Towards scalable network host simulation

Jan Stiborek
Czech Technical University in Prague, Czech Republic
CISCO Systems, Inc.
jastibor@cisco.com

Martin Rehák
CISCO Systems, Inc.

Tomáš Pevný
Czech Technical University in Prague, Czech Republic

ABSTRACT

Anomaly detection techniques in network security face significant challenges on configuration and evaluation, as collecting data for accurate analysis is difficult or nearly impossible. One viable approach is to avoid live data collection and replace if by the agent-based simulation of the network traffic with models of user's behavior. In this paper we propose three approaches differing by the level of detail with which user behavior is modeled. They are well suited for generating NetFlow/IPFIX data that can be used for evaluation and optimal configuration of anomaly detection techniques. First two techniques use simple statistical model that is easy to implement and does not require large amount of training data. The third leverages sophisticated model of the user's behavior covering different aspects of the network traffic not captured by the simpler models. In experimental evaluation it is demonstrated that the complex model generates data indistinguishable for current state-of-the-art anomaly detection methods from the real-world samples, which makes it well-suited for their evaluation and configuration.

Categories and Subject Descriptors

[**Network**]: Network simulations; [**Security and privacy**]: Intrusion detection systems

General Terms

Security,Algorithms

Keywords

Net-Flow simulation,anomaly detection,evaluation

1. INTRODUCTION

This paper presents a user's behavior models for agent-based simulation network traffic that can be used for evaluation of an intrusion detection system (IDS) composed of NetFlow-based anomaly detectors [1, 2]. The main reason why researchers focus on the NetFlow data is that it captures only high level statistics which have been shown sufficient [3] for detecting threats and thus allows to process data from high speed backbone networks which cannot be achieved with other techniques (e.g. deep packet inspection).

The main and prevalent problem is the lack of ground truth data which is due to: (i) unrealistic properties of the ground truth generated in the closed lab without real-world background traffic; (ii) huge volumes of data to label in real environments; and (iii) varying characteristics between different environments making model transfer difficult.

We show that a viable solution to the above problems is to simulate the network traffic. Existing approaches can be divided into following areas: (i) context-free packet generators that correctly capture properties of individual packets but not the high level properties of user's behavior [4–8]; (ii) testbed systems that can generate the network traffic on packet level but are extremely difficult to setup [9]; (iii) lightweight simulator generating only the NetFlow data [10–13].

Our work belongs to the last category of lightweight simulators. Our goal is a realistic simulation of the background traffic (NetFlow records) with correct high level properties, for which we propose three techniques. The first two uses a simple statistical model to generate training data. They are both easy to implement but do not capture sophisticated aspects of user's behavior, such as time variance of the user's behavior, dependency between inter-flow features, etc. The third one, which we advocate, addresses these deficiencies and based on our results is able to mimic the user's traffic in a way that even a combination of state-of-the-art detection algorithms is not able to distinguish.

2. RELATED WORK

Network simulation provides viable approach for ground truth generation as discussed in [14, 15]. Authors argue that static data sets and manual labeling suffer serious problems that network simulation can overcome (manual labeling does not scale, bias in the labeled data, privacy issues with sharing of labeled data, etc.).

First group of traffic generators are context-free packet generators that generate full packet captures, such as NS-2 and NS-3 simulators [4, 5], OMNET++ [6] or NeSSi [7] and its ancestor NeSSi2 [8]. However, these tools were primary designed to test low level algorithms (e.g. routing algorithms) and do not model the high level statistics necessary for evaluation of an IDS system. Moreover, these tools require precise configuration which drastically increases the cost of their deployment.

Next option for generating ground truth data is the testbed systems that emulate the behavior of the network. Such solution was proposed in project LARIAT [9] where authors used virtual machines with service that emulates user's behavior. However, as authors argued, LARIAT requires careful tuning which lowers the chance for practical deployment (authors claim that it takes approximately four months to

setup LARIAT for evaluation of new IDS system). Our work is inspired by the high level design of the LARIAT system. However, the key difference between our work and LARIAT system is that we simulate the user's behavior with statistical models trained from the data and LARIAT uses emulation which requires manual setup.

Another approach is the flow-level simulation that models particular type of behavior, e.g. SSH brute-force attack, which is mixed with background traffic and used for evaluating an IDS system. Such approach is proposed in [10, 11].

The most advanced and the most challenging approach is to model set of users or the whole network and generate the complete dataset (without the need to mix with background traffic). Such approach is well suited for evaluation of an IDS system because the data (if generated correctly) mimic the behavior of the whole network, can be arbitrarily tuned (duration, volume of traffic, number of users, etc.) and can be shared between researches without any privacy concerns. In [13] authors propose host-agent based simulator where single class of network behavior is represented by an autonomous agent (trained from sample traces or malicious traffic model). The traffic is then generated from interaction between agents. In [12] authors propose approach based on modified version of *Traffic Dispersion Graphs* which define the connectivity patterns for given service. The port-based TDGs augmented with additional statistics such as distribution of packets, bytes or duration serve as a model that is able to generate the traffic traces for the whole network. Our solutions use similar statistics to describe the communication between single user and requested service but the connectivity pattern is modeled by probability distributions rather than TDGs. It is designed to precisely model the behavior of single user, not the whole network. However, we can couple together multiple instances of models with different training data and simulate the behavior of the whole network.

3. BASIC MODELS

The design of a simulation model needs to address the common trade-off between complexity and performance. Before introducing our key model proposal in the next section we first discuss two simpler models. We show that simplifying assumptions about NetFlow traffic allow for models of low complexity. At the same time we will show where simplified models fail. The identified flaws then inspire the definition of the improved model presented in the next section.

3.1 Random sampling

The simplest approach to simulation of behavior of a single user is to generate standard NetFlow fields (as listed in Table 1) independently inspired by technique proposed in [16] and technological solutions such as BreakingPoint[1]. Such approach does not take into account any properties of the NetFlow (e.g. distribution of bytes, distribution of source ports, etc.), relation between fields of the NetFlow (e.g. bytes/packet ratio) or relations between individual NetFlows (e.g. request/responses relations). The only condition that has to be satisfied is the validity of NetFlow, i.e. all fields have to be in their allowed ranges and the following condition must be met

$$0 < \text{number of bytes} \leq \text{number of packets} \times 65535. \quad (1)$$

Note that the only parameter of this algorithm that has to be specified in advance is the IP address of the simulated user.

The random sampling algorithm generates all but two individual NetFlow features randomly with respect to the condition of validity of the generated NetFlow. The two exceptions are the thinking time and source and destination IP addresses. Instead of generating the starting time directly we generate the user's thinking time—time delay between two consequential NetFlows. The new starting time is then computed as sum last starting time and the current thinking time. This approach allows us to generate infinite stream of NetFlows. Similarly to the thinking time, the source and destination IP addresses are not generated directly. Instead, we randomly choose whether given flow is request or response. If the flow is generated as request the source IP field is set the value of user's IP address and the destination IP is chosen randomly. In the case of response it is vice versa.

The main benefit of this algorithm is its independence on any training data or manual tuning, because the only parameter that has to be set in advance is the user's IP address. However at the same time, the complete randomness is the main disadvantage because it can generate completely unrealistic data. Therefore, we use this approach only for syntax testing and as a baseline for the comparison to more sophisticated methods.

Table 1: List of NetFlow fields.

Field name	Description
Starting time	Time stamp of the first packet of the flow
Duration	Length of the flow
Protocol	TCP, UDP, ICMP, etc.
Source IP	IP address of the source
Source port	
Destination IP	IP address of the target
Destination port	
Bytes	Number of bytes transferred in the flow
Packets	Number of packets transferred in the flow
TCP flags	Not used for non-TCP protocols

3.2 Sampling with independent intra-flow relations—marginal model

The marginal model provides different approach to NetFlow simulation. It uses training data of a single user in order to train the statistical model of individual NetFlow features (e.g. distribution of bytes or distribution of user's thinking times, etc). Unlike the random sampling discussed above, the marginal model considers NetFlows in request/response pairs[2]. Therefore it is able to partially model inter-

[1]http://www.ixiacom.com/breakingpoint

[2]The model focus on the modeling of user's behavior and thus we consider outgoing flow as *request* and incoming flow

Table 2: List of NetFlow features to be modeled in order to create NetFlow data that correctly reflect requests and responses. Note that TCP flags for request and response are empty for all non-TCP NetFlows.

Feature name	Description
Client's thinking time	Time difference between two consequential client's requests
Client port	Source port of the request
Request bytes	Number of bytes in request
Request packets	Number of packets in request
Request protocol	TCP, UDP, ICMP, etc.
Request flags	TCP flags in request
Request length	Duration of request
Server thinking time	Time difference between request and response
Server IP	IP address of server
Server port	Port number of service used by user
Response bytes	Number of bytes in response
Response packets	Number of packets in response
Response length	Duration of response
Response flags	TCP flags in response
Has response	Is there corresponding response to the request?

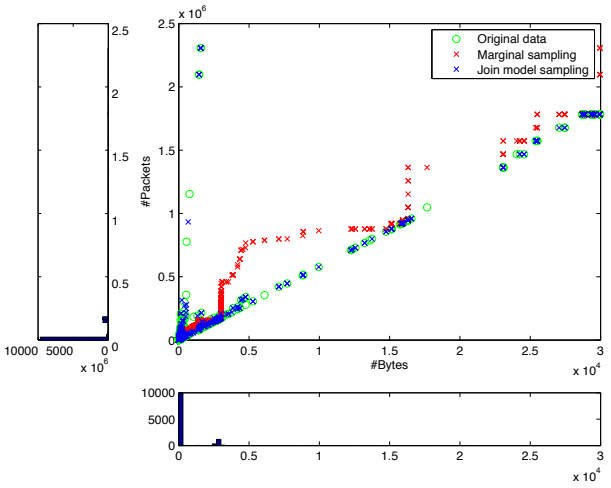

Figure 1: Artifacts of marginal sampling in real-life example. This figure shows that marginal sampling (red crosses) is not able to mimic the data correctly and thus creates serious sampling artifacts—it creates data that did not appear in the original data—whereas the joint model sampling that respects the dependency between features generates the data correctly.

flow relations (the relation between individual flows) as well because it captures request/response relations but not the sequential character of the user's behavior. For example, it is able to model the HTTP request/response pairs but not the download of the whole Google home page. Next, this model assumes that the modeled features are independent and thus it does not take into the account intra-flow relations (e.g. bytes/packets ratio, etc.). The full list of modeled features is listed in Table 2. This assumption is a limitation that affects the variance of the internal model and can cause serious sampling artifacts (see Figure 1).

Marginal models are created as follows. The first step of the sampling algorithm is the preprocessing of the training data during which we pair the requests with the corresponding responses. We assume that the user behaves only as a client and thus every NetFlow with user's IP address as source IP is considered as a request. The corresponding response is matched as NetFlow with following properties:

$$\begin{aligned} \text{source IP}_{\text{response}} &= \text{destination IP}_{\text{request}}, \\ \text{source port}_{\text{response}} &= \text{destination port}_{\text{request}}, \\ \text{destination IP}_{\text{response}} &= \text{user's IP}, \\ \text{destination port}_{\text{response}} &= \text{source port}_{\text{response}}, \\ \text{protocol}_{\text{response}} &= \text{protocol}_{\text{request}}. \end{aligned} \quad (2)$$

Note that there is a maximal delay τ between request and response in order to avoid incorrectly paired flows. In current settings the τ is set to 2 seconds.

After the data preprocessing the model estimates the distributions of all individual features using non-parametric estimates (histogram for continuous features or relative frequencies for categorical features).

Once model training is finished, request/response pairs are sampled as follows. At first we randomly choose whether there will be a response or it will be only request (the feature *Has response*). Next, values of individual fields are sampled from distributions of corresponding features estimated from the training data. Note that starting times of the request and response are not generated in the same manner. The starting time of the request is generated as sum of the last starting time and the user's thinking time (estimated from the training data) and the response starting time is computed as sum of starting time of the request and thinking time of the server (again, estimated from the training data). This approach is similar to the Random sampling discussed in Section 3.1. The source address in the request/destination IP in the response is set to the user's IP address (parameter of the algorithm) and the destination address in the request/source address in the response is sampled from the distribution estimated from the training data.

as *response*.

4. TIME VARIANT JOIN PROBABILITY MODEL

In this section we will discuss the main contribution of this paper. In previous sections we have described the simulation techniques that uses simple statistical model and thus miss more complicated aspects of the user's behavior which leads to following issues:

- no intra-flow relations—single HTTP connection will not likely transfer 60GB in 2 seconds,
- no time variant restrictions—user activity differs during the night and day,
- no reflection of sequential character of the user—HTTP request precedes a DNS request

The main problem of such approaches is the inability to model *relations between individual features*. The sampling

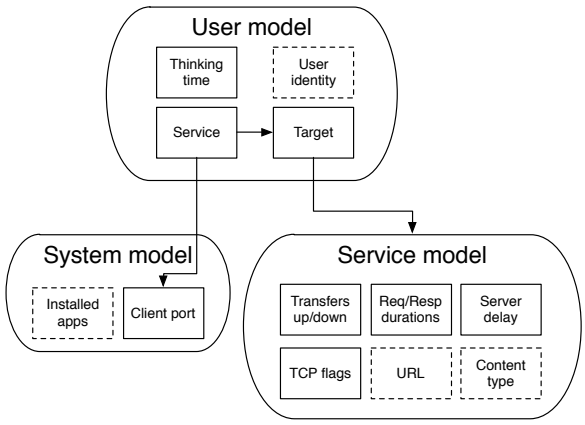

Figure 2: The schema of the proposed model. The figure shows three main components and their relations. The separation of user model and underlying system and service model is inspired by LARIAT project [9] which uses model of the user's behavior on top of the real emulated operating system and services. Note that the dashed boxes are omitted from our implementation in order to reduce the complexity of the current implementation.

with marginal model assumes that NetFlow features are independent, which if not satisfied leads to serious sampling artifacts (see Figure 1).

Next problem is caused by *changes of the user's behavior*. Usually during the night there is no network traffic generated by the user's machine or the volume of the traffic is very low (only the automated behavior of the machine, e.g. periodic updates). However, during the working hours its network activity increases rapidly and it fluctuates during the working hours. For example during the lunch break there is a drop in the volume of the network traffic followed by a spike when users return to work. Note that the profile of the user's behavior changes through the day as user uses different services at different time of day. The model described in previous section does not reflect such changes and thus the quality of the generated data is lower.

Third problem that we have to consider is the *sequential character of the user's behavior*. The typical example is the e-mail usage. At first, user's e-mail client has to resolve the domain name of the IMAP server which will be seen in NetFlows as communication with the DNS server on port 53 over UDP protocol. Next, it synchronizes the e-mails in user's folders—it is represented as opening connection to the IMAP server on port 143 over TCP protocol. In the received e-mails there can be an interesting link that the user clicks to visit. This will appear as request to the DNS server that resolves the domain name from the link followed by a number of different connections to port 80 over TCP protocol to the HTTP server. From this example we can see that users exhibit sequential behavior and thus probability of two consequential services is not independent.

4.1 Model structure

In order to address all the problems discussed above, we propose a model (see Figure 2) composed from three components, all parametrized by day time.

The first component—*user model*—describes different aspects of the user's behavior, such as the timings between requests (*thinking time*), which *service* will he use and which *server/target* will he contact, its identity, etc. Note that the user does not necessary have to be human but some automated agent in the operating system as well (e.g. automatic updates).

The second component—the *system model*—models the automatic behavior of the user's operating system. This component models the process of assigning client ports.

The last component—*service model*—models the behavior of the remote service contacted by the user. This includes the amount of data transferred between client and server (bytes and packets as well), the duration of the response, the delay of the server and TCP flags (if the connection uses TCP protocol).

In order to simplify interactions between components and its internal models we adopt following assumptions:

a) The thinking time (T) depends only on the daytime and not on other aspects of user's behavior,

b) the client port ($cPort$) depends only on the service and day time—source ports for the outgoing connections are assigned by the operating system without any user' interaction. However, most operating systems simply increment the last used port until the range is depleted and thus for different daytime different range of ephemeral ports is used. The dependency on service is caused mainly by long persistent connections that are split by the NetFlow probe into several NetFlow records. All these flows have the same client port and in the statistics it appears as the service prefers single source port.

The negative impact of the assumption is that proposed model is not able to correctly capture the periodical behavior of particular service, however, it is outweighed by the benefit of the simplification of relations and internal models of individual components. The relaxation of adopted assumptions is left to future work.

4.2 Individual model components

The simplified interactions between internal models are following:

- *Thinking time*: Thinking time does not have any interaction or dependency on other components (see Assumption a)).
- *Service and target*: Target contacted by the user depends on the service requested by the user. This corresponds to the fact that there are specialized servers that serves only some (or even a single) services (HTTP server, database server, etc).
- *Client port*: Client port depends only on the service requested by the user (see Assumption b)).
- *Remote service model*: Model of remote depends on the service and target contacted by the user. For example different services on different servers has different profile (volume of the network traffic, server delay, content type, etc.)

In the following we will discuss internal models of each component separately.

Table 3: Set of features that describe the behavior of a service. Note that TCP flags for request and response are empty for all non-TCP connections.

Feature name	Description
Request bytes	Number of bytes in request
Request packets	Number of packets in request
Request protocol	TCP, UDP, ICMP, etc.
Request flags	TCP flags in request
Request length	Duration of request
Server thinking time	Time difference between request and response
Response bytes	Number of bytes in response
Response packets	Number of packets in response
Response length	Duration of response
Response flags	TCP flags in response
Has response	Is there corresponding response to the request?

4.2.1 Thinking time

The approach with marginal model we have proposed, models the starting time of the request as a sum of the starting time of the previous request and the thinking time of the client. However, such approach cannot be directly adopted in this model due to the fact that if we parametrize the thinking time with the daytime, it does not capture the time intervals when the activity of the user is very low. This causes artifacts that results in unrealistic data. For example, when first request occurs at 09:35 and next occurs at 11:20 we cannot compute the thinking time of flows between 10:00 and 11:00.

To avoid this issue, we do not estimate the thinking time directly. Instead we model the number of request n generated by user in given time interval[3]. The thinking time is then computed as follows

$$T = \frac{L}{n} \tag{3}$$

where L is the duration of usual time interval in seconds.

To denoise the input data—number of requests $[n_1, \ldots, n_i, \ldots]$ in five minute time window—we smooth the data with sliding window as follows

$$n_t = \frac{1}{l} \sum_{i=t-\frac{l}{2}}^{t+\frac{l}{2}} n_i, \tag{4}$$

where l is the width of the sliding window. It controls the smoothness of the estimate—if the window is too long, the value does not follow the trends in the data, and if it is too short the estimated value is too noisy.

Next, we divide the list of the number of requests $N = [n_1, \ldots, n_i, \ldots]$ into one-day long sets forming matrix N' defined in Equation 5. For every time interval $t \in [1, \ldots, 288]$, that is represented by a row in matrix N', we have k samples where k is the length of the training data in days.

$$N' = \underbrace{\begin{pmatrix} n_1 & n_{289} & \cdots \\ n_2 & n_{290} & \cdots \\ \vdots & \vdots & \\ n_{288} & n_{576} & \cdots \end{pmatrix}}_{k} \tag{5}$$

We estimate the distribution of number of requests for interval t with histogram $\mathcal{H}_t$ with non-linearly distributed bins $[1, 11), [11, 21), [21, 41), [41, 81), [81, 201), [201, \infty)$. Furthermore, for every bin of the histogram $\mathcal{H}_t$ we define distribution of values. If the number of samples that fit into this bin is 1, we assume uniform distribution of values in this particular bin. If there is more samples, we assume normal distribution with parameters estimated from the samples that fit into the bin. This setup helps us to overcome the lack of data as we have only k samples.

The sampling procedure of the thinking time for given time interval is listed in Algorithm 1.

Algorithm 1 Sampling of thinking time

```
procedure THINKINGTIME(t) ▷ Sampling thinking time
                                for time interval t
    b ~ H_t  ▷ Select bin with respect to distribution H_t
    if |b| = 1 then ▷ If there is only one training sample
                        that fits in the bin b
        n_t ~ U(b_a, b_b)        ▷ Sample from uniform dist.
                                   defined by boundaries
                                   b_a and b_b of the bin b
    else
        n_t ~ N(μ, σ) ▷ Sample from normal distribution
                          with parameters estimated from
                          training samples
    end if
    T = L/n_t ▷ Compute thinking time, L is the length
                of the time interval (in our case L = 300)
    return T
end procedure
```

4.2.2 Service and target

One of the problems of the models described in previous sections was the inability to precisely model the sequential character of the user's behavior. To address this issue we separate the probability of the service defined by following equation

$$p(s|t) \tag{6}$$

where $s \in S$ represents the service (the server port and protocol tuple) requested by the user and h is the day time. The sequential character is naturally modeled by *Markov chain* [19].

Next, we have to discuss model that describes which target will be contacted by the user. In Section 4 we have defined that the target contacted by the user depends on requested service and the day time. This is summarized into following probability

$$p(\text{dIP}|s, t) \tag{7}$$

where dIP is the destination IP of contacted server (target), s is the requested service and t is the day time.

[3]The length of the time interval is 5 minutes—the usual length of the batch in anomaly detection [17, 18].

4.2.3 Client port

As we have discussed above, in order to generate NetFlow data we can simplify the operating system model to model only the client ports. In assumption b) we stated that client port depends on the service and the daytime. Using this assumption we can model the assigning of the ephemeral ports by the probability defined as follows

$$p(\text{cPort}|s,t) \tag{8}$$

where cPort represents the client port assigned by the operating system and service s. This model captures the long term connections to a service that appear in the data as different NetFlows with the same client port (connection to e-mail server, long term SSH connection, etc.) as well as different strategies used by operating system to assign the ephemeral source port to outgoing connections.

4.2.4 Remote service model

Internal models that we have discussed in previous paragraphs addressed only single feature of the component. However in order to capture the relations between individual NetFlow features we model the whole component together by single joint probability defined as follows

$$p(x_s|\text{dIP},s,t) \tag{9}$$

where $x_s \in X_s$ represents the space defined by features of the service listed in Table 3, dIP is the contacted server, s is the requested service and t is the day time. Using this model we can precisely capture the behavior of the service. However, such model captures precisely the behavior that appeared in the data and it is not able to generate completely new values. To overcome this issue we can add the gaussian noise to the Equation 9 and thus generate new previously unseen data. The strength of the noise λ is the parameter of the model and allows us to control how "realistic" the generated data should be.

Before we generate new flows we train the model as described in Sections 4.2.1, 4.2.2 and 4.2.4. Next, we simulate the data as described in Algorithm 2.

Algorithm 2 Sampling of the NetFlow data

```
1: procedure SAMPLEFLOW(length)          ▷ Sampling single flow
2:     F ← ∅
3:     t ← 0                              ▷ Set current time to 0
4:     repeat
5:         T ← thinkingTime(t)            ▷ Sample thinking time
6:         s ← p(s|t)                     ▷ Sample service
7:         dIP ← p(dIP|s,t)               ▷ Sample target
8:         cPort ← p(cPort|s,t)           ▷ Sample client port
9:         x_s ← p(x_s|dIP,s,t)           ▷ Sample remaining features
10:        flow ← t,s,dIP,cPort,x_s       ▷ Build flow
11:        F ← F ∪ {flow}
12:        t ← t + T                      ▷ Increment current time
13:    until t ≤ length
14:    return F
15: end procedure
```

4.3 Possible extensions

In previous paragraphs we have discussed complex solution for generating NetFlow data. However, the general

Table 4: Capabilities of presented models. The table summarizes capabilities of individual models. It shows wheter given property of the traffic can be (✓), can not be (✗) or can be with specific settings ($\frac{1}{2}$) captured by given model. Note that Time variant joint probability model is able to capture the sequential character of the user's behavior when the service model is replaced by Markov chain.

	Random sampling (Section 3.1)	Sampling with marginal model (Section 3.2)	Time variant joint model (Section 4)
Properties of fields of NetFlows (e.g. distribution of bytes, source ports, etc.)	✗	✓	✓
Intraflow relations (e.g. Packet/Bytes ration, etc.)	✗	✗	✓
Interflow relations (e.g. request/response ration)	✗	✓	✓
Changes in the user's behavior	✗	✗	✓
Sequential character of the user's behavior	✗	✗	$\frac{1}{2}$

schema of the proposed model can be extended to capture not only NetFlow data but different types of communication as well. Note also that the the *system model* can be extended to model installed applications. It can affect the number of connection to the server, duration of the request and the thinking time of the server (different versions of an internet browser uses different number of concurrent connections). Another component that can be extended is the *service model*. As it controls the behavior of the remote service it is natural to extend it with application specific features such as URL or content type. However, these extensions, though beneficial, are out of the scope of this paper and will be considered in future work.

5. EVALUATION

In this section we evaluate the quality of the data generated by individual sampling approaches defined in Section 3. We have implemented a set of state-of-the-art detection algorithms to evaluate the simulated data and compared this data with real-world traffic. Using the *Jensen-Shannon divergence* (JSD) [20] we then measure the distance between distribution of anomaly values of real-world and artificially generated data.

5.1 Selected anomaly detection algorithms

The goal of presented model is to generate data for evaluation of an anomaly detection algorithm. Therefore, we have implemented various types of algorithms based on different detection paradigms. This allows us to measure the quality of the generated data under different conditions.

Algorithms proposed by Pevný et al. [21] and Lakhina et al. [22, 23] use the principal component analysis to detect anomalies in the traffic. However, there are several key differences between these methods. First difference is in the features that are used for the definition of the model of the individual detectors. Second difference is the measure used for assigning the anomaly value (Lakhina proposes to use *reconstruction error* and Pevný uses *mahalanobis distance in sub-spaces*). Note that we have implemented four different versions of algorithm proposed by Pevný denoted in the results as *Pevný-f-dIP*, *Pevný-f-sIP*, *Pevný-$f^{\perp}$-dIP* and *Pevný-$f^{\perp}$-sIP* (all described in [21]), and two versions of Lakhina's algorithm where version listed in the results as *Lak.Vol.-sIP* models the traffic with respect to a source IP and version denoted as *Lak.Vol.-dIP* models the traffic with respect to a destination IP.

Second type of algorithm that we have implemented is a modified version of *Minnesota Intrusion Detection System–MINDS* [24]. It uses an internal model of the network traffic but unlike the algorithms proposed by Pevný and Lakhina it does not uses the PCA but measures the difference between last and current time window. In order to overcome the performance issues we have modified the algorithm from the originally proposed version. The modifications are described in [25].

The last group of algorithms does not use any internal model of the network traffic. Method originally published by Kuai Xu *et. al.* [26] uses basic assumption that all network traffic could be classified into several categories using set of static thresholds. In addition to the original algorithm (denoted as *Xu-sIP* in out evaluation) we have implemented modified version (denoted as *Xu-dIP*) that uses complementary features relating to the destination IP.

5.2 Training and evaluation data

To evaluate the quality of the simulated traffic we have used the data recorded on university campus during the one week in April 2013 (further denoted as $\mathcal{D}_{\text{orig}}$). From the recorded data we have selected a set of full-time employees with various user profiles (developers, scientists, managers and administrative staff). We have separated their traffic based on the IP address of selected users and use it as training data for two models defined in Sections 3.2 and 4. The remaining traffic formed the reduced dataset $\mathcal{D}_{\text{red}}$ and was used as background that was mixed with the simulated traffic.

The evaluation of quality of the generated data was separated into two stages. During the first stage we processed the original data $\mathcal{D}_{\text{orig}}$ separately by all anomaly detection methods and for every detection method we estimated the distribution of anomaly values of selected users. These distributions then served in the comparison as a baseline.

In the second stage we simulated the user's behavior using four approaches proposed in this paper: (1) the random sampling (referred as *Random*, see Section 3.1), (2) sampling with marginal model (*Marginal*, see Section 3.2), (3) sampling with time variant joint probability model (*Model*, see Section 4) and (4) sampling with time variant joint probability model with additional noise that affected the sampled data (*ModelN*, strength of the noise $\lambda = 10^{-5}$). The generated data were mixed with the reduced dataset $\mathcal{D}_{\text{red}}$ and separately processed by all anomaly detection algorithms. Next, for every detection algorithm we have again estimated the distribution of the anomaly values of the simulated traffic and measure the value of JSD between the distribution of the simulated traffic and real traffic. This process was repeated 20 times. Next we have used Kruskal-Wallis statistical test on significance level $\alpha = 0.05$ to detect whether the results for different simulation approaches are significantly different or not. The test proved that the results for different simulation approaches are different enough to compare only the mean values.

5.3 Quality of the generated data

The results are summarized in Table 5. It shows that the random sampling generates the least realistic data. The value of JSD is by order of magnitude larger compared to the two remaining models. This confirms the expectations that the random sampling can be used only for syntax testing as we have discussed in Section 3.1. The second approach, the sampling with marginal model, provides significantly better results compared to the random sampling. The results show that correct estimation of the marginal distribution of individual features improves the results by order of magnitude. However, assumptions that (1) all inter-flow features are independent and (2) user's behavior does not depend on the day time clearly do not hold. Therefore, the most advanced approach, the sampling from the time variant joint probability model, provides the results on average 2.3× better than sampling with marginal model. The last approach shows that by adding the low volume of Gaussian noise into the model we can generate data that are completely new but still follow the original user profiles. Such approach is important for testing the detection boundaries of the detection algorithms.

The last results shown in Figures 3 and 4 visualize the distributions of anomaly value of real data and data sim-

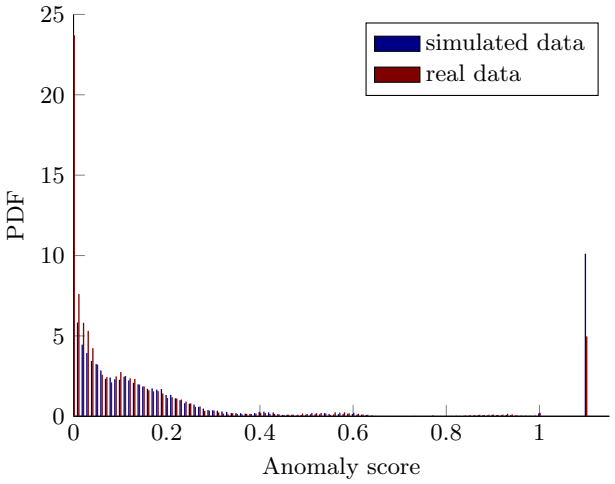

Figure 3: Distribution of anomaly values of Pevný-f-dIP method for the real traffic and traffic simulated by the time variant joint probability model.

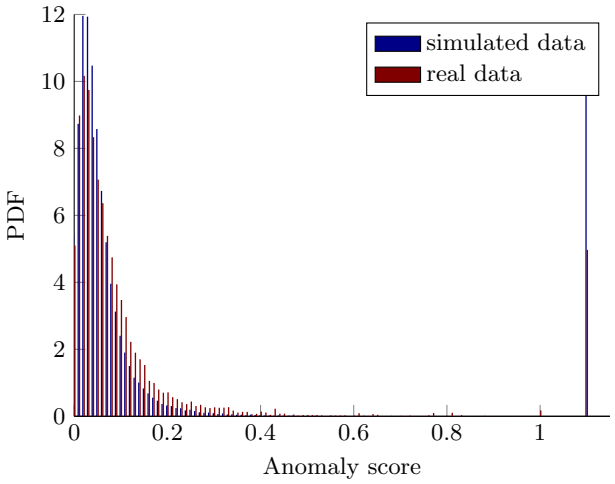

Figure 4: Distribution of anomaly values of Pevný-$f^{\perp}$-dIP method for the real traffic and traffic simulated by the time variant joint probability model.

Table 5: Jensen-Shannon divergence between distribution of anomaly values of real and simulated traffic.

Detection alg.	Model	Marginal	Random	ModelN
Pevný-f-dIP [21]	0.0133	0.0324	0.4596	0.0128
Pevný-f-sIP [21]	0.0146	0.0312	0.4780	0.0122
Pevný-$f^{\perp}$-dIP [21]	0.0167	0.0322	0.4675	0.0138
Pevný-$f^{\perp}$-sIP [21]	0.0175	0.0320	0.4519	0.0130
Lak.Ent. [23]	0.0413	0.0906	0.1198	0.0414
Lak.Vol.-sIP [22]	0.0199	0.0756	0.1076	0.0211
Lak.Vol.-dIP [22]	0.0241	0.0670	0.0938	0.0250
MINDS [24]	0.0184	0.0557	0.1703	0.0170
Xu-sIP [26]	0.0172	0.0188	0.0908	0.0172
Xu-dIP [26]	0.0193	0.0405	0.2507	0.0183
Average	0.0202	0.0476	0.2690	0.0192

ulated by time variant joint probability model. Note that anomaly score 1.1 represents the flows where the particular detector provided no results (due to its limitations). These figures show that our model generates traffic that triggers response of anomaly detection algorithms practically indistinguishable from the response to real traffic.

6. CONCLUSION

We proposed a solution for generating realistic NetFlow data that can be used for evaluation and configuration of anomaly detectors. We introduced the *time variant joint probability model* that is able to capture inter- and intra-flow relations as well as sequential character of user's behavior. We compared the proposed solution with two other simpler models (*random sampling* and *sampling with marginal model*) and have shown that our solution provides more than 2.3× better.

In future work we will focus on relaxing the assumptions adopted in this paper that limits the quality of the generated data. We will extend the general schema to capture the application specific aspects of the network traffic. The second issue that we will address is the modeling of the whole actions and not individual flows that will enable us to correctly model for example the full load of the HTML page.

7. ACKNOWLEDGMENT

The work presented in this paper was supported by Ministry of the Interior project VG20122014079. The work of T. Pevný was also supported by the Grant Agency of Czech Republic under the project P103/12/P514.

REFERENCES

[1] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection," *ACM Computing Surveys*, vol. 41, no. 3, pp. 1–58, Jul. 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1541880.1541882 http://portal.acm.org/citation.cfm?doid=1541880.1541882

[2] A. Patcha and J.-M. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Computer Networks*, vol. 51, no. 12, pp. 3448–3470, Aug. 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S138912860700

[3] P. Barford and D. Plonka, "Characteristics of network traffic flow anomalies," *Proceedings of the 1st ACM SIGCOMM Workshop ...*, 2001. [Online]. Available: http://dl.acm.org/citation.cfm?id=505211

[4] "The network simulatorâĂŞNS-2," http://nsnam.isi.edu/nsnam, [Online; accessed June 20th, 2012].

[5] T. Henderson and M. Lacage, "Network simulations with the ns-3 simulator," *SIGCOMM*, p. 2006, 2008. [Online]. Available: http://conferences.sigcomm.org/sigcomm/2008/papers/p527-hendersonA.pdf

[6] A. Varga and R. Hornig, "AN OVERVIEW OF THE OMNeT++ SIMULATION ENVIRONMENT," *Proceedings of the First International ICST Conference on Simulation Tools and Techniques for Communications Networks and Systems*, 2008.

[Online]. Available: http://eudl.eu/doi/10.4108/ICST.SIMUTOOLS2008.3027

[7] R. Bye, S. Schmidt, K. Luther, and S. Albayrak, "Application-level simulation for network security," *Proceedings of the First International ICST Conference on Simulation Tools and Techniques for Communications, Networks and Systems*, 2008. [Online]. Available: http://eudl.eu/?id=2961

[8] D. Grunewald, R. Bye, K. Bsufka, and S. Albayrak, "Agent-based Network Security Simulation (Demonstration)," *AAMAS*, pp. 1325–1326, 2011.

[9] L. M. Rossey, J. C. Rabek, R. K. Cunningham, D. J. Fried, R. P. Lippmann, and M. A. Zissman, "LARIAT : Lincoln Adaptable Real-time Information Assurance Testbed," pp. 1–27, 2001.

[10] A. Sperotto, R. Sadre, P.-t. D. Boer, and A. Pras, "Hidden Markov Model modeling of SSH brute-force attacks," *9th IEEE International Workshop on IP Operations and Management (IPOM 09)*, p. 13, 2009.

[11] D. Brauckhoff and A. Wagner, "FLAME: a flow-level anomaly modeling engine," in *The conference on Cyber security*, 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1496663

[12] P. Siska, M. P. Stoecklin, A. Kind, and T. Braun, "A flow trace generator using graph-based traffic classification techniques," *Proceedings of the 6th International Wireless Communications and Mobile Computing Conference on ZZZ - IWCMC '10*, p. 457, 2010. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1815396.1815503

[13] J. Sonchack and A. Aviv, "LESS Is More: Host-Agent Based Simulator for Large-Scale Evaluation of Security Systems," *Computer Security-ESORICS 2014*, pp. 365–382, 2014. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-319-11212-1_21

[14] S. Floyd and V. Paxson, "Difficulties in simulating the Internet," *IEEE/ACM Transactions on Networking (TON)*, vol. 9, no. 4, pp. 392–403, 2001. [Online]. Available: http://dl.acm.org/citation.cfm?id=504642

[15] H. Ringberg, M. Roughan, and J. Rexford, "The need for simulation in evaluating anomaly detectors," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 1, p. 55, Jan. 2008. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1341431.1341443

[16] J. Sommers, H. Kim, and P. Barford, "Harpoon: a flow-level traffic generator for router and network tests," *ACM SIGMETRICS Performance ...*, pp. 392–393, 2004. [Online]. Available: http://dl.acm.org/citation.cfm?id=1005733

[17] Y. Zhang, Z. Ge, A. Greenberg, and M. Roughan, "Network anomography," p. 30, Oct. 2005. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251086.1251116

[18] F. Silveira, C. Diot, N. Taft, and R. Govindan, "ASTUTE: Detecting a different class of traffic anomalies," *ACM SIGCOMM Computer ...*, 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1851215

[19] K. Murphy, *Machine Learning: a Probabilistic Perspective*, 2012.

[20] D. Endres and J. Schindelin, "A new metric for probability distributions," *IEEE Transactions on Information Theory*, vol. 49, no. 7, pp. 1858–1860, Jul. 2003. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber

[21] T. Pevny, M. Rehak, and M. Grill, "Detecting anomalous network hosts by means of PCA," in *2012 IEEE International Workshop on Information Forensics and Security (WIFS)*. IEEE, Dec. 2012, pp. 103–108. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber

[22] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications - SIGCOMM '04*, p. 219, 2004. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1015467.1015492

[23] ——, "Mining anomalies using traffic feature distributions," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 4, p. 217, Oct. 2005. [Online]. Available: http://dl.acm.org/citation.cfm?id=1090191.1080118 http://dl.acm.org/citation.cfm?id=1080118

[24] L. Ertoz, E. Eilertson, A. Lazarevic, P.-N. Tan, V. Kumar, J. Srivastava, and P. Dokas, "Minds-minnesota intrusion detection system," *Next Generation Data Mining*, 2004. [Online]. Available: http://www.it.iitb.ac.in/ deepak/deepak/courses/mtp/papers/m minnesota intrusion detection system.pdf

[25] M. Rehak, M. Pechoucek, K. Bartos, M. Grill, and P. Celeda, "Network Intrusion Detection by Means of Community of Trusting Agents," *2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'07)*, pp. 498–504, Nov. 2007. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber

[26] K. Xu, Z. Zhang, and S. Bhattacharyya, "Reducing unwanted traffic in a backbone network," *Usenix Workshop on Steps to Reduce Un- wanted Traffic in the Internet (SRUTI 05)*, 2005. [Online]. Available: https://www.usenix.org/event/sruti05/tech/talks/xu.pdf