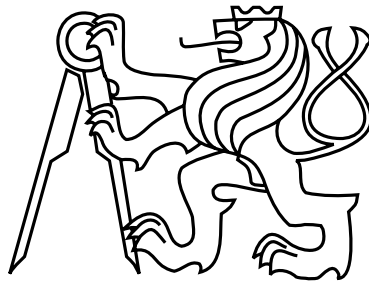


Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science



Master's Thesis

**Black-box attack on network intrusion detection
systems**

Bc. Peter Hroško

Advisor: Ing. Tomáš Pevný, PhD.

Open Informatics, Master degree program

Artificial Intelligence

May 12, 2014

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Computer Science and Engineering

DIPLOMA THESIS ASSIGNMENT

Student: **Bc. Peter Hroško**

Study programme: Open Informatics
Specialisation: Artificial Intelligence

Title of Diploma Thesis: **Black-Box Attack on Network Intrusion Detection Systems**

Guidelines:

- Search and study the prior art on detection avoidance.
- Implementation of network anomaly detection systems.
- Design of algorithms crafting an undetectable attack and their implementation.
- Experimental evaluation of the implemented algorithms on real data.

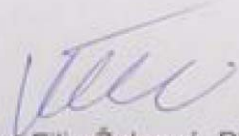
Bibliography/Sources:

Blind Newton sensitivity attack, P. Comesana, L. Perez-Freire and F. Perez-Gonzalez, 2006

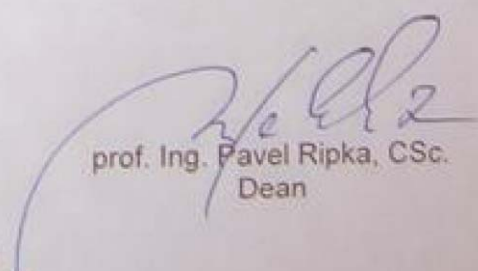
Marius Kloft, Pavel Laskov: Online Anomaly Detection under Adversarial Impact, 2010

Diploma Thesis Supervisor: Ing. Tomáš Pevný, Ph.D.

Valid until the end of the summer semester of academic year 2014/2015


doc. Ing. Filip Železný, Ph.D.
Head of Department




prof. Ing. Pavel Ripka, CSc.
Dean

Prague, March 3, 2014

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze 12.května 2014

A handwritten signature in black ink, consisting of a stylized 'R' followed by a horizontal line extending to the right.

Abstract / Abstrakt

Network security can be viewed as a long lasting arms race between adversaries and intrusion detection systems designers, each of them trying to reach their goals by reacting to the oponent's actions. A potential breakthrough in this race could be made by anticipating the adversary's next move. We attempted to achieve it by devising a procedure, which serves for designing optimal attacks against a particular network detector based on anomaly detection. The proposed method is tested against exact optimal values, and verified to efficiently produce maximal attacks on the mentioned detector. Our findings are important for the evaluation of the security of the analysed detection system and can be used for its improvement.

Na síťovou bezpečnost lze nahlížet jako na dlouhotrvající závod ve zbrojení mezi útočníky a tvůrci detektorů síťových útoků, ve kterém se obě strany snaží dosáhnout svých cílů reagováním na akce svých protivníků. Potenciálním zlomem v tomto závodě by mohlo být předpovídání útočnickových dalších kroků. Toho jsme se pokusili dosáhnout navržením procedury, která má sloužit k přípravě optimálních útoků na konkrétní detektor, fungující na principu detekce anomálií. Navrhovaný algoritmus je poté testován oproti známým ideálním hodnotám a je ověřeno, že efektivně produkuje maximální útoky na zmiňovaný detektor. Výsledky této práce jsou důležité pro odhad bezpečnosti zkoumaného detekčního systému a mohou být použity k jeho zdokonalení.

Contents

1	Introduction	6
1.1	Prior art	7
1.2	Problem formulation	9
2	Anomaly detection	12
2.1	NetFlow format	13
2.2	Detector	14
2.2.1	Detector properties	17
3	Planning the attack	20
3.1	Attack scenarios	21
3.1.1	Scenario 1	21
3.1.2	Scenario 2	22
3.2	Gradient method	23
3.2.1	Discretization	28
3.2.2	Algorithm	29
3.2.3	Single window and Multiple windows attack	31
3.3	Exhaustive search	31
3.4	Implementation	33
3.4.1	Flow injection	33
3.4.2	Detector optimization	34
4	Experimental evaluation	36
4.1	Real traffic	36
4.2	Comparison of attack scenarios	37
4.3	Single window attack	39

4.4	Efficiency of the procedure	41
5	Conclusion	43
A	Attachments	46
B	Attachments on cd	48
B.1	Practical part	48
B.2	Theoretical part	48

Chapter 1

Introduction

Complexity of security systems is growing as a reaction to ever more advanced and refined attacks. Particularly in the area of network intrusion detection systems (IDS) a long lasting arms race can be observed between IDS designers and adversaries, each of them attempting to reach their goals by reacting to opponent's actions.

There are two basic approaches in detection of network intrusion: signature based and anomaly based detection.

Signature based detection works with a database of known attacks. The detection can be then imagined as a comparison of actual events in the network with the known attacks in the database, so called signatures. This approach is limited by the quality of the database, which needs to be updated regularly. Signature based detection is vulnerable to the *zero-day* attacks, i.e. new unknown attacks.

Anomaly based detection's aim is to model legitimate traffic and discover the abnormal events, possibly attacks. This approach may detect the *zero-day* attacks, but usually has other disadvantages, such as high false positive rate (i.e., high frequency of false alarms), because not every anomaly is necessary malicious.

A framework for empirical evaluation of security systems proposed in [1] could be a potential break through in the competition between IDS

designers and adversaries. The framework is based on a general model of adversary. Also rather than a mere reaction to a new attack, a step ahead is suggested: IDS designers should search for the vulnerabilities in their systems and design attacks themselves.

The motivation for making an attack can be rendered in two main points. The first of them is assessing security of the detection system. Knowing where the weaknesses of the detector lie is a very valuable information. It can be then decided, whether the weakness can be harmfully exploited by an adversary and needs a countermeasure, or can be disregarded. Which brings us to the second point. A well designed attack can be viewed as the proactive step in the aforementioned arms race between IDS designers and adversaries. By anticipating the adversary's future actions and designing appropriate countermeasures the security of the system can be increased and eventual damage prevented.

Inspired by this framework we first analyse a particular network intrusion detector in Chapter 2. Then we proceed with planning of an attack against the detector in Chapter 3, which is the main contribution of this thesis. Our goal is to find a method for designing optimal attacks against the analysed detector. Before drawing the final conclusions, the proposed method for designing attacks is thoroughly tested in Chapter 4.

1.1 Prior art

Adversarial machine learning is an emerging field of study, which concerns how learning process and its outcomes can be altered by an adversary. Adaptive detection systems are popular in the area of network security, because they can react on new threats and to continual changes of user's behaviour. However, the adversary can attempt to spoil the learning phase or he can try to make his attack improperly classified as a legitimate behavior. There are various ways how adaptive systems can be subverted. Variety of

examples concerning such malicious behavior can be found in the work of Barreno et al. [3][4]. They introduce a taxonomy of attacks, based on the influence, specificity, and security violation of the attacks, and also possible countermeasures are suggested.

The general approach of describing attacks by their taxonomy is further extended by Biggio et al. [1]. The adversary is modeled there according to his goal, knowledge, capability and attack strategy. Also an example of attack on detectors of malware at test time is presented. It comprises of hiding malicious software into PDF files. The attack is based on gradient descent optimization method and is tested against two types of machine learning detectors: support vector machines and multi-layer perceptron (artificial neural network). In both cases the detection function is available to the attacker and it is analytically differentiable, therefore gradient method is feasible for finding the optimal attack by minimizing the detection function. A mimicry method is incorporated, which attempts to imitate the negative (not malicious) samples potentially helping the gradient method to escape local minima.

Various poisoning attacks have been described in [1][5]. The main distinction of this kind of attacks is they are aimed on the learning phase of the detector, while earlier mentioned attacks targeted test phase. The training set is poisoned by injecting malicious samples, either targeting one attack sample to be misclassified, or causing high error rate of the learned classifier in general.

Two very important ideas in the context of this thesis can be found in the work of Comesaña et al. [2]. It is the “blindness” of the attack, meaning the limited knowledge of the adversary about the detector, and the application of numeric optimization method (there Newton method, here gradient ascent method). They describe a blind attack on a picture watermarks detector in the study. Picture watermarking is a different area,

but the general approach can be applied elsewhere, particularly in network intrusion detection.

Two scenarios are explored in the study: in one of them the aim is to remove watermark from a picture with minimal picture distortion. The other scenario is focused on creating forgeries i.e. finding minimal change of an unwatermarked picture so that it is classified as watermarked.

A similar scenario as in the watermarking example can be found in the work of Nelson et al. [6]. They describe an adversary, who has a limited knowledge about the detector, but is able to query the detector in order to obtain valuable information about it, which can be used to evade detection. The proposed attack is near-optimal and minimizes the difference from the intended ideal attack. Targeted detector belongs to the family of convex-inducing classifiers, similarly as linear classifiers, most of the anomaly-based detectors and other popular detectors.

1.2 Problem formulation

Let's define network flow as a tuple (x_1, x_2, x_3, x_4) , where the elements x_1 to x_4 represent source IP address, source port, destination port and destination IP address, respectively.

Network background traffic \mathcal{B} is defined as a set of network flows $\mathcal{B} = \{(x_1, x_2, x_3, x_4), \dots\}$. Network traffic can be split into time windows $t_1, t_2, \dots: \mathcal{B}_{t_1}, \mathcal{B}_{t_2}, \dots$

We expect adversary's resources to be limited, therefore we define the set of adversary's available IP addresses \mathcal{I} and the set of adversary's available ports \mathcal{P} . Using the resources the adversary designs an attack, formally a set of attack network flows $\mathcal{A} = \{(i_1, p_1, p_d, i_d), (i_2, p_2, p_d, i_d), \dots\}$, where source IPs $i_1, i_2, \dots \in \mathcal{I}$ and source ports $p_1, p_2, \dots \in \mathcal{P}$ can be set by the adversary, but the target ports and IPs are fixed and determined by the attack.

The network traffic seen by IDS consists of the background traffic \mathcal{B} and injected attack flows \mathcal{A} .

$$\mathcal{T} = \mathcal{B} \cup \mathcal{A}$$

The traffic is monitored by a detector, which aims to separate the background and the attack traffic. It comprises of a detection threshold δ , which is constant for the detector, and a detection function f . The function f is a mapping from network traffic \mathcal{T} to a real number for each of the IP addresses \mathcal{K} involved in the traffic:

$$f : \mathcal{T} \rightarrow \mathbb{R}^{|\mathcal{K}|}$$

The goal of the adversary is to attack a system protected by IDS, which means to insert as many attack flows, as possible without being detected. Naturally the attack should be as quick as possible which is represented as the injection of maximal number of network flows to the target in minimal time, in our case one time window t . We can then formalize adversary's task for attack time t as:

$$\max_{i,p} \sum_{i \in \mathcal{I}} n_i$$

subject to:

$$f(\mathcal{B} \cup \mathcal{A}) < \delta$$

Where n_i is the number of flows injected from IP address i . The maximal number of attack flows is sought over IP addresses i and ports p used by adversary. The task is constrained by the detection function f and the detection threshold δ , which expresses the condition of the attack not being detected.

The adversary does not have direct access to f and δ and needs to treat the detector d as a black box:

$$d : \mathcal{T} \rightarrow \{true, false\}^{|\mathcal{K}|}$$

Detector computes the value of detection function f for each of the IP addresses \mathcal{K} involved in the analyzed traffic and compares the value with detection threshold δ , producing output *true* (*false*) for *malicious* (*legitimate*) traffic respectively. The adversary can see only the output and thus has access solely to the information, whether the constraint was satisfied, or not.

Instead of a single detector, an ensemble of m detectors f_1 to f_m can be used together with detection thresholds δ_1 to δ_m , whereas the worst case scenario for the attacker is assumed - none of the detectors can give positive response to the attack.

$$\begin{aligned} f_1(\mathcal{B} \cup \mathcal{A}) &< \delta_1 \\ f_2(\mathcal{B} \cup \mathcal{A}) &< \delta_2 \\ &\vdots \\ f_m(\mathcal{B} \cup \mathcal{A}) &< \delta_m \end{aligned}$$

A scenario with a more informed adversary could be assumed, if also the values of detection functions f_0 to f_m were available to the adversary in addition to the binary output classifying traffic as malicious/legitimate. This case will not be concerned in this thesis.

We can further extend the attack to multiple time windows by replacing single attack window t by a range of time windows $t = 0, 1, 2, \dots, n - 1$, assuming $\mathcal{A} = \bigcup_{t=0}^{n-1} \mathcal{A}_t$.

Clearly, the task is a NP complete problem, because with no additional information about the detection function the only way to design optimal attack is to search all possible combinations of attack IP addresses and ports and find the maximal number of attack flows, which are not detectable.

However, for a special category of detectors - *convex boundary inducing classifiers* - there is an efficient way how to solve the task. It will be discussed in more details in following chapters.

Chapter 2

Anomaly detection

Anomaly detection is based on modelling legitimate behavior of systems. Events which do not conform to the model are considered anomalous, possibly malicious. In this chapter we will discuss how malicious network traffic can be distinguished from normal traffic by a particular anomaly detector. After describing the details about its internals, we will take a look at some of the detector's main properties: monotonicity of its detection function, which quantifies the level of anomaly of the examined traffic, and the detection boundary, which splits the traffic to legitimate and anomalous.

We assume that decision boundary of most intrusion detection systems is convex, because any convex combination of legitimate behaviors should be legitimate as well. Also when various detectors are combined together and individual detectors within the ensemble have convex decision boundary, the final boundary is convex.

The main advantage of such detectors from the adversary's point of view is the possibility of employing a simple optimization method, namely the gradient ascend method, to find the global maximum, representing an optimal attack, in polynomial time. In other types of detectors with non-convex decision boundary the search would often stop in local minima, which would lead to suboptimal results.

This chapter explains the details of the detector, which is later used in

the experimental evaluation of the proposed method. First, there will be described the format of data, the detector operates with. Then the details about how the detector is created and how it works will be presented.

Unfortunately not all assumptions can be correct all the time, which also happened to our assumption about the detector. Before the end of this chapter we will study the detector properties and come to an interesting conclusion: Even though the detector is based on anomaly detection, its decision boundary is not convex.

2.1 NetFlow format

In order to perform network traffic analysis it is necessary to capture its main characteristics and store them in some compact representation. Network traffic is a complex system, so naturally only a fraction of the information describing it can be stored and not every kind of information is useful for the analysis.

One convenient way of recording the traffic is to use the standard NetFlow format, because it can be easily exported into logs directly from network routers. The content of the network packets is disregarded, only the metadata about the traffic are stored, describing the communication inside the network in terms of so called network flows. These can be imagined as an unidirectional connection between two points in the network. Each connection has its origin and destination (ip address and port), time of start and duration, type of network protocol, information about the volume of the communication (number of packets, number of bytes, number of flows) and some additional information (type of service and flags for TCP protocol). These are the metadata stored in NetFlow logs. An example of such a log can be seen on Figure [2.1](#).

Date flow start	Duration	Proto	Src IP Addr:Port	Dst IP Addr:Port	Tos	Packets	Bytes	Flows
2010-10-31 23:57:08.060	0.000	UDP	147.32.85.34:57305 ->	147.32.80.9:53	0	1	71	1
2010-10-31 23:57:08.061	0.000	UDP	147.32.80.9:53 ->	147.32.85.34:57305	0	1	224	1
2010-10-31 23:57:08.061	0.000	UDP	147.32.85.34:53470 ->	147.32.80.9:53	0	1	71	1
2010-10-31 23:57:08.061	0.000	UDP	147.32.80.9:53 ->	147.32.85.34:53470	0	1	227	1
2010-10-31 23:57:08.061	0.000	UDP	147.32.85.34:49518 ->	147.32.80.9:53	0	1	71	1
2010-10-31 23:57:08.061	0.000	UDP	147.32.80.9:53 ->	147.32.85.34:49518	0	1	224	1

Figure 2.1: Sample NetFlow log

Such a description allows analysis of greater and more abstract patterns in the behavior of users participating in network traffic, rather than analysis from the perspective of the content of packets.

2.2 Detector

This thesis is focused on designing a block-box attack, which means adversary has no knowledge about the detector. However, the detector is a vital part of the experimental evaluation and was implemented specifically for this thesis. Also various optimization adjustments were made to the detector, which are necessary for the attack design. Thus, we dedicate this whole section to its thorough description.

Our detector belongs to the anomaly based approach in network detection described earlier in this chapter. It is based on a detector introduced in [7]. The detection is performed on the data of NetFlow logs described in previous section (see 2.1). As was previously said the detector does not inspect the content of network packets, it only uses information about source and destination ip addresses and port numbers.

The detection consists of three main building blocks, first two defining the detector and the third one represents the input data. First block is the model of regular traffic, detection thresholds constitute the second building block, and the last block is the features, extracted from the traffic to be detected. In the following sections we will describe the details necessary for creating the detector and extracting detection features from the data.

Feature space

The detector does not use the flows from NetFlow logs directly, it rather works with the distributions of attributes characterizing the network traffic such as ip addresses and port numbers. Also not the entire information about the distributions is necessary. Convenient compressed way for description of distributions is the *average unpredictability*, which is known as the Shannon information entropy. The entropy exactly expresses the information needed for anomaly detection, because the detector models the *predictable* - legitimate - traffic, and the *unpredictable* traffic is suspicious and can be considered malicious or further analysed.

In order to compute the entropies, flows are grouped with respect to ip addresses. This can be done in two ways - aggregation according to *source* and *destination* ip addresses, resulting in two possibilities how to construct the detector.

Memory

Before the input vectors for detection can be created from raw data, there is one significant aspect of the detector, which needs to be clarified. The detector does not work online, meaning that it is not checking each new flow as soon as it appears in the NetFlow log. Instead, the time is discretized in time windows of 5 minutes length and every five minutes new NetFlow log is exported and processed by the detector. Also the detector has its own memory. The data it is being created on are aggregated from 5 such time windows, which means the memory has length of $5 \times 5 = 25$ minutes.

The whole process can be imagined as a 25 minutes long sliding window, where every 5 minutes some flows (from the oldest time window of memory) are discarded, and some new flows are added.

Level of anomaly

The features extracted from raw data compose input vectors x for each ip address present in the actual traffic. Input vectors have length 15: 5 time windows, each producing 3 items: entropy of source ports, entropy of destination ports and entropy of source/destination ip addresses, depending on the type of aggregation. The matrix of input vectors is then transformed with PCT (principal component transformation), obtaining set of eigenvectors $\{y_j \in \mathbb{R}^{15}\}_{j=1}^r$ and eigenvalues $\lambda_1 > \lambda_2 > \dots > \lambda_r$. All eigenvectors corresponding to eigenvalues lesser than 10^{-6} are discarded for numerical stability.

For assessing the value of *detection function* f , sometimes called the *level of anomaly*, following formula is used on each ip address i :

$$f(x^t(i)) = \sum_{j=1}^k \frac{(y_j^T x^t(i))^2}{\lambda_j} \quad (2.1)$$

$$f^\perp(x^t(i)) = \sum_{j=k+1}^r \frac{(y_j^T x^t(i))^2}{\lambda_j} \quad (2.2)$$

From these two equations the values of two detectors are acquired. Equation 2.1 employs the first k major components from PCT, defining one detector. The rest $(r - k)$ minor components in Equation 2.2 result in another detector. According to the original source of the detector [7], the constant k was set to 1.

Combining two ways of aggregating flows (source/destination IP addresses) with two ways of computing detection function (major/minor PCT components) altogether four detectors are created (see 2.1).

	major PCT components	minor PCT components
source IPs	detector ₁	detector ₂
destination IPs	detector ₃	detector ₄

Table 2.1: Four different detectors are created.

Thresholds

Having computed the level of anomaly of each ip address appearing in the network traffic, there must be defined a treshold for discerning the anomalous activity from the legitimate one. We can assume that 5% of the traffic can be investigated by a human operator, so the 5% most anomalous ip addresses will be reported by the detector as potentially malicious. The absolute value of the treshold is computed independetly for each of the four detectors, and relative to the level of anomaly of all ip addresses currently active in the traffic.

Intersection of detectors

Each detector splits the feature space into two parts. One part is called the *decision region*, where traffic is considered legitimate, the other part is marked malicious. In order to construct the final detector, the four simple detectors described earlier in this chapter are combined together. The most strict way how to do it is intersecting their decision regions. This corresponds to the safe strategy of the attacker, when he does not want to be detected by any detector from the ensemble.

2.2.1 Detector properties

As the last step before we can start designing an attack, we need to analyse the detector's basic properties. First of them is the monotonicity of the detection function. In other words we need to verify that the value of

detection function will increase (or at least not decrease) whenever we inject some attack flows.

A simple test has been done, where the known attack flows captured during the NetFlow log (see Chapter 4.1) were injected into the traffic one by one. The value of the detection function of each of the 4 detectors can be seen on Figure 2.2.

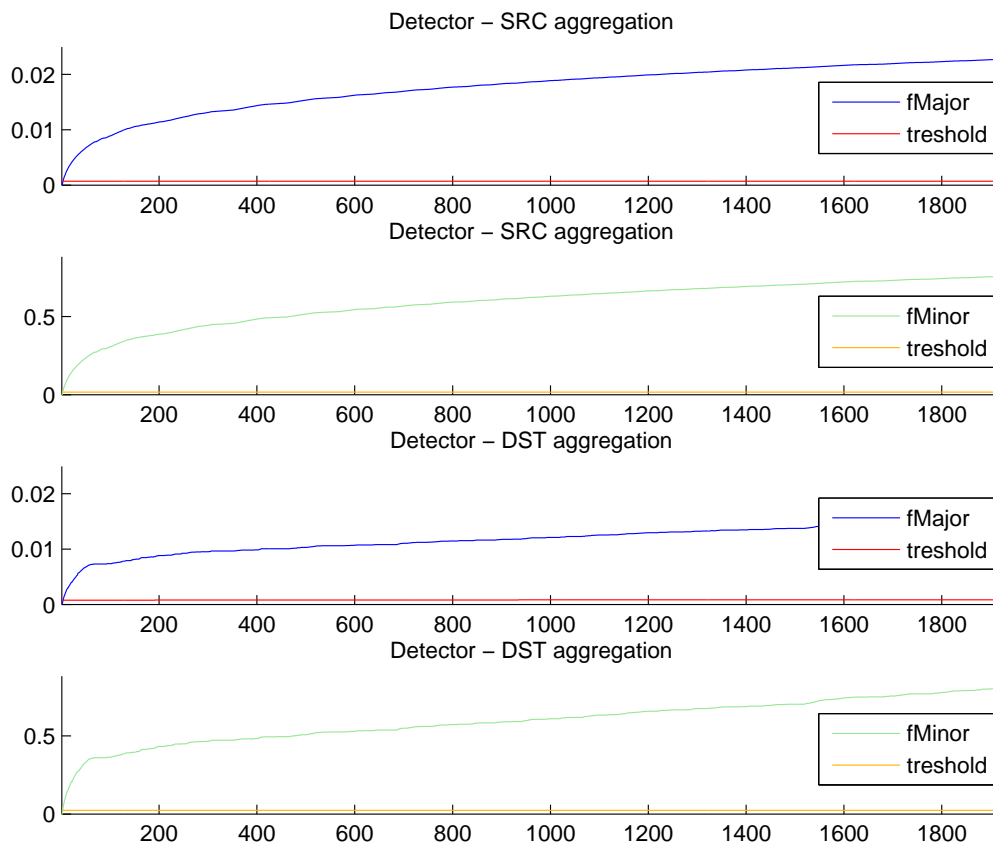


Figure 2.2: Monotonicity check: Horizontal scan

Similar tests were done for other two attacks, which were also captured in the university’s network traffic. The results are analogical and can be seen in Appendices A.1 and A.2

We have explored the characteristics of detection function under conditions of a progressing attack. From the monotonicity test we can conclude,

that its behavior coincides with our expectations of an increasing function.

As the next step we will be examining the decision boundary of the detector. The boundary is studied according to the number of flows injected from given number of ip addresses. The number of ip addresses defines the dimension of the space, in which the detection boundary is observed. On Figure 2.3 there can be seen the decision boundary plotted for 3 ip addresses.

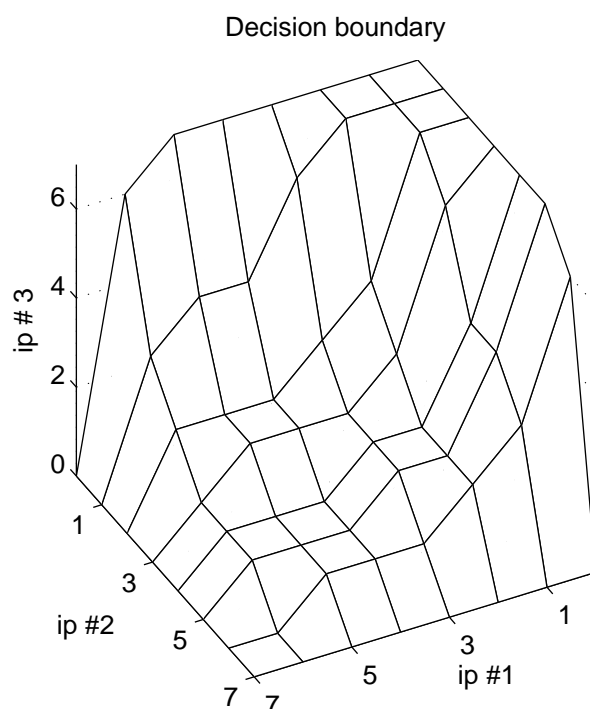


Figure 2.3: Decision boundary

There can be observed an interesting property of the detector on the figure. The boundary is non-convex, which has some grave consequences for the designing of optimal attacks. Unlike the previous test of monotonicity, this result was quite unexpected. Some major adjustments had to be done to address this problem and they will be further described in the next chapter.

Chapter 3

Planning the attack

We devised two scenarios for the purposes of designing an attack. They differ from each other in terms of how and when the detector is used, so naturally the choice of the scenario strongly influences obtained results.

After the description of scenarios we present a classic optimization technique: gradient ascent method. It is combined with computation of numerical gradient, which allows to treat the detector as a black-box, whereby simulating an adversary's attempt to design an attack with no knowledge about the detector.

The gradient method is adapted for the domain of network communication, specifically for designing an optimal attack. Besides a thorough description of the proposed method, also an explanation is included of how we managed the problem of the decision boundary non-convexity.

For estimating quality of results obtained from the proposed method we need a reference solution, which will be acquired by a procedure called exhaustive search. This procedure is described in this chapter and later used in [Chapter 4](#).

The gradient method is then employed in designing two types of attack. First of them is quite simple, because it is limited in duration to a single time window. The number of different feasible (undetectable) attacks is low

enough to explore them all in exhaustive search, so the result of our method can be compared with exact optimal values.

Second attack is much more complex, because it extends to multiple time windows, and it will be investigated only theoretically.

To finalize the attack planning some important implementation details will be explained, before proceeding to the experimental evaluation in the next chapter.

3.1 Attack scenarios

The detection consists of three building blocks: *model* of normal traffic, detection *thresholds* and *input vectors* representing the traffic to be detected. The details, about how these blocks are built, were described in Chapter 2.2, but so far it hasn't been mentioned, how the blocks work together and what data they are built on. We devised two scenarios, defining how the detection is performed and how the detector is used for the design of attack.

3.1.1 Scenario 1

Scenario 1 is the most straightforward option. Blocks 1 and 2 of the detector are computed on time windows t_{-5} to t_{-1} and also the attack is designed based on this detector and time window t_{-1} . After the attack is ready, it is injected in the time window t_0 . The injected attack is then verified by a second detector, whose first part ($model_1$) remains the same as in the first detector, but thresholds are created from data of t_{-4} to t_0 . The concept can be better understood by looking at Figure 3.1.

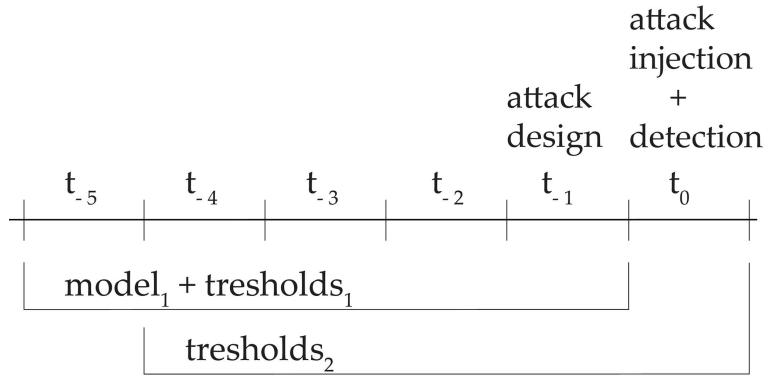


Figure 3.1: Scheme of Scenario 1

3.1.2 Scenario 2

Second scenario is similar to the first one in the point when the attack is designed (time t_{-1}) and when it is injected to the traffic (time t_0). Also the computation of thresholds is based on the same time windows as in the previous scenario. Unlike in scenario 1 the $model_1$ is computed one window earlier. This way the first detector is obtained, which serves for the purposes of attack design.

The feasibility of the attack is again verified with a second detector, which is computed on the same basis as the first one in this scenario, but shifted one time window ahead. A better idea of the process of creating the detectors and preparation of the attack can be gained from the Figure 3.2.

The difference between both scenarios is following: In scenario 1 the detector is trained on the same data as attack, which can make it more biased towards the detector. We can thus anticipate the attack to be stronger, but also with higher chance of being detected by the second detector.

As opposed to the first scenario, the building blocks in scenario 2 are more spread in time and thus cover more data. Consequently the attack should be less biased to the detector used for its design. It follows, that the strength will be probably lower, but the same holds for the chances of the attack being detected by the second detector.

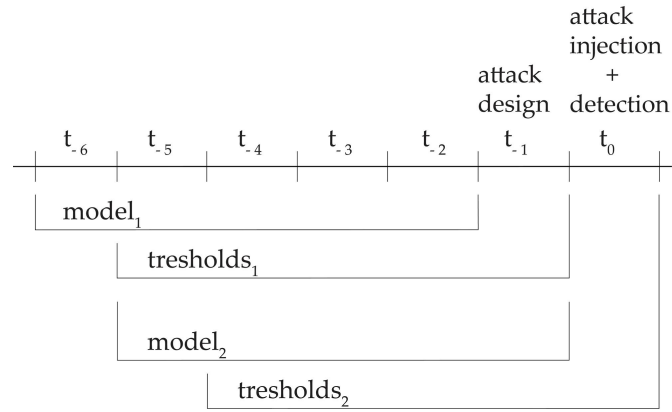


Figure 3.2: Scheme of Scenario 2

3.2 Gradient method

In this section a well known optimization method will be presented, which is often used for its simplicity combined with good effectivity in domains with continuous space representation: The gradient ascent method. For this thesis the method is combined with numerical computation of gradient, which allows us to use it even with limited knowledge about the detector, which is treated as a black-box. Otherwise it would be possible to compute the gradient analytically.

As was said in Chapter 1.2, our goal is to design an optimal attack against the introduced detector. The quality or strength of the attack is defined as the number of flows injected to the target server without being detected.

In the Problem formulation the attack was defined as a set of flows. However there are infinitely many possible representations and for each purpose different representation can be the most efficient.

For gradient method the following one proved to give the best results. The attack is defined as a distribution of flows α between attacker's source ip addresses:

$$\alpha = (\alpha_1, \alpha_2, \dots, \alpha_k),$$

where k is the number of ip addresses the adversary can use. We ensure α to be a valid distribution by laying

$$\alpha_i \geq 0 \text{ and } \sum_{\forall i \in \mathcal{I}} \alpha_i = 1$$

The number of flows n_i to be injected from ip address i can be calculated from the distribution by the formula

$$n_i = \alpha_i \cdot n,$$

where n is the total number of flows to be injected.

The search space is consituted by all possible distributions. The method is called gradient ascent, because starting in some initial point it iteratively traverses the search space in the direction of increasing gradient of the objective function. The search is stopped once the gradient drops to (or below) zero, which means a local, or in case of convex objective function, global maximum is reached.

The basic iteration cycle is composed of two actions: compute gradient in actual point of the search space, make a step in this direction and repeat. We will now describe these two actions in detail.

Gradient

Numerical computation of gradient allows us to calculate the gradient with a mere binary output from the detector. In contrary if we assumed the adversary's knowledge about the detector is not limited, we suppose it would be possible to determine the gradient analytically.

Numerical gradient is computed via a well known formula [3.1](#). Although there is a speciality connected to our representation of attacks. A distribution α of k variables can be expressed in $k - 1$ independent variables, because

$$\alpha_k = 1 - \sum_{i=1}^{k-1} \alpha_i, \text{ induced from the condition that } \sum_{\forall i \in \mathcal{I}} \alpha_i = 1 \text{ to be a valid}$$

distribution.

Gradient is then computed with respect to the $k - 1$ independent variables only:

$$\nabla f(\alpha) = \left[\frac{\partial f(\alpha)}{\partial \alpha_1}, \dots, \frac{\partial f(\alpha)}{\partial \alpha_{k-1}} \right] \quad (3.1)$$

where

$$\frac{\partial f(\alpha)}{\partial \alpha_u} = \frac{f(\alpha_1, \dots, \alpha_u + h, \dots, \alpha_k - h) - f(\alpha_1, \dots, \alpha_u, \dots, \alpha_k)}{2h} \quad (3.2)$$

Another speciality in the computation of gradient is the subtraction of h from α_k in Equation 3.2. The reason is again in the condition of α being a valid distribution. If we add h to an element of α_u , we must subtract it from the dependent variable α_k in order to keep $\sum_{i=1}^k \alpha_i = 1$.

Line search

So far we haven't discussed, what the objective function $f(\alpha)$ is, that we are optimizing. It is a mapping:

$$f : \mathbb{R}^k \rightarrow \mathbb{N}$$

The function $f(\alpha)$ takes a distribution $\alpha \in \mathbb{R}^k$ as its input and returns the maximal number of flows $n \in \mathbb{N}$, which can be injected under the given distribution.

More precisely it iteratively increases n from zero until the attack $n \cdot \alpha$ is detectable. Last n which produced an undetectable attack is the required value. This procedure is called *line search* and encloses a simple one-dimensional optimization task of finding maximal attack along a fixed direction, given by the input distribution.

On Figure 3.3 there is the line search procedure with input direction $\alpha = [\alpha_1, \alpha_2]$, which projects the distribution to the decision boundary. It is displayed in 2D space, representing the numbers of flows n_1 and n_2 injected from 2 attack ip addresses.

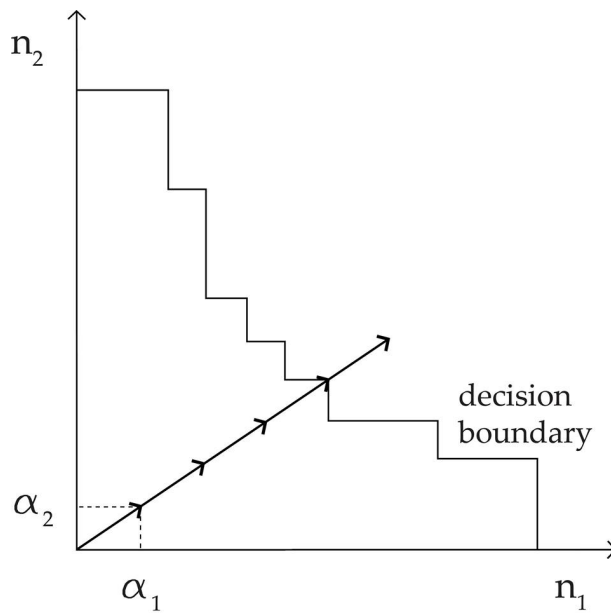


Figure 3.3: Line search procedure

Step size

We already described, that gradient is computed in each point of the search α_s . The gradient leads the search towards the optimum. After gradient is computed a step to the next point α_{s+1} needs to be made. Usually in gradient method optimization there is some step size coefficient δ , which may be fixed or adapted during the search.

$$\alpha_{s+1} = \alpha_s + \delta \cdot \nabla f(\alpha_s) \quad (3.3)$$

On Figure 3.4 there can be seen how *delta* is employed in the search. Gradient $\nabla f(\alpha_s)$ is shortened there to g .

Finding δ could be seen as another one-dimensional optimization task. To circumvent this task we used a greedy approach, where we pick the first improving δ and rely on the gradient, that it will adjust the search direction in the next step, even though the chosen δ was not optimal.

The first improving step size is sought on an interval, determined by

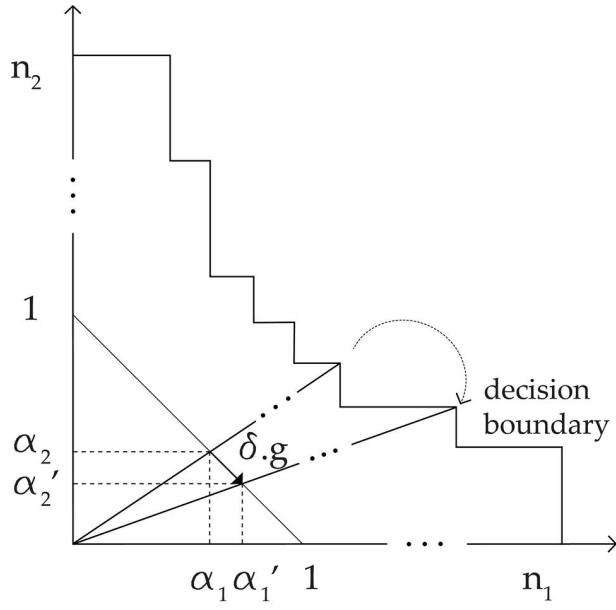


Figure 3.4: Step size computation

conditions of α remaining a distribution. It must hold that

$$\sum_{i=1}^k \alpha_{s+1,i} = 1$$

and thus:

$$0 \leq \sum_{i=1}^{k-1} \alpha_{s+1,i} \leq 1$$

By employing Equation 3.3 we get:

$$0 \leq \sum_{i=1}^{k-1} (\alpha_{s,i} + \delta \cdot \frac{\partial f(\alpha_s)}{\partial \alpha_{s,i}}) \leq 1$$

The lower and upper limits for the step size can be derived by isolating δ :

$$\delta \geq -\frac{\sum_{i=1}^{k-1} \alpha_{s,i}}{\sum_{i=1}^{k-1} \frac{\partial f(\alpha_s)}{\partial \alpha_{s,i}}} \quad \delta \leq \frac{1 - \sum_{i=1}^{k-1} \alpha_{s,i}}{\sum_{i=1}^{k-1} \frac{\partial f(\alpha_s)}{\partial \alpha_{s,i}}}$$

3.2.1 Discretization

We encountered two main problems during implementation of the gradient method. Firstly, the decision boundary is not convex, which means the gradient method can not consistently find global maxima, because it is a local optimization tool. Depending on the starting point the search terminates in some of the optima (local or global).

Second problem is the discrete nature of the search space. When we project the decision boundary - originally a smooth continuous function - to the discrete space, it breaks up into a combination of step functions. The steps can be viewed as alternating local minima and maxima of the objective function, which is proportional to the distance between the boundary and the origin (see Figure 3.4).

Practically, both these problems collapse to a single challenge of escaping local maxima. We managed to solve it by gathering more information about the neighborhood of the searched point. The gradient is computed not for just one size of the step in numerical gradient, but for various values and both negative and positive. What can be accomplished by this approach can be seen on Figure 3.5.

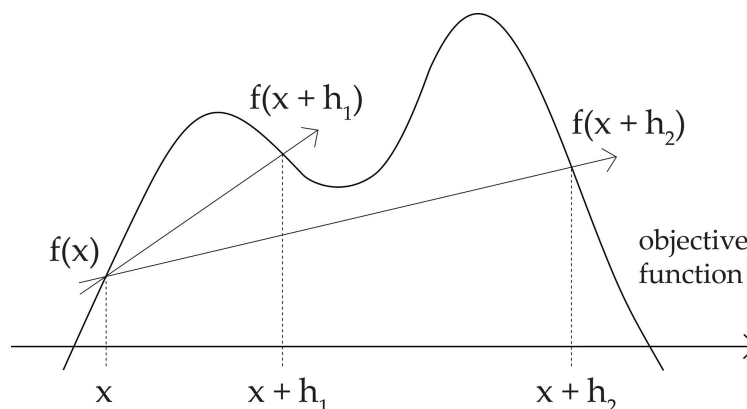


Figure 3.5: Various values of h in computation of gradient can lead the search towards different local maxima. The objective function is only illustrative.

This way the local maxima caused by the discretization can be overcome. More precisely any local maxima with locality less than h_{max} (the greatest h from all employed in the search). Locality can be expressed as the diameter of an area, for which the generally *local* maximum represents a *global* maximum.

So far nothing has been said about the local maxima possibly connected with the non-convexity of the decision boundary. Can they be overcome by searching with gradient of different h -sizes? A general statement can't be concluded, but based on our available data, if there exists more than one local maxima, all of them have equal value. Further details about the effectivity of the proposed adjustment can be found in Chapter (4.3).

3.2.2 Algorithm

To summarize the gradient method, we present here the pseudocode of the earlier described procedure.

Algorithm 1: Gradient method

```

Input : init_point
Output: max_attack

actual ← init_point;
actual_flows ← sum(init_point);
max_flows ← 0;
while actual_flows > max_flows do
    max_flows ← actual_flows;
    for  $h \in \{h_1, \dots, h_n\}$  do
        calculate numerical gradient  $g$  in actual point (Equation 3.1);
        calculate first improving step size  $\delta$  (Section 3.2);
        update actual according to  $g$  and  $\delta$  (Equation 3.3);
        temp_flows ← sum(actual);
        if temp_flows > actual_flows then // find maximum
            actual_flows ← temp_flows; // over different
            max_attack ← actual; // sizes of h (3.2.1)
        end
    end
end

```

Both for computation of gradient and step size the value of objective function is needed. This value is acquired by line search procedure, which was already described earlier in Section 3.2. In addition we show the pseudocode of Algorithm 2, because this procedure is crucial for the whole process of attack design.

Algorithm 2: Line search (Section 3.2)

Input : direction
Output: max_flows, max_attack

malicious \leftarrow false;
max_attack \leftarrow empty;
max_flows \leftarrow 0;
actual_flows \leftarrow 0;
coefficient $c \leftarrow$ maximal element from the direction vector;
 $n \leftarrow 0$;
while *not malicious* **do**
 $n \leftarrow n + 1$;
 if $actual_flows \geq max_flows$ **then**
 max_flows \leftarrow actual_flows;
 max_attack \leftarrow actual;
 end
 actual \leftarrow direction $\cdot \frac{n}{c}$;
 actual_flows \leftarrow sum(actual);
 malicious \leftarrow use detector on actual
end

Line search projects any input vector (usually an attack distribution) to the detection boundary of the detector. In other words, it finds out the maximal possible attack, which is achievable under the given distribution. The return value is the number of flows injected by such attack and for convenience also the attack itself. The procedure also guarantees the returned attack is not detectable.

3.2.3 Single window and Multiple windows attack

The introduced algorithm represents a method for designing attacks of limited duration. The limiting factor is the length of a time window defined by the detector - 5 minutes. If the attack was to exceed this limit, a new detector would have to be incorporated into the design to perform detection in the new window.

In reality the change to the proposed method wouldn't be too dramatic. The method regards the detector as a block box and needs solely a binary response from it (legitimate/malicious). Thus the verification part wouldn't have to change at all. Only the representation of the attack would have to be altered. Also when more time windows are concerned, new possibilities of how to combine the attacks between them come into play. The search space increases exponentially with each new time window.

Due to the rising number of combinations it also wouldn't be possible to compare the attack with an exhaustive search, because the number of possibilities is simply too high. Other ways of acquiring accuracy of the method would have to be introduced, but we doubt an exact result could be obtained. Probably some approximate techniques, such as *Monte-Carlo sampling*, would help.

Due to these reasons the attack to multiple windows will not be regarded in this thesis, although we claim the proposed method should be able to solve this task and the necessary modifications would have merely an implementation character.

3.3 Exhaustive search

Going through all possible combinations of attacks is the only possible way of verifying that our proposed method finds the global maximum. We can apply the monotonicity property of the detector to limit the infinite space of

all combinations (of course adversary's resources are in practice limited, so the space of all combinations is not infinite, but its size is still enormous). So the monotonicity can be employed by the means that once the attack is detected, it can never become undetectable again by *adding* more flows.

Again we use a different attack representation for this method. In this case we can take advantage of enumerating all possible combinations of attacks, if expressed by the number of flows injected from each of the adversary's ip addresses. It wouldn't be possible if we used the representation by distributions, because distributions are expressed by real numbers, thus there are infinitely many of them even though they project to the same finite set of attacks. An example of such representation is a vector [7,5,2,1], describing an attack consisting of 7 flows injected from one ip, 5 flows from another one, etc.

Procedure

In exhaustive search all possible combinations of attacks must be traversed and checked by a detector. The strongest undetectable attack found is returned as a results. Theoretically the search space of all combinations is infinite, but we can limit it by employing the monotonicity property of the detector. It means that when an attack is detectable, it can never become undetectable again by increasing its strength.

Note that by combinations we mean combinations with repetitions, because the adversary's ip addresses are undistinguished (vector [7 2 1 1] describes the same attack as [2 7 1 1]).

3.4 Implementation

3.4.1 Flow injection

An inseparable part of the process of attack design is the flow injection. The proposed methods (both gradient method, and exhaustive search) are iterative procedures, which means that before the optimal attack is found, some intermediate attacks must be checked, whether they are detectable. The detection is done by inserting the attack flows into the data representing the traffic. One can imagine the flow injection as a simulation of the actual attack. Naturally, if the flows were injected into the real traffic, it couldn't be undone, once the attack was detected or a better one was found. So it is only simulated instead.

The cycle of designing an attack, simulating it, and finding out whether it already reached the detection boundary, or can be further increased, is obvious. Now we will discuss the details of the flows insertion.

The target is identified by its ip address and its port number. Generally, the variables that can be adjusted by the adversary are the source ip address and the source port number. But usually the source port number is set by operating system and is set to different values for each connection. If each of the attack flows was for example a http request, they all would have different port numbers. We assume this case, when inserting the attack flows, because it is a more difficult position for the adversary. So if we successfully design an attack under this condition, it is a stronger evidence than if we were successful with an easier attack.

The last variable: ip addresses used and the number of flows injected from them, are defined in the attack design process, so now we have all information necessary for attack injection.

However, for better understanding how the traffic works, it is helpful to know, that flows usually exist in pairs of request-response. If the request is

successful, i.e. the destination ip address and port are available through the network, there always follows the response. This fact is extremely important for modelling artificial traffic (i.e. when designing an attack), because if only flows of one direction with no response would be created, it wouldn't be a realistic traffic.

There are more domain-specific specialities of network traffic such as linking of http requests with requests to the DNS servers (web browsing), etc. These links are not concerned when artificial attack flows are generated.

3.4.2 Detector optimization

Routine working cycle of the detector consists of updating the detector (model and thresholds) every 5 minutes (when new data come in), extracting features from the data of actual traffic and performing the detection on them. The whole process was described in more details in Chapter 2.2.

For the purpose of designing an attack, the life cycle is a bit different. The detector doesn't need to be updated, only the input data change when an attack is injected. The general procedure could generally look like this:

1. inject attack into actual traffic (simulation)
2. extract features
3. perform detection on all attack ip addresses
4. reset actual traffic and repeat until best attack is found

However, this is a very inefficient way of designing an attack, because not all of the operations need to be repeated each iteration. The background traffic (the rest of the traffic, excluding adversary's and target ip addresses) doesn't change upon injection of attack. We can thus extract the detection features beforehand and update only those values affected by the inserted flows.

This simple modification speeds up the process substantially and turns the attack design from being rather theoretical to practically applicable in terms of time complexity. However, there is one more option how to accelerate it by a simple time-memory trade-off. The outputs of the detector can be *cached*, which means when we receive a response from the detector stating about an attack it is legitimate/malicious, the response is stored and next time the detector is queried for the same attack, which happens quite often because of the repeated gradient computation, it will restore the old result from memory instead of performing the demanding detection again.

Also before adding new attack into the cache or before looking it up, the attack needs to be sorted. The reason was already mentioned in Chapter 3.3 - the attack [7,2,1,1] is equivalent with [2,7,1,1], so by sorting it we circumvent this redundancy.

Chapter 4

Experimental evaluation

This chapter is dedicated to verification of the assumptions we have made during the planning of the attack and to the evaluation of the proposed procedure for designing network attacks.

First we will present the data set, on which the experiments will be performed. Then we will proceed with comparison of the attack scenarios described in Chapter 3.1. Based on the results of this comparison, the more suitable scenario will be chosen for the next test. We will show, that even though our method is an application of local optimization tool and the objective function is not convex, we are able to consistently obtain optimal results and successfully design an optimal attack against the presented detector of anomalies in network traffic.

We will also compare the gradient method with the exhaustive search in terms of efficiency, which can be expressed in number of detector calls necessary for finding an optimal attack.

4.1 Real traffic

The NetFlow data, which were made available for this thesis, consist of a 3 hours and 25 minutes long capture of traffic in our university network. It contains over 6.5 million entries (almost 800MB of data).

Worth mentioning is that the capture contained a real attack on the university server. The attack comprised of *horizontal scan* to identify victim, *password cracking* to gain access to the victim, and finally from *downloading* a large file simulating stealing the data.

Preprocessing

The raw data in NetFlow format have to be preprocessed before they can be used for further analysis. The preprocessing consists of parsing, extracting the desired information and storing in a matlab *.mat* file. The detection is performed on the source and destination ip addresses and ports. Together with the number of flows in each connection, only these 5 data items are stored for each entry in the NetFlow log.

4.2 Comparison of attack scenarios

Two possible attack scenarios were described in Chapter 3.1. They differ from each other in how the detector and the data of traffic are used for the design of attack. First scenario is more compact, while the second scenario spreads to more time windows and thus is expected to better capture the unpredictability of the traffic, producing weaker but less detectable attacks.

The comparison was executed in two separate runs of exhaustive search, each of them for a different scenario. Both scenarios have the same form: first detector is used for attack design, the prepared attack is then injected into the following time window and verified by a second detector.

During the runs four values were tracked: number of true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN). These values are often used in the area of classification, although their meaning is slightly shifted here. In this test we observe in which cases both the first detector, used for attack design, and the second detector, used for the

verification, give the same response (then it is TP if both detect, TN if both miss the attack) and where each of them gives different response (FN if attack detected during verification, FP otherwise). See Table 4.1 for a better understanding of these terms.

		1st detector	
		attack	no attack
2nd detector	attack	TP	FN
	no attack	FP	TN

Table 4.1: Explanation of the values tracked in the test

In Table 4.2 the four tracked values are charted. Rows express the number of IP addresses used in the attack (dimension) and the column in the middle shows the number of different attacks for the given dimension.

Dim	Scenario 1				total	Scenario 2			
	TP	TN	FP	FN		TP	TN	FP	FN
1	1	8	0	0	9	1	7	1	0
2	9	34	0	2	45	11	28	6	0
3	107	9	0	49	165	120	7	2	36
4	480	0	0	15	495	482	0	0	13

Table 4.2: Comparison of attack scenarios. Scenario 2 exhibits lower number of FN, which means it produces more robust attacks.

For both scenarios the exhaustive search was limited to 4 attack IP addresses with 9 flows injected from each of them in maximum. Beyond these limits any attack would be marked as malicious by both detectors from both scenarios, so we don't have to inspect these combinations.

We can see from the TN column, that in scenario 1 a higher number of attacks was declared feasible by both detectors. However, this partial advantage is outweighed by the FN column, which expresses the number of attacks, which were legitimate for the first detector, but malicious for the second one. For a more robust attack we need to minimize the number of false negatives, which is clearly lower in the second scenario. Comparing

the false negative rate ($\frac{FN \cdot 100\%}{FN+TP}$) we get almost 10% versus 7.4% favoring scenario 2.

Our expectations about the scenarios described at the end of Chapter 3.1 were confirmed by the test, so we can commit to scenario 2 and start using it in further evaluation. Just to clarify, the second detector is very important for discovering how well an attack is designed in terms of unpredictability of traffic (i.e, how robust the attack is). Though, optimizing the attack for *robustness* is not the aim of this thesis and is well out of the scope. For this reason we use just one detector for attack design and we optimize for *strength*.

4.3 Single window attack

We have put some serious challenges on our gradient method by placing this continuous-space optimization tool into a discrete space and requiring it to find global maximum of a non-convex objective function. Can the proposed method fulfill these requirements? And what is the right way to verify them?

We could track the intermediate points of the search and decide, whether it behaves *reasonably*. This could be maybe done, but it would be very inefficient and probably subjective. We need a statistical evidence, that the method does, what we expect it to.

Local optimization methods are known to get stuck in local optima, but sometimes they accidentally end up in a global optimum. It only depends on the starting point of the search. In the next test we will run the gradient method from various starting points and watch the results. The results will be averaged and compared with the exact optimal values obtained by exhaustive search.

To gain the necessary statistical significance of the test the starting points

will be composed of all the possible combinations for an attack under the same limits as in exhaustive search.

In Table 4.3 you can see the results of the test. Rows express the number of ip addresses used in the attack (dimension). In the first column the exact optimal values of attacks are printed. These values were acquired by exhaustive search through all possible attacks. The second column contains the average result values of our proposed gradient method. For completeness the number of combinations of all possible attacks and thus the number of different starting points for the test is specified in the third column.

Dim	Exhaustive search	Gradient method	Combinations
1	7	7	7
2	14	14	28
3	14	14	84
4	11	14	210

Table 4.3: Comparison of Gradient method with Exhaustive search

Besides the fact, that the gradient method was able to find the optimal attack from all the 329 tested starting points, there is one interesting detail worth noting. The attack from 4 ip addresses has the exact optimal value equal to 11, while the gradient method has an average of 14 over all 210 starting points. How is it possible to reach a *more than optimal* value?

Explanation is simple and shows the flexibility of the gradient method. Exhaustive search generates only the combinations, where all the ip addresses have non-zero number of flows. This limits the maximum strength of the attack, because an attack lead from 4 ip addresses is easier to detect. Gradient method has reached better results, because it will find the ideal distribution (possibly by setting some ip addresses to zero flows) even if the ip addresses have more than zero flows at the beginning of the search.

4.4 Efficiency of the procedure

After proving that our method finds optimal attacks, we were interested in stating how efficient the method is. Because measuring computation time is dependent on the testing environment, mainly on the computation power of the testing machine, we decided to compare the efficiency in terms of number of detector calls.

The gradient method was again compared to the exhaustive search. In Chapter 3.4.2 we described, that the detector is optimized by caching its outputs. It ensures, that the detection is performed at most once per each attack combination, and we can measure only these unique detector calls. The look-ups in the cache are not regarded in the test, because they are much faster relative to the actual detection.

In exhaustive search the number of detector calls is equal to the number of different possible attacks. In gradient method this number depends on the starting point of the method. To address this variability we ran the gradient method from all possible starting points (i.e., all the attacks examined by exhaustive search) and averaged the results.

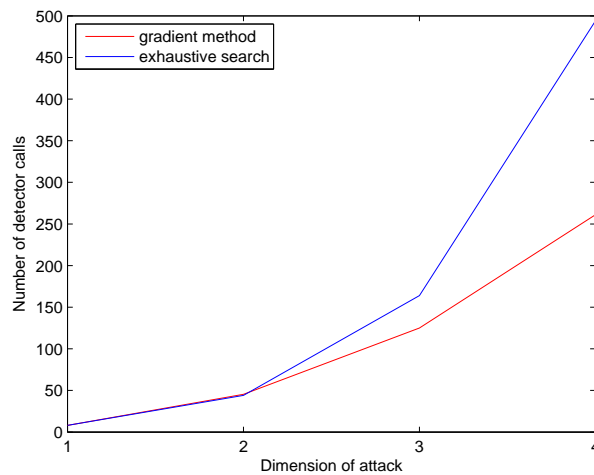


Figure 4.1: Gradient method and exhaustive search compared in number of detector calls per attack dimension.

On Figure 4.1 the results are plotted according to the dimension of the attack, which is equal to the number of ip addresses involved in the attack.

We can observe that in low dimensional attacks both methods are comparable in the number of detector calls, but in higher dimensions the gradient method performs better than exhaustive search. Note that there is still possible speed-up in the gradient method. For example the line search procedure, which finds the decision boundary with linear time complexity relative to the boundary's distance from the origin, could be replaced by a binary search, which has logarithmic complexity. However, we leave such improvements to the future work.

Chapter 5

Conclusion

The goal of this thesis was to design a procedure finding an optimal attack against a particular network detector based on anomaly detection. We defined optimality of an attack in the sense of attack strength, expressed in number of flows injected to the target, with condition of undetectability of the attack.

The advantage of our approach over the prior art is that the intrusion detection system is treated as a black-box, which means that the attacker does not need to know anything about its internals. The mere binary output is sufficient to find the attack.

We found a suitable representation of attacks, which allowed the use of gradient method for optimization. We encountered various problems connected to the discrete nature of the domain of network communication. Other serious obstacle was the non-convexity of the decision boundary of the detector.

In Chapter 4 we proved by various tests, that our solution of the encountered problems, as well as solution of the whole task, can be considered successful. The tests included a comparison of two different attack scenarios (4.2), which have been devised in Chapter 3.1. Based on this comparison the more suitable attack scenario was chosen, producing more robust attacks. We also proved, that our proposed method consistently gives optimal

results (4.3). The optimality was checked against exact values obtained by exhaustive search.

We also tested the efficiency of the gradient method compared to the exhaustive search (4.4). We found out that in attacks, which involve less ip addresses (1 or 2), both methods are comparable in efficiency, because the search space is small, however for more ip addresses (3 or 4) our proposed method performs significantly better. A speed-up is still possible and a concrete improvement was suggested for future work.

As was suggested in Chapter 3, it is possible to analyse the detection systems by designing attacks against them. Our method can be used to find vulnerabilities in IDS or to assess its level of security. More informed IDS designers can then find better countermeasures long before any real attack occurs.

Our proposed method does not take into account some of the domain specifics, such as communication with DNS servers. These specifics could be viewed as side effects in real attacks, which make them more abnormal and easier to detect. By not including the side effects to our method we produce stronger attacks and thus when assessing the level of security of a system we acquire the upper limit. This is important, because in security the most interesting information is the worst case scenario results.

There are some challenges left in the task of designing attacks. For example it would be interesting to study the effect of unpredictability of the network traffic on the feasibility of designed attacks. We optimized on strength of the attack, but it is also possible to optimize on robustness. Such attacks would have higher chance of not being detected.

Although we were successful with the single window attack, we would like to extend its duration to multiple time windows. We argue, the gradient method could work in general for designing attacks of unlimited duration, and hopefully we will be able to implement the extension in the future.

Bibliography

- [1] Biggio, B., Corona, I., Nelson, B., Rubinstein, B. I. P., Maiorca, D., Fumera, G., Giacinto, G., Roli, F.: *Security Evaluation of Support Vector Machines in Adversarial Environments*. CoRR abs/1401.7727 (2014)
- [2] Comesana, P., Perez-Freire, L., Perez-Gonzalez, F.: *Blind Newton Sensitivity Attack*. In: Proceedings of SPIE 6072, Security, Steganography, and Watermarking of Multimedia Contents VIII, 60720E (2006).
- [3] Barreno, M., Nelson, B., Sears, R., Joseph, A.D., Tygar, J.D.: *Can machine learning be secure?* In: ASIACCS '06: Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security, pp. 16–25. ACM, New York, NY, USA (2006).
- [4] Barreno, M., Nelson, B., Joseph, A., Tygar, J.: *The security of machine learning*. Machine Learning 81, 121–148 (2010).
- [5] Kloft, M., Laskov, P.: *Online anomaly detection under adversarial impact*. In: Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS), pp. 405–412 (2010).
- [6] Nelson, B., Rubinstein, B.I., Huang, L., Joseph, A.D., Lee, S.J., Rao, S., Tygar, J.D.: *Query strategies for evading convex-inducing classifiers*. Journal of Machine Learning Research 13, 1293–1332 (2012).
- [7] Pevný T., Reháč M. and Grill M. Identifying suspicious users in corporate networks. In *IEEE International Workshop on Information Forensics and Security 2012*.

Appendix A

Attachments

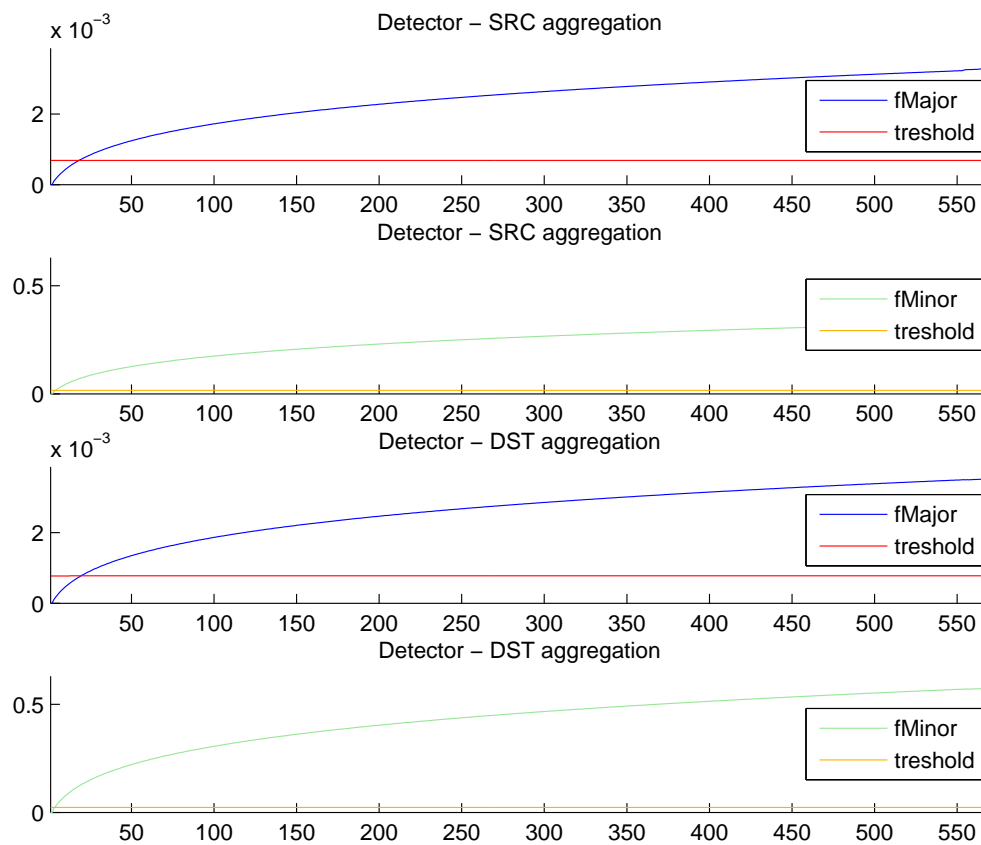


Figure A.1: Monotonicity check: SSH Password Cracking

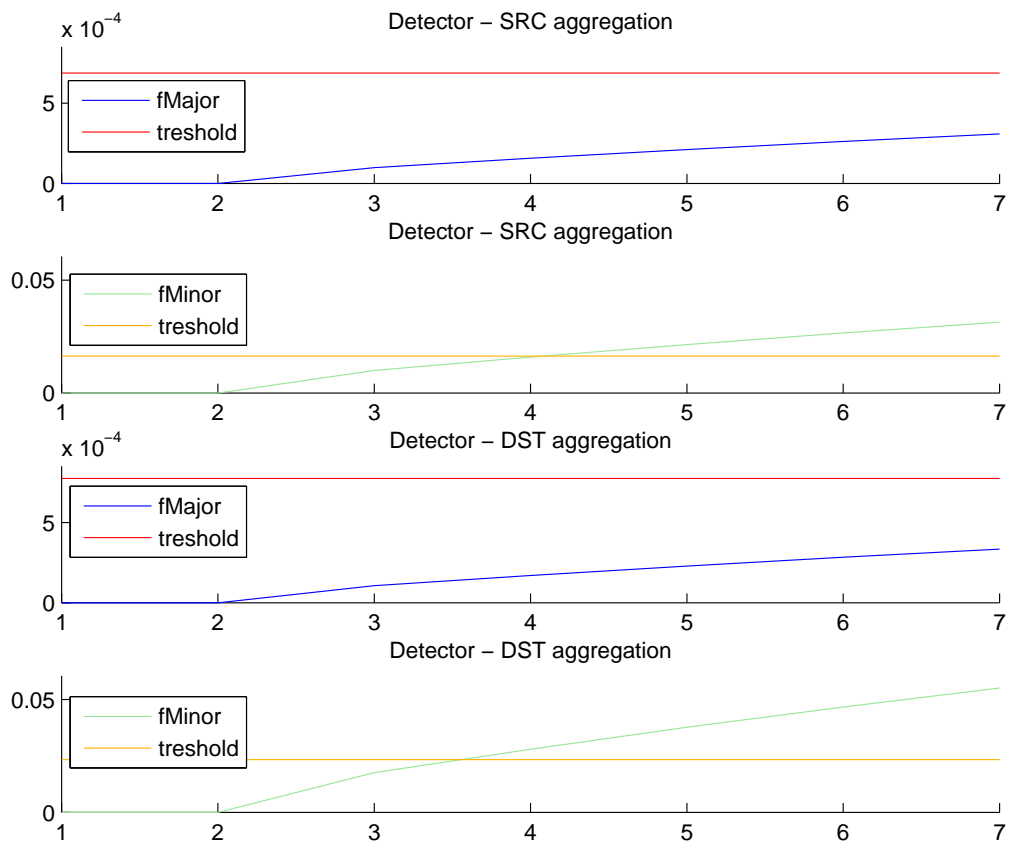


Figure A.2: Monotonicity check: Download

Appendix B

Attachments on cd

B.1 Practical part

original NetFlow logs

precomputed data

source code files¹

B.2 Theoretical part

pdf file containing the text of the thesis

Latex source code of the thesis

¹With courtesy to Mr. Pevný a file *loader.cpp* is included, who allowed this function to be used in this thesis. It serves for parsing network traffic to matlab.