification in Behavioral Analysis of Network Traffic Intended for Targeted Attack Detection.* Brno, 2020. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Ivan Homoliak, Ph.D.

Rozšířený abstrakt

Síťově orientované systémy pro odhalení průniku, které jsou založené strojovém učení, jsou schopny pomocí behaviorální analýzy síťové komunikace detekovat i pro ně neznámé útoky, a to i bez nahlížení na vnitřní data jednotlivých paketů. Problémem je, že pomocí obfuskačních metod lze tyto klasifikátory oklamat, a tudíž je možné pro obfuskovaný útok přes takové systémy proniknout bez povšimnutí.

V této práci se řeší, jak vylepšit schopnost klasifikátorů detekovat adversariální útoky založené na obfuskacích a zjistit, jak se od sebe navzájem liší jednotlivé klasifikátory. Dále je cílem zjistit, jak dobře dokážou detekci obejít různé obfuskační techniky a jak odolné jsou klasifikátory vůči neznámým obfuskačním technikám. Ke konci je práce se zaměřena na to, jaký vliv má použití různých datasetů pro trénování a testování daných klasifikátorů.

V rámci práce byly v Národní databázi zranitelností Národního institutu standardů a technologie, který spadá pod Ministerstvo obchodu Spojených států amerických, dohledány zranitelnosti z roku 2018 a 2019. Následně byly v databázi od společnosti Offensive Security nalezeny exploity, které daných zranitelností zneužívají k průniku do cílového systému. Ke každé zranitelné službě byl vytvořen virtuální stroj, na který byla služba nainstalovaná. K virtuálním strojům byl připojen útočný počítač, který používal nástroj Metasploit, pomocí kterého se stažené exploity použily k útokům na každý virtuální stroj. Celá komunikace všech útoků byla nahrána pomocí nástroje tcpdump. Dále byl na útočný stroj přidán mně dodaný Non-Payload-based (NPBO) framework, který sloužil jako nástroj pro obfuskaci útoků provedených staženými exploity.

K obfuskaci byly použity techniky, které měnily různé behaviorální vlastnosti útoků pomocí změn, které nezasahovaly do vnitřních dat paketů. Obfuskační techniky prováděly např. simulaci nespolehlivého síťového kanálu pomocí umělého poškozování určitého procenta paketů, přidávání zpoždění přenášených paketů, změnu pořadí paketů, různé kombinace jmenovaných technik atd. Pomocí NPBO frameworku se automaticky nebo poloautomaticky útočilo obfuskovanými útoky na všechny zranitelné služby běžící na virtuálních strojích. NPBO framework veškerou komunikaci zaznamenal pomocí automatického spouštění nástroje tcpdump. Dále byly vytvořeny záznamy komunikace legitimního provozu zranitelných služeb (opět pomocí nástroje tcpdump), což obsahovalo připojení se k dané službě a tam případnou autentizaci na nějakého uživatele a různé změny nastavení na dané službě, vytváření nových uživatelských účtů, vytváření webových stránek, nahrávání a stahování souborů apod. Zkrátka se vykonávaly běžné úkony, které by mohly být obvyklé v případě každé služby. Nový dataset se tedy skládá ze záznamů legitimního provozu každé služby, záznamů přímých útoků na danou službu a záznamů obfuskovaných útoků na ni. Slabinou tohoto datasetu je fakt, že veškeré útoky a simulace legitimního provozu byly prováděny v laboratorních podmínkách, a to dokonce mimo jakéhokoli jiného provozu, tudíž nashromážděná data dokonale neodpovídají reálnému provozu v praxi.

Nashromážděná data byla potom předána dalšímu mně dodanému nástroji *recurs-walker*, který provedl extrakci jednotlivých TCP spojení ze záznamů komunikace a přidal je do nové PostgreSQL databáze. Následně byl spuštěn další mně dodaný nástroj *metrics-extractor*, který analyzoval TCP spojení v databázi a z nich extrahoval Advanced Security Network Metrics (ASNM) rysy, např. průměr velikosti zdrojových paketů (statistický rys), počet přenesených bajtů za sekundu (dynamický rys), zdrojová IP adresa (lokalizační rys), počet paketů za sekundu distribuovaných do 10 intervalů (distribuovaný rys), aproximace délek příchozích paketů polynomem 5. řádu (behaviorální rys). V průběhu extrakce rysů se provedla i anonymizace koncových bodů ve spojeních pomocí změn IP adres. Dále byly implementovány procesy v nástroji RapidMiner, které provedly další zpracování TCP spo-

jení a přípravu dat pro jednotlivé klasifikátory, např. odstranění lokalizačních rysů, protože v laboratorních podmínkách pro klasifikaci dat, která neobsahují žádná jiná spojení než je útok, nebo jen legitimní provoz dané aplikace jedním uživatelem, nemá daný rys význam. V dalším procesu jsou potom z připravených dat vybrány rysy pomocí algoritmu Forward Feature Selection za použití klasifikátoru Naïve Bayes s Kernel Density Estimation nejprve z dat obsahujích přímé útoky (DL data) a potom z dat, které obsahují i obfuskované útoky (DOL data). Následně je provedena křížová validace na datech DL a DOL, srovnány rozdíly mezi klasifikátory za použití vybraných rysů z těchto dat a otestována odolnost klasifikátorů se znalostmi o obfuskovaných útocích vůči pro ně neznámým obfuskacím.

Vyhodnocení nového datasetu bylo zaměřeno na šest klasifikátorů. Nejdříve byly klasifikátory natrénované jen na legitimním provozu a přímých útocích. V křížové validaci bylo správně detekováno od 97.63% do 100.00% přímých útoků. Následně byly testovány schopnosti těchto klasifikátorů detekovat obfuskované útoky a experiment ukázal, že mnoho útoků se detekci vyhnulo. Výsledky ukázaly, že klasifikace obfuskovaných útoků dopadla o 0.35% až 83.55% hůř, než křížová validace nad přímými útoky a legitimním provozem. Oproti tomu v experimentu křížové validace na datech, která obsahují i obfuskované útoky, klasifikace přímých útoků v kombinaci s obfuskovanými útoky dosáhla zlepšení o 0.04% až 81.05% v závislosti na typu klasifikátoru. Trénování klasifikátorů na obfuskovaných útocích se tedy ukázalo jako velmi důležité.

Nad novým datasetem a datasetem ASNM-NPBO-v1 [39] bylo provedeno křížové vyhodnocení pomocí čtyř klasifikátorů. Úspěšnost detekce útoků klasifikátorů trénovaných na novém datasetu a testovaných na datasetu ASNM-NPBO-v1 dosahuje od 54.37% do 86.41% a klasifikátory trénované a testované naopak dosáhly 9.2% až 37.57%. Při hlubší analýze výsledků byly nalezeny značné rozdíly ve výsledcích různých zranitelných služeb. Klasifikátory trénované na novém datasetu detekovaly průměrně 96.76% útoků na Apache a jen 30.43% útoků na MSSQL. Podobné dva extrémní případy byly objeveny i v opačném případě, kdy byly klasifikátory trénované na datasetu ASNM-NPBO-v1, kde detekovaly průměrně 70.38% útoků na Gitstack a 0% útoků na FTPShell. Bylo ověřeno, že klasifikátory jsou schopny úspěšně detekovat velké procento pro ně absolutně neznámých útoků na některé zranitelnosti, ale zároveň existují i útoky zaměřené proti jiným zranitelnostem, které jsou pro ně nedetekovatelné.

Improvement of Adversarial Classification in Behavioral Analysis of Network Traffic Intended for Targeted Attack Detection

Declaration

I hereby declare that this term project was prepared as an original work by the author under the supervision of Ing. Ivan Homoliak, Ph.D. and I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

. . . .
Ondřej Sedlo
June 3, 2020

Acknowledgements

My thanks go to my supervisor Ing. Ivan Homoliak, Ph.D. for the useful comments, remarks and engagement through the learning process of this term project. I would like to thank also Mgr. Kamil Malinka, Ph.D., who was available as a consultant.

Contents

Chapter 1

Introduction

In the list of top 10 cybersecurity threats[1] are attacks, such as Ryuk, Maze, Nemty ransomware, campaigns like Operation ZeroCleare, etc. Mentioned attacks targeted organizations, which are capable of paying the large ransom demanded, governments, industrial sectors, telecommunications providers, or regular people to steal sensitive data, encrypt data and require money for decryption, cause damage to machines, etc. Another problem is unpatched software for example from Adobe, Microsoft, or Oracle[2], which is being targeted mostly. Therefore, there is a necessity for defense against attacks like them, which might be acquired with the use of intrusion detection systems.

Knowledge-based (a.k.a. misuse-based) intrusion detection systems have difficulties detecting zero-day attacks and they are also vulnerable to attacks that were modified using polymorphism. The vulnerability originates from the fact there are no signatures for novel attacks and polymorphic modification of known attacks might prevent the positive signature match as well. Hence there is a requirement for new ways of network intrusion detection, which would eliminate mentioned defects. A possible solution to the problem is anomaly detection systems. Anomaly-based intrusion detection systems build profiles of users, which represent their normal behavior. Anomalies are recognized as deviations from users' profiles. There is a drawback of an anomaly-based approach though because these systems tend to have a high false-positive rate, unlike a knowledge-based approach. Another interesting approach is classification-based detection, which combines the advantages of both techniques. The principle of a classification-based detection system resides in constructing its model from malicious traffic as well as benign traffic. To classify an input the detector just compares the input to both models and assigns the more similar class to it. Anomaly detection systems and classification-based systems are capable of new attack detection, but they have problems detecting attacks based on obfuscation techniques [42].

Due to data encryption and also for efficiency reasons this thesis is concerned only with classification-based network intrusion detection systems, which are not performing deep packet inspection that analyzes the packet payload. On the contrary, these systems inspect only headers of packets that are attributed to particular TCP connections. We also assume that adversaries know all details about principles of the classification-based system, because the system should be secure even if the adversary knows everything about it.[3] The adversary is capable only to modify the input of the system, but he has to adhere to protocols of the TCP/IP stack specification. Hence there are several things he

[1] https://www.mcafee.com/enterprise/en-us/threat-center.html

[2] https://www.us-cert.gov/ncas/alerts/TA15-119A

[3] The assumption is derived from the Kerckhoffs's principle.

might execute, for example: exploit code modification, adding padding at the application layer of exploit code or he may manipulate network or transport layer protocols. When an adversary needs to attack a big count of targets and use many different exploits it is very impractical for him to manually modify exploit codes or add padding to each of them separately. Therefore he might find it useful to implement non-payload-based obfuscation techniques, which would transform known intrusions in an exploit-independent way. Non-payload-based obfuscation methods' goal is to camouflage intrusions to make them look similar to legitimate traffic. The Non-Payload-Based Obfuscation framework described in this thesis follows this idea and achieves exploit-independent obfuscation by modifying given exploits at network and transport layers of TCP/IP stack. Considering a classification based on generalized architecture of intrusion detection system (see Section 3.2), this approach belongs to the measurement phase-based attacks. In the case of the categorization by Barreno et. al [21] (see Section 4.0.1), our framework's method belongs to the exploratory type of attacks as far as influence is concerned, integrity attack in case of security violation, and it belongs to the category of indiscriminate attacks in case of specificity [42].

Contributions

In this thesis, a novel dataset was created. The dataset consists of ASNM features extracted from network traces of legitimate traffic, direct attacks, and obfuscated attacks. Attacks in the dataset are executed on contemporary vulnerabilities present in 11 services. Vulnerabilities that we selected were disclosed to the National Vulnerability Database of NIST in 2018 and 2019.

Then the evaluation of the novel dataset was performed using 6 classifiers. First, the classifiers were trained on legitimate traffic and direct attacks only. In cross-validation, the true-positive rate (TPR) of direct attack detection achieved values from 97.63% to 100.00% (with a low false-positive rate). Next, the detection capability of obfuscated attacks was tested using these classifiers, and the experiment proved that many attacks evaded the detection, and thus caused false-negative predictions. The results of the prediction of (unknown) obfuscated attacks showed that deterioration of TPR in contrast to the initial cross-validation experiment ranges from 0.35% to 83.55%. Further, when we included obfuscated attacks into the training process of the classifier, the ability to detect direct and obfuscated attacks was raised by range from 0.04% to 81.05%, depending on a particular classifier. Therefore, including some obfuscated attacks into the training process of classification-based models showed to be very important.

Cross-dataset evaluation using four classifiers was performed with the novel dataset and ASNM-NPBO-v1 dataset [39]. TPR of classifiers trained on novel dataset tested on the ASNM-NPBO-v1 dataset ranges from 54.37% to 86.41%, and TPR TPR of classifiers trained and tested vice versa achieves values ranging from 9.2% to 37.57%. In the detailed analysis of the results, significant differences between various vulnerabilities were found. Classifiers trained using novel dataset and validated on the ASNM-NPBO-v1 dataset detected 96.76% of Apache attacks and only 30.43% of MSSQL attacks. In the opposite situation (i.e., training on ASNM-NPBO-v1 dataset and validation using our dataset), two extreme cases can be found in the results – detecting on average 70.38% of attacks on Gitstack and 0% FTPShell attacks. Therefore, classifiers can successfully detect a high percentage of completely unknown attacks targeted to some vulnerabilities, but also some of the unknown attacks targeted to different vulnerabilities are undetectable to them.

Organization

The structure of this document consists of these chapters: Chapter 2 describes the taxonomy of network intrusion detection systems, Chapter 3 describes adversarial attacks divided by the intrusion detection system's phases, in Chapter 4 there are attacks which are designed against classification intrusion detection systems, and in chapter 5 there is described the non-payload-based obfuscation framework. The novel dataset is described in Chapter 6, in Chapter 7 is noted data preparation process, forward feature selection, and experiments with models without knowledge about obfuscated attacks. In Chapter 8 are listed experiments with obfuscated attacks and cross-dataset evaluation.

Chapter 2

Taxonomy of Network Intrusion Detection Systems

Network intrusion detection systems could be based on one of these three fundamental principles: anomaly detection, misuse detection and there are also classification-based intrusion detection systems that combine both latter approaches.

Anomaly detection works with a normal behavior model and it detects abnormal deviations from the model, which could be later identified as attacks. The fact knowledge of intrusions is not required implicates it is possible to detect new unknown attack techniques and we do not need to update the system with new attack information.

Misuse detection needs intrusions specifications for its operation. This approach aims to detect concrete attack patterns and it searches for weak spots in the monitored system. Audit data streams are checked for the intrusion patterns and attack signatures, and if a successful match occurs an alarm is generated. Thanks to precise specifications this approach is easier to understand and thus implement for developers and analysts The main disadvantage of this method compared to anomaly detection is the fact it faces problems detecting novel attacks, because of its requirements for specification [45].

Classification-based intrusion detection systems combine the advantages of both mentioned approaches. The principle is based on modeling of legitimate behavior, but also on the modeling of intrusions. The classification of inspected data is based on similarities with those models.

2.1 Signature detection principles

Most of the information in this section is based on an article called „Intrusion Detection Systems: A Survey and Taxonomy“ by Stefan Axelsson [20]. Signature detection principles use quit different approach than anomaly detection ones. The difference is that the core of the detector is built upon a knowledge-base, where are defined all patterns of what signals legal or malicious behavior. Observation data are being compared to intrusion knowledge and then the decision is made.

Normal behavior is not modeled in this approach, thus it is not critical to the detector what the observed systems look like because it is not taken into consideration while making a decision if the intrusion occurred. These systems have acceptable detection and false alarm rate. The taxonomy of signature detection principles is illustrated in Figure 2.1.

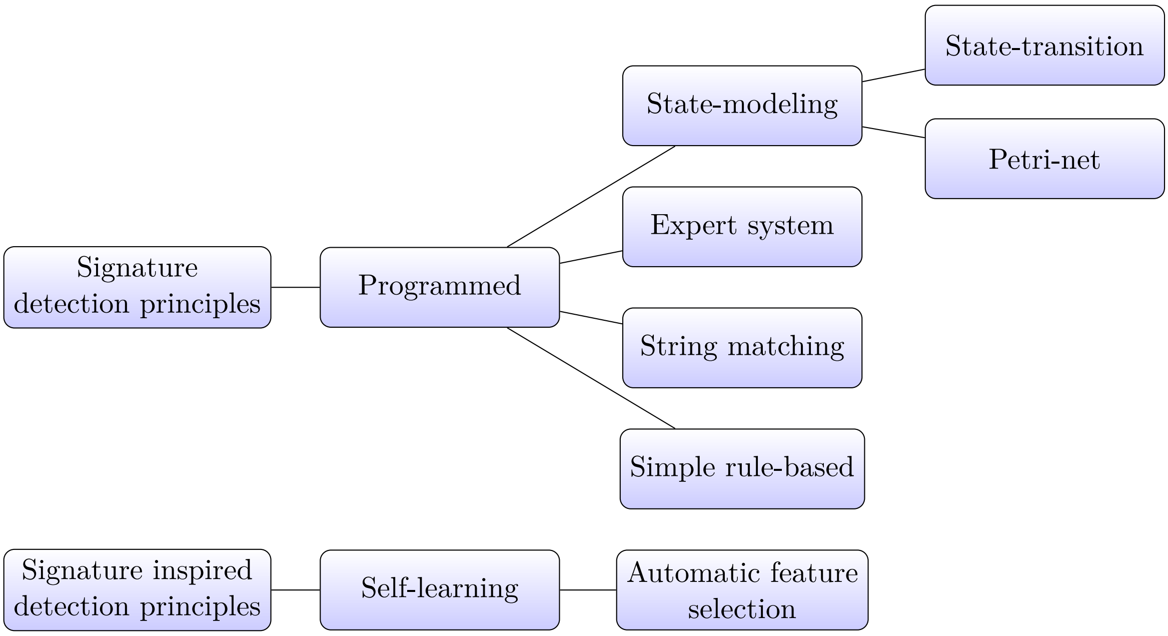

Figure 2.1: Classification of Signature detection and inspired principles by Axelsson [20].

2.1.1 Programmed

These systems are programmed with an explicit decision rule. In the decision rule, there is precisely programmed what is expected to happen from a specific intrusion. The principle is to implement attack traces that are checked, and the decision is made based on it. This approach is common in the law field, the detector is based on illegal behavior listing.

State-modeling

The intrusion is encoded as a number of different states in the observation space. These systems are using time series models. This approach can be split into two subclasses:

State-transition, where the intrusion forms a simple chain, where every part should take place in order. The second is Petri-net based. The Petri-net is a tree structure, which consists of states connected with transitions. The states might be fulfilled in parallel until the destination is reached, thus this approach is more general than the former one.

Expert system

An expert creates the rules, which are later applied to audit data. The core of the detection system is a set of rules. Typically it is a forward-chaining production-based tool. They are flexible and use powerful operations such as unification but at the cost of execution speed.

String matching

A very simple method, which uses extremely efficient algorithms searching for substrings. It is easy to understand for its developers and users. Usually, the system is case sensitive.

Simple rule-based

The system makes decisions using a base with a set of rules. It is similar to expert systems, but it is not as sophisticated as them. However, as simpler systems, they have higher performance.

2.2 Anomaly-based detection principles

Information in this section is based on „Network Anomaly Detection System: The State of Art of Network Behaviour Analysis“ article by Shu Yun Lim et al. [45] and an article called „Intrusion Detection Systems: A Survey and Taxonomy“ by Stefan Axelsson [20].

Three components participate in generic anomaly detection system:

- The Sensor subsystem monitors input traffic which is used for later anomaly detection.
- The Modeling subsystem is responsible for normal behavior model generation.
- The Detection system looks for events with suspicious characteristics in real-time and flags anomalous activities.

The network anomaly detection system operates in two modes: model construction and detection. To make the system working correctly it is necessary to train the anomaly detection sensor. It monitors network traffic events for a period of time, for example, a few days or weeks. While observing the network traffic it gradually builds a picture of all hosts. After the time period expires the system generates a measure for the data using a profiling method. Now it has a baseline of the system's normal behavior, which consists of extracted data characteristics for example the state of the network's traffic load, protocol, and typical packet size. The behavior model then serves as a pattern of correct network traffic characteristics in the detection phase. If any abnormal network activity occurs in the model construction phase the system has the wrong normal behavior model and does not detect attacks that are based on that anomaly. The complete taxonomy scheme by Lim is depicted in Figure 2.2 and anomaly detection from Axelsson's point of view is portrayed in Figure 2.3.

2.2.1 Learnt Model by Lim

There are two phases of operation in this approach [45]: the learning phase and the anomaly detection phase. In the former phase, the detection system creates a profile based on the normal behavior of the specific network or host using machine learning techniques. It is necessary to train on the system, where it will be later used for anomaly detection because every network has its special characteristics. There are three approaches based on this anomaly detection model: Rule-based, Model-based and Statistical-based.

Rules-based

The normal behavior of the monitored network or host is represented by a set of rules. Those rules are based on comparing a high-level state of the system change patterns that were derived from the audit data, with penetration state change scenarios.

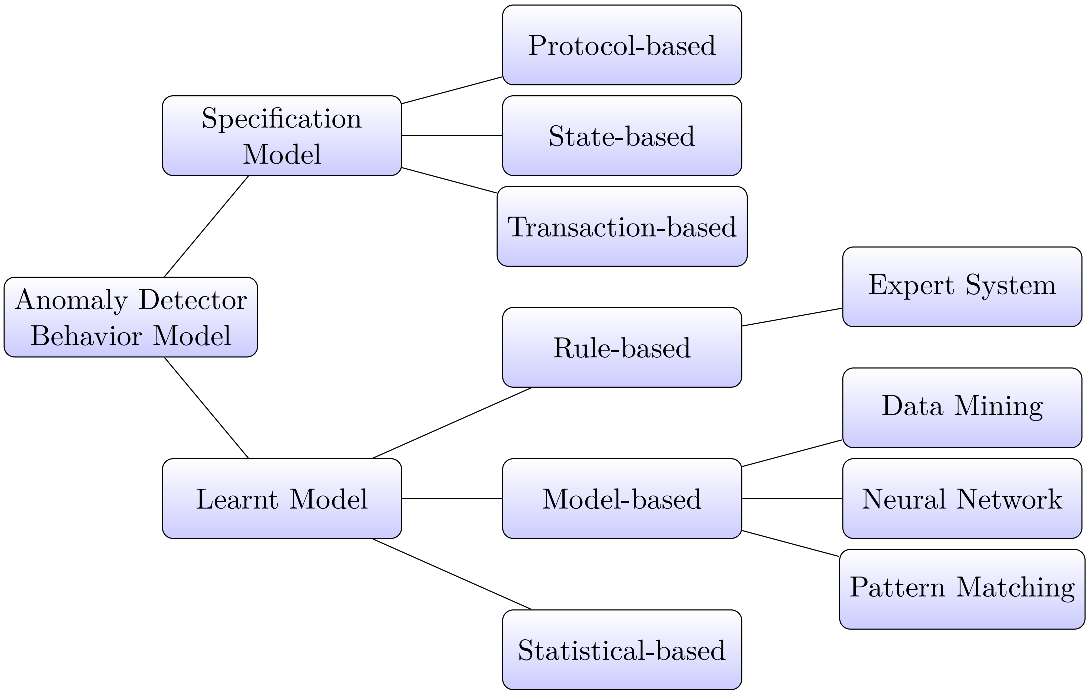

Figure 2.2: Network Anomaly Analysis Taxonomy by Lim [45].

Expert System

Expert System extends rule-based systems to a more complex approach. The core of the system is based on knowledge-base, which has two parts. The first part is a fact-based represented by a set of assertions which are applied to input audit data, and the second one is a rule-base which contains a set of rules which describe known intrusion techniques' scenarios in the system. When a match of an assertion from the fact-base and some rule's predecessor pattern is found, the system creates a rule-fact binding. If all the patterns of one specific rule have been bound with some facts, then the binding analysis process is triggered. The binding analysis assures the consistency of all the associated variables in the rule with their binding.

Model-based

The difference of model-based intrusion detection technique compared to Rules-based is it works on a higher level of abstraction instead of binding audit records to expert rules. Anomaly detection is based on a model of the normal behavior of the current monitored network or host. This approach might process input data faster because it uses just audit data which are relevant to the more general behavior model. Anomalies are detected by comparing actual input information to the normal behavior model, unusual deviations from the model are considered suspicious. Thanks to the higher abstraction of the model there are more intuitive explanations of intrusions which allow us to predict intruder's next action. In the Lim's taxonomy of network anomaly analysis there are 3 examples of model-based approaches: Data Mining, Neural Networks and Pattern matching.

Data Mining

Data Mining method at first extracts relevant features from audit data for individual network connections and host sessions. Then a data mining program is applied to those features. The result is models describing the behavior of intrusions and normal activities. The system adaptively builds models from a large amount of data, thus its models are up-to-date and the whole process is efficient.

There are data mining-based frameworks principles that can be split into these three parts. Meta classification, which allows it to learn correlations between intrusions, association rules creation for link analysis, and frequent incident matters of sequence analysis. The whole process is based on mining audition data, and then using extracted patterns to train classifiers, which are able to detect intrusions using its knowledge.

Neural Networks

Neural Networks are very efficient at learning system-call sequences, one of the significant reasons for it is probably the fact neural networks work with high abstraction level input data extracted from audit data. When trained on a representative command sequences of a user, the net derives the generalized profile of the user's normal behavior.

Neural Network Intrusion Detector (NNID) is a useful approach for off-line monitoring user profiles. It is a backpropagation neural network which is run at the end of each day by an administrator to learn from what users did at their workstations to build their profiles. The NNID uses each user profile built on the user's past behaviors to recognize him from the current day. If the user has behaved differently in a suspicious way the investigation of the incident starts.

Neural Networks approach is a very perspective area of network anomaly detection especially for individual user anomaly detection because they have as high performance that it is possible to use them for real-time detection.

Pattern Matching

Pattern Matching builds normal traffic profiles based on symptom-specific feature vectors, for instance, link utilization, packet loss, and the number of collisions. The learning process is performed online. Traffic profiles building is very sensitive to a monitored network, thus it is not possible to use them in a different network environment. The tolerance limits need to be set after traffic profiles are finished because the thresholds are derived from normal traffic behavior. If new input features exceed the thresholds of set tolerance an anomaly is recognized.

Statistical-based

The anomaly detector for all subjects generates profiles representing their normal behavior base. These profiles are stored in very little memory consuming way, and they are required to be updated fast and efficiently. The reason for it is that profiles might be updated with each audit record. The system periodically generates quantitative measures of its stored normal profiles.

Anomalies are being recognized using multiple statistical methods such as the integral of absolute difference of two functions over a time interval, which were calculated from profiles data. The difference must not raise above the tolerance threshold, otherwise, it would be considered malicious. Another method could be based on for example multiple of the standard deviation on any side of the mean. There are even more statistics-based methods, for instance, Bayesian statistics, covariance matrices, and Chi-square statistics.

A serious disadvantage connected with statistical-based approaches is the fact statistics are not sensitive to the order of events in an intrusion, thus there is considerable information loss. It could be fixed using an enhancement which checks intrusion event sequences. There's also a problem with setting thresholds right, because if they are too tolerant it leads to false negatives and otherwise false positives, thus it is critical to the system how precise thresholds are.

2.2.2 Specification Model by Lim

The specification model is not based on mathematics as much as previous approaches, but it is based on human expert knowledge. The model is built upon a logic-based description of expected behavior. Different system element monitoring is combined in this method, monitored elements range from application to network traffic.

Protocol-based

Protocol-based intrusion detection does not use any statistical-based model, instead, it is based on the exact specification of the current TCP/IP protocol. The idea of designing a detector by protocol's specification has a significant impact on normal model construction accuracy compared to statistic model-based approaches because statistic models generalize their view of data and thus have very limited knowledge of monitored network protocols. For this approach detect anomalies much easier, because of very precise documentation of a protocol that specifies normal use states any deviation from described usage is considered suspicious. Hence the basic explanation of the fundamental principle of an anomaly filter may be described as simply searching for a protocol misuse.

The protocol is defined as a set of rules describing the interaction between communicating sides. The official definition of protocol theoretical rules is in the description document (for example RFC), they might also be derived from practical usage of the protocol.

Systems based on this approach do not require any signature database updates, because they are only based on the protocol description. Hence they are able to detect any attacks including novel ones.

State-based

This method is making use of the idea that all connection-based network protocols have states which conform to individual connection parts. Thus there is exactly defined what is expected at a certain time in the connection for both communicating sides. If something different happens and thus an unusual change of the state occurs, the anomaly is recognized. The state model is represented by a state machine.

Transaction-based

The fundamental idea of this approach is that we describe positive behavior cases. Expected behavior consists of a set of desired actions and a sequence of actions. Specific transactions that correspond to expected actions are defined. The set of transactions is an integral part of the security policy.

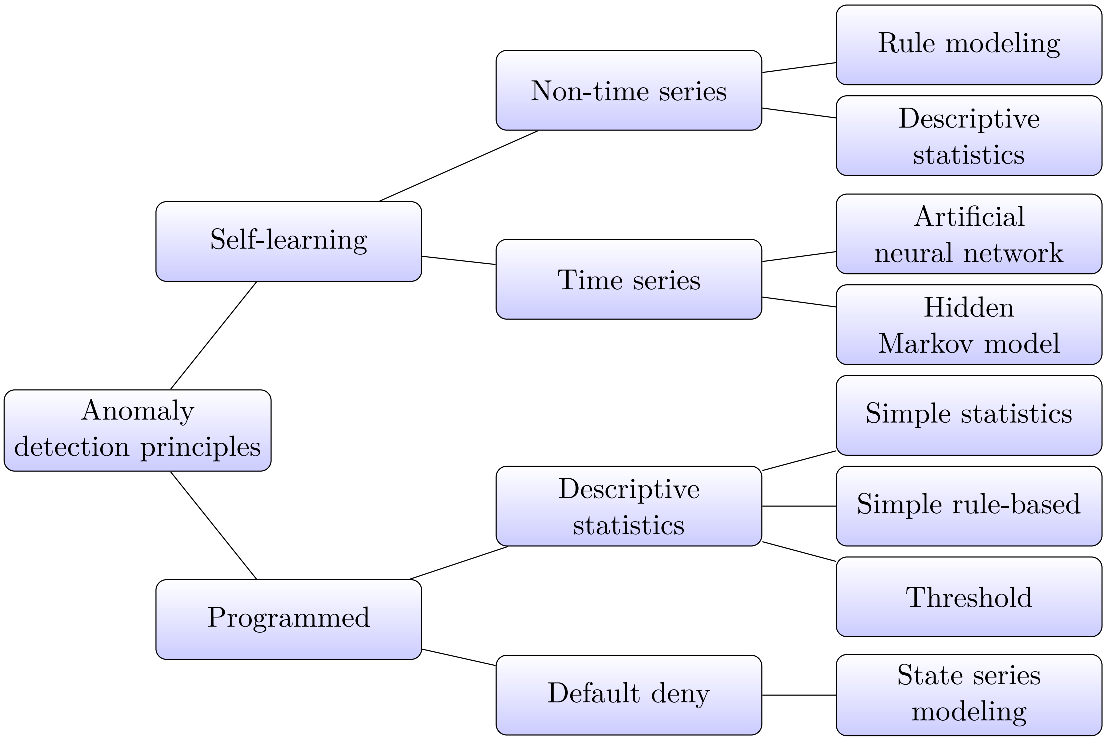

Figure 2.3: Classification of Anomaly detection principles by Axelsson [20].

This concept originates from the area of database management systems, but unlike there it is not necessary for distinct transactions to be executed. The detection system only monitors the host or network for potential conflicts.

2.2.3 Self-learning systems by Axelsson

These systems learn what is normal behavior of a monitored system. Usually, it takes some time of observing the network communication while constructing a model of the normal state.

Non-time series detectors

Non-time series detectors model normal behavior using stochastic models that do not take time-series behavior into account. There are two approaches to them: rule modeling and descriptive statistics.

Rule modeling uses information gained from the monitoring of the traffic to create a number of rules which have to be respected. If a poor match occurs the detector raises an alarm and the whole situation needs to be investigated.

The latter approach is based on descriptive statistics. It constructs a profile using collected simple, descriptive, mono-modal statistics and derives a distance-vector for the traffic and the profile. If the distance exceeds tolerated limit the alarm is raised.

Time series detectors

Time series detectors are generally more complex than non-time series detectors. There are several approaches based on this principle, for instance, a Hidden Markov Model and an Artificial Neural Network, which is described below.

An Artificial Neural Network (ANN) learns what normal behavior of the monitored communication is like from observed traffic data. Then ANN's output is compared with measured communication data, thus the result can be used in the intrusion detection decision. In a concrete system, the final decision might be done for example using a second stage, which would be implemented as an expert system that decides if the result of the comparison mentioned above signals an intrusion.

2.2.4 Programmed systems by Axelsson

Programmed systems have to be implemented by someone, thus all intrusions' principles are derived from his knowledge. What is considered abnormal decisions are based on the opinion of the user of the system.

Descriptive statistics

Descriptive statistics based systems build a model of the normal behavior of the monitored system with parameters. Usable parameters might be for instance the number of unsuccessful logins, network connections count, the number of commands with error returns, etc. This principle can be divided into three characteristic approaches:

Simple statistics based systems, which consist of higher-level components that use collected statistics for the final decision.

Simple rule-based systems that need rules provided for the user. These rules are later applied to the collected statistic data.

Threshold systems are the simplest version of descriptive statistics detectors. The user sets thresholds, which trigger alarms. Thresholds might be represented as simple ranges or simple conditions, for example, the number of unsuccessful login attempts > 3.

Default deny

Default deny system needs specific circumstances, in which the monitored system operates in a security-benign manner, to be set. All deviations from this operation, which are not explicitly permitted, are then labeled as intrusive.

State series modeling is a method, which based on the state machine theory. The policy is encoded as a set of states of the state machine and the transitions between them are implicit in the model. Only explicitly allowed actions do not cause the detector to raise the alarm. If any action, which was not set is done, then a transition between states occurs and thus the alarm is raised by the detection system. Alarm triggering actions might be for instance file accesses, the opening of ports that are considered secure, etc. The rule matching engine is simpler and not as powerful as a full expert system.

2.2.5 Other Anomaly intrusion detection principles by Debar

This section is based on the article „A revised taxonomy for intrusion-detection systems“ by Hervé Debar et al. [30]. These principle models normal behavior and the model is built upon information collected by monitoring of users or traffic communication. An intrusion is detected when the behavior deviates from the normal state in an unusual way. In other words, the detector compares current behavior with its normal behavior model and then makes a decision if an alarm should be raised. Anything anomalous is considered intrusive.

These systems have a useful advantage in their possibility to detect new intrusions. Thus it is also possible to use them for the semiautomatic discovery of novel attacks. They are able to detect „abuse of privilege“ attack types, which do not use any security vulnerability exploitation, as well.

On the other hand, the mentioned advantages are accomplished at the cost of a high false-alarm rate. Another problem is that when the observed system changes it is necessary to actualize the detector's normal behavior model. Hence retraining of the detection system has to be performed and while retraining the system is unable to detect any attacks and if any intrusion occurs during the training process it learns it as normal behavior.

Several approaches have been proposed for the behavior intrusion detection: User Intention Identification and Computer Immunology. Some approaches have been already described in Section 2.2: Statistical-based, Expert systems and Neural networks.

User Intention Identification by Debar

This approach models normal user behavior using the model which consists of a set of high-level tasks, which the user has to perform. High-level tasks are then transformed into actions, which are associated with collected audit data from the monitored system. If any action does not fit to the task pattern, the system raises an alarm.

Computer Immunology by Debar

This approach models the normal behavior of services instead of users. In the model, there are used short sequences of system calls, which are usual for the modeled service. Intrusions tend to use extraordinary system calls because they need to open specific files, which are not otherwise used very commonly. The systems get audit references from the reference table, which includes all the known allowed sequences of system calls. This technique work as an online monitoring detector.

There is a very low false-positive rate if the reference table is complex enough. The problem of this technique is it does not detect intrusions based on configuration errors, because such attacks use legitimate actions to gain unauthorized access.

2.3 Classification-based detection principles

The classification-based systems are in Axelsson's article called as signature inspired detection principles, which are portrayed in Figure 2.1.

These systems use both points of view on the problem, it models normal behavior of observed elements and the intrusive behavior of the intruder. However, they are called „signature inspired“, because the intrusion model is much stronger and more explicit than the model describing normal behavior. These systems are good at detecting very advanced intrusions because they have the intrusion knowledge and combine it with normal behavior knowledge gained from the normal model. These detectors are in some senses respected as the most advanced intrusion detection systems in this survey.

Self-learning. This approach's idea is to learn normal behavior and the same behavior infected with intrusions, thus the detector is going to recognize malicious traffic specifically for the monitored network or host. The learning process requires examples with normal behavior and prepared attacks, which have to be labeled as intrusive. These systems might also use methods, such as Automatic Feature Selection, where the detector learns automatically what features are important for intrusion recognition.

2.3.1 Machine learning

In this section are described some widely used techniques approached in models, which are used by classifiers of intrusions detection systems [25].

Support Vector Machines (SVM)

SVMs are supervised learning models, where data is represented as points in the space, and its goal is to construct maximum-margin hyperplane and divide these points into classes. This algorithm is searching for maximally wide spaces between bordering points of different classes.

Artificial Neural Networks (ANN)

ANNs are networks consisting of mutually connected perceptrons, called neurons, which were inspired by biological neurons. These neurons' outputs connected to inputs of some other neurons and their behavior is defined by weights and an activation function. In the learning process weights are updated using the back-propagation algorithm.

Deep Neural Networks (DNN)

DNNs are ANN which include multiple layers that are connected to previous and succeeding layers, except for the input layer and output layer. Layers that have predecessors and successors are called hidden. The advantage of DNNs is that their input might be unlabeled and unstructured data because they are able to extract related features on their own.

Convolutional Neural Networks (CNN). CNNs consist of two main parts, which have different functionalities: feature learning part and classification part. The former part is made of convolutional or sub-sampling layers and its task is to create a feature map and extract important information from it. It sub-samples its input in order to reduce the dimensionality of each feature map. The latter part consisting of one or two layers is fully connected to the last layer of the previous part and classifies the data.

2.4 Honeypots

Honeypots have no functionality in the production system, except being bait for adversaries. There should not be any traffic communicating with the honeypot. If any communication directed to honeypot occurs or the honeypot itself starts sending packets to the network, then it is considered malicious. Honeypots can be split into two categories based on how much activity might an attacker do there [44]:

2.4.1 Low-Interactive

These systems are designed to emulate only some functionalities of the system that they appear as. Usually, these systems emulate some services, which allow the attacker just limited interaction, because the emulation is not full. For example, FTP service limited to login function and some basic commands. The advantage of these systems is low maintenance requirements and the fact it does not use the whole operating system. There is also not a high risk of an attacker, because he only controls the partial emulation of service, which is immediately reported to honeypot's logs. There is also a risk that the attacker discovers the fact he is interacting with only the honeypot if he uses some of the not implemented commands.

2.4.2 High-Interactive

High-Interactive honeypots are more sophisticated than Low-Interactive ones. They are run at real systems with real operating systems. Services used by these honeypots are completely installed on the system. There is a higher risk of using these systems because a potential attacker takes control over the whole operating system and might use it in order to intrude on other hosts in the network. The main advantage of these systems is that their owners might deeply investigate what is the attacker doing and deduce what is his aim. On the other hand, its drawback is the fact these systems require a full machine, which only baits potential attackers, which might be quite expensive.

Chapter 3

Taxonomy of Adversarial Attacks

This chapter is mostly based on the article „Adversarial Attacks against Intrusion Detection systems: Taxonomy, Solutions and Open Issues“ by Igino Corona et al. [28] and „Adversarial Attacks and Defences: A Survey“ by Anirban Chakraborty et al. [25]. And the chapter was written with a focus on network-based attacks because host-based attacks are beyond the scope of this thesis.

Intrusion detection systems with other tools constitute the computer security infrastructure, hence they might be vulnerable to attacks by the same intrusions they try to detect. Thus intrusion detection systems became targets of attacks, and if they are successfully exploited they might transform themselves into attacking tools serving an intruder.

3.1 Goals of attacks on IDSs

There are six main goals of attacks against intrusion detection systems [28]:

Evasion

It modifies the intrusion pattern to make the attack undetected. Hence, it causes that an intrusion attempt evinces the characteristics of benign network communication.

Overstimulation

In overstimulation attacks, the attacker tries to generate false positive alerts of IDS by creating a benign communication that evinces aspects of malicious intrusions. Many attack patterns are applied to overstimulate the detection system. These attacks are less popular than evasion attacks since their effect does not lead to compromise of the system but to exhaustion of the operator who analyzes the alerts.

Poisoning

The aim of this approach is to insert malicious patterns into the training set for the intrusion detection system and mislead the learning phase into a state, where the detector will be unable to detect prepared intrusions.

Denial Of Service (DoS)

Attacks of this type use patterns to disable or at least slow down the detector sensor. If the network traffic is not allowed to transfer into the internal network of the attacked system without proper inspection by the intrusion detection system, then slowing down the detector might cause network traffic transfer delay and packet drops.

Response Hijacking

This type of attack tries to fake alarm descriptions for response units to make them react inappropriately, for instance, to make them block some legitimate connections.

Reverse Engineering

The goal is to extract information about the intrusion detection system's internal implementation and use the gained information to design new attack patterns that take advantage of it.

3.2 Attack classification based on generalized architecture of Intrusion Detection Systems

There are many different architectures that constitute Intrusion Detection Systems. However, most of them are based on a relatively general architectural framework, which consists of these four components: event generator, event analyzers, response units, and event databases. The framework's operation can be divided into these three phases: Measurement, Classification and Response [28].

3.2.1 Measurement phase

Measurement is performed by event generators. A vector of measurements (aka features) is used to characterize and event pattern. The detector uses features to differentiate intrusions from legitimate actions.

There are four categories of attacks which target network measurements in this phase:

Set of Measures

The attacker can exploit limits in the discriminant capability of the chosen set of measurements in order to evade detection. Even if intruders are not capable to perform the previously mentioned attack, they might try to evade using novel or small variations of a known intrusion. Thus the intrusion detection system should be designed counting with the possibility that attacks can evolve.

Input Data

The attacker modifies the input data for example in a system call, which returns the list of running processes. This method is usual at the host level, but this problem might encounter in a network environment as well, for example, if the attacker has control over a router.

Event Reconstruction

These attacks are critical especially for network sensors and here are listed some examples:

- Tunneling - Using the type of traffic, that is not observable by the network sensor.
- Desynchronization - Evading the network sensor view, for example using TTL[1] to elude the network sensor by taking advantage of the network topology.
- Encoding Variations - Acquiring different semantic of data on each communicating side, which can be handled into intrusion.
- Segmentation - The network traffic is divided into segments different on the source side differently than on the destination side and taking advantage of the fact some OSs[2] have a different policy when dealing with duplicate or overlapping segments.

Integrity and Availability Attacks

There are more techniques in this category, but most of them are not listed, because this thesis focuses on network-based analysis. An example of availability attacks is overloading the network sensor with too much traffic, which it cannot inspect as fast as they come, so it starts dropping packets.

3.2.2 Classification phase

The classification process is implemented in event analyzers. Internal models are created and rules are applied for event pattern classification to decide if the pattern is legitimate or intrusive. If the pattern has been determined as intrusive, then an alarm is generated and the incident is presented human-readable form. This process is performed usually in real-time.

In modern classifiers, there is a trend to develop an event analyzer based on machine learning, which requires statistical representative patterns. There are three main problems associated with statistical representativity described bellow:

- Privacy - Collecting legitimate users' data may involve sensitive information and cause problem with privacy.
- Real-world intrusions - It is necessary to keep a set of known real-world intrusions and update it as quickly as possible.
- Ground Truth - It is critical to have training data for the system validated properly. Validation requires deep expertise and training data amount is huge. This problem is the reason why it is practically possible for intruders to use poisoning attacks.

Here are described some attack issues against event analyzers:

Difference between Alert Space and Intrusion Space

Let us define intrusion space (I) and alert space (A):
$I = set\ of\ all\ intrusive\ patterns,$

[1] Time To Live in https://tools.ietf.org/html/rfc791

[2] Operating Systems

$A = set\ of\ patterns\ causing\ an\ alarm\ raising$,
then
$M = I - A = set\ of\ missed\ alarms$,
$F = A - I = set\ of\ fake\ alarms$.
Sets M and F might be used for evasion or overstimulation attacks in techniques listed below:

- Contextual Information Exploitation, for example about observed hosts and services.
- Mimicry Attacks - this type of attack tries to mimic legitimate patterns.
- Cost-Sensitive Classification - classifying intrusions depending on damage cost.
- Classifier Ensembles - using multiple parallel classifiers, where each of them is designed to detect its individual intrusion class.
- Automatic Evaluation - evaluation of how vulnerable are different classifying algorithms against evasion and overstimulation intrusion methods.

Pattern Matching

Pattern Matching algorithms might be slowed down using a specially crafted pattern which causes for example worst-case complexity scenario of the algorithm, which results in DoS[3] attack.

Description of Intrusive Events

An attacker might create intrusion, which causes the detector to produce too general or wrong alert description.This could lead for example into triggering defense mechanisms, which would create wrong firewall rules blocking some legitimate users.

There are several defense techniques dealing with this problem. Classification confidence measuring added to every alert description. Automated Attack Inference - Anomaly detection systems might try to classify the unknown attack to most similar attack patterns automatically. Model of the Adversary - modeling intruder's goals and behavior.

Poisoning Attacks

There are several machine learning (ML) techniques, which are being used in intrusion detection systems: Support Vector Machines (SVM), Hidden Markov Models (HMM), N-grams, Decision Trees and Artificial Neural Networks.

However, ML-based algorithms might be vulnerable in their learning process. If an attacker successfully inserts his prepared intrusions into the set of training examples, the algorithm learns the wrong pattern and will not be able to detect the attack associated with the inserted intrusion.

There are defense methods against poisoning attacks, for example, Training Data Manipulation based methods:

Reject On Negative Impact, which creates data sets with including and excluding a test sample, and if the sample decreases the trained model success rate it is excluded as a poisoning attack.

[3] Denial of Service

Or the method, which works with multiple intrusion detectors, which all train on their own randomly selected training samples, thus it can statistically determine from variations of these detectors results which sample is poisoning.

3.2.3 Response phase

Response units' function is to react to the raised alarm in order to save the defended system. For example, a new firewall rule might be added to block the current attack. There are some issues related to the response phase described below:

Response Effectiveness

To make the intrusion detector as effective as possible it is useful to evaluate response effectiveness. The actions detector makes might have a good impact, but it may also cause damage to the system, for example, DoS attack against infrastructure using firewall blocks.

There are several techniques trying to solve problems associated with this issue:

Game Theory, which requires the definition of an attacker model, potential costs of each action it can perform against him, values of each protected element, etc.

Response Frameworks with their own infrastructures, which perform necessary actions to prevent intrusion.

Cost-sensitive models, which calculate with costs of intrusion defense actions and costs of potential damage dealt by intrusions.

There is also problematic with response time because for a successful intrusion prevention process it is necessary to be faster than the attack.

Response Feedback

Response Feedback mechanisms might be used to prevent for instance DoS attack mentioned in the previous paragraph. We might try to achieve it by simple checking the blocked traffic characteristics, because in the case of well-known intrusion it may be possible to estimate its consequences.

Response Evaluation

The idea of this issue is that we could simulate an attacked infrastructure and intrusions attacking it. From the simulation, we could potentially evaluate the costs of an intrusion and damage dealt.

Chapter 4

Taxonomy of Attacks against Classification-Based IDSs

This chapter is based on the article „*Adversarial attacks and defences: A survey*“ by A. Chakraborty [25]. There are three types of adversarial attacks, which are being designed against classification-based intrusion detection systems to explore it, evade detection, or poison the classifier. Hence there are three fundamental types of these attacks: Exploratory attacks (in Section 4.1), Evasion attacks (in Section 4.2) and Poisoning attacks (in Section 4.3). The attacks might be also categorized based on other properties, which is described in the following subsection:

4.0.1 Categorization by Barreno

The attacks might be categorized based on Influence, Security Violation, and Specificity. This subsection is based on the article: „The security of machine learning“ by M. Barreno et al. [21].

Influence

These categories discriminate against the capability of the attacker.

- Causative - the attacker influences training data for the classifier.
- Exploratory - the attacker cannot influence training data for the classifier, but he sends new instances to the classifier and examines its decisions.

Security Violation

These categories depend on the harm the attacker might cause.

- Integrity - the attacker's intrusive data is able to evade the classifier and go through as false negatives.
- Availability - the attack leads into Denial of Service, mostly it is caused using false positives.

Specificity

This categorization distinguishes how specific are targets of the attack.

- Targeted - the attack specializes against a concrete instance.
- Indiscriminate - the attack tries to manipulate the classification of a wide distribution of instances.

4.1 Exploratory attacks

These attacks do not try to manipulate the training set, but they are designed to extract as much information about the classifier as possible. They are being used in the testing phase of the attacked intrusion detection system. Attacks of this type look the same as legitimate traffic and do not cause any harm to the system, thus they evade the detection and gain information about tested learner [25].

4.1.1 Model Inversion

In this approach, the aim is to perform model inversion in order to get information about its inputs using output data. The linear regression model f estimates the patient's drug dosage from his medical history information and genetic markers. Then the mentioned model f is used as a white-box and an example of data $(X = x_1, x_2, ..., x_n, y)$, it is possible to gain genetic marker x_1 from the inversion of model f. This method can be enhanced and also used for black-box models, for example, to recover images in case of face recognition [25].

4.1.2 Model extraction via APIs

This attack is focused against Machine learning APIs. The attacker has no information about the model or training data, but the target API returns him precision confidence values and class labels. Therefore, the attacker tries to solve it mathematically. Parameters or features can be calculated from equations with supplied confidence values. The attacker needs to perform $d + 1$ queries with d-dimensional inputs in order to calculate $d + 1$ parameters [25].

4.1.3 Member Inference Attack

The black-box target model is attacked in this method. The attackers send queries with his dataset to the target model. The target model returns him information in the form of vectors of probabilities, which specify recognized classes of queried data. Using the queried dataset as training dataset and output vectors from the target model the attacker builds shadow models. Then he constructs a training structure for the attack model. Shadow models have input from the training dataset and also a new testing dataset, then their outputs are labeled depending on input sets. The labeled dataset is used as training data for the attack model, thus the attack model is trained to categorize its input data whether are they from the training dataset or the testing dataset. Therefore, the attack model is able to estimate which data were in the training dataset of the black-box target model [25].

4.1.4 Information Inference

An attacker uses a meta-classifier to extract applicable information from a machine learning system. The attack requires information about training data, but no information about the target system internal, thus it attacks a black-box. An example of this attack is a public

API of the speech recognition system, which is based on Hidden Markov Models. For an attacker, it is possible to extract the information he should not be able to extract, for instance, the accent of the users [25].

4.2 Evasion attacks

Information in this section was adopted from the article „Adversarial Attacks and Defences: A Survey“ by Anirban Chakraborty et al. [25]. Evasion attacks try to evade the detector, in other words, to not be recognized as an attack and intrude the system.

4.2.1 Adversarial Examples Generation

Changing samples in order to damage the classifier to make it unable to detect the intrusion. This approach is divided into two categories based on which phase of the classifier implementation it is trying to manipulate.

Training Phase Modification

Training data might be modified in two manners:

Label Manipulation - The adversary is able to modify the training labels only. In the study [22] researchers randomly flipped 40% of the training data labels, which worsened the classifier enough for their adversary task.

Input Manipulation - The attacker is capable to modify also the input features. Thus he is able to influence the decision boundary of the classifier into his favor.

Testing Phase Generation

There are two types of this approach depending on the knowledge of the tested setup.

White-Box Attacks - The framework (from [62]) is searching for the perturbations which are added to input data samples for the attacked classifier. Final found perturbations should be able to manipulate the classifier to classify the modified sample differently. At first direction sensitivity estimation process is performed, then the perturbation is selected, which is then checked for its ability to be misclassified by the neural network and the feedback is sent back to the direction sensitivity estimation process.

Black-Box Attacks - An example of this approach is the technique called Jacobian based Data Augmentation. This technique's aim is to learn a substitute for the attacked model, which is later used to scheme new inputs for the black-box. The idea is that new inputs will be classified by the black-box the way the intruder wants [25, 61].

Transferability of Adversarial Samples

The idea of this principle is that samples generated by one model might affect the second model. It might be intra-technique, where are both models of the same type, for example, Neural Networks, or cross-technique, where are models of different types, for instance, Neural Network and Support Network Machine.

4.2.2 Generative Adversarial Attack (GAN)

There are two deep learning networks in the GAN procedure, which play different roles in the learning procedure. The first is a generative deep learning network, whose task is to generate samples that cannot be differentiated from the training set for the second network. The second one is the discriminative deep learning network, which has to determine if its input samples come from a generative network or from the training set. These two networks compete with each other and that leads to their accuracy advancement. The training finishes when the discriminative network makes a mistake.

4.2.3 GAN based attack in Collaborative Deep Learning

There are two collaborative neural networks, one of them is the victim training classification of its training set and the second is the attacker, which uses GAN for generating samples similar to the victim's training set. The GAN has access to honest outputs of the victim. The goal is to amass as much information about the victim's training set classes as possible.

4.2.4 Adversarial Classification attack based on Game Theory

The idea is based on game theory, where the classifier stands against the adversary. The adversary's goal is to modify the training samples to make them be classified as negative instead of positive. The classifier's goal is to classify even modified samples as intrusions. The classifier's game strategy is based on a cost-sensitive Bayes learner, which searches for the minimum cost of its action while expecting the adversary to use the best possible strategy.

4.2.5 Obfuscated Attacks

The obfuscation principle is changing adversary network traffic characteristics in order to appear as legitimate traffic. Hence the obfuscated attack cannot be detected by the classifier and is falsely classified as legitimate [41].

Tunneling

In the work [40] is discussed an idea of an intruder having a machine that is cooperating with him before he starts attacking the network hosts. In such a situation the tunneling method might be quite useful for the adversary. The adversary is using buffer overflow attacks which are tunneled through HTTP or HTTPS traffic in order to evade the IDS and Network Behavioral Analysis (NBA) system. The attack uses two modules: the cooperating machine called the Callback in the target network and the adversary's outer machine called the Fake HTTP Server, which waits for connection by the Callback. The attack is performed through the Callback, which is controlled by the Fake HTTP Server, while all communication between them, which is protocol independent, is camouflaged in the HTTP/HTTPS traffic. There was discovered that the classifier, which was trained on direct attacks and legitimate traffic only, was incapable to detect tunneled attacks and there was a very significant improvement when the classifier was trained on dataset extended of tunneled attacks.

Non-payload-based

These attacks are focused against legitimate behavior model. Data is not important in this attack, but it achieves obfuscation by modifying packets' headers and communication behavioral characteristics.

In [41] were performed experiments, where the obfuscation tool was used in order to improve the classification intrusion detection system. The obfuscation tool was based on Advanced Security Network Metrics. Experiments proved that training the classifier on a dataset including obfuscated attacks improved its capability to detect the same or similar obfuscated attacks.

4.3 Poisoning attacks

Poisoning attacks are used to contaminate the training data set and influence the network intrusion system. Here are listed some of the evasion and poisoning attacks. This section based on a survey in the article „Adversarial Attacks and Defences: A Survey“ by Anirban Chakraborty et al. [25].

4.3.1 Attack on Support Vector Machines

SVM's training and testing data are provided from the same distribution, but in adversarial learning, it is possible to exploit the system using data modification. For adversary, it may not be possible to get access to the SVM's training dataset, but he might find datasets with similar distributions.

4.3.2 Poisoning attacks on Collaborative Systems

There are poisoning attacks, which require a thorough knowledge of the learning system, that are able to generate data which significantly decreases the system's effectiveness.

Three types of these attacks have been announced: Availability Attack, where the attacker tries to raise the error of the collaborative filtering system as much as possible. Integrity Attack, where the adversary's goal is to maneuver the acceptance of a subset of items. And Hybrid Attack, which is the combination of both mentioned attacks.

4.3.3 Adversarial attacks on Anomaly Detection Systems

The aim of these attacks is to move the centroid of the normal behavior space to the distribution of prepared intrusion characteristics. The adversary's goal is to include its intrusion into the set of negatively classified items.

Chapter 5

Non-Payload-Based Obfuscation Framework

In this chapter, there are described Advanced Security Network Metrics (ASNM) features, and the Non-Payload-Based Obfuscation (NPBO) framework is specified. This chapter is based on two articles: „*Improving Network Intrusion Detection Classifiers by Non-payload-Based Exploit-Independent Obfuscations: An Adversarial Approach*“ by I. Homoliak et al. [42] and „*ASNM Datasets: A Collection of Network Traffic Features for Testing of Adversarial Classifiers and Network Intrusion Detectors*“ by I. Homoliak et al. [39].

The obfuscation framework was designed in order to create a non-payload-based obfuscation tool, which can modify the exploit a remote attack in such a way that the target classifier is not able to detect it as an intrusion. The Behavioral state diagram of the obfuscation tool is depicted in Figure 5.1. Then a new classifier is trained on the dataset, which includes obfuscated exploits' features and thus it is better at detecting other obfuscated attacks. The mentioned hypothesis has been proven to hold in [42]. The ASNM features are extracted from the observed network communication by the framework in order to describe network traffic characteristics.

5.1 ASNM

The original ASNM feature list was introduced in the Master's thesis [37] including 167 features, and it was formally described in [36]. Then the content of the ASNM features list was expanded to the number equal to 194 features in [38]. These features were split up into five categories based on their principle.

5.1.1 Statistical Features

In this category, there are features that express the statistical properties of TCP connections. Statistical operations are being used in this approach, such as count, mode, median, mean, standard deviation, there are also some other features like ratios of specific packet header fields or whole packets. In these features there are is also information about their incidence time, but there is no context available for them, unlike in the dynamic features category. There is also a dichotomy of some features based on if they were going inward or outward.

5.1.2 Dynamic Features

The role of features in this category is to present the dynamic properties of TCP connections. These features do not have to be necessarily real, but they also might be simulated. Some dynamic features respect the context of the inspected TCP connection. The main difference between dynamic and static features is that dynamic features reflect speed or error rate of the analyzed TCP connection, and also this category pays attention to how many acknowledgment packets were delivered, etc. Some of these features discriminate the direction of observed packets as well.

5.1.3 Localization Features

Features in this category represent information about communicating endpoints of the inspected TCP connection. They all share an aspect that they do not change in time, but stay static until the connection ends. Most of these features also respect directions of analyzed TCP connection flows. Another characteristic of these features is they do not deal with the context of the TCP connection.

5.1.4 Distributed Features

The most important trait of distributed features is the fact they are distributed into time intervals. These intervals lengths are constraint in logarithmical scale, for example, 1s, 2s, 4s, or 8s. Measured features distributed in a constant count of intervals might be for instance count or lengths of packets observed. Distributed features respect the context of the inspected TCP connection and packets' directions as well.

5.1.5 Behavioral Features

Behavioral features express properties related to the behavior of an analyzed TCP connection. For instance successful or prohibited connection closing, a number of new TCP connections since the beginning of a TCP connection. There are also some more complicate operations over captured data about inspected TCP connection, such as the polynomial approximation of packet lengths in a time domain or a packet index number domain, coefficients of Fourier series with respect to the direction of a TCP connection, etc. [39]

5.2 NPBO Framework Specification

The NPBO Framework description, which also includes all the following definitions in this section is from the article: „*Improving Network Intrusion Detection Classifiers by Non-payload-Based Exploit-Independent Obfuscations: An Adversarial Approach*" by I. Homoliak et al. [42].

The framework looks at internet communication as a session between two sides: the client and the server. Both participants of a session communicate using the application protocol of the TCP/IP stack, which intervenes in data transfer between them. The application data transfer TCP/IP stack is represented as *connection* k, which is bounded to connection-oriented protocol TCP at L4, Internet protocol IP at L3 and Ethernet protocol at L2. The connection k consists of start and end timestamps, ports of the client and the server, IP addresses of the client and the server, sets of packets by the *client* P_c, and by the *server* P_s.

5.2.1 Features Extraction

The features extraction process is defined as a function that maps a connection k into space of features F:

$$\begin{gathered} f(k) \mapsto F, \\ F = (F_1,\, F_2, \ldots, F_n), \end{gathered} \tag{5.1}$$

where n means the count of defined features. Every particular function f_i, which extracts feature i is defined as a mapping of a connection k into feature space F_i:

$$f_i(k) \mapsto F_i, \quad i \in \{1, \ldots, n\}, \tag{5.2}$$

and each element[1] of codomain F_i is defined as

$$\begin{gathered} e = (e_0, \ldots, e_n),\ n \in \mathbb{N}_0, \\ e_i \in \mathbb{N} \ \mid\ e_i \in \mathbb{R}\ \mid e_i \in \Gamma^+,\ i \in \{0, \ldots, n\}, \\ \Gamma = \{a - z, A - Z, 0 - 9\}, \end{gathered} \tag{5.3}$$

where Γ^+ denotes positive iteration of the set Γ.

5.2.2 Intrusion Detection Classification

Let us define V as the space of samples, where a sample means the vector of the network features, which were extracted from a specific connection. And let Y be the space of possible labels. Then let us define $X = V \times Y$ as the space of labeled samples. Let $D_{tr} = \{x_1, x_2, \ldots, x_n\}$ be a training dataset consisting of n labeled samples, where $x_i = (v_i \in V,\ y_i \in Y)$. The classifier C maps unlabeled sample $v \in V$ to a label $y \in Y$:

$$y = C(v), \tag{5.4}$$

and learning algorithm A maps the given dataset D to a classifier C:

$$C = A(D_{tr}). \tag{5.5}$$

The notation $y_{predict} = A(D_{tr}, v)$ stands for the label assigned to an unlabeled sample v by the classifier C, build by learning algorithm A on the dataset D_{tr}. All features extracted from the connection k can be used as an input of the trained classifier C which predicts the target label:

$$y_{predict} = A\big(D_{tr}, f(k)\big), \tag{5.6}$$

where $y_{predict} \in \{Intrusion, Legitimate\}$.

5.2.3 Non-Payload-Based Obfuscations

When a remote not obfuscated attack occurs, its communication is expressed as a connection k_a. Then, features extracted from k_a can be defined as

$$f(k_a) \mapsto F^a = (F_1^a, F_2^a, \ldots, F_n^a). \tag{5.7}$$

They are distributed to the formerly trained classifier C. Let us assume that the target label is predicted by the classifier C as an intrusion correctly. Because the connection k_a,

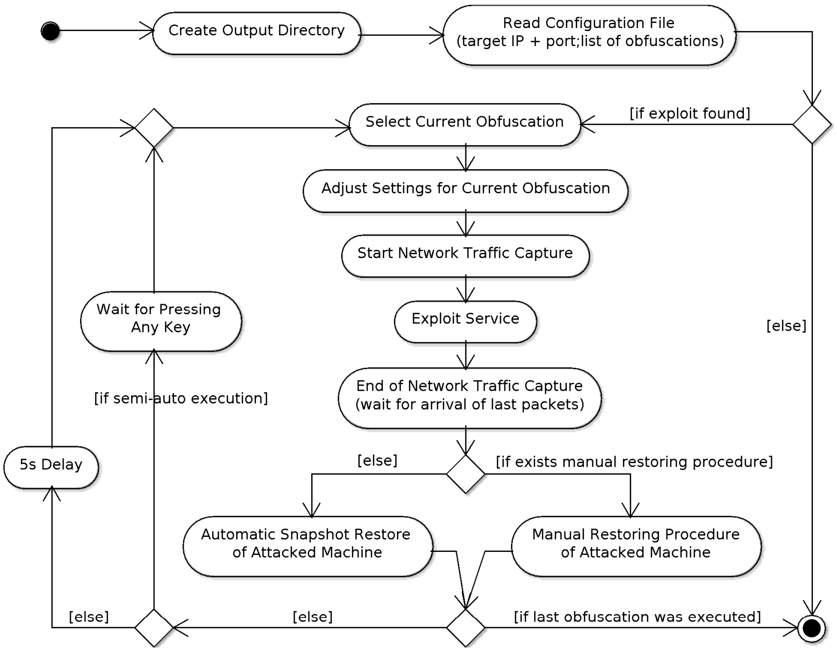
Create Output Directory
Read Configuration File (target IP + port;list of obfuscations)
[if exploit found]
Select Current Obfuscation
Adjust Settings for Current Obfuscation
Start Network Traffic Capture
Exploit Service
Wait for Pressing Any Key
[else]
End of Network Traffic Capture (wait for arrival of last packets)
[if semi-auto execution]
5s Delay
[else]
[if exists manual restoring procedure]
Automatic Snapshot Restore of Attacked Machine
Manual Restoring Procedure of Attacked Machine
[else]
[else]
[if last obfuscation was executed]

Figure 5.1: Behavioral state diagram of the obfuscation tool from [42].

or connection with similar behavior properties to k_a, was included in the dataset D_{tr} and the classifier C was trained on the dataset D_{tr}.

When the non-payload-based obfuscator is used to create obfuscated version of a remote attack with connection k_a, its connection is defined as k'_a. The connection k'_a differs from the original connection k_a by its modifications, which changed its network behavioral properties.

The obfuscation tool uses operations, such as insertion, removal, and transformation of the packets, in order to modify the P_c and P_s packet sets of the modified connection k_a.

The modifications of packet sets P_c and P_s of the connection k_a might transform its features F^a to different ones. Therefore, features, which are extracted from connection k'_a are defined as

$$f(k'_a) \mapsto F^{a'} = (F_1^{a'}, F_2^{a'}, \ldots, F_n^{a'}) \tag{5.8}$$

Hence, here is an assumption that the likelihood of a correct prediction of features $F^{a'}$ by the classifier C is lower than the likelihood of a correct prediction of features F^a. In addition to the previous assumption, let us assume that the classifier C' trained by learning algorithm A on training dataset D'_{tr}, which includes some obfuscated intrusions, is going to be better at the prediction of unknown obfuscated intrusions than classifier C, which was not trained on any obfuscated intrusions. These assumptions have been fulfilled with experiments in [42].

[1] Representing a specific dimension of a feature.

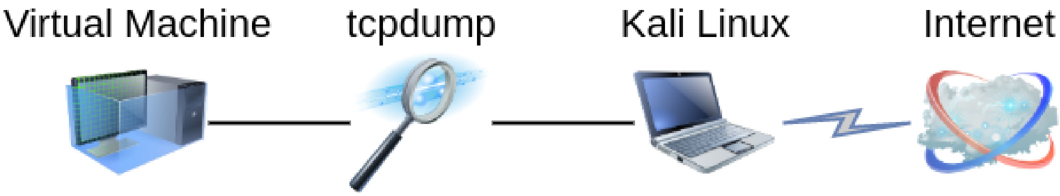

Figure 6.1: Network infrastructure for capturing attacks and legitimate traffic.

Chapter 6

The Novel Dataset

The goal of this work is to construct a novel dataset intended for the evaluation of network intrusion detection systems. Beside plain versions of network attacks, the dataset should contain obfuscated attack instances, which makes the detection more challenging. Although such a dataset has been already proposed in the literature [42], the vulnerable services that were exploited in that dataset are obsolete, and thus the detection of contemporary obfuscated attacks by classifiers using ASNM features is questionable. We aim to address this limitation by creating a novel dataset with the most recently discovered remote vulnerabilities (i.e., years 2018 and 2019).

All attacks were performed using Metasploit framework [66] on a machine with Kali Linux [57] operating system. All target machines are virtual appliances running on Oracle VM VirtualBox [58] and during attacks, they were connected to the host machine using Host-Only Adapter [59]. The scheme with devices used in order to record attacks and legitimate traffic is depicted in Figure 6.1. All used obfuscation techniques and their instances, which are supported by the NPBO framework, are listed in Table 6.1. Several attacks were performed and recorded using tcpdump, each of them generated multiple TCP objects, which however included some legitimate connections as well, because always there was a legitimate connection to a created shell, so they are listed in the Other Legitimate Traffic row. Some legitimate communications with vulnerable services were simulated in the virtual environment using the same machines as were used for attack simulations. All TPC objects are listed in Table 6.2.

6.1 Data Capture and Metrics Extraction

Whole metrics extraction procedure is depicted in Figure 6.2. All attacks were recorded using the NBPO framework, which uses tcpdump [81] as a tool for packet transfer capturing. Thus every attack corresponds to one pcap TCP dump file [81]. These files are stored

Technique	Parametrized Instance	ID
Spread out packets in time	• constant delay: 1s	(a)
	• constant delay: 8s	(b)
	• normal distribution of delay with 5s mean 2.5s standard deviation (25% correlation)	(c)
Packets' loss	• 25% of packets	(d)
Unreliable network channel simulation	• 25% of packets damaged	(e)
	• 35% of packets damaged	(f)
	• 35% of packets damaged with 25% correlation	(g)
Packets' duplication	• 5% of packets	(h)
Packets' order modifications	• reordering of 25% packets; reordered packets are sent with 10ms delay and 50% correlation	(i)
	• reordering of 50% packets; reordered packets are sent with 10ms delay and 50% correlation	(j)
Fragmentation	• MTU 1000	(k)
	• MTU 750	(l)
	• MTU 500	(m)
	• MTU 250	(n)
Combinations	• normal distribution delay ($\mu = 10ms$, $\sigma = 20ms$) and 25% correlation; loss: 23% of packets; corrupt: 23% of packets; reorder: 23% of packets	(o)
	• normal distribution delay ($\mu = 7750ms$, $\sigma = 150ms$) and 25% correlation; loss: 0.1% of packets; corrupt: 0.1% of packets; duplication: 0.1% of packets; reorder: 0.1% of packets	(p)
	• normal distribution delay ($\mu = 6800ms$, $\sigma = 150ms$) and 25% correlation; loss: 1% of packets; corrupt: 1% of packets; duplication: 1% of packets; reorder 1% of packets	(q)

Table 6.1: Experimental obfuscation techniques with parameters and IDs [42].

in a specially ordered folder structure, which defines what type of attack it is and on which service was the attack directed against. There is also a folder structure for legitimate communication, which was captured using tcpdump as well. And the purpose of the structure is similar to the attacks' folder structure, it identifies which services were used in the packet communication.

Both file structures were passed to special scripts collection called *recurs-walker*, which was supplied to me by the supervisor of this thesis. The recurs-walker read all the data, connected to PostgreSQL service [80] running on the host machine, and created a database with connections.

Then the second script used (called *metrics-extractor*) was intended for extraction of ASNM features and collection was also supplied to me by my supervisor. The metrics-extractor fetched connection data from the PostgreSQL database and performed the extraction of ASNM metrics. Extracted metrics were then saved in a new table in the database and written into an output file using CSV format [72]. The CSV file was then imported into RapidMiner Studio [70] local repository. In the importation process, some unnecessary data were removed and datatypes of other data were set. The imported dataset was then fixed in RapidMiner Studio using processes described in Section 7.2.

Service	Legitimate	Direct Attacks	Obfuscated Attacks	Summary
Confluence	99	275	275	649
Drupal	67	177	399	643
FTPShell Client	80	65	98	243
GetSimple CMS	63	396	1173	1632
GitStack	447	296	398	1141
jQuery-File-Upload	79	230	318	627
LibreNMS	76	264	368	708
Nagios XI 5.4.12	145	572	657	1374
Nagios XI 5.5.6	224	205	145	574
rConfig	650	136	232	1018
Webmin	831	168	276	1275
Other Legitimate Traffic	6827	N/A	N/A	6827
Summary	9588	2784	4339	16711

Table 6.2: Number of TCP objects in the dataset

6.2 Vulnerable Services

Vulnerable services were found in the National Vulnerability Database [55], where are listed Common Vulnerabilities and Exposures (CVE). In order to make an up-to-date dataset of vulnerabilities and exploits, CVEs from the years 2018 and 2019 were searched. Hence two NVD JSON Data Feeds, each corresponding to one year, were downloaded [55]:

- dataset from 2018 in file *nvdcve-1.1-2018.json* with last modification on 25th November 2019 and with SHA-256 checksum:
$ef87d6f377bbef6e035504ac605d088acece09defb511a2d0036c231e79d7a2c$

- dataset from 2019 in file *nvdcve-1.1-2019.json* with last modification on 12th November 2019 and with SHA-256 checksum:
$653f95912e8cda06ca1d4faccd05aabc4d4d98fa02d969136befc24f6ed40c84$

Also corresponding Official Common Platform Enumeration (CPE) Dictionary file *official-cpe-dictionary_v2.3.xml* was downloaded [55]. CPEs include important information about vulnerable technology systems, software, and packages, which are mentioned in each CVE record. An example of CVE JSON record can be found in Appendix B [55]. In order to process information in CVE JSON objects and in CPE XML objects a parser was implemented and will be described later.

After finding suitable vulnerabilities (i.e., remote, critical, compatible with Windows or Linux operating systems, etc.) it was necessary to find exploits for them, which was performed using Exploit Database by Offensive Security [56]. When CVEs and corresponding exploits were found, some of those which had the vulnerable version easily and were freely accessible were chosen to be added to the new dataset. In the following, we describe all remotely vulnerable services that were found in the selected NIST CVE dataset files.

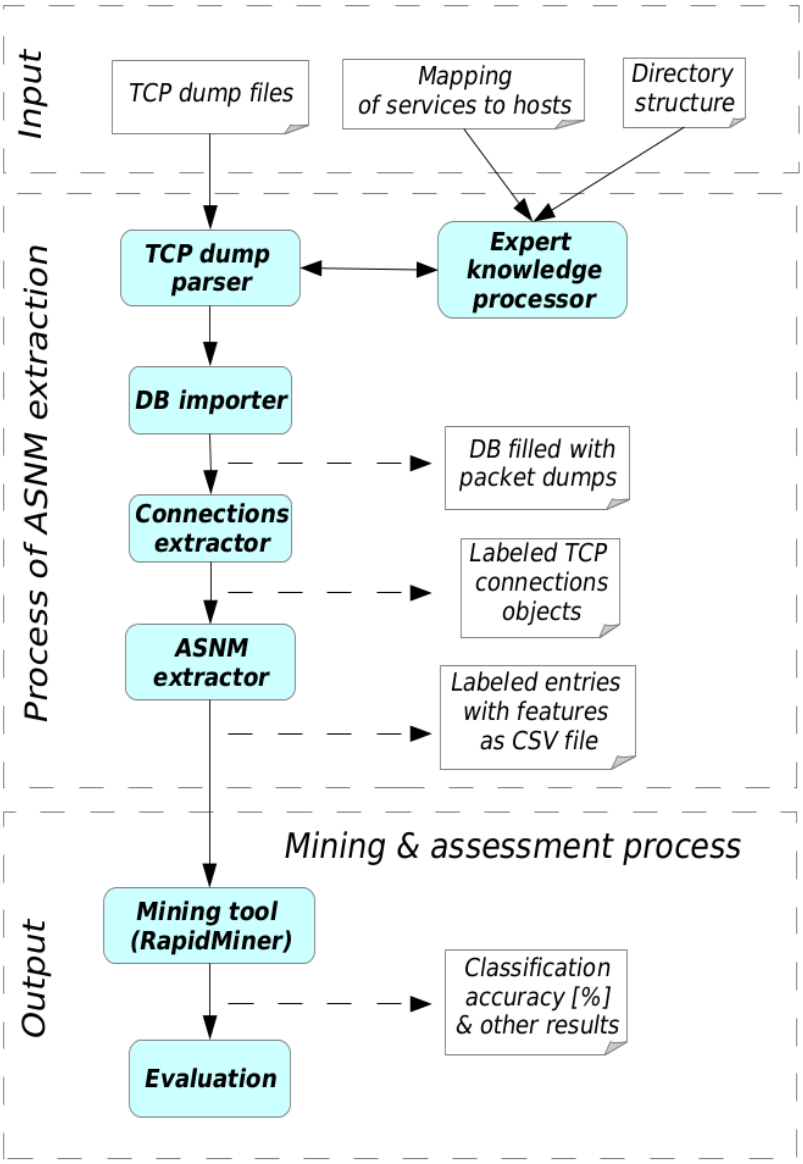
Input
TCP dump files
Mapping of services to hosts
Directory structure
Process of ASNM extraction
TCP dump parser
Expert knowledge processor
DB importer
DB filled with packet dumps
Connections extractor
Labeled TCP connections objects
ASNM extractor
Labeled entries with features as CSV file
Mining & assessment process
Output
Mining tool (RapidMiner)
Classification accuracy [%] & other results
Evaluation

Figure 6.2: Attacks and legitimate traffic recording network infrastructure [38].

CVE JSON and CPE XML Parser

The parser was developed using Python programming language and consists of couple of classes. The main class is called *CVEReader*, whose constructor requires two parameters with names of input CVE JSON file and CPE XML file. In the constructor two more objects are initialized, which are instances of *ParserJSON* and *ParseXML* classes. Next the *parse* method is launched. The method *parse* invokes methods in both parser objects, which are called the same as the calling method. After parsing is finished the *lookup* method is called. The *lookup* method performs infinite loop, which listens to new commands from the standard input of the program and executes them. Three simple lookup commands are implemented: *CPE*, *CVE* and *index*. These commands search for records with given CPE, CVE or index of the record in given dataset and then print it. *ParserJSON* and *ParseXML* classes implement methods which parse or print given CVE JSON files or CPE XML files. *ParseXML* class also uses *CPEHandler* and *CPE_XML* classes, which handle individual CPE objects.

6.2.1 Drupal

Drupal is an open-source project, which provides a content-management system [24]. Drupal is used in the development of 1.6 % websites worldwide and its content-management system market share is 2.8% [64].

In this dataset, the attack is based on Metasploit exploit 44557 [73], which is used to exploit CVE-2018-7602 [7] vulnerability. The exploit requires an attacker to be authenticated as Drupal user and be able to delete a node, then the malicious POST method request [33] can be crafted and sent to the server. The vulnerability allows an attacker to remotely execute code on the machine running Drupal [23, 73]. The target machine with Debian 4.9.130-2 is running Drupal 7.57.

6.2.2 FTPShell Client

FTPShell Client is a program, which enables an user to connect to a SFTP [88] or FTPS [34] server and upload or download files. The application is compatible with Windows operating system and it supports LDAP based Active Directory [50] and Windows NTLM [51] authentication.

This attack is based on Metasploit exploit ftpshell_cli_bof [65], which is used to exploit CVE-2018-7573 [6] vulnerability. An attacking machine starts listening on port 21 and pretending it is FTP [63] server. Then its target has to try to connect to the attacker's exploit. The exploit sends a response consisting of 400 characters of 'F' together with the FTP 220 response code, which leads to the target's application crash caused by the buffer overflow. After the overflow, the attacker is able to execute code on the target machine [6]. The target machine with Windows 10 Enterprise Evaluation is running FTPShell Client 6.70.

6.2.3 GitStack

Gitstack is an open-source git [35] server for the Windows platform. The application is based on msysgit [52] and apache web server [78] [74].

This attack is based on Metasploit exploit gitstack_rce [77], which is used to exploit CVE-2018-5955 [5] vulnerability. In the authentication process, the password is not being sanitized and still, it is passed to the exec function. Therefore an attacker might execute code on target system [76]. The target machine with Windows 7 Professional SP1 is running GitStack 2.3.10.

6.2.4 jQuery-File-Upload

jQuery File Upload is a file upload widget, which supports chunked and resumable file upload and download. The program was developed in order to support multiple server platforms. It is also possible to preview images, videos, and audio [84].

This attack is based on Metasploit exploit jquery_file_upload [85], which is used to exploit CVE-2018-9206 [12] vulnerability. Due to default configuration in Apache 2.3.9 [78] and newer versions the .htaccess file in this widget might not be enabled. Hence the attacker is able to upload arbitrary PHP file with payload to the server and then execute it using GET method [33]. The target machine with Windows 10 Enterprise Evaluation is running jQuery File Upload 9.22.0.

6.2.5 LibreNMS

LibreNMS is an open-source network management software, which supports several operating systems and network hardware. The application automatically discovers network using multiple protocols and is accessible via the web interface and API [17].

This attack is based on Metasploit exploit librenms_addhost_cmd_inject [49], which is used to exploit CVE-2018-20434 [4] vulnerability. The exploit module requires LibreNMS user credentials to authenticate to the application. The attacker injects his payload into the community parameter, which is used in the POST request [33]. The community parameter, which was not sanitized is then passed to popen function, thus attacker's code is executed on the target machine. The target appliance with Ubuntu 18.04 is running LibreNMS 1.46.

6.2.6 Nagios XI 5.4.12

Nagios XI is an open-source application, service and network monitoring software. It monitors network devices, application and database servers, etc. It is able to communicate with its users via the web interface, emails, short messages, and other communication channels [32].

This attack is based on Metasploit exploit nagios_xi_chained_rce_2_electric_boogaloo [75], which is used to exploit CVE-2018-8733 [8], CVE-2018-8734 [9], CVE-2018-8735 [10] and CVE-2018-8736 [11] vulnerabilities.

At first, the attacker sends specially crafted POST method request [33] to a vulnerable PHP file, which sets the database user to root [8]. Then he sends another crafted POST method request, which takes advantage of SQL injection vulnerability in selInfoKey1 parameter in another PHP file, which allows the attacker to enumerate API keys [9]. The next step is an addition of new Nagios administrative user with the next POST method request using gained API keys [10]. Then the attacker authenticates as the created user. When authenticated the attacker sends another crafted POST method request, which injects a command with nopasswd sudo to a PHP file causing root shell creation for him [11]. The last step is to remove database user and Nagios administrative user, which were created during exploitation [75, 43].

The target machine with CentOS 7 is running Nagios XI 5.4.12.

6.2.7 Nagios XI 5.5.6

Nagios XI is described in Section 6.2.6. This attack is based on Metasploit exploit nagios_xi_magpie_debug [47], which is used to exploit CVE-2018-15708 [2] and CVE-2018-15710 [3] vulnerabilities.

The attacker sets up his own web server, which responds to access requests with PHP code payload. Then he injects crafted parameters into URL he accesses on the target system, which are passed to curl [1] command that is executed on the target server. The attacked application accesses the attacker's web server with PHP payload and writes it into a new local PHP file. Now the attacker is able to execute commands as a local user by accessing crafted URL on the target system via uploaded PHP file [2, 46].

The second part of the exploitation is privilege escalation on the target server. It is possible to run commands as root by running a specific PHP file on the server, which is enabled to be runnable as root without a password. The PHP file is vulnerable to command injection into one parameter, which leads to launching a new process executing the attackers command with root privileges [3, 46].

The target appliance with Ubuntu 14.04 is running Nagios XI 5.5.6.

6.2.8 Confluence

Confluence is collaborative software and covers 1.38 % market share [29]. It is useful for new project organization, decision making, setting goals, etc. The Confluence Server can be accessed using a web interface and is compatible with multiple platforms [19].

This attack is based on Metasploit exploit confluence_widget_connector [31], which is used to exploit CVE-2019-3396 [16] vulnerability. There is a vulnerability in some renders, where their parameters including Velocity Template [79] file path are not sanitized and run. The attackers starts his own FTP [63] server with Velocity Template files. Then he injects 2 crafted Velocity Template files with Java code payload into _template parameter using the POST method request [33], which enables him to remotely execute his code. The attacker does not need to be authenticated [31, 26]. The target machine with Ubuntu 14.04 is running Confluence 6.9.0.

6.2.9 GetSimple CMS

GetSimple CMS is an open-source content management system. The philosophy of the program is a simple web interface, which includes everything needed, but does not cover unnecessary features. This software has already been downloaded over 120,000 times [27].

This attack is based on Metasploit exploit getsimplecms_unauth_code_exec [82], which is used to exploit CVE-2019-11231 [13] vulnerability. Due to the new default configuration in Apache [78] the .htaccess file does not override the Apache configuration. The attackers get multiple files with sensitive information from the target server. The sensitive information such as apikey is then used to calculate hashes, which are necessary to create a cookie. The cookie is then used to access CSRF [60] nonce from a php page using POST method request [33]. The PHP page is vulnerable to path traversal, however, it is not even necessary for the exploit, because there is already the .htaccess file vulnerability. The vulnerable PHP page allows the attacker to upload his crafted file and does not check it. Therefore the attacker uploads crafted file with his payload and then accesses it, in order to execute his code on the target machine [13, 82, 83]. The target appliance with Debian 8.11 is running Getsimple CMS 3.3.15.

6.2.10 rConfig

rConfig is an open-source network device configuration management software. It is possible to create snapshots of network device configurations, automate miscellaneous tasks, etc. The tool also supports customization addition. The software has over 8,000 users and manages over 2 million network devices [71].

This attack is based on Metasploit exploit rconfig_install_cmd_exec [48], which is used to exploit CVE-2019-16662 [15] vulnerability. The exploit requires the install subdirectory to not be removed, which does not happen automatically. The attacker crafts the GET method request [33] to a vulnerable PHP page, which is accessible for unauthenticated users. The page uses the rootUname parameter from the GET method in 2 commands, while not being sanitized. Therefore the attacker can remotely execute code on the target machine. Note that since the payload is executed in 2 commands it is executed twice, thus with default configuration 2 meterpreter sessions are opened [48, 18]. The target appliance with CentOS 7 is running rConfig 3.9.2.

6.2.11 Webmin

Webmin is a utility for system administration of Unix-like operating systems. A user controls the system using a web interface. It is possible to manage services, system configuration, sharing, open-source applications, etc. The software removes the need for manual modification of operating system configuration files. Webmin has been downloaded over a million times [86].

This attack is based on Metasploit exploit 47230 [54], which is used to exploit CVE-2019-15107 [14] vulnerability. The user password change must be enabled for exploitation to be working. An administrator of the target system has to have „Prompt users with expired passwords to enter a new one“ option checked. The attacker sends crafted POST method request [33] to a vulnerable CGI file. There is a command injection vulnerability in parameter old processing while trying to change the password of a user. It is not necessary for a user to exist and password to be correct. Note that the vulnerability was injected intentionally into the Perl source code by an unknown attacker, who created this backdoor [54, 53]. The target appliance with Ubuntu 14.04 is running Webmin 1.910.

Chapter 7

Data Preprocessing, Forward Feature Selection and DL Models

In this chapter, we describe the RapidMiner tool and classifiers used in our experiments. Data preprocessing, which consists of several tasks, is spread across multiple sections. The first task, which prepares data is described in Section 7.2, then interesting features are selected using Forward Feature Selection (see Section 7.3), and finally some of the data preprocessing techniques are performed in process of model training, which is described in Section 7.4. Finally, Cross-Validation was performed with models of classifiers trained on direct and legitimate traffic (also referred to as *DL models*), which we focus on in Section 7.5.1.

7.1 RapidMiner

RapidMiner is a data mining platform used in over 40,000 organizations [70]. All classifiers and processes, which prepare data, train classifiers, and test them are implemented using RapidMiner Studio. In this chapter are described individual processes implemented in RapidMiner. Sections are organized by specific tasks, which were performed using RapidMiner processes. Information about operators in sections in this chapter were derived from RapidMiner documentation [69] and RapidMiner operator manual [68].

Notation

For the purpose of unified explanation, we adopt the following terminology from the RapidMiner Studio [67]:

- **Example** represents a TCP connection data with ASNM features, i.e. a row in a table.
- **Attribute** represents an ASNM feature, i.e. a column in a table.
- **Operator** is a node in a graph inside a process that can be run, by the process, and thus perform some specific task, which depends on the operator's type.
- **Process** is a collection of operators, which can be run and thus launch them in a defined order.

- **Subprocess** is a process inside some operator, so when the operator is launched the subprocess starts.

7.1.1 Classifiers

We used 6 classifiers for dataset evaluation: Naïve Bayes (Kernels), Naïve Bayes, Decision Tree, Random Forest, Support Vector Machine and Logistic Regression.

Naïve Bayes is a parametric model, which uses Bayes' theorem in order to learn and classify data. It is a simple probabilistic classifier, which does not require very much data to learn. The classifier is also one of the faster ones. A fundamental property of the classifier is the fact it assumes that all learned features are independent [69]. In performed experiments is also used Naïve Bayes with Kernel Density Estimation, i.e. Naïve Bayes (Kernels), which makes it non-parametric.

Decision Tree is a classifier that creates a tree structure, where each node corresponds to a feature of the data. In every node, there is also a condition, which depends on the related feature. The result of the condition evaluation determines which child node to go when classifying the data using already trained Decision Tree. Leafs of the tree represent the final decision of the classifier [69].

Random Forest's training starts with splitting the input dataset into random subsets. Then for each subset, a new Decision Tree is generated. Each example of data given to the Random Forest in order to get classified is classified by all Decision Trees. The result of classification is then voted by all trees [69].

Support Vector Machine's basic principle is described in Section 2.3.1.

Logistic Regression is a method that estimates one dependent variable, which has only two possible values. The estimated variable depends on multiple independent variables, which might be miscellaneous data types. The Logistic Regression uses an S-shaped logistic distribution function in order to model the data classifier [87].

7.2 Data Preparation

Data preparation is divided into two separate parts. One of them, described in Section 7.2.1, transforms the data into a form with better readable information about services and attack types. In the case of the second part, we filter the data and label their attack types (see Section 7.2.2).

7.2.1 Data Transformation

The first process is called *Data-Transformation-A*, which uses several operators with miscellaneous functionalities. At first, the Retrieve operator is used to fetch a table with ASNM metrics, which was imported to RapidMiner from a CSV input file, that was generated using metrics-extractor described in Section 6.1. Then the id role is set to id attribute in the table using a Set Role operator. Two next operators of Replace type rename label attribute in all examples of Nagios XI service because there are two versions of the service each with different vulnerabilities and exploits. It is useful to rename them into an easily readable form, which discriminates each version of the service. Then there are some new attributes generated using Generate Attributes operator, which include attack information of each example. The process continues setting label role to a label attribute, which signifies that label attribute is the information about what class is the attack in. So the classifier

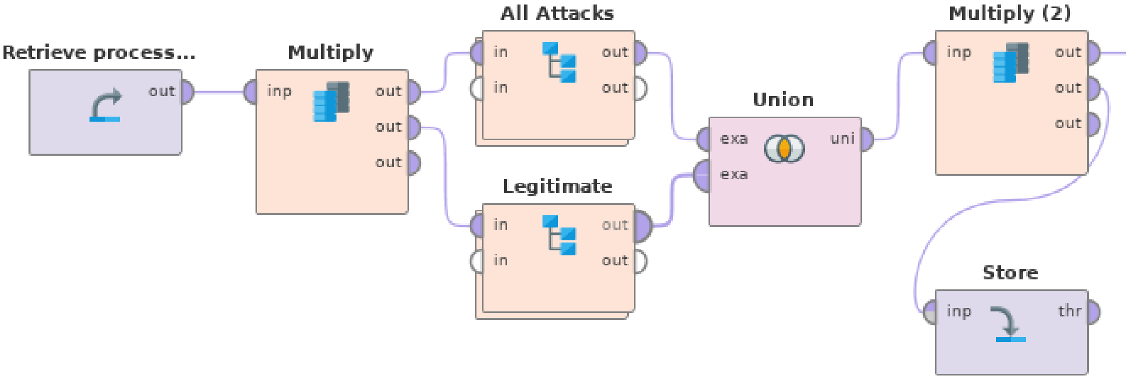

Figure 7.1: Dataset-Repairment-B process scheme

is trained to guess the label attribute. Next, there are removed some no more needed attributes and renamed recently created attributes. In the end, the attributes are reordered, in order to make it easier to work with the dataset, some are renamed to a better form and the final table is written to an output file using Store operator.

7.2.2 Data Repairment

The second process for data preparation is called *Dataset-Repairment-B*, which is depicted in Figure 7.1. At the beginning of the process, a Retrieve operator reads the input file with a data table, which was the output of the Data-Transformation-A process. Then the table is passed into two parallel subprocesses All Attacks and Legitimate processes using a Multiply operator.

The All Attacks subprocess starts with filtering its input data using Filter Examples operator, which matches examples with attack data only. The filtered table is then distributed to 11 subprocesses, each corresponding to one of the attacked services. These 11 subprocesses all do almost the same thing. Each of them starts by filtering examples with data related to its service only. Then each one's data is filtered by the destination port of the attack and unmatched data, which has different destination port number is labeled as other traffic using three Replace attributes. Therefore the attack data is detached from other traffic, which was captured by tcpdump [81] during attack realization. All data from each subprocess is then united into one table with all attacks and then the data labeled as other traffic is labeled as legitimate as well.

The Legitimate process includes only 1 operator of Filter Examples type, which filters only traffic labeled as legitimate. Both processes outputs are then united into one table with all labeled data and saved in the output file using the Store operator.

7.3 Forward Feature Selection

In order to increase efficiency of classifiers Forward Feature Selection was performed. There are two types of classifiers in this thesis, the ones trained on legitimate traffic and direct attacks' data and the ones, which are trained on a dataset including obfuscation attacks. Therefore each group of them is trained on different datasets, so the feature selection has to be performed twice. Selected features from direct attacks and legitimate traffic are called

DL-FFS. Features from the second run were selected from all data, hence they are called DOL-FFS, where O stands for obfuscated attacks.

All selected features are listed in Table A.1. Both tasks were performed using classifiers of type Naive Bayes (Kernels). The number of DL-FFS is 11 and 14 features were selected during the DOL-FFS selection process. The only feature, which was selected in both processes is *intervalsIPsSig*, which represents the standard deviation of time intervals between consecutive connections of the two hosts running on the same IP addresses as an analyzed connection [38]. Both sets include mostly features based on approximation of communication by polynomials, Fast Fourier Transformation of packet sizes, and normalized products of packet sizes with some Gaussian curves and lengths of packets in intervals of time.

7.3.1 Implementation

Forward Feature Selection is implemented in process *2cl-FFS*, which works with label that has 2 classes. As input there can be used DOL data, i.e. direct attacks, obfuscated attacks, and legitimate traffic data, or DL data, i.e. DOL without obfuscated attacks. At first 2 Set Macros operators set necessary macros for the process, then DOL112 or DL12 file is read using Retrieve operator and the data is passed to a Remove Useless Attributes operator, which has same settings as described in Section 7.4.1. Then the data is fixed in operator Nominal to Binominal, which sets label attribute type to binominal. The fixed data table is passed to the Forward Selection operator.

The Forward Selection operator has a subprocess, that returns a Performance Vector. The selection process starts with an empty list of selected features. The process consists of rounds, which have to be performed in order to construct a list of selected features. In each round, unused features are appended to a new list. For each feature in the current round, the performance is measured using some operators included in the subprocess. In the case of forward feature selection in this theses, the cross-validation technique was used. The Cross-Validation operator parameters are configured the same way as described in Section 7.5.1 with the only difference in local random seed attribute, which is here set to 1987. The Cross-Validation operator includes the same operators as in Section 7.5.1 as well, but the main criterion in the Performance operator in the testing phase is set differently. The main criterion is accuracy and other calculated criterions are AUC (optimistic), AUC, precision, and recall. The outputs of the Cross-Validation operator are then passed to a Branch operator using Multiply operator and at the same time, the number of currently selected attributes is extracted into attribs macro.

The Branch operator compares the attribs macro with the counter macro and if they are equal it performs its Then subprocess, else Else subprocess is performed instead. In the Then subprocess the model and performance vector are saved to files using Store operators and then the counter macro is incremented using Generate Macro operator. In the Else subprocess, there are 2 Store operators, which write the model and performance vector to files.

The feature with the best performance is then selected at the end of the round and added to the selection list.

The Forward Selection operator contains parameters, which define stopping the behavior, maximal number of attributes, and speculative rounds count. The stopping behavior parameter was set to „without increase“ value, which influences an operator’s behavior to make it stop when a round does not increase classification performance. This setting was combined with the second parameter, which was set to the maximal number of 15 attributes and there are 2 speculative rounds enabled. Speculative rounds parameter is the count of rounds that are allowed to be performed without any increase of performance, which is useful in order to deal with local optimums.

In the end, final selected attributes are saved to the output file using a Store operator.

7.4 Model Training

In this section are models trained on direct attacks and legitimate traffic, i.e. DL data, and on a dataset including both DL data and obfuscated attacks, i.e. DOL data. And that is also the reason why these models are called DL Models and DOL Models. Forward Feature Selection algorithm was also used on that data, so DL-FFS and DOL-FFS features were selected. Therefore all models in this section were trained on DL or DOL data limited to DL-FFS or DOL-FFS features, which means that all testing data passed to these models are limited to such features as well.

At first splitting train data is done in *2cl-Split-Train-Data-attALL* process and is described bellow in Section 7.4.1. Then the data is filtered and normalized if needed (see Section 7.4.2). After it is done there are three data tables with fixed DL12, OL12, and DOL112 data, which are prepared for classification. Before training processes can be launched the data needs to be split into subsets and forward feature selection has to be done. The forward feature selection is described in Section 7.3. Next, the model training phase can start, because three data tables are prepared and features as well. The training phase is described in Section 7.4.3.

7.4.1 Split train data

In the beginning, the process read input data using the Retrieve operator. Input data is the data table, which was produced by *Dataset-Repairment-B* process described in Section 7.2.2. Then some necessary macros are set using 2 Set Macros operators. The next step is to remove all attributes, which contain information about the attack and service type of the connection using Select Attributes operator and label role is set to a special attribute label to define what the classifier should learn. The next operator is to Remove Useless Attributes, which removes attributes from the table based on user-specified thresholds, which are defined in the operator's parameters.

The numerical min deviation parameters are set to 0, which means numerical attributes, which have the standard deviation less than or equal to this deviation threshold are removed. Therefore all attributes, where all examples have the same value are removed. The nominal useless above parameter sets the threshold for the ratio of most frequent values to the total number of examples and removes all nominal attributes above the user-defined ratio. However it is set to 1.0, thus latter mentioned parameter does not remove anything. The nominal useless bellow parameter is similar to the latter one, but it removes nominal attributes with the ratio of least frequent values to the total number of examples, in other words, attributes with most different values. The latter parameter was set to 0, hence it removed nothing.

The process continues with Filter Examples operator, which performs removal of all examples, which have missing values. Then the data is stored in the DOL123 file, which signifies, that the file includes direct attacks with label 1, obfuscated attacks, which are labeled as 2 and legitimate traffic, which is labeled as 3. And after saving to the file the data is distributed to 3 subprocesses using the Multiply operator.

The first subprocess is called *(DL->12)*, which means it filters direct attacks and legitimate traffic only and assigns their label to 1 and 2 in that order. It consists of 2 operators, where the former one filters obfuscated attacks out of the data and the second, which changes legitimate traffic label to 2. The second subprocess is called *(OL->12)*, which filters obfuscated attacks and legitimate traffic only and changes their labels to 1 and 2 in that order. It uses 3 operators, where the first one filters the communication based on the type of attack/legitimate label and the other two operators change labels in a proper way. The third subprocess is called *(DOL->112)* and thus the direct attack is labeled 1, the obfuscated attack is labeled as 1 as well and legitimate communication is labeled as 2. Thus the latter subprocess loses information about the type of each attack. However, the information about what type of attack the specific example is might be later acquired using join operations with data from the original table because all examples still have their id attribute kept. Each of the 3 mentioned subprocess passes its output to a Retrieve operator, which saves the data table into a file. The first one, with data table from *(DL->12)* subprocess saves it in the file called DL12. The next one, with data table from *(OL->12)* subprocess writes the table into the file called OL12. The last one, with data table from *(DOL->112)* subprocess uses the file called DOL112 to store it.

7.4.2 Data Filtering and Normalization

Training processes both start by reading prepared DOL123 data using Retrieve operator, then 2 Set Macros operators prepare necessary macros and the data is passed to Select by Weights operator. Then the attributes file is read using another Retrieve operator and then delivered to the Select by Weights, which has a weight relation parameter set to „greater equals“ value, and the weight parameter is set to 1 value. Since the attributes file contains only 1 or 0 values only attributes with value 1 are selected and these values correspond only to attributes that were selected by forward feature selection processes. Then the data with FFS features are distributed using Multiply operator to 3 subprocesses called *Prepare DL*, *Prepare OL*, and *Prepare DOL*. Mentioned subprocesses are same as subprocesses called *(DL->12)*, *(OL->12)* and *(DOL->112)*, which are described in Section 7.4.1 bellow. Then the data prepared in these processes are stored in files the same way as in the mentioned paragraph. Next, each of 3 data tables is passed to the corresponding Nominal to Binominal operator, which changes the type of label attribute from nominal to binominal, and thus the data fit for binary classifiers.

In case of usage of the SVM model or Logistic Regression model, two more operators are performed in order to prepare data for the classifier. The first is Nominal to Numerical operator, which changes data types of attributes defined by its parameters. Its parameter coding type is set to dummy coding, so for every value of an attribute with nominal data type except comparison group a new attribute is created. The created attribute has value 1 in case of all examples, which have the created attribute's ancestor value and 0 if the ancestor's value is different. The second is the Normalize operator, which performs normalization of examples.

7.4.3 Training Phase

The training phase starts with 3 data table lines with fixed DL12, OL12, and DOL112 data, which are prepared for classification. In the case of *2cl-Train-on-DL-attDLFFS* the fixed DL12 data table is distributed using the Multiple operator to two operators, based on their purpose. In case of *2cl-Train-on-DOL-attDOLFFS* the fixed DOL112 data table is distributed instead. The first of those operators with a given data table is an operator, which generates a model, for instance, Naïve Bayes and the second is an Apply Model operator. The model generating operator uses these data for training a new model and puts the model into its output. The trained model is then distributed using the Multiply operator to three Apply Model operators. Generally, Apply Model operators use their input models to label their input data. Each of the mentioned 3 Apply Model operators has the trained model as its input model and different data. The first one uses the supplied model to label DL12 data, the second uses the same model to label OL12 data and the last one uses the model to label DOL112 data. Labeled data from each operator is then passed to its Performance (Binominal Classification) operator, while all of them are set up the same way.

Performance operators calculate a bunch of criterions and their outputs are Performance Vectors [69]. The main criterion is recall, which means it is used for Performance Vector's comparison. There were set up to be calculated these criterions: recall, accuracy, AUC (optimistic), AUC, AUC (pessimistic), precision and f measure.

- The recall criterion is calculated as follows:
  $recall = (true\_positive\_predictions)/(number\_of\_positive\_examples)$
- The accuracy criterion is calculated as follows:
  $accuracy = (correct\_predictions)/(number\_of\_examples)$
- The AUC (optimistic), AUC and AUC (pessimistic), where AUC stands for Area Under the Curve, the curve is from ROC graph. In the beginning predictions are sorted by their score from highest to lowest and then the graph is plotted example by example. The optimistic AUC plots positive examples before negative ones. The pessimistic AUC plots negative examples before positive ones. The AUC plots plots average between optimistic and pessimistic AUCs.
- The precision criterion is calculated as follows:
  $precision = (true\_positive\_predictions)/(all\_positive\_predictions)$
- The f measure criterion is calculated as follows:
  $F_1 = 2(precision \times recall)/(precision + recall)$

The process finishes by saving all three Performance Vectors and the trained model to files using four Store operators.

7.5 DL Cross Validation

Cross Validation is a method to rank the accuracy of a model on a given dataset. The dataset is split into N subsets with an equal count of samples. Then subsets are iterated in order to train and validate a model. There are two phases of each iteration, the first is the training phase and the second is the testing phase. In the training phase, $N - 1$ subsets are used for training a new model. When the model is trained the testing phase launches. Testing data, which is the last subset of the dataset that was not used for training, is passed to the prepared model. Therefore the model is tested on the data unknown for it. Then the next iteration is performed with another combination of training and testing data. Cross validation implementation is described in Section 7.5.1.

Classifier	TPR	FPR	F_1 (↑)	Avg. Recall
Random Forest	99.98%	0.09%	99.84%	99.95%
Naïve Bayes (Kernels)	99.34%	0.02%	99.63%	99.66%
Decision Tree	98.94%	0.58%	98.49%	99.18%
Logistic Regression	97.63%	0.60%	97.79%	98.52%
Support Vector Machine	97.66%	0.97%	97.17%	98.35%
Naïve Bayes	100.00%	60.57%	48.97%	69.72%

Table 7.1: Cross validation of direct attacks and legitimate traffic

Because the classification is binary the 5-fold cross-validation is used in order to evaluate the classifiers. Cross validation of DL data is presented in Table 7.1. The table is sorted by the F_1 score in descending order. In the table can be seen that the most successful is the Random Forest classifier with 99.84% F_1 score and the least successful is Naïve Bayes with 48.97% F_1 score. However, Naïve Bayes classifier has the highest TPR, which is 100.00%, but probably on behalf of FPR, that is 60.57%, which is abnormally bad compared to other classifiers, because they all have FPR under 1%.

- TPR stands for True Positive Rate and is calculated as follows: $TPR = (true\_positive\_predictions)/(number\_of\_positive\_examples)$
- FPR stands for False Positive Rate and is calculated as follows: $FPR = (false\_positive\_predictions)/(number\_of\_negative\_examples)$
- F_1 stands for F_1 score, i.e. F measure, which is harmonic mean of precision and sensitivity and it is definition can be found in Section 7.4.3.
- Avg. Recall stands for Average Recall, which is described in Section 7.4.3.

7.5.1 Cross Validation Implementation

Cross-validation is implemented in *2cl-X-val-inDL* and *2cl-X-val-inDOL* processes. From these processes' names can be derived that the classification is binary and data are a set of direct attacks and legitimate traffic of services with DL-FFS attributes only in *2cl-X-val-inDL* process and all data including the mentioned and obfuscated attacks as well, but limited to DOL-FFS attributes only are in *2cl-X-val-inDOL* process. Most operators in both processes are the same and therefore they are both described in this section. There are two parts of cross validation processes, the first is Data Preparation, and the second is Cross Validation Loop.

Data Preparation

The process starts with two Set Macros operators, which set macros with information about the model and working folders of the process. Then a Retrieve operator DL12 or DOL112 input data, which are data labeled with numbers depending on the type of attack as can be seen in the name of input data, e.g. in DOL112, direct and obfuscated attacks are labeled as 1 and legitimate traffic as 2. The data table is then fixed for a binary classifier using Nominal to Binominal operator. In the case of usage of the SVM model or the

Logistic Regression model, two more operators are performed in order to prepare data for the classifier as described in Section 7.4.2.

Normalization is a process of data example values modification in order to make them fit into the required range, which is necessary for some types of classifiers to work properly. The Normalize operator supports four methods for normalizing data, and in this case, it was configured to use the Z-Transformation method. Z-Transformation, i.e. Statistical Normalization is a method that at first subtracts the mean of given data from each value and then divides every value by the standard deviation. As a result, the mean of values is zero and the variance is one. Z-Transformation is a widely used normalization method.

Cross Validation Loop

A fixed data table is passed to the main Loop operator, which has a number of iterations parameter set to iteration macro, so it loops for as many times as is set in the iteration macro. The iteration macro is set to 100, because the macro is used the number iterations is accessible to inner subprocesses, which are executed within every iteration.

In the loop subprocess in the beginning the Generate Macro operator calculates seed macro from iteration macro as follows: $seed = iteration + 100$. The subprocess continues by passing its input to Cross-Validation operator, which includes 2 subprocesses, one for training and one for testing a trained model. The number of folds parameter, which defines the number of subsets that are iterated, is set to 5. The sampling type parameter is configured to stratified sampling, so the subsets are built randomly, thus the class distribution is the same as in the whole dataset. In the case of this classifier, which is binominal the stratified sampling creates subsets that have approximately the same number of samples with label attribute of its two values. The Cross-Validation operator is also using a local random seed value, which is defined by prepared seed macro.

In the training phase, there is one model generating operator, which trains the model on given training data and then passes it to the training phase subprocess output. In the testing phase, the model given from the previous phase is applied to the testing data using the Apply Model operator. The labeled testing data is then passed to Performance (Binominal Classification) operator that is configured to calculate these criterions: AUC, precision, recall, and f measure, while the recall criterion is set as the main criterion. The criterions are described in Section 7.4.3. All Performance Vectors produced from the mentioned Performance operator in each iteration of the validation are averaged into single Performance Vector, which is then stored in an output file using a Store operator and passed into the output of the whole Loop operator.

Therefore the output of the loop subprocess is a collection of averaged Performance Vectors, which is passed to the Average operator that produces a final Performance Vector with averaged values of the collection. The final Performance Vector is in the end stored in an output file using a Store operator.

Chapter 8

Obfuscated Attacks Detection

Several experiments were performed in order to evaluate the dataset. There are experiments, which tested classifiers with knowledge about legitimate traffic and direct attacks and their performance of obfuscated attacks classification (see Section 8.1). Then experiments with classifiers with knowledge widened of obfuscated attacks (DOL Models), were performed in Section 8.2 to test their improvement of obfuscated attacks classification, and there are also experiments testing their durability against unknown obfuscation techniques and instances. And Finally Cross-Dataset Evaluation is summarized in Section 8.3.

8.1 Prediction of Attacks by DL Models

Experiments comparing all attacks prediction and obfuscated attacks prediction are summarized in Section 8.1.1. The results for successfully obfuscated attacks are described in Section 8.1.2. Attacks prediction implementation is described in Section 8.1.3. An analysis of predicted data implementation is described in Section 8.1.4. Three experiments were performed in order to evaluate DL Models trained on the novel dataset described in sections below.

8.1.1 All Attacks vs Obfuscated Attacks Prediction

Models trained on direct attacks and legitimate traffic were tested on all attacks, containing obfuscated attacks, which are unknown for them. In Table 8.1b are listed results of the experiment, including the difference of TPR compared to the Table 7.1 with cross-validation on direct attacks and legitimate traffic. There is a significant difference, which shows that all attacks struggled to detect data, which contained unknown obfuscations. Only in case of Naïve Bayes with difference 0.21% the problem is not as serious as in case of other classifiers.

The second experiment was the classification of obfuscated attacks only using the same classifiers. As can be seen in Table 8.1a with the result of the experiment, which contains a comparison with cross-validation as well, all classifiers were doing even worse than in the case of the combination of direct and obfuscated attacks. Therefore we can state that obfuscated attacks successfully evaded the classification. Again similar situation with all classifiers took place as in the case of the previous experiment, Naïve Bayes stayed as the best at TPR and other classifiers did not get better than any other one.

Classifier	TPR	ΔTPR
Naïve Bayes	99.65%	-0.35%
Random Forest	88.61%	-11.37%
Decision Tree	87.44%	-11.50%
Naïve Bayes (Kernels)	70.02%	-29.32%
Support Vector Machine	55.24%	-42.42%
Logistic Regression	14.08%	-83.55%

(a) Obfuscated attacks only.

Classifier	TPR	ΔTPR
Naïve Bayes	99.79%	-0.21%
Random Forest	93.06%	-6.92%
Decision Tree	92.31%	-6.63%
Naïve Bayes (Kernels)	81.74%	-17.60%
Support Vector Machine	60.14%	-37.52%
Logistic Regression	15.27%	-82.36%

(b) All attacks.

Table 8.1: Prediction of attacks including obfuscated ones.

8.1.2 Obfuscated Attacks Evasions per Service

This experiment show how much prone are individual services to obfuscation attacks' evasions. All classifiers were trained on direct attacks and legitimate traffic, and the training set was limited to attributes, which were selected by forward feature selection also without knowledge about obfuscated attacks. Therefore classifiers have no knowledge about obfuscations and in the experiment can be seen how did classifiers struggle with obfuscations depending on individual services.

Every obfuscated attack, which was classified as legitimate traffic is considered as evasion, thus the more evasions the less successful classification was. The results of this experiment are listed in Table 8.2, where are samples counts of obfuscated attacks, the percentage of evasions for each classifier per service, and finally average of all classifier evasions for each service. In two last rows in the table are calculated aggregation functions per each classifier, which show how successful obfuscations were.

The most prone service to obfuscated attacks is FTPShell, which is different from all other services in one feature. The FTPShell exploit nuance is that the vulnerable application is on the client's side, in every other attacks' cases the vulnerability in on the server's side. However, other attacks evasions rate is between 26.8% and 36.78%, so the difference in the obfuscation proneness between them and the FTPShell with 44.61% is not significant.

8.1.3 Attacks Prediction by DL Models Implementation

Attacks prediction is implemented in two processes based on what attack data are tested. These two processes are called *2cl-Prediciton-inDO-attDLFFS* and *2cl-Prediction-inOL-attDLFFS*, which means the former one applies given model to all attacks, and the latter one tests obfuscated attacks only.

In the beginning, the process reads the input model file using a Retrieve operator and passes it to the Apply Model operator through two Set Macros operators. Therefore the model file is read and before it goes to the next usage, macros with information about input data, model, and output folder are set. Then another Retrieve operator launches and reads OL12 in case of obfuscated attacks prediction or DOL112 data files in case of all attack prediction.

In the process, intended for prediction of all attacks, the data is passed to Filter Examples operator, which removes legitimate traffic from it, and hence there are attacks only left all with label 1. Next, in both processes, the label attribute in data is fixed for binary classifiers using Nominal to Binominal operator. Then the data is copied using the Multiply operator to the Apply Model operator, which already has an input model. The

Service	Samples Count	Naïve Bayes (Kernels)	Naïve Bayes	Decision Tree	Random Forest	Support Vector Machine	Logistic Regression	Average
FTPShell	98	35.33%	0.00%	16.70%	15.65%	100.00%	100.00%	44.61%
Nagiosb	145	19.33%	0.00%	6.00%	4.67%	90.67%	100.00%	36.78%
Confluence	275	28.57%	0.00%	27.03%	10.47%	59.04%	94.51%	36.60%
Drupal	399	41.48%	0.75%	20.03%	18.25%	64.43%	68.89%	35.64%
Gitstack	398	22.07%	0.98%	4.33%	7.57%	75.08%	100.00%	35.00%
LibreNMS	368	40.72%	1.30%	16.61%	15.05%	36.84%	88.18%	33.12%
GetSimple	1173	29.58%	0.00%	11.95%	13.96%	37.75%	88.88%	30.35%
Webmin	276	38.73%	0.00%	35.38%	25.53%	23.60%	52.03%	29.21%
jQuery-File -Upload	318	25.55%	2.85%	7.93%	12.48%	37.21%	88.94%	29.16%
Nagiosa	657	31.46%	0.00%	7.28%	6.92%	35.28%	92.53%	28.91%
rConfig	232	38.85%	0.00%	11.24%	10.57%	27.87%	72.27%	26.80%
Average		31.97%	0.53%	14.95%	12.83%	53.43%	86.02%	
Std. Dev.		7.67%	0.91%	9.58%	5.87%	26.14%	15.36%	

Table 8.2: Evasions of Obfuscated Attacks per Service.

Apply Model operator uses the model to classify the data and passes labeled data to the Performance operator, which calculates these criterions, which are listed in Section 7.4.3, while the AUC criterion is set as the main one. A Performance Vector is created from calculated criterions and passed to the Store operator, which stores them in an output file. The Performance operator also passes labeled examples to the second Store operator, which saves them in a file and passes them to the Wrong predictions filter operator of type Filter examples with configured condition class parameter to wrong predictions value. In so far as obfuscated attacks prediction process is concerned there is also a Filter Examples operator, that is configured to remove all examples with label attribute set to 2, thus legitimate traffic is removed from the data. Then the wrong predictions data is stored in the third output file using Store operator.

8.1.4 Analysis of Predicted Data

The analysis starts in *2cl-AnalysisO-inOLres* process by defining macros with data and directories information using two Set Macros operators. Then the *Retrieve and Split* subprocess, is performed and its five outputs are connected to five branches.

The first branch stores data with obfuscated attacks and legitimate traffic to an output file using a Store operator, which also passes it to the Nominal to Binominal operator, which prepares the data for Binominal Classification Performance operator. The Performance operator calculates these criterions: accuracy, classification error, AUC, precision, recall, and F_1 measure. All mentioned criterions except classification error defined bellow are described in Section 7.4.3 [69].

- $classification\_error = (incorrect\_predictions)/(number\_of\_examples)$

The Accuracy criterion is configured as main criterion. The Performance operator produces a performance vector, which is passed to the Store operator and written to an output file.

The second branch uses a Store operator to save the collection of data sets, which include obfuscated attacks and are split by obfuscation instances. Then the data are delivered using Multiply operator to *Loop (O-split-perfs) Collection* operator and *Loop, Store and Log* subprocess, that are described in their own subsections bellow. All outputs of both operators are then connected to Store operators, which save them in files. The third branch only saves wrong obfuscated attacks' data into a file using a Store operator. The forth branch does the same thing as the previous one, except the data is wrong obfuscated attacks split into a collection of data sets. Finally, the fifth branch consists of the same operators as the first branch, the only difference is data, which is limited to obfuscated attacks only.

After *2cl-AnalysisO-inOLres* process is done the process called *2cl-Analysis-PrintToConsole*, which is described in *Analysis Print To Console* subsection bellow, can be run in order to extract important information from all generated files and print it in a usable form.

Retrieve and Split

The Retrieve and split subprocess begins by reading 3 files with input data by launching 3 Retrieve operators. The first one reads labeled OL12 data, which were produced by obfuscated attacks prediction process (see Section 8.1.3) and passes them to the first of 2 Join operators. The second Retrieve operator reads wrong labeled data, that were also produced by the process for prediction of all attacks (see Section 8.1.3) and deliver them to the second Join operator. The third one reads the data file, which contains all data information about examples including attack types and services, produced by the data preparation process described in Section 7.2.2.

The third Retrieve operator passes the data to a Set Role operator, that defines the role of the id attribute. Then the data is passed to Select Attribute operator, which selects only attributes with information about attack type, service, and id of each example. Next, the selected data go to the Generate Attributes operator, which creates a new attribute with obfuscation information. Later the data is delivered to the right inputs of mentioned 2 Join operators using Multiply operator.

Therefore the first Join operator has all classified OL data on its left input and the data with attack type, services, and label information on the right input. The operator is configured to perform left join operation with the data. Then the joined data is passed to a Reorder Attributes operator, which puts the data in the correct order for easier work with it and passes it to a Multiply operator. Next, the Multiply operator delivers the data to the three following operators. The first of them is the Sort operator, which sorts the data by *obfus_label* attribute and puts it into the first output of the *Retrieve and Split* subprocess. The second operator of type Loop is called *Split obfuscations*, and in each of its 17 iterations, 2 operators are performed of these types: Generate Macros and Filter Examples. The Generate Macros operator sets the ob (i.e. obfuscation) macro based on the current iteration, which is read from iteration macro. Then Filter Examples operator then uses mentioned ob macro in order to filter only examples with one type of obfuscation corresponding to the current iteration. Hence *Split obfuscations* Loop operator's output is a collection of selected example sets, which were selected in each iteration. Next, the collection is passed to a Loop Collection operator, which iterates over the collection and performs its subprocess for each example set. In each iteration, the example set is passed to the Generate Attributes operator in order to generate the wrong attribute, and then attributes are reordered using the Reorder Attributes operator. The processed data from all iterations constitute a new collection of example sets, which is put into the second output of *Retrieve and Split* subprocess. The third operator is of Filter Examples type and is configured to select obfuscated attacks only and then pass it to the fifth output of *Retrieve and Split* subprocess.

The second Join operator in *Retrieve and Split* subprocess has all wrong OL data on its left input and on the right input is connected the data with attack type, services, and label information. The operator is configured to perform left join operation. The joined data is treated the same way as the first Join operator's data as described in the previous paragraph. The only difference is that there are only 2 branches, the one with Sort operator and the one with Split obfuscations Loop operator and Loop Collection operator. Both branches are configured the same as in the case of the previous paragraph, however, obviously, their outputs are connected to different outputs of *Retrieve and Split* subprocess. The sorted data goes from the Sort operator to the third output and the collection of data sets going from the Loop Collection operator is connected to the fourth output of *Retrieve and Split* subprocess.

Loop (O-split-perfs) Collection

This loop iterates over the given collection and in each iteration it multiplies a given set of data into two inputs of the Branch operator. The first input is used for condition evaluation, which is defined by condition type and condition value parameters. Condition type is set to *min_examples* and value is set to 1, thus the Then subprocess of the Branch operator will be performed only if there is at least 1 example in the input data. The Then subprocess passes input data to the Nominal to Binominal operator, which fixes the data for the next connected operator of type Performance (Binominal Classification), which is the same as the Performance operator described at the beginning of Section 8.1.4. The output performance vector is passed to the output of the Branch operator. The Else subprocess just passes its input data to the same output, which is then propagated to the output of whole *Loop (O-split-perfs) Collection*.

Loop, Store and Log

The subprocess begins by preparing necessary macros using Set Macros operator and passing its input to the Loop Collection operator called *Loop and Store*.

Loop and Store operator iterates over a collection of data sets, which were split by obfuscation instances. In each iteration the input data is passed to the inner *Split by label_polyX* subprocess, whose outputs are all connected to outputs of *Loop and Store* operator.

Split by label_polyX subprocess starts by sorting input data by *label_polyX* attribute using Sort operator. The sorted data is then passed through Set Macros operator, which prepares needed macros, to an Execute Script operator that executes a manually programmed script. The script goes through the whole data set and finds all *label_polyX* attribute values, then they are written into *split-label_polyX-list* macro. The script also passes its input data to the output, which is connected to the input of a Loop operator. The Loop operator's outputs are all passed to outputs of *Split by label_polyX*.

The Loop operator includes scripts which are programmed the way to enable it to be performed collaterally. At first, an Execute Script operator performs a script. The script extracts a value from *split-label_polyX-list* macro depending on what the current iteration of the Loop operator is running, and then sets *current-label_polyX* macro. The data is passed to a Filter Examples operator, which selects only data based on prepared *current-label_polyX* macro. Selected data is then sent to another Execute Script operator, which reads the *label_obfus* attribute from the data and set *obfus-label* macro. Next, the data is delivered to the Nominal to Binominal operator, which fixes data for the following operator of type Performance (Binominal Classification), which is the same as the Performance operator described at the start of Section 8.1.4. The consequent performance vector is then saved to an output file using the Store operator, which uses macros mentioned above to name the file. The input data of the Performance operator is also delivered to Store operators, which are configured likewise as the latter one. All the saved data are passed from Store operators to the output of the Loop operator.

All outputs of the *Loop and Store* operator are then saved in files using Store operators.

Analysis Print To Console

Process *2cl-Analysis-PrintToConsole* starts by performing a Set Macros operator, which sets the macro with information about the input folder, which contains files with performance vectors. Then a subprocess called *Work* is run.

Work subprocess launches a Retrieve operator, which reads a special file, which includes information about the folder structure, which is organized per services and obfuscations. The content of the special file is a collection and thus is processed by Flatten Collection operator. Next, the data is sent to Execute Script operator, which creates a set of examples with filenames and passes it to a Loop Values operator. The Loop Values operator then iterates over the filenames and in each iteration sets the current filename to a macro. Inside the loop, a Retrieve operator reads the current file and passes it to Execute Script operator, which parses the file and prints important information from it to the console.

8.2 DOL Models

DOL models are models, which were trained on legitimate traffic, direct attacks, and obfuscated attacks as well, so they were trained on all data, i.e. DOL data. DOL-FFS features were selected from the training data using the Forward Feature Selection algorithm. Cross validation experiment of all data was proceeded (see Section 8.2.1). Other experiments were performed in order to test how resistant the classifiers are against new unknown obfuscation techniques or instances. In each round, one obfuscation instance or technique was chosen to be removed from training data for classifiers, and then after the model was trained, the unknown obfuscation instance was used as testing data. The results of experiments with unknown obfuscation instances are in Section 8.2.2, and the experiment with unknown obfuscation technique is described in Seciton 8.2.3.

8.2.1 DOL Cross Validation

An implementation of this experiment is described in Section 7.5.1. Two calculations were performed in order to test both sets of selected features. The first one in Table 8.3a tested a feature set, which was selected by forward feature selection without knowledge about obfuscated attacks and the second one in Table 8.3b tested a set of features, which were selected using forward feature selection with knowledge about all attacks.

The column with ΔTPR in both mentioned tables signs that the results are the difference between TPR from the current table and the TPR from the table with all attacks classified by DL models in Table 8.1b. The column with ΔFPR means a comparison of the current FPR with Table 7.1, which includes results of DL cross-validation.

Classifier	TPR	FPR	ΔTPR	ΔFPR	F_1 (↑)	Avg. Recall
Random Forest	99.94%	0.43%	6.88%	0.34%	99.68%	99.76%
Decision Tree	99.89%	1.15%	7.58%	0.57%	99.17%	99.37%
Naïve Bayes (Kernels)	97.33%	0.80%	15.59%	0.78%	98.11%	98.27%
Logistic Regression	96.32%	2.06%	81.05%	1.46%	96.76%	97.13%
Support Vector Machine	96.79%	2.48%	36.65%	1.51%	96.73%	97.16%
Naïve Bayes	99.83%	60.56%	0.04%	-0.01%	70.98%	69.64%

(a) DOL Cross Validation with DLFFS features.

Classifier	TPR	FPR	ΔTPR	ΔFPR	F_1 (↑)	Avg. Recall
Random Forest	99.97%	0.11%	6.91%	0.02%	99.91%	99.93%
Decision Tree	99.71%	0.18%	7.40%	-0.40%	99.73%	99.77%
Naïve Bayes (Kernels)	98.76%	0.08%	17.02%	0.06%	99.33%	99.34%
Support Vector Machine	92.07%	3.45%	31.93%	2.48%	93.61%	94.31%
Logistic Regression	87.69%	5.57%	72.42%	4.97%	89.87%	91.06%
Naïve Bayes	97.32%	63.25%	-2.47%	2.68%	68.93%	67.04%

(b) DOL Cross Validation with DOLFFS features.

Table 8.3: DOL Cross Validation.

8.2.2 Single Unknown Obfuscation Instance Detection

Results of the experiment are listed in Table 8.4. Different classifiers were tested in this experiment and therefore each row in the table with results corresponds to one unknown obfuscation instance. At the end of each row, there is an average score of all classifications of the corresponding obfuscation instance calculated. In two last rows, the average score and standard deviation of the related classifier can be found.

The most dangerous unknown obfuscation instances seem to be the ones, which use fragmentation into smaller lengths, unreliable network simulating ones, and the one that uses the normal distribution of packet transmission delay. The least dangerous probably are the ones, which reorder packets and instances that simulate slight differences in fragmentation. Therefore the amount of fragmentation or packet transmission delay is very critical for the instance to be successful.

8.2.3 Single Unknown Obfuscation Technique Detection

The purpose of this experiment is to assess the ability of classifiers to detect unknown obfuscation techniques. This experiment was based on iterating over obfuscation techniques. In each round, one technique was reserved for testing and other ones were used as training data for a classifier.

In this experiment, most of the tested classifiers have worse results than in the experiment with unknown obfuscation instances as expected, because more data was unknown here for the classifiers. Results in this experiment also usually differentiate less than in the one with obfuscation instances, because instances with most deviated score were arranged in their techniques' groups.

As we can see in Table 8.5, where all results from the experiment are stored, the most successful obfuscation techniques are the ones, which use fragmentation, simulation of un-

Instance	Samples Count	Naïve Bayes (Kernels)	Naïve Bayes	Decision Tree	Random Forest	Support Vector Machine	Logistic Regression	Average
(k)	288	94.79%	96.88%	98.96%	100.00%	93.75%	86.46%	95.14%
(i)	303	100.00%	98.35%	98.02%	100.00%	93.07%	80.86%	95.05%
(o)	276	99.64%	97.83%	97.83%	100.00%	92.03%	82.97%	95.05%
(j)	304	100.00%	97.70%	97.70%	100.00%	93.42%	81.25%	95.01%
(l)	281	94.31%	97.87%	100.00%	100.00%	90.75%	86.83%	94.96%
(p)	290	99.66%	98.28%	97.93%	100.00%	92.76%	80.69%	94.89%
(q)	285	99.30%	97.54%	97.19%	100.00%	92.98%	80.00%	94.50%
(b)	18	88.89%	94.44%	100.00%	100.00%	100.00%	83.33%	94.44%
(d)	271	95.57%	95.20%	95.20%	98.89%	87.82%	82.29%	92.50%
(e)	281	97.51%	96.80%	97.87%	100.00%	87.19%	73.31%	92.11%
(h)	320	96.56%	98.13%	96.88%	100.00%	87.81%	73.13%	92.08%
(a)	286	98.25%	97.20%	97.55%	100.00%	84.27%	75.18%	92.07%
(f)	257	94.94%	97.28%	96.89%	99.61%	85.60%	76.27%	91.76%
(m)	281	89.68%	100.00%	92.17%	100.00%	88.97%	78.29%	91.52%
(g)	266	91.73%	95.87%	95.11%	98.87%	85.71%	73.31%	90.10%
(c)	48	72.92%	85.42%	85.42%	100.00%	93.75%	75.00%	85.42%
(n)	284	80.63%	99.30%	92.25%	100.00%	78.52%	59.16%	84.98%
Average		93.79%	96.71%	96.29%	99.85%	89.91%	78.14%	
Std. Dev.		7.40%	3.21%	3.57%	0.38%	4.94%	6.58%	

Table 8.4: Single Unknown Obfuscation Instance

reliable network channel and the ones which combine multiple obfuscation approaches. The least successful and thus the most easily detectable obfuscation techniques are the ones, which use packets' loss simulation, packets' duplication, and time delay of the packet transmission.

8.2.4 Implementation of Unknown Obfuscation Instances and Techniques

Evaluation of the obfuscated attacks which differentiates instances and techniques is implemented in *2cl-TrainPrediction-inOL-attDOLFFS-perInstance* and *2cl-TrainPrediction-inOL-attDOLFFS-perTechnique* processes. From the names of those processes can be determined that they are designed for binary classification, the input data are expected to be obfuscated attacks and legitimate traffic and attributes are limited to DOL-FFS only. Both processes are very similar, thus they are described together and all differences are explicitly mentioned.

At first, the input data, which was produced by *Dataset-Repairment-B* process described in Section 7.2.2, is read using a Retrieve operator. Then in case of use of SVM or LR operators Nominal to Numerical and Normalize operators prepare the read data, which are the same as the ones described in Section 7.4.2. Read data is then passed to *Prepare OL* subprocess (see Section 7.4.2), which is delivers it to the Generate Attributes operator, which adds new attribute called *label_obfus* and delivers the data to the Select attributes operator. The Select attributes operator filters just attributes that include information about attack types and services. Selected data is connected to the right input of the Join operator and on the left input is connected OL12 data, which is read by another Retrieve operator. Therefore the Join operator, which is configured to perform a left join operation, takes obfuscated attacks and legitimate traffic from the left input and adds information

Technique	Samples Count	Naïve Bayes (Kernels)	Naïve Bayes	Decision Tree	Random Forest	Support Vector Machine	Logistic Regression	Average
(d)	271	95.57%	95.20%	95.20%	98.52%	87.82%	82.29%	92.44%
(h)	320	96.56%	98.13%	96.88%	100.00%	87.81%	73.13%	92.08%
(abc)	352	92.33%	95.74%	95.46%	100.00%	85.23%	75.57%	90.72%
(ij)	607	100.00%	98.02%	71.66%	100.00%	93.25%	80.73%	90.61%
(opq)	851	99.30%	98.12%	73.91%	100.00%	92.60%	77.67%	90.27%
(efg)	804	90.92%	96.52%	96.77%	99.75%	83.83%	70.15%	89.66%
(klmn)	1134	80.60%	98.68%	70.19%	99.12%	85.27%	70.90%	84.13%
Average		93.61%	97.20%	85.72%	99.63%	87.97%	75.78%	
Std. Dev.		6.63%	1.36%	12.97%	0.59%	3.68%	4.71%	

Table 8.5: Single Unknown Obfuscation Technique

about which obfuscation was used and what service was attacked from the right input. Then the data is prepared using Reorder Attributes and Nominal to Binominal operators for the binominal performance evaluation and are sent to the next operator of type Loop.

In the case of training and prediction per instance process, the Loop operator is called *Predict per Instance* and has 17 iterations set in its parameters, because there are 17 obfuscation instances. However, in the case of a process, that trains and predicts per technique, the operator is called *Predict per Technique* and is configured for just 7 iterations, because there are just 7 obfuscation techniques. Both variants of the Loop operator are quite similar, so they are described together with the same as the processes.

Predict per Instance/Technique

Each iteration of the loop starts with 2 Set Macros operators, which prepare necessary macros, and then 2 most important operators, which constitute the core of the process proceed. The first one is a Generate Macro operator, that reads the iteration macro, uses it to determine which obfuscation should be selected in this iteration, and writes it to ob macro in case of prediction per instance. In so far as prediction per technique is concerned the Generate Macro operator sets 4 macros with information about what instances are present in the current technique. The second core operator is Filter Examples operator, which uses generated macros in order to select the right instance or instances of obfuscation attacks for the current iteration and also sends all unmatched examples to its second output. Thanks to this design it is possible to run the whole loop collaterally.

Both matched and unmatched data are then saved in files using Store operators and then passed to Select Attributes operators, which remove excrescence labels about services and attack types, which could also corrupt learning and prediction procedures. The unmatched data, which do not include current obfuscation instances are passed to the model generating operator, which trains a model on them. Next, the model is stored using a Store operator and then delivered to the Apply Model operator. The matched data is connected to the Apply Model as well, so the model is tested on them. Resulting labeled data is then sent to Performance (Binominal Classification) operator, which calculates criterions described in Section 7.4.3. Then the produced performance vector is saved to a file using Store operator, and in the end, it is delivered to the Execute Script operator, which performs a script that extracts critical information from the data and macros and prints results.

Classifier	TPR	FPR	F_1 (↑)
Decision Tree	86.41%	5.54%	61.72%
Random Forest	63.28%	3.24%	58.06%
Naïve Bayes (Kernels)	54.37%	3.02%	52.97%
Naïve Bayes	62.81%	9.40%	39.07%

(a) Classifiers trained on the novel dataset in Section 6 tested on ASNM-NPBO-v1 dataset [39].

Classifier	TPR	FPR	F_1 (↑)
Naïve Bayes (Kernels)	37.57%	13.35%	48.31%
Naïve Bayes	28.61%	11.46%	39.73%
Decision Tree	24.67%	2.16%	38.67%
Random Forest	9.20%	0.06%	16.83%

(b) Classifiers trained on ASNM-NPBO-v1 dataset [39] tested on the novel dataset in Section 6.

Table 8.6: Cross-Dataset Evaluation.

8.3 Cross-Dataset Evaluation

In Cross-Dataset evaluation the novel dataset (described in Section 6) and the state-of-the-art dataset ASNM-NPBO-v1 [39] are cross evaluated. The Evaluation process consists of training a classifier on the first dataset and then testing in on the second one and vice versa. Cross-dataset evaludation experiments implementation is described in Section 8.3.1. All data including obfuscated attacks were used in this experiment and the result can be found in Table 8.6. There are significant differences between models of the same classifiers trained on different datasets, for instance, Decision Tree reaches much better TPR in the case of testing on ASNM-NPBO-v1 dataset than in the case of doing it vice versa. However, in the case of Decision Tree tested on the novel dataset, the classifier's FPR is significantly better. After sorting the results, no classifier ended on the same "rank" in both tables. An interesting fact is also that classifiers trained on the novel dataset performed better than classifiers trained on the ASNM-NPBO-v1 dataset. The reason might be the fact novel dataset contains more vulnerabilities than the ASNM-NPBO-v1 dataset.

8.3.1 Cross-Dataset Evaluation Implementation

Cross-dataset evaluation is implemented in *2cl-X-Dataset-Evaluation* process. The process starts by reading input datasets with DOL123 data using two Retrieve operators. Then a file with DOLFFS attributes selected from the novel dataset is read using another Retrieve operator. After it is done, the attributes are filtered from the dataset table using Select by Weights operator, which uses the dataset table and attribute table as its inputs. DOLFFS attributes selected from ASNM-NPBO-v1 dataset [39] are filtered using the Select Attributes operator, which includes information about the attributes. There are also two more DOLFFS filtering operators, which are the same as above, but they filter different datasets than attributes. Datasets and attributes are delivered using three Multiply operators. Therefore four branches with data are prepared as a result of previous operations.

Next in the first branch (with DOLFFS and DOL123 from the novel dataset) are performed two Set Macros operators, which prepare macros with model, data, and folders information. All four branches continue by passing their data to Prepare DOL112 attFFS subprocesses, which are all the same. Prepare DOL112 attFFS subprocess consists of two Replace operators only, which change values of label attribute from 2 to 1 (obfuscated attacks get label 1 same as direct attacks) and 3 to 2 (legitimate traffic gets label 2). Then Nominal to Binominal operators fix label data types for binary classifiers.

The branches, which include data with attributes both from the same dataset are delivered using the Multiply operators to two new branches for each. One of each two subbranches passes the data to the model generating operator and the second passes the data to the Apply Model operator, which got a model from the model generating operator. Both trained models are delivered using the Multiply operators to two Apply Model operators, the mentioned ones, which apply the models to their training data and the ones that apply models to data from different datasets. All labeled data from each Apply Model operator are passed to Performance (Binominal Classification) operators, which are configured the same as described in Section 7.4.3. All calculated performance vectors are then saved in output files using Store operators, and the same is done with trained models. The labeled data is saved as well, and then it is passed to Filter Examples operators, which are configured to filter wrong classified examples only, which are then stored too. In the end, the data is analyzed using the process described in Section 8.1.4.

8.3.2 Cross-Dataset Evaluation per Service

In Table 8.7 with obfuscated attacks detected by classifiers trained on the novel dataset in Section 6 tested on ASNM-NPBO-v1 dataset [39] per services can be seen that Apache, Samba and DistCC services were detected much more easily than other ones. An average TPR of mentioned services ranges from 88.97% to 96.76%. The second group consists of Server and PostgreSQL, which were not detected as well as in the first case, but the average TPR is significantly influenced by the Naïve Bayes (Kernels), which classified them hardly with 4.9% and 8.04% only success. The toughest problem for classifiers is obfuscated attacked targeted to MSSQL. The most successful classifier is Decision Tree with 93.82% of detected obfuscated attacks.

The second table presents the results of obfuscated attacks detected by classifiers trained on the ASNM-NPBO-v1 dataset [39] tested on the novel dataset in Section 6 per services. Most services ranged from 15.3% to 47.39% detection success on average. There are two obfuscated service attacks hard to detect Confluence and FTPShell. The interesting fact about FTPShell is that the client application is attacked by the server, maybe that is the reason no attacks targeted to FTPShell was detected because it phenomenally differentiates from other attacks. The best performance was measured in Gitstack detection with 70.38%, where was especially successful Naïve Bayes (Kernels) classifier with 91.33% detected obfuscated attacks.

8.3.3 Cross-Dataset Evaluation per Obfuscation Instance

Obfuscated attacks detected by classifiers trained on the novel dataset in Section 6 tested on ASNM-NPBO-v1 dataset [39] per obfuscation instances are listed in Table 8.9. The results per instance on average range from 54.54% to 82.22%. The easiest to detect are obfuscation instances that modify packets' order or duplicate packets. The classifiers struggled to detect instances, which delayed the communication, fragmented packets into low size and instances that combined miscellaneous obfuscation techniques.

Service	**Samples Count**	**Naïve Bayes (Kernels)**	**Naïve Bayes**	**Decision Tree**	**Random Forest**	**Average**
Apache	163	91.79%	96.52%	98.75%	100.00%	96.76%
Samba	44	84.80%	95.10%	94.12%	94.12%	92.03%
DistCC	23	61.76%	94.12%	100.00%	100.00%	88.97%
PostgreSQL	45	4.90%	66.67%	100.00%	77.45%	62.25%
Server	100	8.04%	64.12%	99.02%	62.75%	58.48%
MSSQL	103	48.88%	0.00%	71.01%	1.82%	30.43%
Average		50.03%	69.42%	93.82%	72.69%	
Std. Dev.		37.13%	37.04%	11.39%	37.67%	

Table 8.7: Obfuscated attacks detected by classifiers trained on the novel dataset in Section 6 and tested on ASNM-NPBO-v1 dataset [39] per services.

Service	**Samples Count**	**Naïve Bayes (Kernels)**	**Naïve Bayes**	**Decision Tree**	**Random Forest**	**Average**
Gitstack	398	91.33%	63.86%	38.75%	11.26%	70.38%
rConfig	232	15.47%	95.56%	13.75%	7.16%	47.39%
Drupal	399	41.34%	34.66%	82.29%	19.37%	45.60%
LibreNMS	368	47.24%	33.68%	38.57%	35.02%	39.20%
Nagiosb	145	18.00%	63.33%	8.67%	0.00%	35.65%
Webmin	276	17.97%	31.12%	33.40%	8.19%	26.18%
Nagiosa	657	36.26%	21.89%	8.86%	7.08%	25.94%
GetSimple	1173	37.08%	13.70%	19.61%	2.79%	24.32%
jQuery-File-Upload	318	33.83%	0.00%	13.10%	0.00%	15.30%
Confluence	275	3.01%	0.00%	0.00%	0.00%	1.24%
FTPShell	98	0.00%	0.00%	0.00%	0.00%	0.00%
Average		31.05%	32.53%	23.36%	8.26%	
Std. Dev.		33.06%	25.95%	31.27%	8.05%	

Table 8.8: Obfuscated attacks detected by classifiers trained on ASNM-NPBO-v1 dataset [39] tested on the novel dataset in Section 6 per services

Instance	**Samples Count**	**Naïve Bayes (Kernels)**	**Naïve Bayes**	**Decision Tree**	**Random Forest**	**Average**
(i)	27	65.00%	83.33%	100.00%	80.56%	82.22%
(h)	30	63.89%	83.33%	100.00%	80.56%	81.94%
(j)	27	50.00%	83.33%	100.00%	83.33%	79.17%
(k)	27	52.78%	83.33%	100.00%	80.56%	79.17%
(g)	26	62.78%	73.33%	94.44%	83.33%	78.47%
(l)	27	52.78%	83.33%	100.00%	77.78%	78.47%
(e)	26	66.67%	77.78%	100.00%	63.89%	77.08%
(o)	28	57.41%	83.33%	83.33%	83.33%	76.85%
(m)	27	49.44%	83.33%	100.00%	66.67%	74.86%
(d)	30	44.44%	72.22%	97.22%	77.78%	72.92%
(a)	28	61.11%	66.67%	83.33%	75.00%	71.53%
(f)	28	40.00%	60.56%	88.89%	77.78%	66.81%
(p)	33	40.15%	48.48%	88.89%	66.67%	61.05%
(q)	35	40.28%	38.89%	91.67%	69.44%	60.07%
(n)	27	36.11%	83.33%	77.22%	33.33%	57.50%
(b)	22	26.67%	40.00%	96.67%	60.00%	55.83%
(c)	30	30.16%	26.11%	92.86%	69.05%	54.54%
Average		49.39%	68.86%	93.80%	72.30%	
Std. Dev.		12.50%	19.09%	7.23%	12.47%	

Table 8.9: Obfuscated attacks detected by classifiers trained on the novel dataset in Section 6 and tested on ASNM-NPBO-v1 dataset [39] per obfuscation instances.

In Table 8.10 there are obfuscated attacks detected by classifiers trained on ASNM-NPBO-v1 dataset [39] tested on the novel dataset in Section 6 per obfuscation instances. Most instances' scores ranged from 29.23% to 32.36%, however, there were significant differences between individual classifiers. Most attacks evaded using obfuscation instances that used the normal distribution of packets transmission delay, packets' duplication, and the most extreme fragmentation technique instance, which fragmented the communication into the smallest data objects.

Instance	**Samples Count**	**Naïve Bayes (Kernels)**	**Naïve Bayes**	**Decision Tree**	**Random Forest**	**Average**
(m)	281	32.13%	32.60%	16.72%	4.55%	32.36%
(b)	18	58.75%	25.00%	12.50%	6.25%	32.08%
(e)	281	33.54%	30.61%	26.17%	13.25%	32.08%
(q)	285	30.24%	33.45%	25.50%	8.91%	31.85%
(g)	266	37.35%	31.37%	26.59%	13.80%	31.77%
(p)	290	31.03%	32.11%	25.01%	9.66%	31.57%
(i)	303	31.74%	33.72%	25.76%	9.65%	31.31%
(l)	281	34.64%	32.59%	16.96%	5.68%	31.17%
(o)	276	30.02%	32.23%	24.68%	9.41%	31.13%
(f)	257	33.82%	31.27%	27.81%	14.78%	31.10%
(d)	271	34.36%	31.38%	25.04%	11.09%	30.26%
(k)	288	32.16%	33.40%	24.32%	4.17%	29.96%
(a)	286	36.57%	26.14%	26.01%	6.47%	29.57%
(h)	320	31.32%	31.24%	25.19%	7.63%	29.25%
(j)	304	32.12%	32.52%	25.20%	9.88%	29.23%
(n)	284	23.25%	30.71%	29.55%	4.55%	26.98%
(c)	48	20.54%	23.61%	18.55%	4.17%	20.90%
Average		33.15%	30.82%	23.62%	8.46%	
Std. Dev.		7.81%	3.00%	4.57%	3.44%	

Table 8.10: Obfuscated attacks detected by classifiers trained on ASNM-NPBO-v1 dataset [39] and tested on the novel dataset in Section 6 per obfuscation instances.

Chapter 9

Conclusion

In the first part of this work, we focus on the existing taxonomies of intrusion detection systems. Articles that describe network intrusion systems taxonomy differ in many cases [45, 20, 30]. Thus it was necessary to split descriptions by the origin of the description, in cases of those which differentiated from each other, for example, anomaly-based detection principles. Based on fundamental characteristics the taxonomy was united and extended from mentioned works.

Adversarial attacks were divided by phases of the intrusion detection system which is considered as the target of these attacks. The division is based on a taxonomy by Igino Corona et al. [28], where his general description of intrusion detection system architecture consists of three parts: *event generators*, *event analyzers*, and *response units*, where each part represents one functionality phase.

The taxonomy of attacks against classification-based intrusion detection systems consists of three main types of attacks based on their objective as described in [25]. The first of them is *exploratory attack*, which is designed to gain as much information as possible from the attacked system. There is also *evasion type* of attacks, whose objective is to intrude the system, and its tactic is based on evasion of the intrusion classifier. The last type of attack is *poisoning attacks*, which tries to contaminate the training data set of the target system in order to manipulate its recognition capabilities.

In the second part of our work, we focus on evasion attacks performed using Non-Payload-based Obfuscations. In detail, we start by the description of the Non-Payload-based Obfuscation framework [42] and Advanced Security Network Metrics [39]. The framework provides an ability to obfuscate exploits in order to evade detection of the target intrusion detection system. A big advantage of this framework is that it is working in an exploit-independent way, thus it is able to obfuscate given attack without the need for manual modification of it. Experiments in [42] showed that by adding obfuscated exploits into training datasets for the classifier of the IDS, the performance of other obfuscated attacks detection of such trained IDS was improved. However, these results were obtained using outdated vulnerabilities and they were not proven for the recent vulnerabilities and techniques of targeted attacks, which is the goal of this work.

In order to develop, test, and improve classification-based intrusion detection systems, the novel dataset was created in this work. The dataset consists of ASNM features extracted from records of legitimate traffic, direct attacks, and obfuscated attacks, which were targeted against 11 vulnerable services. All vulnerabilities were found in the National Vulnerability Database [55], where they can be identified as Common Vulnerabilities and

Exposures (CVE). CVEs in the dataset were published in 2018 and 2019. All exploits used to attack the vulnerabilities were downloaded from the Exploit Database by Offensive Security [56].

In our experiments, Forward Feature Selection was used in order to select the best ASNM features for attack detection. Six classifiers were tested in this thesis. For example, the performance of Naïve Bayes with Kernel Density Estimation classifier, which had knowledge about direct attacks and legitimate traffic only, achieved 99.34% true positive rate (TPR) of detecting direct attacks in cross-validation experiment, while the false-positive rate was very low (i.e., 0.02%). Then obfuscated attacks were classified by the mentioned classifier resulting in 70.02% TPR, which worse by 29.32%. Therefore, the obfuscated attacks successfully evaded the classification process. The improvement of the classifier was accomplished by widening its knowledge of obfuscated attacks, which were added into training data. The classifier achieved 98.76% TPR in cross-validation over the whole dataset, which is 17.02% better than the score of classification of all attacks using classifier without knowledge about obfuscated attacks, while TPR was deteriorated only slightly (i.e., by 0.06%).

Next, the detection capability of classifiers to unknown obfuscation instance was tested. For example, the Naïve Bayes with Kernel Density Estimation classifier was on average able to detect 93.79% of obfuscated attacks, which were obfuscated using obfuscation instance unknown for the classifier. A similar experiment was performed with obfuscation techniques, where the classifier scored 93.61% TPR in the case of detecting attacks obfuscated using an unknown obfuscation technique.

Finally, a cross-dataset evaluation was performed with the novel dataset and ASNM-NPBO-v1 dataset [39]. The Naïve Bayes with Kernel Density Estimation classifier trained on the novel dataset achieved 54.37% TPR and 3.02% FPR on ASNM-NPBO-v1 dataset attacks detection. In the case of the same classifier trained on the ASNM-NPBO-v1 dataset and validated on the novel dataset, the TPR achieved was equal to 37.57% while FPR was equal to 13.35%. The best classifier was Decision Tree trained on the novel dataset, which resulted in 86.41% TPR and 5.54% FPR score when doing validation on the ASNM-NPBO-v1 dataset. In sum, the “backward” detection achieved better results than the forward detection. This indicates the importance of retraining the classifiers with the novel datasets and techniques used in the contemporary attacks, while classifiers trained using the old vulnerabilities are more susceptible to targeted attacks with obfuscations.

In future work, the research might focus on widening training datasets and comparing attacks focused on different vulnerable services, because in the cross-dataset evaluation significant differences between detection ability of unknown attacks focusing on various services were discovered. For instance, classifiers trained on the novel dataset detected 96.76% Apache attacks on average and just 30.43% MSSQL attacks.

Bibliography

[1] *Curl(1) Linux User's Manual.* August 2014.

[2] *CVE-2018-15708.* [Available from MITRE, CVE-ID CVE-2018-15708.]. 2018 [cit. 2020-04-18]. Available at: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-15708.

[3] *CVE-2018-15710.* [Available from MITRE, CVE-ID CVE-2018-15710.]. 2018 [cit. 2020-04-19]. Available at: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-15710.

[4] *CVE-2018-20434.* [Available from MITRE, CVE-ID CVE-2018-20434.]. 2018 [cit. 2020-04-18]. Available at: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-20434.

[5] *CVE-2018-5955.* [Available from MITRE, CVE-ID CVE-2018-5955.]. 2018 [cit. 2020-04-18]. Available at: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-5955.

[6] *CVE-2018-7573.* [Available from MITRE, CVE-ID CVE-2018-7573.]. 2018 [cit. 2020-04-17]. Available at: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-7573.

[7] *CVE-2018-7602.* [Available from MITRE, CVE-ID CVE-2018-7602.]. 2018 [cit. 2020-04-17]. Available at: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-7602.

[8] *CVE-2018-8733.* [Available from MITRE, CVE-ID CVE-2018-8733.]. 2018 [cit. 2020-04-18]. Available at: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-8733.

[9] *CVE-2018-8734.* [Available from MITRE, CVE-ID CVE-2018-8734.]. 2018 [cit. 2020-04-18]. Available at: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-8734.

[10] *CVE-2018-8735.* [Available from MITRE, CVE-ID CVE-2018-8735.]. 2018 [cit. 2020-04-18]. Available at: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-8735.

[11] *CVE-2018-8736.* [Available from MITRE, CVE-ID CVE-2018-8736.]. 2018 [cit. 2020-04-18]. Available at: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-8736.

[12] *CVE-2018-9206.* [Available from MITRE, CVE-ID CVE-2018-9206.]. 2018 [cit. 2020-04-18]. Available at: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-9206.

[13] *CVE-2019-11231.* [Available from MITRE, CVE-ID CVE-2019-11231.]. 2019 [cit. 2020-04-18]. Available at: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-11231.

[14] *CVE-2019-15107.* [Available from MITRE, CVE-ID CVE-2019-15107.]. 2019 [cit. 2020-04-18]. Available at: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-15107.

[15] *CVE-2019-16662.* [Available from MITRE, CVE-ID CVE-2019-16662.]. 2019 [cit. 2020-04-18]. Available at: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-16662.

[16] *CVE-2019-3396.* [Available from MITRE, CVE-ID CVE-2019-3396.]. 2019 [cit. 2020-04-18]. Available at: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-3396.

[17] Armstrong, A. and LibreNMS contributors. *LibreNMS* [online]. 2020 [cit. 2020-04-18]. Available at: https://www.librenms.org/.

[18] Askar. *RConfig v3.9.2 authenticated and unauthenticated RCE (CVE-2019-16663) and (CVE-2019-16662)* [online]. 2019 [cit. 2020-04-19]. Available at: https://shells.systems/rconfig-v3-9-2-authenticated-and-unauthenticated-rce-cve-2019-16663-and-cve-2019-16662/.

[19] Atlassian. *Confluence* [online]. 2020 [cit. 2020-04-18]. Available at: https://www.atlassian.com/software/confluence.

[20] Axelsson, S. *Intrusion detection systems: A survey and taxonomy.* Technical report, 2000.

[21] Barreno, M., Nelson, B., Joseph, A. D. and Tygar, J. D. The security of machine learning. *Machine Learning.* Springer. 2010, vol. 81, no. 2, p. 121–148.

[22] Biggio, B., Nelson, B. and Laskov, P. Support vector machines under adversarial label noise. In: *Asian Conference on Machine Learning.* 2011, p. 97–112.

[23] Blaklis. *Drupal < 7.58 - 'Drupalgeddon3' (Authenticated) Remote Code Execution (PoC)* [online]. 2018 [cit. 2020-04-17]. Available at: https://www.exploit-db.com/exploits/44542.

[24] Buytaert, D. and Drupal Community. *Drupal About* [online]. 2020 [cit. 2020-04-17]. Available at: https://www.drupal.org/about.

[25] Chakraborty, A., Alam, M., Dey, V., Chattopadhyay, A. and Mukhopadhyay, D. Adversarial attacks and defences: A survey. *ArXiv preprint arXiv:1810.00069.* 2018.

[26] Chybeta. *SSTI and RCE in Confluence Server via Widget Connector* [online]. 2019 [cit. 2020-04-19]. Available at: https://chybeta.github.io/2019/04/06/Analysis-for-%E3%80%90CVE-2019-3396%E3%80%91-SSTI-and-RCE-in-Confluence-Server-via-Widget-Connector/.

[27] CMS, G. *GetSimple CMS* [online]. 2020 [cit. 2020-04-18]. Available at: http://get-simple.info/.

[28] Corona, I., Giacinto, G. and Roli, F. Adversarial attacks against intrusion detection systems: Taxonomy, solutions and open issues. *Information Sciences*. Elsevier. 2013, vol. 239, p. 201–225.

[29] Datanyze. *Confluence Market Share* [online]. 2020 [cit. 2020-04-20]. Available at: http://web.archive.org/save/https://www.datanyze.com/market-share/team-collaboration--267/confluence-market-share.

[30] Debar, H., Dacier, M. and Wespi, A. A revised taxonomy for intrusion-detection systems. In: Springer. *Annales des télécommunications*. 2000, vol. 55, 7-8, p. 361–378.

[31] Dmitriev, D. and Shchannikov, D. r. *Atlassian Confluence Widget Connector Macro Velocity Template Injection* [online]. 2019 [cit. 2020-04-18]. Available at: https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/multi/http/confluence_widget_connector.rb.

[32] Enterprises, N. *Nagios XI* [online]. 2020 [cit. 2020-04-18]. Available at: https://www.nagios.com/products/nagios-xi/.

[33] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L. et al. *Hypertext Transfer Protocol – HTTP/1.1* [Internet Requests for Comments]. RFC 2616. RFC Editor, June 1999. Available at: https://tools.ietf.org/html/rfc2616.

[34] Ford Hutchinson, P. *Securing FTP with TLS* [Internet Requests for Comments]. RFC 4217. RFC Editor, Oct 2005. Available at: https://tools.ietf.org/html/rfc4217.

[35] Git community. *Git* [online]. 2020 [cit. 2020-05-01]. Available at: https://git-scm.com/.

[36] Homoliak, I., Barabas, M., Chmelar, P., Drozd, M. and Hanacek, P. ASNM: Advanced Security Network Metrics for Attack Vector Description. In: *Conference on Security & Management*. 2013, p. 350–358. ISBN 1-60132-259-3.

[37] Homoliak, I. *Metrics for Intrusion Detection in Network Traffic*. 2011. Master's thesis. University of Technology Brno, Faculty of Information Technology, Department of Intelligent Systems. In Slovak Language.

[38] Homoliak, I. *Intrusion Detection in Network Traffic*. 2016. Dissertation. Faculty of Information Technology, University of Technology Brno.

[39] Homoliak, I. and Hanacek, P. ASNM Datasets: A Collection of Network Traffic Features for Testing of Adversarial Classifiers and Network Intrusion Detectors. *ArXiv preprint arXiv:1910.10528*. 2019.

[40] Homoliak, I., Ovsonka, D., Gregr, M. and Hanacek, P. NBA of obfuscated network vulnerabilities' exploitation hidden into HTTPS traffic. In: IEEE. *The 9th International Conference for Internet Technology and Secured Transactions (ICITST-2014).* 2014, p. 310–317.

[41] Homoliak, I., Teknos, M., Barabas, M. and Hanacek, P. Exploitation of NetEm Utility for Non-payload-based Obfuscation Techniques Improving Network Anomaly Detection. In: Springer. *International Conference on Security and Privacy in Communication Systems.* 2016, p. 770–773.

[42] Homoliak, I., Teknos, M., Ochoa, M., Breitenbacher, D., Hosseini, S. et al. Improving Network Intrusion Detection Classifiers by Non-payload-Based Exploit-Independent Obfuscations: An Adversarial Approach. *ICST Transactions on Security and Safety.* European Alliance for Innovation. Jan 2019, vol. 5, no. 17, p. 156245. DOI: 10.4108/eai.10-1-2019.156245. ISSN 2032-9393. Available at: http://dx.doi.org/10.4108/eai.10-1-2019.156245.

[43] Husted, B., Arave, J. and Smith, C. *NagiosXI remote root vulnerability CVE-2018-8733, CVE-2018-8734, CVE-2018-8735, CVE-2018-8736* [online]. 2018 [cit. 2020-04-19]. Available at: https://gist.github.com/caleBot/f0a93b5a98574393e0139104eacc2d0f.

[44] Kuwatly, I., Sraj, M., Al Masri, Z. and Artail, H. A dynamic honeypot design for intrusion detection. In: IEEE. *The IEEE/ACS International Conference onPervasive Services, 2004. ICPS 2004. Proceedings.* 2004, p. 95–104.

[45] Lim, S. Y. and Jones, A. Network anomaly detection system: The state of art of network behaviour analysis. In: IEEE. *2008 International Conference on Convergence and Hybrid Information Technology.* 2008, p. 459–465.

[46] Lyne, C. *[R2] Nagios XI Multiple Vulnerabilities* [online]. 2018 [cit. 2020-04-19]. Available at: https://www.tenable.com/security/research/tra-2018-37.

[47] Lyne, C. and André, G. *Nagios XI Magpie debug.php Root Remote Code Execution* [online]. 2018 [cit. 2020-04-18]. Available at: https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/linux/http/nagios_xi_magpie_debug.rb.

[48] mhaskar and bcoles. *RConfig install Command Execution* [online]. 2019 [cit. 2020-04-18]. Available at: https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/unix/webapp/rconfig_install_cmd_exec.rb.

[49] mhaskar and Pace, S. *LibreNMS addhost Command Injection* [online]. 2018 [cit. 2020-04-18]. Available at: https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/linux/http/librenms_addhost_cmd_inject.rb.

[50] Microsoft. *Authenticate Machine Learning Server users against LDAP AD or Azure Active Directory* [online]. 2020 [cit. 2020-05-01]. Available at: https://docs.microsoft.com/en-us/machine-learning-server/operationalize/configure-authentication.

[51] Microsoft. *Microsoft NTLM* [online]. 2020 [cit. 2020-05-01]. Available at: https://docs.microsoft.com/en-us/windows/win32/secauthn/microsoft-ntlm.

[52] msysgit. *Msysgit* [online]. 2020 [cit. 2020-05-01]. Available at: https://github.com/msysgit/msysgit.

[53] Mustafa Akkuş Özkan. *Webmin <= 1.920 - Unauthenticated RCE* [online]. 2019 [cit. 2020-04-19]. Available at: https://pentest.com.tr/exploits/DEFCON-Webmin-1920-Unauthenticated-Remote-Command-Execution.html.

[54] Mustafa Akkuş Özkan. *Webmin 1.920 Unauthenticated RCE* [online]. 2019 [cit. 2020-04-18]. Available at: https://www.exploit-db.com/exploits/47230.

[55] National Institute of Standards and Technology. *National Vulnerability Database* [online]. 2020 [cit. 2020-05-02]. Available at: https://nvd.nist.gov/vuln/data-feeds.

[56] OffSec Services Limited. *Exploit Database by Offensive Security* [online]. 2020 [cit. 2020-05-08]. Available at: https://www.exploit-db.com/.

[57] OffSec Services Limited. *Kali Linux* [online]. 2020 [cit. 2020-05-01]. Available at: https://www.kali.org/.

[58] Oracle Corporation. *Oracle VM VirtualBox* [online]. 2020 [cit. 2020-05-01]. Available at: https://www.virtualbox.org/.

[59] Oracle Corporation. *Oracle VM VirtualBox User Manual* [online]. 2020 [cit. 2020-05-01]. Available at: https://www.virtualbox.org/manual/ch06.html.

[60] OWASP Foundation, Inc.. *Cross Site Request Forgery (CSRF)* [online]. 2020 [cit. 2020-05-01]. Available at: https://owasp.org/www-community/attacks/csrf.

[61] Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z. B. et al. Practical black-box attacks against machine learning. In: ACM. *Proceedings of the 2017 ACM on Asia conference on computer and communications security*. 2017, p. 506–519.

[62] Papernot, N., McDaniel, P., Wu, X., Jha, S. and Swami, A. Distillation as a defense to adversarial perturbations against deep neural networks. In: IEEE. *2016 IEEE Symposium on Security and Privacy (SP)*. 2016, p. 582–597.

[63] Postel, J. and Reynolds, J. *FILE TRANSFER PROTOCOL (FTP)* [Internet Requests for Comments]. RFC 959. RFC Editor, Oct 1985. Available at: https://tools.ietf.org/html/rfc959.

[64] Q Success. *W3techs Usage statistics of content management systems* [online]. 2020 [cit. 2020-04-17]. Available at: http://web.archive.org/web/20200416002708/https://w3techs.com/technologies/overview/content_management.

[65] r4wd3r and Teixeira, D. *FTPShell client 6.70 (Enterprise edition) Stack Buffer Overflow* [online]. 2018 [cit. 2020-04-17]. Available at: https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/windows/ftp/ftpshell_cli_bof.rb.

[66] Rapid7. *Metasploit* [online]. 2020 [cit. 2020-05-01]. Available at: https://www.metasploit.com/.

[67] RapidMiner. *RapidMiner Studio Manual* [online]. 2014 [cit. 2020-05-14]. Available at: https://docs.rapidminer.com/downloads/RapidMiner-v6-user-manual.pdf.

[68] RapidMiner GmbH. *RapidMiner 9 Operator Reference Manual* [online]. 2019 [cit. 2020-05-14]. Available at: https://docs.rapidminer.com/latest/studio/operators/rapidminer-studio-operator-reference.pdf.

[69] RapidMiner GmbH. *RapidMiner Documentation* [online]. 2020 [cit. 2020-05-14]. Available at: https://docs.rapidminer.com/.

[70] RapidMiner, Inc.. *RapidMiner* [online]. 2020 [cit. 2020-05-02]. Available at: https://rapidminer.com/.

[71] rConfig. *RConfig Network Management* [online]. 2020 [cit. 2020-04-18]. Available at: https://www.rconfig.com/.

[72] Shafranovich, Y. *Common Format and MIME Type for Comma-Separated Values (CSV) Files* [Internet Requests for Comments]. RFC 4180. RFC Editor, Oct 2005. Available at: https://tools.ietf.org/html/rfc4180.

[73] SixP4ck3r. *Drupal < 7.58 - 'Drupalgeddon3' (Authenticated) Remote Code (Metasploit)* [online]. 2018 [cit. 2020-04-17]. Available at: https://www.exploit-db.com/exploits/44557.

[74] Smart Mobile Software. *GitStack* [online]. 2020 [cit. 2020-04-18]. Available at: https://gitstack.com/.

[75] Smith, C., Husted, B. and Arave, J. *Nagios XI Chained Remote Code Execution* [online]. 2018 [cit. 2020-04-18]. Available at: https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/linux/http/nagios_xi_chained_rce_2_electric_boogaloo.rb.

[76] Szurek, K. *GitStack 2.3.10 Unauthenticated Remote Code Execution* [online]. 2018 [cit. 2020-04-18]. Available at: https://security.szurek.pl/en/gitstack-2310-unauthenticated-rce.html.

[77] Szurek, K. and Robles, J. *GitStack Unsanitized Argument RCE* [online]. 2018 [cit. 2020-04-18]. Available at: https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/windows/http/gitstack_rce.rb.

[78] The Apache Software Foundation. *The Apache HTTP Server Project* [online]. 2020 [cit. 2020-05-01]. Available at: https://httpd.apache.org/.

[79] The Apache Software Foundation. *The Apache Velocity Project* [online]. 2020 [cit. 2020-05-01]. Available at: http://velocity.apache.org/.

[80] The PostgreSQL Global Development Group. *PostgreSQL* [online]. 2020 [cit. 2020-05-02]. Available at: https://www.postgresql.org/.

[81] The Tcpdump Group and MartinGarcia, L. *Tcpdump and libcap* [online]. 2020 [cit. 2020-05-02]. Available at: https://www.tcpdump.org/.

[82] TRUERAND0M. *GetSimpleCMS Unauthenticated RCE* [online]. 2019 [cit. 2020-04-18]. Available at: https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/multi/http/getsimplecms_unauth_code_exec.rb.

[83] TRUERAND0M. *SSD Advisory – GetSimple CMS Unauthenticated Remote Code Execution* [online]. 2019 [cit. 2020-04-19]. Available at: https://ssd-disclosure.com/ssd-advisory-getcms-unauthenticated-remote-code-execution/.

[84] TSCHAN, S. *JQuery File Upload* [online]. 2020 [cit. 2020-04-18]. Available at: https://github.com/blueimp/jQuery-File-Upload.

[85] VIVIANI, C., CASHDOLLAR, L. W. and WVU. *Blueimp's jQuery (Arbitrary) File Upload* [online]. 2018 [cit. 2020-04-18]. Available at: https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/unix/webapp/jquery_file_upload.rb.

[86] WEBMIN. *Webmin* [online]. 2020 [cit. 2020-04-18]. Available at: http://www.webmin.com/.

[87] WHITEHEAD, J. *An Introduction to Logistic Regression* [online]. 2020 [cit. 2020-05-22]. Available at: https://www.appstate.edu/~whiteheadjc/service/logit/index.htm.

[88] YLONEN, T. and C. LONVICK, E. *The Secure Shell (SSH) Transport Layer Protocol* [Internet Requests for Comments]. RFC 4253. RFC Editor, Jan 2006. Available at: https://tools.ietf.org/html/rfc4253.

Appendix A

Employed ASNM Features

Feature	**Description**	**FFS DOL**	**FFS DL**
MedTdiff2PktsIn	Median of packet IAT (inter-arrival times) in inbound traffic.		X
InPktLen64s10i[3]	Lengths of inbound packets occurred in the first 64 seconds of a connection which are distributed into 10 intervals. The feature represents totaled inbound packet lengths of the 4th interval.		X
Bytes3WH2FIN	The number of all transferred bytes from the start to the end of a communication including session initiation and destruction packets.		X
SigPktLenOut	Standard deviation of outbound packet lengths.		X
PolyInd5ordOut[2]	Approximation of outbound communication by polynomial of 5th order in the index domain of packet occurrences. The feature represents the 3rd coefficient of the approximation.		X
PolyInd10ordOut[8]	Approximation of outbound communication by polynomial of 10th order in the index domain of packet occurrences. The feature represents the 9th coefficient of the approximation.		X
PolyInd13ordIn[10]	Approximation of inbound communication by polynomial of 13th order in the index domain of packet occurrences. The feature represents the 11th coefficient of the approximation.		X
fourGonModulIn[1]	Fast Fourier Transformation (FFT) of inbound packet sizes. The feature represents the angle of the 2nd coefficient of the FFT in goniometric representation.		X
fourGonModulOut[1]	FFT of outbound packet sizes. The feature represents the angle of the 2nd coefficient of the FFT in goniometric representation.		X
intervalsIPsSig	Standard deviation of time intervals between consecutive connections of the two hosts running on the same IP addresses as an analyzed connection. The feature assumes only beginnings of connection for computation of intervals.	X	X
gaussProds8Out[1]	Normalized products of outbound packet sizes with 8 Gaussian curves. Packets are divided into 2 slices and products are computed per each slice by summing of products of relevant packets with fitted Gaussian function. Each product is normalized by the number of packets in a slice. The feature represents a product of the 2nd slice of packets.		X

Table A.1: FFS (Part 1/2)

Feature	Description	FFS DOL	FFS DL
sumSessPerPort	The number of TCP sessions in interval ± 5 minutes from the current session, which have the same port number.	X	
InPktLen64s10i[6]	Lengths of inbound packets occurred in the first 64 seconds of a connection which are distributed into 10 intervals. The feature represents totaled inbound packet lengths of the 7th interval.	X	
OuPktLen32s10i[3]	Lengths of outbound packets occurred in the first 32 seconds of a connection which are distributed into 10 intervals. The feature represents totaled inbound packet lengths of the 4th interval.	X	
OuPktLen32s10i[6]	Lengths of outbound packets occurred in the first 32 seconds of a connection which are distributed into 10 intervals. The feature represents totaled inbound packet lengths of the 7th interval.	X	
OuPktLen64s10i[8]	Lengths of outbound packets occurred in the first 64 seconds of a connection which are distributed into 10 intervals. The feature represents totaled inbound packet lengths of the 9th interval.	X	
BytesPerSessIn	The number of transferred bytes during TCP session in inbound direction.	X	
MedPktLenOut	Median of packet sizes in outbound traffic of a connection.	X	
MedPktLenIn	Median of packet sizes in inbound traffic of a connection.	X	
ModPktLenIn	Mode of packet sizes in inbound traffic of a connection.	X	
polyInd3ordOut[3]	Approximation of outbound communication by polynomial of 3rd order in the index domain of packet occurrences. The feature represents the 4th coefficient of the approximation.	X	
polyInd5ordIn[4]	Approximation of inbound communication by polynomial of 5th order in the index domain of packet occurrences. The feature represents the 5th coefficient of the approximation.	X	
fourGonModulOut[2]	FFT of outbound packet sizes. The feature represents the angle of the 3rd coefficient of the FFT in goniometric representation.	X	
fourGonModulOut[3]	FFT of outbound packet sizes. The feature represents the angle of the 4th coefficient of the FFT in goniometric representation.	X	
gaussProds8AllNeg[1]	Normalized products of all packet sizes with 8 Gaussian curves. The feature represents a product of the 2nd slice of packets with a Gaussian function which fits to the interval of the packets' slice.	X	

Table A.2: FFS (Part 2/2)

Appendix B

CVE JSON Record Example

```
{
        "cve" : {
                "data_type" : "CVE",
                "data_format" : "MITRE",
                "data_version" : "4.0",
                "CVE_data_meta" : {
                        "ID" : "CVE-2018-7573",
                        "ASSIGNER" : "cve@mitre.org"
                },
                "problemtype" : {
                        "problemtype_data" : [ {
                                "description" : [ {
                                        "lang" : "en",
                                        "value" : "CWE-119"
                                } ]
                        } ]
                },
                "references" : {
                        "reference_data" : [ {
                                "url" : "https://cxsecurity.com/issue/WLB-2018030011",
                                "name" : "https://cxsecurity.com/issue/WLB-2018030011",
                                "refsource" : "MISC",
                                "tags" : [ "Exploit", "Third Party Advisory" ]
                        }, {
                                "url" : "https://www.exploit-db.com/exploits/44596/",
                                "name" : "44596",
                                "refsource" : "EXPLOIT-DB",
                                "tags" : [ "Exploit", "Third Party Advisory", "VDB Entry" ]
                        }, {
```

```
                "url" : "https://www.exploit-db.com/exploits/44968/",
                "name" : "44968",
                "refsource" : "EXPLOIT-DB",
                "tags" : [ "Exploit", "Third Party Advisory", "VDB Entry" ]
            } ]
        },
        "description" : {
            "description_data" : [ {
                "lang" : "en",
                "value" : "An issue was discovered in FTPShell Client 6.7. A remote FTP server can send 400 characters of 'F' in conjunction with the FTP 220 response code to crash the application; after this overflow, one can run arbitrary code on the victim machine. This is similar to CVE-2009-3364 and CVE-2017-6465."
            } ]
        }
    },
    "configurations" : {

        ...

    },
    "impact" : {

        ...

    },
    "publishedDate" : "2018-03-01T17:29Z",
    "lastModifiedDate" : "2019-03-01T18:27Z"
},
```

Appendix C

Contents of the DVD

The enclosed DVD contains the following files:

- 0-exploitator/ - source code of the extended NPBO framework
- cves/ - source code of the CVE JSON and CPE XML Parser
- doc/ - source files of this thesis
- rapidminer/ - RapidMiner repository files including datasets, models, processes sources and results of all dataset evaluation experiments
- nvdcve-1.1-2018.json.zip - ZIP file with NVD JSON Data Feed 2018
- nvdcve-1.1-2019.json.zip - ZIP file with NVD JSON Data Feed 2019
- official-cpe-dictionary_v2.3.xml.zip - ZIP file with Official CPE Dictionary
- thesis.pdf - PDF of this thesis