Available online at www.sciencedirect.com
SciVerse ScienceDirect
journal homepage: www.elsevier.com/locate/cose

Computers & Security

# Toward developing a systematic approach to generate benchmark datasets for intrusion detection

*Ali Shiravi*[*], *Hadi Shiravi, Mahbod Tavallaee, Ali A. Ghorbani*

*Information Security Centre of Excellence (ISCX), Faculty of Computer Science, University of New Brunswick, 540 Windsor Street, Fredericton, New Brunswick, Canada E3B 5A3*

ARTICLE INFO

*Article history:*
Received 28 February 2011
Received in revised form
2 December 2011
Accepted 27 December 2011

*Keywords:*
Intrusion detection
Dataset generation
Network traffic profile

ABSTRACT

In network intrusion detection, anomaly-based approaches in particular suffer from accurate evaluation, comparison, and deployment which originates from the scarcity of adequate datasets. Many such datasets are internal and cannot be shared due to privacy issues, others are heavily anonymized and do not reflect current trends, or they lack certain statistical characteristics. These deficiencies are primarily the reasons why a *perfect* dataset is yet to exist. Thus, researchers must resort to datasets that are often suboptimal. As network behaviors and patterns change and intrusions evolve, it has very much become necessary to move away from *static* and *one-time* datasets toward more dynamically generated datasets which not only reflect the traffic compositions and intrusions of that time, but are also modifiable, extensible, and reproducible. In this paper, a systematic approach to generate the required datasets is introduced to address this need. The underlying notion is based on the concept of *profiles* which contain detailed descriptions of intrusions and abstract distribution models for applications, protocols, or lower level network entities. Real traces are analyzed to create profiles for agents that generate real traffic for HTTP, SMTP, SSH, IMAP, POP3, and FTP. In this regard, a set of guidelines is established to outline *valid* datasets, which set the basis for generating profiles. These guidelines are vital for the effectiveness of the dataset in terms of realism, evaluation capabilities, total capture, completeness, and malicious activity. The profiles are then employed in an experiment to generate the desirable dataset in a testbed environment. Various multi-stage attacks scenarios were subsequently carried out to supply the anomalous portion of the dataset. The intent for this dataset is to assist various researchers in acquiring datasets of this kind for testing, evaluation, and comparison purposes, through sharing the generated datasets and profiles.

## 1. Introduction

Intrusion detection has attracted the attention of many researchers in identifying the ever-increasing issue of intrusive activities. In particular, *anomaly detection* has been the main focus of many researchers due to its potential in detecting novel attacks. However, its adoption to real-world applications has been hampered due to system complexity as these systems require a substantial amount of testing, evaluation, and tuning prior to deployment. Running these

[*] *Corresponding author*. Tel.: +1 506 4534901; fax: +1 506 4533566.
E-mail addresses: ali.shiravi@unb.ca (A. Shiravi), hadi.shiravi@unb.ca (H. Shiravi), m.tavallaee@unb.ca (M. Tavallaee), ghorbani@unb.ca (A.A. Ghorbani).

0167-4048/$ – see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cose.2011.12.012

systems over *real* labeled network traces with a comprehensive and extensive set of intrusions and abnormal behavior is the most idealistic methodology for testing and evaluation. This itself is a significant challenge, since the availability of such datasets are extremely rare, and thus systems are evaluated over one or more *obtainable* datasets that lack sufficient comprehensiveness. Examples of this are the publicly available datasets such as CAIDA (2011) and Lawrence Berkeley National Laboratory and ICSI, which are heavily anonymized with the payload entirely removed resulting in decreased utility to researchers.

Another significant challenge is the appropriate comparison of intrusion detection systems (IDS) against one another. The scarcity of an appropriate public dataset severely impairs the evaluation of IDSs, particularly affecting anomaly-based detectors. This has been pointed out in numerous work such as Tavallaee et al. (2010) and has also been argued in Sommer and Paxson (2010) that "*the most significant challenge an evaluation faces is the lack of appropriate public datasets for assessing anomaly detection systems.*"

Despite the significant contributions of DARPA (Lincoln Laboratory, 2011) and KDD (University of California, 2011) datasets in the intrusion detection domain, their accuracy and ability to reflect real-world conditions has been extensively criticized in McHugh (2000) and Brown et al. (2009). This inability to evaluate intrusion detection systems against current and evolving intrusions and network traffic patterns is a major practical concern. This shift demands newer and more dynamically generated datasets, which reflect trending traffic patterns and intrusions as they evolve. This is in contrast to *static* datasets that are widely used today but are outdated, unmodifiable, inextensible, and irreproducible.

To overcome these shortcomings, a systematic approach has been devised to generate datasets in order to analyze, test, and evaluate intrusion detection systems, with a focus toward network based anomaly detectors.

The authors' vision is to enable researchers to generate datasets from *profiles* that can be combined to create a diverse set of datasets, each with a unique set of features that cover a portion of the evaluation domain. These profiles contain abstract representations of events and behaviors seen on the network. They are implemented using various autonomous agents or applied through human assistance. For example, the activity of a single host over the HTTP protocol can be abstracted to represent its distribution of packets, flow lengths, requests, end-points, and similar attributes. The same can be said for anomalous behavior where a profile could possibly represent a sequence of attacks or the behavior of an anomalous application on the network. These profiles, as well as the generated datasets, are readily sharable and can be interchanged among collaborators and researchers without (major) privacy issues. This will enable other research groups to *regenerate* the network behavior of various applications, protocols, hosts, and intrusions by utilizing a subset of these profiles according to their needs.

This work has been geared toward reaching the aforementioned vision. As the first step, a set of guidelines regarded as prerequisites for a valid dataset are established. In the second step, a *systematic* approach to traffic generation is delineated and along with the mentioned guidelines, a *sample* dataset is generated through the introduction and utilization of profiles. In practical terms, the likelihood of acquiring or generating a *perfect* dataset is not very strong as there will always be problems and issues with one particular dataset, especially when applied to a plethora of techniques for intrusion detection. Hence, by devising a systematic approach to dataset creation, various *adequate* datasets could potentially be generated for various situations and if done correctly, it will effectively depress the need for such perfect datasets.

In order to generate a dataset, an infrastructure is necessary as profiles only provide the knowledge to simulate one feature or aspect of the overall network. In this work, the natural behavior of network connected nodes at multiple layers has been utilized through the physical implementation of a testbed with real live network devices and workstations.

To simulate user behavior, the behaviors of our Center's users were abstracted into profiles. Agents were then programmed to execute them, effectively mimicking user activity. Attack scenarios were then designed and executed to express real-world cases of malicious behavior. They were applied in real-time from physical devices via human assistance; therefore, avoiding any unintended characteristics of post-merging network attacks with real-time background traffic. The resulting arrangement has the obvious benefit of allowing the network traces to be labeled. This is believed to simplify the evaluation of intrusion detection systems and provide more realistic and comprehensive benchmarks.

The prepared dataset consists of network traces, including full packet payloads, which along with the relevant profiles are publicly available to researchers through our website.[1] To further justify the cause of this work, the principal objectives for an acceptable dataset are elaborated below. These objectives serve as guidelines that outline the vision behind the current approach to dataset generation.

### 1.1. *Realistic network and traffic*

Ideally, a dataset should not exhibit any unintended properties, both network and traffic wise. This is to provide a clearer picture of the real effects of attacks over the network and the corresponding responses of workstations. For this reason, it is necessary for the traffic to look and behave as realistically as possible. This includes both normal and anomalous traffic. As also determined in previous works, any artificial post-capture trace insertion will negatively affect the raw data and introduce possible inconsistencies in the final dataset. Consequently, all such adjustments are highly discouraged.

### 1.2. *Labeled dataset*

A labeled dataset is of immense importance in the evaluation of various detection mechanisms. Hence, creating a dataset in a controlled and deterministic environment allows for the distinction of anomalous activity from normal traffic; therefore, eliminating the impractical process of manual labeling.

[1] http://iscx.ca/datasets.

### 1.3. *Total interaction capture*

The amount of information available to detection mechanisms is of vital importance as this provides the means to detect anomalous behavior. In other words, this information is essential for post-evaluation and the correct interpretation of the results. Thus, it is deemed a major requirement for a dataset to include all network interactions, either within or between internal LANs.

### 1.4. *Complete capture*

Privacy concerns related to sharing real network traces has been one of the major obstacles for network security researchers as data providers are often reluctant to share such information. Consequently, most such traces are either used internally, which limits other researchers from accurately evaluating and comparing their systems, or are heavily anonymized with the payload entirely removed resulting in decreased utility to researchers. Payloads are not only beneficial to evaluating the growing number of systems which rely on analyzing payloads (deep packet inspection), but also to systems which rely on accurate application discovery for the correct evaluation of their systems and methods. For example, assuming port 80 as HTTP traffic does not provide enough information to accept or refuse a packet as HTTP traffic or not. In this work, the foremost objective is to *generate* network traces in a controlled testbed environment, thus completely removing the need for any sanitization and thereby preserving the naturalness of the resulting dataset.

### 1.5. *Diverse intrusion scenarios*

Attacks have increased in frequency, size, variety, and complexity in recent years. The scope of threats has also changed into more complex schemes, including service and application-targeted attacks. Such attacks can cause far more serious disruptions than traditional brute force attempts and also require a more in-depth insight into IP services and applications for their detection. Through executing attack scenarios and applying abnormal behavior, the aim of this objective is to perform a diverse set of multi-stage attacks; each carefully crafted and aimed toward recent trends in security threats. This objective often labels many of the available datasets as ineffective and unfit for evaluating research results.

In this paper, we make the following contributions;

- A set of guidelines is delineated as prerequisites for a valid evaluation dataset.
- Introduction and utilization of profiles that can be combined together to create a diverse set of datasets, each with a unique set of features that cover a portion of the evaluation domain.
- A systematic approach to traffic generation is specified.
- A sample dataset adherent to the mentioned guidelines is generated through the utilization of profiles.

The remainder of this paper is organized as follows. In Section 2, a formal description of profiles and their respective representations are presented as the basis of this work. Section 3 describes the core approach in utilizing profiles for generating the necessary background traffic. Section 4 outlines the topology of the testbed, its underlying systems, and the statistical details of the generated dataset. The attack scenarios are throughly described in Section 5 and some of the underlying technical matters are outlined in Section 6. Section 7 looks at related work and Section 8 provides a detailed comparison of similar datasets. Subsequently the paper is concluded in Section 9 with details on future improvements of the dataset.

## 2. Profiles

A profile contains an abstract representation of certain features and events to facilitate the reproduction of certain real-world behaviors as seen from the network. These profiles are subsequently used by agents or human operators in order to generate events on the network. Their abstract property effectively prevents them to be dependent on a specific topology and thus allows them to be applied to a diverse range of networks with different topologies.

Profiles are used in combinations to generate a dataset with a required set of features and events. Two general classes of profiles, $\alpha$ and $\beta$, are defined as:

- $\alpha$-profiles attempt to describe an attack scenario in a unambiguous manner. As will be shown later, various attack description languages can be used to encode a multi-stage attack scenario. In the simplest case, humans can interpret these profiles and subsequently carry them out. Idealistically, autonomous agents along with compilers would be employed to interpret and execute these scenarios.
- $\beta$-profiles encapsulate extracted mathematical distributions or behaviors of certain entities and are represented as procedures with *pre* and *post* conditions. Examples include the distributions of packet sizes of a protocol, number of packets per flow, certain patterns in the payload, size of payload, request time distribution of a protocol and so forth.

Apart from the fact that $\alpha$-profiles describe attacks and $\beta$-profiles encapsulate entity distributions, $\alpha$-profiles generally require human knowledge and assistance in execution, whereas $\beta$-profiles can be automatically extracted from network traces or any other entity of concern. Specifying this information in the aforementioned formats provides viable means to break away from the original network traces and regenerate new datasets which follow the trends and patterns of the provided $\beta$-profiles while featuring a set of attack scenarios through the execution of $\alpha$-profiles.

This encourages the sharing of these profiles among the community and allows for further improvements to not only these profiles but to the resulting datasets which will be readily sharable, reproducible, and extensible.

Any attack or malicious activity that an intrusion detection system or an anomaly-based method is not trained on can be assumed to be a zero-day attack from the system's perspective and will not reduce the practicality of our approach in generating traffic for evaluating systems that address zero-day attacks.

### 2.1. *Profile representation*

Profiles are created to be used for traffic and attack generation, and to subsequently be shared with others. In order to do this, the gathered profiles must be presented in such a way as to enable (a) automatic generation of traffic and attacks, and (b) provide an effective approach to sharing and collaboration. Therefore, a common representation format for the profiles is necessary.

An $\alpha$-profile encapsulates a description of an attack scenario. In order to describe the steps and stages required to perform an intrusion, it is required to employ a *description language* that considers the various aspects of attacks. Once an attack is (unambiguously) described, agents are then able to interpret and extract the necessary information to carry out the intrusions. In this respect, a specific class of *attack languages* referred to as *exploit languages* are of particular interest. General-purpose languages are usually used to detail the steps of an attack; however, special-purpose languages (Arboi, 2005; Cuppens and Ortalo, 2000; Eckmann et al., 2002; Michel and Mé, 2001; Networks, 1999) have also been defined to support the scripting of attacks. Currently, no common exploit language exists and for this work, ADeLe (Michel and Mé, 2001) was found to adequately support the intended view of attacks and also the necessary expressiveness required to describe an attack scenario. Fig. 1 illustrates an SSH brute force attack expressed using ADeLe.

A $\beta$-profile stores a distribution model of certain entities or their attributes. Its content could range from simple statistical distributions to complex algorithms which abstract a certain behavior. In this respect, representing a $\beta$-profile as a *procedure* in a (supporting) programming language will allow for maximum expressiveness when dealing with a variety of models. A programming language such as Pascal, C, C++, or Java has the advantage of representing a $\beta$-profile non-ambiguously, both to agents and human users, but has the disadvantage of decreased portability among platforms and programming frameworks.

```
Alert SSH_Brute_Force(IN IPaddr targetip){
<EXPLOIT>
  <PRECOND>
#TFTP server is available at local(attacker) host.
     LocalService("TFTP")
  </PRECOND>

  <ATTACK> <LANG>EDL</LANG>
     UsernameGenerator genUName;
     PasswordGenerator genPass;
     SSHSession ssh;
     String username;
     String password;
     Integer ret_val;

     WHILE (TRUE) {
        username = genUName();
        WHILE (TRUE) {
          password = genPass();
          EVENT E0{
            ssh.Login(targetip, username, password, ret_val);
          }
           IF (ret_val != 0){
             EVENT E1{
               # send files to attacker
               ssh.Execute("tftp ....");
             }
           }
        }
     }

  </ATTACK>

  <POSTCOND>
     UserAccountCompromised = TRUE;
  </POSTCOND>

</EXPLOIT> }
```

**Fig. 1 – An SSH brute force attack (Scenario 4) description in ADeLe (Michel and Mé, 2001).**

## 3. $\beta$-Profile generation

One of the highest priorities of this work is to generate realistic background traffic. The foremost concern is to accurately reproduce the quantity and time distribution of flows for a chosen set of protocols.

To achieve this goal we have captured and carefully analyzed four weeks worth of network activity associated with the users and servers of our Centre for any abnormal or malicious activity. This has been achieved through the use of several IDSs and human assistance. Having filtered any suspicious activity, the traffic is used to determine the arrangement of traffic in terms of application and protocol composition. These profiles are subsequently abstracted, thus the likelihood of $\beta$-profiles containing malicious traffic is very diminutive. Fig. 2 depicts the arrangement of protocols and applications used throughout the capturing period.

The following protocols were subsequently chosen to be simulated in the testbed environment: HTTP, SMTP, POP3, IMAP, SSH, and FTP. Other protocols such as NetBios and DNS are readily generated as an indirect consequence of utilizing the aforementioned protocols and also as a result of layer 2 protocols.

To generate the necessary requests, TCP flows of each protocol originating from our center's hosts were extracted, and $\beta$-profiles were generated to store the distributions in terms of the number and timestamp of requests based on a weekday basis. In the following sections, we will elaborate further on the details regarding each protocol and its related $\beta$-profile.

### 3.1. *HTTP*

The majority of traffic (85.34%) observed within our center was HTTP, which was traced back to two major sources; (a) user browsing, and (b) autonomous programs.

To simulate the HTTP request behavior of a single user by an agent, it was required to extract and model the HTTP requests made from a real machine, into a $\beta$-profile for agents to consume. The agents then use the respective $\beta$-profiles and the distributions that they represent to generate a set of request times to autonomously request pages from the Web, similar to that of web crawlers.

To model HTTP requests, several approaches are available. The majority of work in literature is based on the well known statistical distributions. These probability distributions are analytically well described and have the advantage of being compact and easy to evaluate. An early study in analyzing statistical distributions has been conducted by Paxson et al.

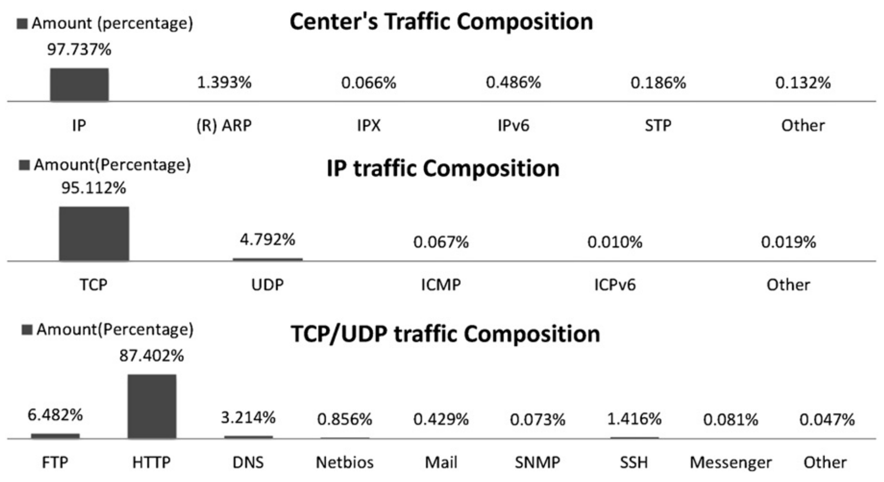


**Fig. 2 – Composition of protocols and applications seen in our Center's traffic.**

(Paxson, 1994) where they illustrated a number of analytic models for describing the characteristics of telnet, nntp, smtp, and ftp connections. However, by analyzing the available traces for HTTP requests, per user a daily basis we failed to observe a particular statistical distribution over the data. The following distributions were considered: Normal, Beta, Weibull, Erlang, Triangular, Gamma, Exponential, Uniform, and Lognormal. Fig. 3 depicts a sample HTTP request distribution for a user on a single working day.

As illustrated in Fig. 5, even the HTTP distributions of a single user varies during the weekdays. These outcomes are in line with previous studies that suggest the necessity of modeling HTTP activity using more complex distributions (Danzig and Jamin, 1991).

An alternative method is to represent probability distributions by their Cumulative Distribution Functions (CDFs), and to use the inverse transformation method (Jain, 1991), to extract and simulate HTTP user requests as it is applied in Danzig and Jamin (1991). While requiring more storage and perhaps a greater execution time, this approach has the advantage of representing arbitrary distributions. The following describes the steps taken to generate the necessary $\beta$-profiles that are used by agents to generate requests:

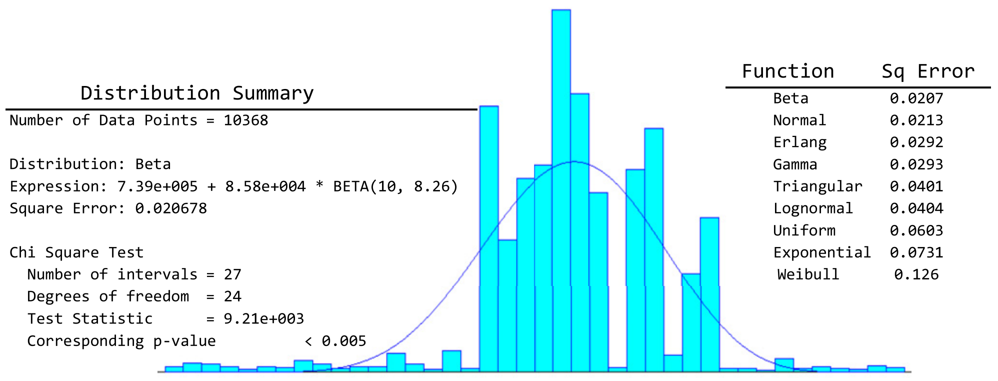
Step 1 The HTTP request timestamps for each user is extracted and grouped by the day of the week. Each set of timestamps for a particular day of the week is named a *D-profile*.

Step 2 *D-profiles* are grouped together for each day of the week and the set is called a *DoW-profile*.

Step 3 For each *D-profile* in a *DoW-profile*, we note the number of requests (NumRequests).

Step 4 For each *DoW-profile* we merge its *D-profiles* together into a single CDF and average its NumRequests. The inverse transformation method is then applied to this CDF to generate pseudorandom times according to the original distribution. The combination of the CDF, its average NumRequests, and an inverse transformation function constitutes the core of $\beta$-profiles.

An agent, upon the start of each day, will first pseudorandomly select a $\beta$-profile from an available collection. The number of requests is then generated through a pseudorandom number generator via a normal distribution with the following parameters: $\mu = Num\ Requests$, and $\sigma = 200$. In a situation where two agents were to choose the same $\beta$-profile, this normal distribution was seen to generate the


```
        Distribution Summary
Number of Data Points = 10368

Distribution: Beta
Expression: 7.39e+005 + 8.58e+004 * BETA(10, 8.26)
Square Error: 0.020678

Chi Square Test
  Number of intervals = 27
  Degrees of freedom  = 24
  Test Statistic      = 9.21e+003
  Corresponding p-value        < 0.005
```

```
Function      Sq Error
  Beta         0.0207
  Normal       0.0213
  Erlang       0.0292
  Gamma        0.0293
  Triangular   0.0401
  Lognormal    0.0404
  Uniform      0.0603
  Exponential  0.0731
   Weibull      0.126
```

**Fig. 3 – Histogram of HTTP request distribution for a user on a single working day.**

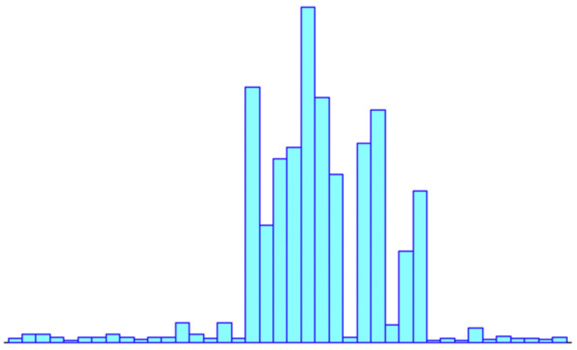


**Fig. 4 – Histogram of HTTP requests made by an agent.**

required pseudorandom noise in the number of requests generated by the agent. Fig. 3 illustrates the similarity between the extracted requests and the corresponding distribution of requests made by an agent is shown in Fig. 4.

To keep the requests as realistic as possible, a slightly modified web-crawling mechanism was devised to reflect the browsing behaviors of users. A very simplified interpretation for an HTTP agent is laid out in Algorithm 1. After generating the necessary number of request times, the `MaintainAgent` procedure is called. The responsibility of this procedure is to always keep a pre-determined number of request threads on standby. After a request thread is terminated, `MaintainAgent` is called again to sustain the required number of threads on standby. The rationality behind this approach is further discussed in Section 6.

Algorithm 1: HTTP Request Agent Logic

```
Data: A set $initialurls$ of URLs; A set $internalurls$ of URLs
begin
    $profile \leftarrow$ GetRandomDoWProfile ($date$) ;
    $NumRequests \leftarrow$ GetRequestNum (profile) ;
    initialize global $requestList$;
    for $i \leftarrow 1$ to $NumRequests$ do
        $requestTime \leftarrow$ GetRandomNum (profile);
        Add $requestTime$ to $requestList$ ;
    sort $requestList$ in ascending order ;
    call MaintainAgent () ;
end
```

```
begin
    static $currentRequestIndex$;
    while $numStandbyThreads < 50$ do
        call CreateHTTPRequestThread ($requestlist[currentRequestIndex]$) ;
        $currentRequestIndex \leftarrow currentRequestIndex + 1$;
end
```

### 3.2. SMTP/IMAP

To accurately simulate email communications within the testbed network, email accounts were created on the main server for each machine and agents were set up accordingly on each machine to access its inbox using IMAP or POP3 protocols. The agents were configured to access their inboxes in a fixed, pseudorandom interval between 1 and 30 min. $\beta$-profiles for SMTP traffic were created from the email accounts of 15 people from the communications of, at least, the past 2 years. They were analyzed thoroughly and for each email, its sent timestamp, size, and size of attachments were extracted and stored. Similar to the approach used for generating HTTP traffic, SMTP $\beta$-profiles were created for each weekday in terms of sent time, size, and attachment size. An agent would then generate a number of emails based on the respective profiles and send them at the specified times.

Fig. 6 depicts the histogram of distributions for email sent times and Fig. 7 depicts the histogram of distributions for the number of emails sent, both for the same $\beta$-profile and for each day of the week. The average throughput of email related traffic in the final dataset is also depicted in Fig. 8.

### 3.3. FTP

The FTP traffic in this dataset is intended to reflect synchronization behavior of hosts with a local server. Due to the fact that this type of behavior was missing from our Centre's real traffic, synthetic $\beta$-profiles where created. The agents synchronize a chosen folder with the main server using $\beta$-profiles that generate synchronization times once between 4:00 a.m. and 6:00 a.m. (Sunday 9:00–10:00 a.m.) and pseudorandom intervals of 1–5 h thereafter. In most cases, files are not transferred mainly due to the fact that the folder does not change between synchronization periods. This trend can clearly be seen in Fig. 9, where the spikes represent the instructed daily morning synchronization.

### 3.4. SSH

In the capturing period, SSH has been generated by several means; (a) anomalous behavior related to the execution of $\alpha$-profiles; (b) $\beta$-profiles related to the daily synchronization of application data between the machines and the main server; and, (c) remote connection to the servers and particular machines. Several $\beta$-profiles were specified synthetically to generate pseudorandom synchronization times within a 10 min interval after midnight, except for Sundays, where no

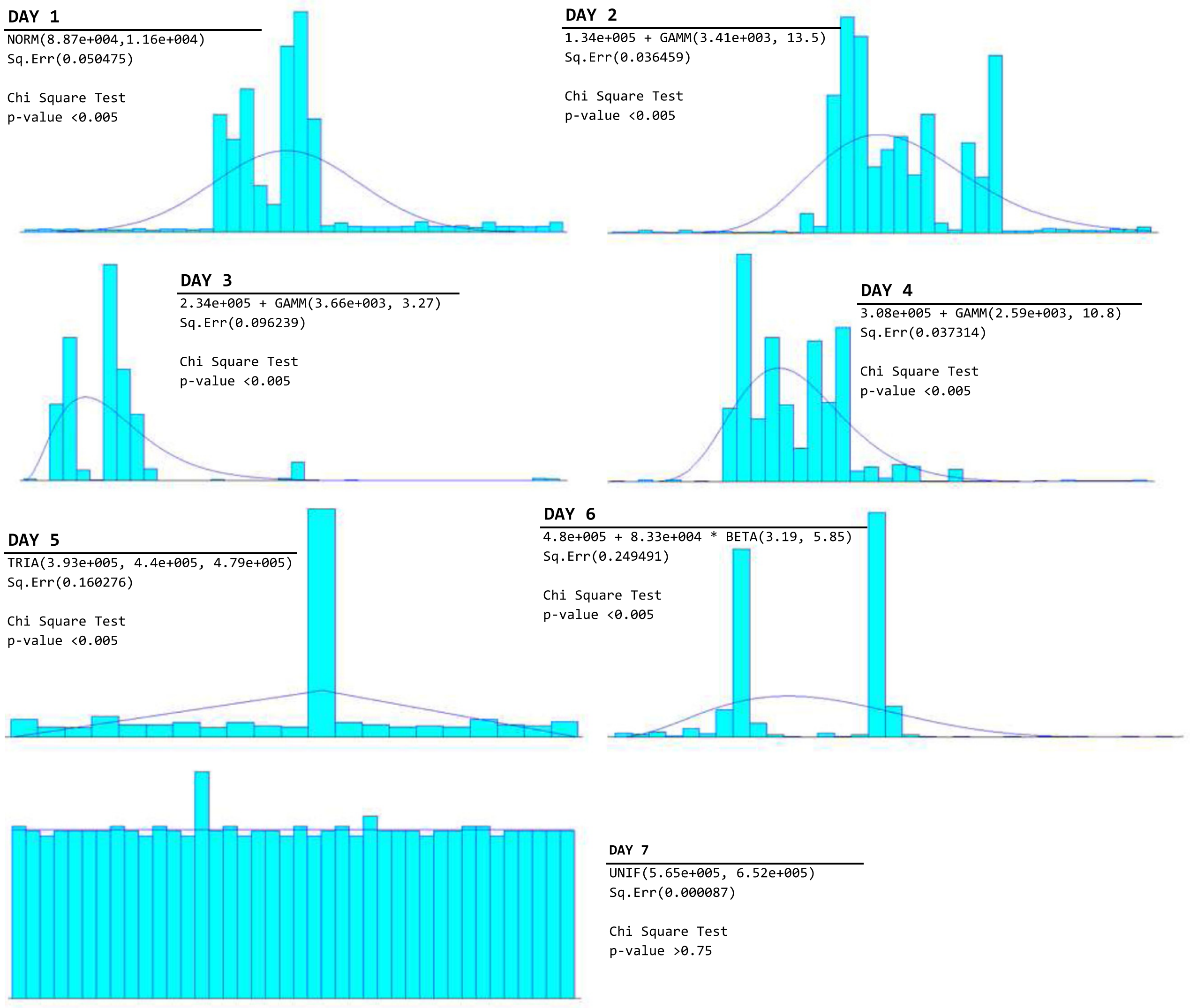


**Fig. 5 – Histograms of HTTP request distribution for a user on multiple days of the week. The corresponding best-fit distribution expression and *p*-values for the Chi Square Test is noted on each histogram.**

synchronization takes place. Fig. 10 depicts this trend in the captured dataset.

## 4. Dataset generation

It is important to note that a profile is a concept and requires an infrastructure in order to be used effectively. The resulting details which arise from the realization of profiles, executing agents, and the supporting infrastructure is discussed in the following sections.

### 4.1. *Testbed network architecture*

The testbed network consists of 21 interconnected Windows workstations. The Windows operating systems is chosen so that it would be possible to exploit a diverse set of known vulnerabilities against the testbed environment. Seventeen workstations were installed with Windows XP SP1, two workstations with SP2, one with SP3, and a workstation with Windows 7.

As shown in Fig. 11, the workstations are divided into 4 distinct LANs in order to effectively construct a real inter-connected network. This by itself has the effect of reducing the broadcast domain for each workstation. The fifth LAN consists of servers that provide web, email, DNS, and Network Address Translation (NAT) services. The NAT server (192.168.5.124) acts as an access provider to the Internet for the entire network. Acting as the entry point of the network, it effectively provides firewall facilities to block unauthorized access while permitting authorized communications. The NAT server is connected to the Internet through two valid IP addresses. One IP address is multiplexed for the workstations accessing the Internet while the other is designated solely for the main server. The main server (192.168.5.122) is responsible for delivering the network's website, providing email services, and acting as the internal name resolver. A secondary server (192.168.5.123) is responsible for internal ASP.NET applications. Both the main and NAT servers are Linux based, with

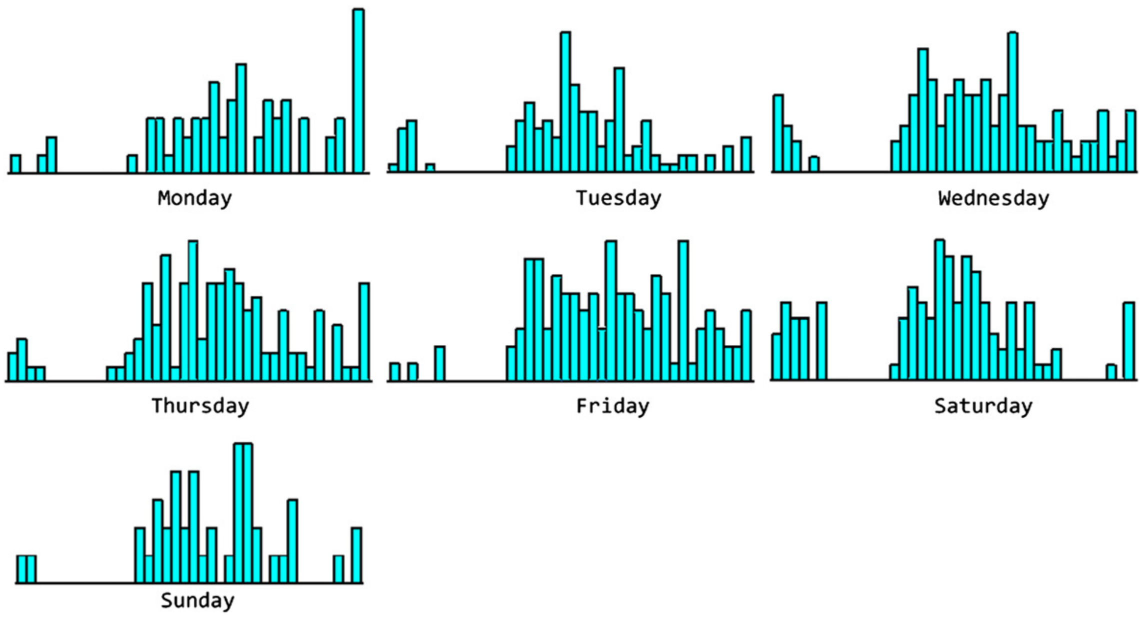


**Fig. 6 – Distribution histograms related to a single email $\beta$-profile, depicting the day time of emails sent for a week. Each bar represents a 36 min interval in 24 h.**

Ubuntu 10.04 installed and configured on them. The secondary server is a Windows Server 2003 machine. Table 1 specifies more details regarding the servers and the various services installed on them.

The sixth LAN provides the means to conduct non-disruptive monitoring and maintenance of workstations and servers. Since the traffic is not captured, tasks such as loading applications and tuning service parameters are made possible.

A single layer 3 switch is utilized to provide the necessary layer 2 and 3 switching and also mirroring of traffic. All connections are explicitly set at 10 Mbit/s. This was seen as more than adequate for the networked devices to operate effectively while keeping the maximum throughput well below the maximum switching capacity. This measure was taken to reduce the probability of packets being dropped by both the switch and the capturing devices.

An Ethernet tap (running at 100 Mbit/s) was efficiently employed to transmit the mirrored traffic to multiple devices without any processing overhead or disruption. These devices provided the means for redundant capturing (e.g. tcpdump), alert generation through various Intrusion Detection Systems (IDS) (e.g. Snort), IDS management systems (e.g. QRadar, OSSIM), and visualization systems (e.g. ntop).

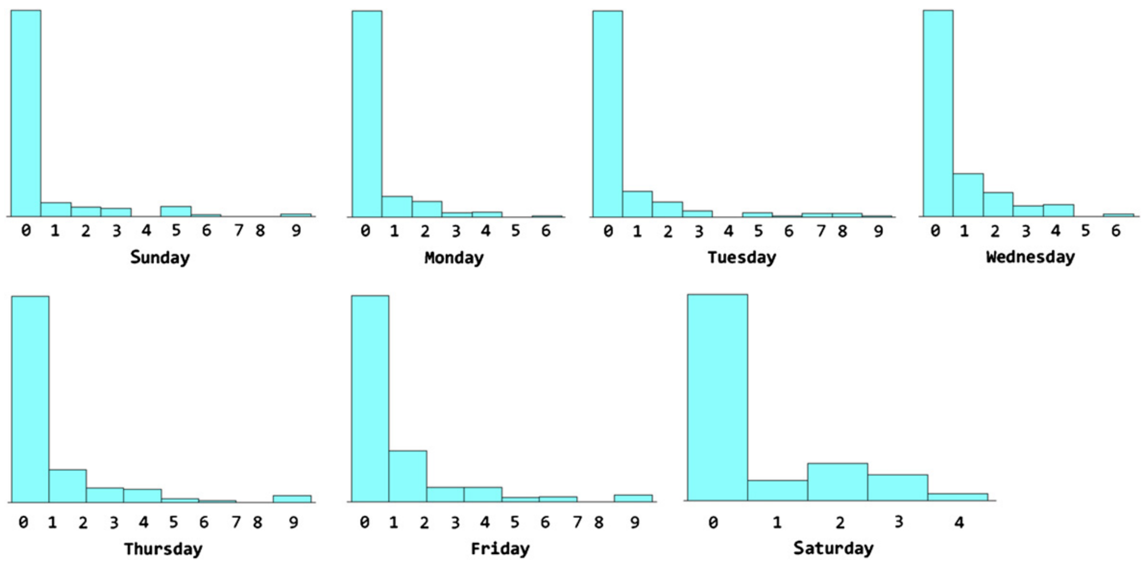


**Fig. 7 – Distribution histograms depicting the number of emails sent for a week. These are related to the email $\beta$-profile in Fig. 6. The x-axis represents the number of relevant email count.**

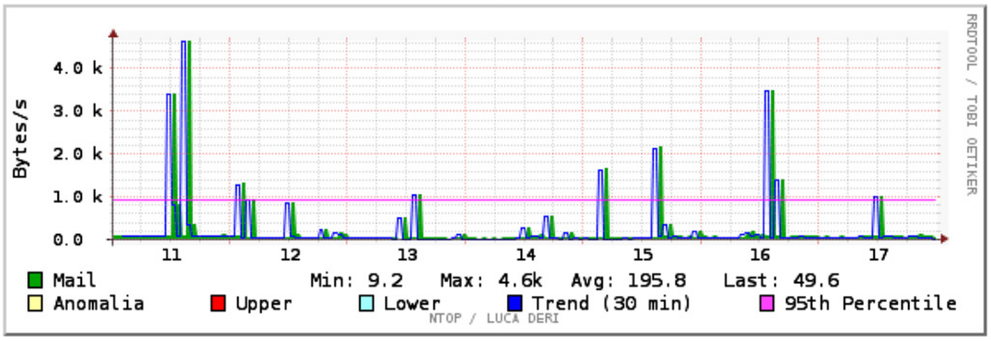


**Fig. 8 – Average throughput of Email bytes seen during the capturing week.**

### 4.2. *Execution*

The capturing period started at 00:01:06 s on Friday June 11th and continuously ran for an exact duration of 7 days, ending at 00:01:06 s on Friday June 18th. Attacks were subsequently executed during this period. As it can been seen from Fig. 12, during the weekdays (Monday–Friday) we can see a substantial rise in the number of flows in the mornings and a slow reduction in the afternoons. This collective behavior is consistent with the individual $\beta$-profiles discussed in Section 3.1 since HTTP traffic constitutes a large portion of this dataset. The overall statistics for the generated dataset is summarized in Table 2 and its composition is illustrated in Table 3.

## 5. Generation and execution of $\alpha$-profiles

Since the proposed dataset is intended for network security and intrusion detection purposes, it would not be complete without a diverse set of attack scenarios. As mentioned previously, a combination of $\alpha$-and $\beta$-profiles are used to generate a dataset with a required set of features and events. This section describes the process of generating and executing $\alpha$-profiles by our team during the capturing period. Each attack scenario is used to create an $\alpha$-profile which is later used to carry out the scenario. Our aim here is to mimic the actions of malicious hackers by performing four multi-stage attack scenarios, each carefully crafted toward achieving a predefined set of goals. A full attack scenario is composed of the following steps;

1. Information gathering and reconnaissance (passive and active)
2. Vulnerability identification and scanning
3. Gaining access and compromising a system
4. Maintaining access and creating backdoors
5. Covering tracks

Reconnaissance or information gathering is the first step in conducting an attack and is perhaps one of the most important. Information gathering can be carried out in both passive and active modes. In passive information gathering, we intend to gather information regarding our target network through secondary Internet sources without directly connecting to the network itself. This includes information gained through WHOIS and DNS databases such as namespaces, server types, server locations, and incorporated software technologies. In active information gathering the boundaries of the target network are identified and are elaborated in greater detail in later stages. Identifying the range of IP addresses or determining whether the target network is behind a NAT server takes place in this stage. Also, if the target network possesses an email server, creating a list of users would be a good idea and can be useful in performing social engineering attacks in later stages.

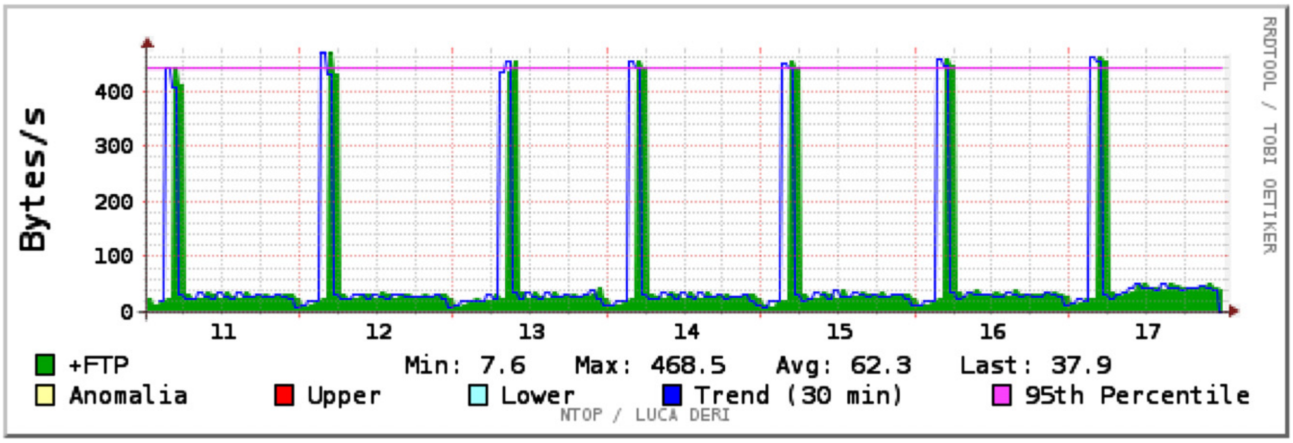


**Fig. 9 – Average throughput of FTP bytes seen during the capturing period.**

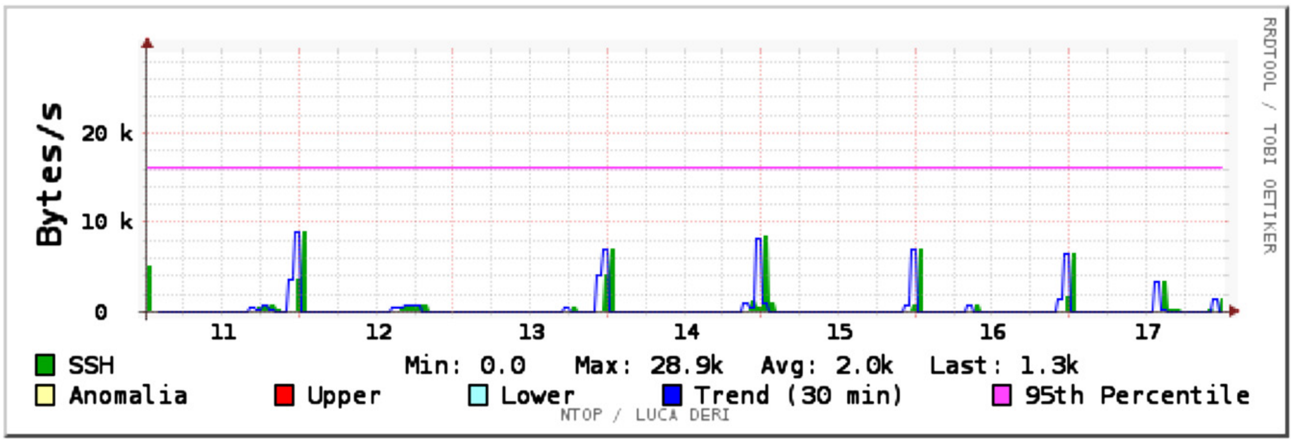


Fig. 10 – Average throughput of SSH bytes seen within the capturing week.

Prior to performing any exploits and after initial reconnaissance, we need to identify the vulnerabilities of the systems residing inside the target network. This requires some information about the type and version of the operating system, along with the running services and open ports. Later this information can be used to query databases containing an up-to-date list of vulnerabilities. Network scanners such as Nmap can be used to identify active hosts and services running inside a network while proprietary vulnerability scanners such as Nessus maintain a database of recent security vulnerabilities updated on a daily basis.

The only way to verify the information gained via the vulnerability assessment phase is to actually implement the attacks. After initial assessment, the potential vulnerabilities are exploited through third-party scripts or exploitation tools such as Metasploit (Rapid7, 2011) which provide a framework for the automation of the attack process. These tools also have the potential to contribute toward enabling autonomous agents to parse and execute $\alpha$-profiles and to enable the full automation of dataset generation.

Once the attacker has acquired access to the network, maintaining that access is crucial since the compromised

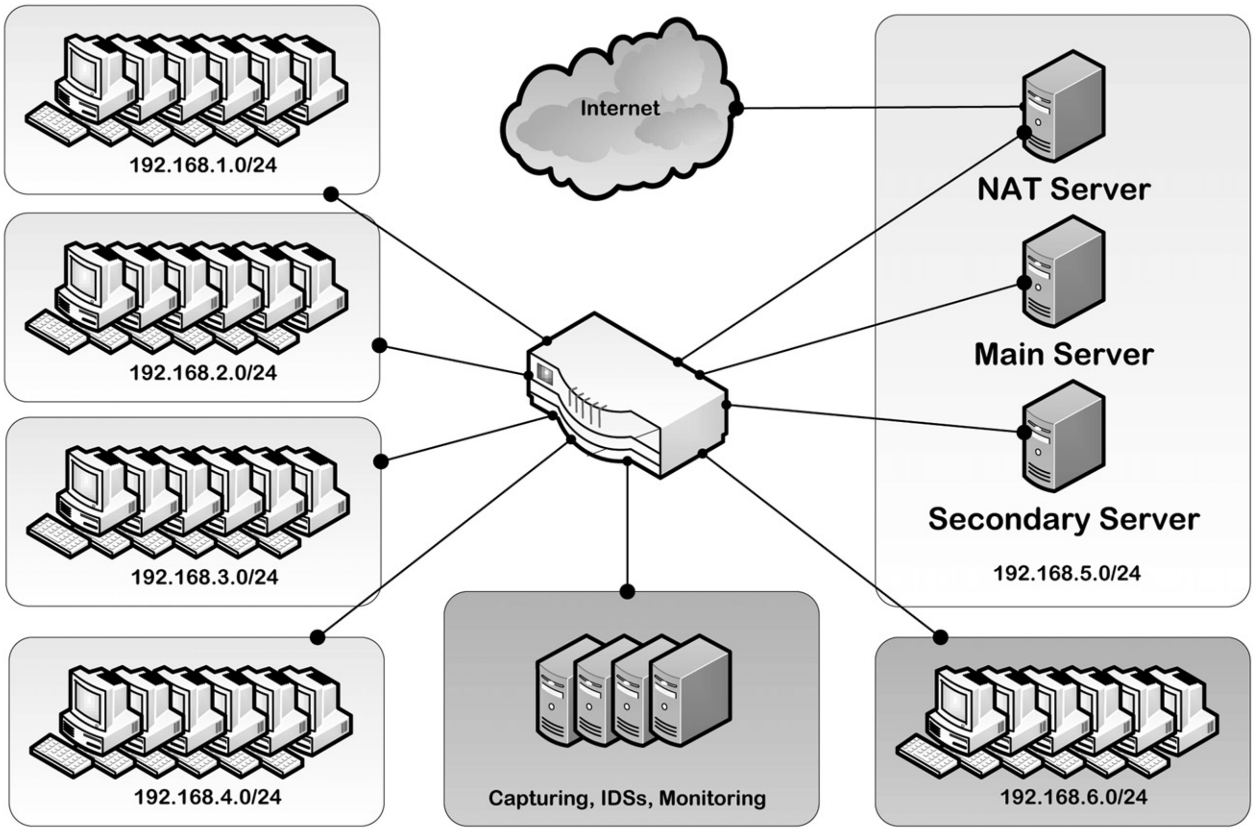


Fig. 11 – Testbed network architecture.

**Table 1 – Various severs and services running on the testbed.**

| Server | Services | Provider |
|---|---|---|
| Main server ubuntu 10.04 | Web | Apache 2.2.9 |
| | eMail | (MTA) Postfix (IMAP/POP3) Dovecot |
| | DNS | Bind 9 |
| | SSH | OpenSSH v5.3 |
| | FTP | vsftp v2.2.2 |
| NAT server ubuntu 10.04 | NAT | IPTables |
| Secondary server Windows server 2003 | Web | IIS v6 |

system may simply reboot, leading to the loss of the initial connection or even get patched forbidding any future attack attempts. Dumping password hash files and installing remote shell connections (or in some cases reverse shell) are typical activities in maintaining access to a computer. Meterpreter is an advanced payload that is integrated in the Metasploit Framework and can be considered as a powerful tool in not only maintaining access to a compromised host but also performing post exploitation tasks such as uploading and downloading of files, file system interaction, network pivoting and port forwarding, and much more.

It is in the intent of some attackers to move through a network as quietly as possible, raising the least number of alarms and leaving minimum tracks behind. Editing or deleting system or application level log files is often a useful mechanism in avoiding detection by system administrators. However, since many attacks on networks are carried out by previously compromised systems, hackers are less than interested in clearing logs and traces. An attacker can simply use many of its underlying compromised hosts for future attacks.

In order to be as realistic as possible, later attack scenarios are based on the results of earlier attacks making them sophisticated, powerful, and harder to detect. This also illustrates how simpler $\alpha$-profiles can be coupled together to create more sophisticated $\alpha$-profiles. Each attack scenario is elaborated further below.

## 5.1. *Scenario 1: infiltrating the network from the inside*

Many of today's networks are built on what is called the eggshell principle: hard on the outside and soft on the inside. This means that if an attacker gains access to a host on the inside, she can then use the compromised host as a pivot to attack systems not previously accessible via the Internet such as a local intranet server or a domain controller.

This attack scenario starts by gathering information about the target including network IP ranges, nameservers, mail servers and user email accounts. This is achieved by querying the DNS for resource records using network administrative tools like Nslookup and dig.

Having performed the initial reconnaissance, it was found that the only system on the target network exposed to the Internet was a NAT server. This makes internal users of the target network inaccessible from outside, since the NAT server acts as a firewall, dropping any initial connections from outside the network. This is where conventional attacking techniques are not useful and we are forced in using client side attack techniques. Based on the `nslookup` output we can start to enumerate through the mail server guessing potential email addresses that are required to penetrate into the system.

The Adobe Reader `util.printf()` buffer overflow vulnerability was used as a starting point for this scenario. An attacker can exploit this issue to execute arbitrary code with the privileges of the user running the application. We create a malicious PDF file using Metasploit and embed a Meterpreter reverse TCP shell on port 5555 inside it. The PDF file is attached to a system upgrade email and sent on behalf of `admin@[...]` to all 21 users of the testbed. We have also set up a listener on port 5555 to capture the reverse connection on our attacking machine. Clicking on the file opens Adobe but shows a gray window that never reveals a PDF but instead makes a reverse TCP connection back to our attacking computer which is listening on port 5555.

The first session is initiated from user5 of the testbed which connects back to the attacking machine by exploiting the Adobe Reader vulnerability. Once gained full access, we need to start identifying potential targets such as internal servers or backup databases. We upload Nmap to user5 using our Meterpreter session and start to scan potential hosts on two consecutive subnets (192.168.1.0/24 and 192.168.2.0/24). User12 is identified as running Windows XP SP1 with a vulnerable SMB authentication protocol on port 445. This vulnerability is exploited and a scan is performed from user12 to the server subnet (192.168.5.0/24). This scan identifies a Windows Server 2003 running an internal Web application using MSSQL Server as its backend database with only port 80 opened. A remote desktop connection to user12 is created to access the Web application and is tunneled back to the attacking machine using the previously established Meterpreter session.

When a machine has only port 80 opened, even the most advanced vulnerability scanner cannot identify anything

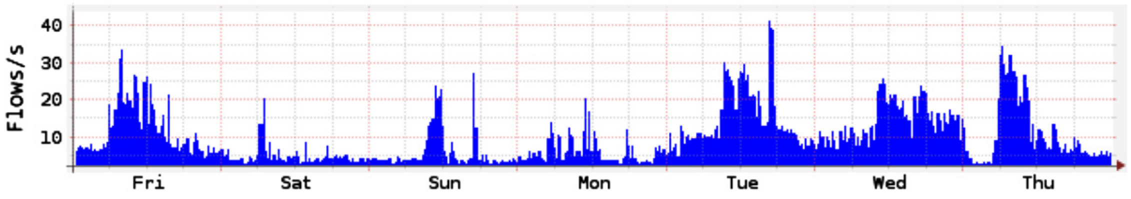


**Fig. 12 – Number of flows seen per second during the capturing period.**

Table 2 – Dataset statistics.

| Day | Flows | Src bytes | Dst bytes | Src packets | Dst packets | Protocol (Flows) | | | Direction of Flow | | Tag | | Dstnct | Dstnct |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | TCP | UDP | ICMP | L2L | L2R | Normal | Attack | Src IP | Dst IP |
| Friday | 378,667 | 884,276,523 | 16,033,149,459 | 8,565,698 | 12,912,796 | 297,398 | 78,786 | 2,478 | 37,365 | 339,668 | 378,667 | 0 | 83 | 7,110 |
| Saturday | 133,193 | 311,170,654 | 4,148,028,048 | 2,340,671 | 3,599,451 | 95,117 | 37,966 | 81 | 27,244 | 103,282 | 131,111 | 2,082 | 44 | 2,610 |
| Sunday | 275,528 | 433,396,736 | 7,881,760,674 | 4,703,338 | 6,734,950 | 221,026 | 54,076 | 374 | 58,274 | 215,722 | 255,170 | 20,358 | 44 | 2,645 |
| Monday | 171,380 | 1,051,987,567 | 6,187,034,339 | 3,879,857 | 5,743,424 | 122,298 | 48,453 | 623 | 25,444 | 144,127 | 167,609 | 3,771 | 448 | 7,957 |
| Tuesday | 571,698 | 1,905,193,956 | 22,842,855,364 | 13,254,945 | 21,746,115 | 441,563 | 124,023 | 6,073 | 74,433 | 490,406 | 534,320 | 37,378 | 2,196 | 26,780 |
| Wednesday | 522,263 | 789,247,707 | 17,879,787,685 | 9,916,103 | 14,625,339 | 434,674 | 87,070 | 513 | 40,158 | 481,356 | 522,263 | 0 | 137 | 8,897 |
| Thursday | 397,595 | 607,242,240 | 12,510,135,591 | 7,007,151 | 10,323,340 | 329,378 | 67,658 | 547 | 38,975 | 352,607 | 392,392 | 5,203 | 115 | 7,243 |
| *Total* | 2,450,324 | 5,982,515,383 | 87,482,751,160 | 49,667,763 | 75,685,415 | 1,941,454 | 498,032 | 10,689 | 301,893 | 2,127,168 | 2,381,532 | 68,792 | | |

Table 3 – Dataset traffic composition.

| Protocol | Size (MB) | Pct |
|---|---|---|
| (a) Total traffic composition | | |
| IP | 76570.67 | 99.99 |
| ARP | 4.39 | 0.01 |
| Ipv6 | 1.74 | 0.00 |
| IPX | 0 | 0.00 |
| STP | 0 | 0.00 |
| Other | 0 | 0.00 |
| (b) TCP/UDP traffic composition | | |
| TCP | 75776 | 98.96 |
| UDP | 792 | 1.03 |
| ICMP | 2.64 | 0.00 |
| ICMPv6 | 0 | 0.00 |
| Other | 0.03 | 0.00 |
| (c) TCP/UDP traffic composition | | |
| FTP | 200.3 | 0.26 |
| HTTP | 72499.2 | 94.69 |
| DNS | 288.6 | 0.38 |
| Netbios | 36.4 | 0.05 |
| Mail | 119.8 | 0.16 |
| SNMP | 0.01 | 0.00 |
| SSH | 241.4 | 0.32 |
| Messenger | 0.54 | 0.00 |
| Other | 3179.08 | 4.15 |

useful. This leads to the use of Web application hacking techniques like SQL injection or Cross Site Scripting. SQL injection is the most common type of Web application hacking which takes advantage of non-validated input vulnerabilities and attempts to pass SQL commands through a web application for execution by the backend database. We start the attack with the *single quote trick* by entering something similar to "`' having 1=1-`" inside the username textbox. The Web application responds with an error message signaling a potential vulnerability. We continue by using the information in the error messages to identify the column names and types of the user table. This is achieved by using multiple union and aggregate queries and analyzing their respected errors. We finally manage to create a username *tree* with password *hello* and insert it into the database. After logging in and taking a look at different sections of the Web application, we decided to delete the user table to avoid any future logins from the testbed users. This is achieved by injecting a simple query to the database such as "`'drop table user;-`". In order to be able to continue the attacks in later days and to avoid undergoing the burdensome process of compromising an internal host, a backdoor is created and configured to connect back to the attacker's machine.

Metasploit's `msfencode` is used to create a standalone executable of a reverse TCP shell connection and is uploaded to user12 through the already established Meterpreter session. Now we can schedule the backdoor to connect back to us at whatever interval we desire. Scheduling tasks in Windows can be done through the command line using the `schtasks` command. The backdoor executable is run by the operating system every hour while making a TCP connection back to the attacker. This marks the final stage of the first attack scenario as we exit all previously established connections.

### 5.2. *Scenario 2: HTTP denial of service*

The second attack scenario is designed toward performing a stealthy, low bandwidth denial of service attack without the need to flood the network. We will be utilizing `Slowloris` as the main tool in this scenario as it has proven to make Web servers completely inaccessible using a single machine. Slowloris starts by making a full TCP connection to the remote server. The tool holds the connection open by sending valid, incomplete HTTP requests to the server at regular intervals to keep the sockets from closing. Since any Web server has a finite ability to serve connections, it will only be a matter of time before all sockets are used up and no other connection can be made.

We start the attack by running a TCP listener on the attacking machine (different from the previous scenario to have a diverse set of attacking IPs) and wait for the operating system of user12 to run the backdoor scheduled task as previously configured in the final stage of scenario 1. Upon receiving a connection from user12 a scan of the 192.168.3.0/24 subnet is performed to identify potential vulnerable hosts. User13 is identified as running a vulnerable SMB authentication protocol on port 445. We then exploit this vulnerability and tunnel back a remote desktop connection from user13 to the attacking machine. `Slowloris` is downloaded from one of our servers and is used to attack the main web server (192.168.5.122) which acts a host for a valid domain. The attack is run for approximately ten minutes leading to the inaccessibility of the server. The same procedure is taken to exploit users 1,6,15, and 17 as `Slowloris` is downloaded and run on each machine resulting in a Denial of Service attack against the Apache Web server.

### 5.3. *Scenario 3: distributed denial of service using an IRC Botnet*

Botnets are an emerging threat to organizations of all kinds as they are the driving force behind spam, data theft and malware distribution. Botnets combine previous maligned behaviors into a single platform, essentially simplifying and assisting the user of such platform to form sophisticated attacks against workstations or networks around the world. Such behaviors include scanning, distributed denial-of-service (DDoS) activities, direct attacks, and deceitful activities taking place across the Internet.

This scenario is designed with the goal of performing a distributed attack using infected hosts on the testbed. An Internet Relay Chat (IRC) bot, written from scratch by our team, is sent as an attachment for an update message to testbed users. The bot has the ability to download malicious programs from remote servers and to execute them with user privileges. A separate Denial of Service program has also been written to perform an HTTP GET attack on a specific target. The program is designed with heavy multithreading to enable each bot to make hundreds of requests at any one time.

The attack is initiated when user12 connects back to the attacking machine using the backdoor configured in the final stage of scenario 1. An IRC server is uploaded to user12 via the Meterpreter session and is configured to start listening on port 6667. An IRC client is also uploaded to allow for commands to be sent to bots on infected machines. Within a period of 30 min, seven users have connected to the IRC server including users 3, 5, 9, 10, 13, 18 and 20. Bots connect to a prespecified channel on the IRC server with a nickname constructed by hashing their primary MAC address. The bots are ordered to download the HTTP GET program using the `download httpget IP` command. As each user finishes its download, a download complete message is sent over the channel. The Distributed Denial of Service attack is started using the `run http IP threads` command and is targeted toward the main Apache Web server. The attack is run for a period of 60 min leading to inaccessibility or slowing down of the server. The attack is stopped using the `stop http` command and bots are ordered to quit and to disconnect.

### 5.4. *Scenario 4: brute force SSH*

Brute force attacks are very common against networks as they tend to break into accounts with weak username and password combinations. The final scenario has been designed with the goal of acquiring an SSH account by running a dictionary brute force attack against the main server. We use *brutessh* as the main tool for this scenario as it can be easily configured to use a custom made dictionary list. The dictionary is composed of over 5000 alphanumerical entries of varying length. We ran the attack for a 30 min time period in which the superuser account credentials were returned. The user credentials were then used to login into the server and to download `/etc/passwd` and `/etc/shadow` files.

## 6. Technicalities

Putting together an operating testbed has its challenges. Aside from the issues with setting up and configuring the many machines, the vital concerns were capturing traffic, executing the attacks, and implementing the agents, as they directly affected the contents of the dataset. These points are elaborated below.

### 6.1. *Capturing traffic*

The network configuration consists of a single layer 3 switch (Omniswitch 6850), which was used as the connecting point for all machines. Virtual LANs were configured on the switch to effectively implement the necessary LANs. This enabled us to capture virtually all communications between all machines which was a priority objective of this experiment.

Directing all of the traffic through a single interface on the switch has its disadvantages, most importantly the dropping of packets due to congestion. To circumvent this matter, the speed of all interfaces on the switch were subsequently lowered to 10 Mbps, effectively lowering the total throughput. Preliminary test runs indicated no substantial effect on the overall quality of the dataset. Consequently, lower hardware specifications were required to capture, monitor and analyze the traffic, therefore, enabling the use of commodity and off-the-shelf hardware. As an example, a simple Ethernet tap was utilized to replicate the mirrored traffic, on-the-fly, to various monitoring devices.

### 6.2. *Agent implementation*

When an HTTP agent generates the required number of requests for a particular day, a `MaintainAgent` function (Section 3.1) is called to create a pre-determined number of threads for standby. This is done so that when the time to execute a request is due, all internal maintenance matters regarding a request have already been executed beforehand and the thread is ready to simply send out the request.

The issue with internal maintenance was particularly apparent when the time of requests were very close to each other. In such cases, the requests were delayed mainly due to several mutexes in place for the arbitration of request lists and thread maintenance. To overcome this issue, threads were create beforehand and were put on standby. This effectively increased the accuracy of request times but substantially increased the complexity of the underlying `MaintainAgent` function.

For SMTP agents, the request times had a lower degree of sensitivity and thus did not require the standby thread model for its implementation. However, as the infrastructure had already been create for HTTP agents, it was nonetheless used to implement the SMTP requests as well.

## 7. Related work

Most existing work on network traffic generation have not focused on applicability in the area of Network Security and evaluation of anomaly-based techniques.

The authors in Hong and Wu (2005) introduce a tool to replay previously recorded TCP sessions and adjust the replay according to a set of traffic parameters. Their work is focused on re-modeling traffic replay whereas our work is centered on the idea of creating profiles for traffic generation.

Model generation techniques such as Cao et al. (2004); Lan and Heidemann (2002); Weigle et al. (2006) attempt to model a set of features from observed real traffic and use it to generate statistically similar distributions in simulation environments. Apart from the fact that they do not address actual traffic generation, their center focus is on extracting distributions on a a wide-area scale in contrast to a more finer user-level characteristics that $\beta$-profiles are recommended to include. However, their results can be complementary to this work to create more realistic $\beta$-profiles that capture certain features of real traffic at large scales.

Other approaches such as Kayacik and Zincir-heywood (2005); Mutz et al. (2003); Sommers et al. (2004); Sommers et al. (2005); Valgenti and Kim (2011) explore methods to assess firewalls and IDSs on specific protocols. Authors in Valgenti and Kim (2011) use IDS signature sets to create traffic with payloads that partially match the signatures. In other works such as Sommers et al. (2005), the authors describe a traffic generation framework to evaluate stateful, protocol-aware intrusion detection systems by employing a trust-based strategy to separate normal traffic from malicious activity in a given trace. This is fundamentally different from our approach which focuses on profile generation rather than replaying traffic. The malicious portion of their traffic is generated via their previous work (Sommers et al., 2004) which defines a framework to flexibly compose attack traffic. Their method, however, is limited to generating simple attack vectors and thus is incapable of being applied to a testbed environment for the generation of sophisticated intrusions. These types of intrusions include reconnaissance, multiple step attacks, host pivoting, tunneling or the generation of an infrastructure as seen in botnets.

The authors in Wright et al. (2010) construct Markov chain models of the way real users interact with each application in terms of event ID, process ID, and arrival time of each COM (Component Object Model) event on a given system. This model is then used to generated event streams and drive applications. Although their objectives are similar to our work, their central idea revolves around simulating the behavior of a user at the operating system GUI-based application level while interacting with lightweight server-side honeypots which simulate local and remote servers. Thus, network activity is a by-product of application execution as opposed to our work where user behavior is limited to what can be observed at the network level (various layers in the OSI model) through the correct simulation of application level protocols with emphasis on real interactions with local servers and also remote hosts on the Internet. Through this, we intend to limit and abstract the complexities of simulating user behavior at the operating system level.

It it worthy to note that works such as Sommer and Paxson (2010) have made interesting observations on anomaly-based network intrusion detection mechanisms and have provided recommendations to further improve research in this field. As part of their recommendations, they state that the main objective of evaluation is to gain insight into a system's capabilities and how it functions. They indicate that inorder to do so, datasets play a key role in demonstrating how well a system behaves. However, they also acknowledge the fact that obtaining such datasets is very difficult and must be done in collaboration with network operators in return for potential viable results. In this work we strive to provide a different means to obtain and share such datasets.

The systematic approach described in this work addresses the evaluation recommendations of Sommer and Paxson (2010). Inheret to this work is the generation of multiple datasets which is deemed vital to the sound evaluation of anomaly detection techniques in various environments. The generation of the datasets in a determinstic manner provides the technicalities to precisely select the required dataset characteristics (through the use of extracted $\beta$-profiles) and inturn provides a more fundamental way to assess the generated dataset and determine its shortcomings. Also by tracking $\alpha$-profiles in the final dataset, investigations can be made quicker, more accurate, and fully automated.

## 8. Comparison with other datasets

Many real network traces are readily available to the research community. This includes CAIDA (CAIDA, 2011), PREDICT (RTI International, 2011), The Internet Traffic Archive (Lawrence Berkeley National Laboratory, 2010), LBNL traces (Lawrence Berkeley National Laboratory and ICSI,), DARPA datasets

(Lincoln Laboratory, 2011), KDD'99 datasets (University of California, 2011), and DEFCON(The Shmoo Group, 2011).

Although many of these traces are invaluable to the research community, many, if not all, fail to satisfy one or more of the objectives described in Section 1. This work is mostly distinguished by the fact that the issue of data generation is approached from what other datasets have been unable to provide, to a satisfactory degree, for the network security research community.

As an example, CAIDA collects many different types of data and makes it available to the research community. Most of CAIDA's datasets are very specific to particular events or attacks. Many of its longer traces are anonymized backbone traces with their payload, sometimes their protocol information, destination, and so forth completely removed. All of the datasets mentioned above have similar shortcomings which will no doubt affect their effectivity as benchmarking datasets.

From other aspects, datasets provided by The Internet Traffic Archive suffer from heavy anonymization and lack the necessary packet information. In addition, they are mostly from the 90's which requires further analysis as to whether they still represent today's traffic patterns.

LBNL's internal enterprise traces are full header network traces, without payload and suffer from heavy anonymization to the extend that scanning traffic has been extracted and separately anonymized as to remove any information which could identify an individual IP.

The series of DARPA datasets were constructed for network security purposes, to simulate the traffic seen in a medium sized US Air Force base. Upon careful examination of DARPA'98 and '99 in Mchugh (2000) and Brown et al. (2009), many issues have been raised including their failing to resemble real traffic, and numerous irregularities due to the synthetic approach to data generation and insertion into the dataset. The KDD'99 dataset is also built based on the data captured in DARPA'98 which nonetheless suffers from the same issues.

The DEFCON dataset are also commonly used for evaluation of IDSs. The traffic produced during their Capture The Flag (CTF) competition is very different from the real-world network traffic since it mainly consists of intrusive traffic as opposed to normal background traffic. Due to this shortcoming, this dataset is usually used for evaluation of alert correlation techniques.

This work attempts to resolve the issues seen in other datasets by employing the concept of profiles in a systematic manner to generate datasets. Table 4 summarizes the comparison between the aforementioned datasets and also the dataset generated through the application of this systematic approach to fulfill the principal objectives outlined for qualifying datasets. All Datasets except for DEFCON were captured or generated over a realistic network configuration. However, DARPA-99 and KDD-99 do not contain a consistent trace due to post-capture synthetic attack traffic insertion.

Most available datasets are unlabeled as labeling is laboursome and requires conducting a comprehensive search to tag malicious activity. Although Intrusion Detection Systems help to reduce the work, there is no guarantee that all malicious activity is labeled. This has been a major issue with all datasets and one of the reasons behind the post-insertion of attack traffic in DARPA-99, so that malicious traffic can be labeled in a deterministic manner. Having seen the inconsistencies produced by traffic merging, this work has adopted a different approach to provide the same level of deterministic behavior with respect to malicious traffic by conducting malicious activity within the capturing period using available network resources and through the utilization of knowledge in $\alpha$-profiles. Through the use of logging, all ill-intended activity can be effectively labeled.

**Table 4 – Comparing various datasets according to the principal objectives outlined for qualifying datasets.**

| | | Realistic network configuration | Realistic traffic | Labeled dataset | Total interaction capture | Complete capture | Diverse/multiple attack scenarios |
|---|---|---|---|---|---|---|---|
| CAIDA (large traffic datasets) | | Yes[a] | Yes | No | No[c] | No[f] | No[b] |
| Internet Traffic Archive | BC | Yes[a] | Yes | No | Yes | No[g] | No[b] |
| | The rest | | | | No | No[e] | |
| LBNL | | Yes[a] | Yes | No | No[d] | No[h] | No[b] |
| DARPA-99 KDD-99 | | Yes | No | Yes | Yes | Yes | Yes[k] |
| DEFCON | | No | No[j] | No | Yes | Yes | Yes |
| Our dataset | | Yes | Yes[i] | Yes | Yes | Yes | Yes |

a No network configuration information available.
b Basic captured network traces.
c Only external traffic seen on the backbone.
d Inter-subnet traffic only.
e No payload available. Most are simply reduced/summarized trace information. Local IP addresses are usually renumbered.
f No payload available. In some cases, protocol information, destination, and flags have been also been removed.
g Contain no packet contents and no host or protocol information.
h No payload available. Suffers from heavy anonymization.
i To the extend specified by the profiles.
j Only intrusive traffic.
k Does not necessarily reflect current trends.

The extend and scope of traffic capture in terms of *complete capture* becomes relevant in situations where the information contained in the traces will breach the privacy of individuals or organizations. In order to prevent privacy issues, almost all publicly available datasets remove any identifying information, such as payload and some others, as indicated in Table 4, anonymize, crop header information, or just summarize flows. DARPA-99, in particular, attempts to overcome this issue by simulating a small network and producing background activity with scripts. These scripts focus on protocol correctness and average overall traffic rates (in 15 min intervals) but fail to properly account for more finer user activity or protocol behavior. The intent of $\beta$-profiles is to enable the storage of various types of information, from overall behavior to finer user and protocol behavior, in a repeatable and deterministic manner. In the currently generated dataset, user behavior has been the focus of the $\beta$-profiles and as previously stated in Section 3 and Section 4.2, it generates a distinct, previously unspecified, group level behavior.

Apart from background traffic, traces must contain malicious activity. Most captured datasets have little control over the malicious activities included in the trace. Other datasets such as DARPA-99 and DEFCON contain an assorted number of attacks, however, a major concern with evaluating anomaly-based detection approaches is the requirement of certain scales of malicious activity. Additionally, malicious activity tend to become outdated with the realization of more sophisticated attacks. To overcome these issues, the $\alpha$-profiles in this work tend to provide a far more deterministic means to generate malicious activity where the attacks can be clearly detailed, published, and replicated to evaluate previous results or future works. It also facilitates the generation of more up to date datasets that reflect current trends and are tailored to evaluate certain characteristics of detection mechanisms which are unique to themselves.

Network simulation environments (e.g. GTNetS (Riley, 2003)) may also appear suitable in generating datasets, but in practice, network simulators only focus on the correct simulation of lower layers of the TCP/IP stack, thus inhibiting the raw behavior seen from operating systems and their collective interactions. In addition, they lack the required functionality to exploit vulnerabilities and conduct attacks.

Network traffic modeling and simulation is also another related field. Very interesting and extensive work has been conducted in the realm of network traffic modeling including work done on modeling Internet traffic (Muscariello, 2006), and specifically HTTP (Mah, 1997) and email traffic (Eckmann et al., 2004; Ebel et al., 2002) which are two of the protocols that have been of focus in this work. Apart from the difficulties in modeling such behavior due to the dynamic behavior of human activity (Barabsi, 2005), it is not the objective of this work to generate realistic traffic for the purpose of network modeling and simulation. The objective was to establish a systematic approach for the generation of a dataset with appropriate background traffic which reflects, to some extent, normal traffic and is complementary to the applied malicious traffic. This was collectively believed to create a complete trace for use in applications of network intrusion detection. The work on network traffic can nonetheless be used in $\beta$-profiles to generate the required behavior.

## 9. Discussion and conclusion

When dealing with dataset generation, many concerns are raised with respect to what constitutes a perfect dataset, what defines the respective qualities of normal, malicious, or realistic traffic. We believe the guidelines presented in this paper provide a path and a template that ultimately leads to a dataset that inhibits normality, maliciousness, and realism. However, the question of whether quantitative measurements can be used to measure the quality of these characteristics is still up for debate. We believe that these attributes cannot be correctly quantified by solely looking at the dataset itself due to undefined boundaries but can be verified through the application of the dataset to specific methods. A method defines a context that clarifies domain boundaries for each characteristic and will consequently enable quantitative measurements. This work, by means of $\alpha$-and $\beta$-profiles, allow the generation of traffic for a particular context with specific requirements and applications.

Examining currently available datasets led this work toward framing a systematic approach to generate datasets with a tendency to avoid the various weak points described in Section 7. In this respect, it was deemed necessary to define a set of requirements for a reasonable dateset, most importantly (a) it should not exhibit any unintended property (both network and traffic wise) due to post-insertion of data, (b) be adequately labeled, (c) should cover all network interactions, and (d) be entirely non-anonymized.

Despite the importance of creating such datasets, there will always be deficiencies in any one particular dataset and primarily the reason why a *perfect* dataset doesn't yet exist and arguably will never exist. It is therefore, very much desired to move away from *static* and *one-time* datasets toward more dynamically generated datasets which not only reflect the traffic compositions and intrusions of that time, but are also modifiable, extensible, and reproducible. In this respect, by devising the systematic approach in this paper, a variety of *adequate* datasets are possible to generate for the purpose of analysis, testing, and evaluation of network intrusion detection systems from multiple different angles.

This systematic approach has been based on the creation of profiles which contain abstract representations of events and behaviors seen on the network. They have been implemented using various autonomous agents or applied through human assistance. By combining these profiles together, a diverse set of datasets can be generated each with a unique set of features, which cover a portion of the evaluation domain. By simply altering the combination of profiles, the composition (e.g. protocols) and statistical characteristics (e.g. request times, packet arrival times, burst rates, volume) of the resulting dataset can be controlled.

These profiles as well as the generated datasets are readily sharable and could be interchanged among collaborators and researchers without (major) privacy issues. For example, the procedures described here to create the background traffic, as well as the scenarios used to generate the attacks can easily be analyzed, modified, and reused accordingly. This will enable others to *recreate* the network behavior of various applications, protocols, hosts, intrusions, and so forth through the utilization of a subset of these profiles and on a need-by basis.

In real-world situations (with descent firewalls) the ratio of attack traffic over normal traffic is very small. This work tends to replicate the same sort of proportions and can be used to evaluate the sensitivity of any approach. Nonetheless, the approach itself is very versatile in generating datasets that can have an arbitrary amount of attack traffic for any purpose, through the addition of more $\alpha$-profiles at the execution or generation stage.

In order to generate a dataset, an infrastructure is necessary as profiles only provide the knowledge to model quantities of interest and describe attacks. In this work, the natural behavior of network connected nodes at multiple layers has been utilized through the physical implementation of a testbed with real live network devices and workstations.

To realistically generate the required background traffic in the dataset, $\beta$-profiles have been generated from the real activity of machines. They were used to generate the necessary request counts and timestamps utilized by agents. One of the concerns with $\beta$-profiles is that they give the impression of distribution models representing the syntactic activities of the respective entities and not their semantic behavior. This is however not the case, as these profiles can be used to describe distributions that represent semantic behavior. Considering the HTTP agents, to bring in semantic activity would mean to model the web addresses according to typical user activity and to have agents crawl those websites proportionally. Although, this seems difficult in cases where state is required to be transferred to a website.

Regarding the generated $\beta$-profiles, modeling the HTTP and SMTP flows using their probability distributions was seen as elegant and sufficient enough for generating a dataset that mimics a small network, close to what was seen within our center. It is to note that $\beta$-profiles are independent of any modeling mechanism and thus more sophisticated modeling approaches can be readily used.

Although much effort has been put into developing a systematic approach to dataset generation, some drawbacks are evident. Much of this revolves around the complexities that arise due to the need to initially generate and subsequently execute $\alpha$-and $\beta$-profiles. $\alpha$-profiles require specific knowledge as to how an attack is constructed. The generation of $\beta$-profiles requires filtered network traces which might not be easily obtainable. Some of this "normal" traffic might contain undetected traces of attacks which will affect the final extracted profile. However, the reusability of these profiles enables them to be readily shareable with the rest of the community once they are generated. Also, depending on the complexities of $\alpha$-profiles, they might require human assistance for execution, which might introduce variances in how an attack is performed and reduce the repeatability quality of a generated dataset. On another note, the generated datasets can be completely shared without privacy concerns which would greatly reduce the need for the regeneration of the datasets for re-evaluation purposes.

In the near future we intend to enhance the current dataset by improving the accuracy of the current $\beta$-profiles to better match user activity. We will also focus on updating and increasing the number of $\alpha$-profiles to diversify abnormal activity and to also raise the relative portion of malicious traffic. In respect to the general systematic approach, we intend to improve our execution infrastructure to enable automatic agent deployment through (semi-)automatic interpretation of $\alpha$-profiles and by that we aim to move toward complete dataset generation using autonomous agents.

## REFERENCES

Arboi M. The NASL2 reference manual. Nessus; 2005.

Barabsi AL. The origin of bursts and heavy tails in human dynamics. Nature 2005;435:207–11.

Brown C, Cowperthwaite A, Hijazi A, Somayaji A. Analysis of the 1999 DARPA/Lincoln laboratory IDS evaluation data with netadhict. In: Proceedings of the second IEEE international conference on computational intelligence for security and defense applications. Piscataway, NJ, USA: IEEE Press; 2009. p. 67–73.

CAIDA. The cooperative association for internet data analysis, http://www.caida.org/; 2011.

Cao J, Cleveland WS, Gao Y, Jeffay K, Smith FD, Weigle MC. Stochastic models for generating synthetic HTTP source traffic. In: IEEE INFOCOM; 2004.

Cuppens F, Ortalo R. LAMBDA: a language to model a database for detection of attacks. In: RAID '00: proc. of the third international workshop on recent advances in intrusion detection; 2000. p. 197–216.

Danzig PB, Jamin S. tcplib: a library of internetwork traffic characteristics. Technical Report. Computer Science Department, USC; 1991.

Ebel H, Mielsch LI, Bornholdt S. Scale-free topology of e-mail networks. Phys Rev E 2002;66:35–103.

Eckmann ST, Vigna G, Kemmerer RA. STATL: an attack language for state-based intrusion detection. J Comput Secur 2002;10: 71–103.

Eckmann JP, Moses E, Sergi D. Entropy of dialogues creates coherent structures in e-mail traffic. Proc Nat Acad Sci U S A 2004;101:14333–7.

Hong SS, Wu SF. On interactive internet traffic replay. In: RAID'05; 2005. p. 247–64.

Jain R. The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling. Wiley; 1991.

Kayacik HG, Zincir-heywood AN. Generating representative traffic for intrusion detection system benchmarking. In: Conference on communication networks and services research; 2005. p. 112–7.

Lan KC, Heidemann JS. Rapid model parameterization from traffic measurements. ACM T Model Comput S 2002;12:201–29.

Lawrence Berkeley National Laboratory and ICSI. LBNL/ICSI enterprise tracing project. www.icir.org/enterprise-tracing/.

Lawrence Berkeley National Laboratory. The internet traffic archive, http://ita.ee.lbl.gov/index.html; 2010.

Lincoln Laboratory MIT. DARPA intrusion detection evaluation, http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/index.html; 2011.

Mah BA. An empirical model of HTTP network traffic. In: Proceedings of the INFOCOM '97. Sixteenth annual joint conference of the IEEE computer and communications societies. Washington, DC, USA: Driving the Information Revolution, IEEE Computer Society; 1997. p. 592.

Mchugh J. Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln laboratory. ACM Trans Inf Syst Secur 2000;3:262–94.

McHugh J. The 1998 Lincoln laboratory IDS evaluation. In: Proceedings of the third international workshop on recent

advances in intrusion detection (RAID). London, UK: Springer-Verlag; 2000. p. 145–61.
Michel C, Mé L. ADeLe: an attack description language for knowledge-based intrustion detection. In: Sec '01: proceedings of the 16th international conference on Information security: trusted information. Norwell, MA, USA: Kluwer Academic Publishers; 2001. p. 353–68.
Muscariello, L., 2006. On internet traffic measurements, characterization and modelling. Ph.D. thesis.
Mutz D, Vigna G, Kemmerer RA. An experience developing an IDS stimulator for the black-box testing of network intrusion detection systems. In: Annual computer security applications conference; 2003. p. 374–83.
Networks S. Custom attack simulation language (CASL); 1999.
Paxson V. Empirically derived analytic models of wide-area TCP connections. IEEE/ACM Trans Netw 1994;2:316–36.
Rapid7. Metasploit penetration testing, http://www.metasploit.com; 2011.
Riley GF. The Georgia tech network simulator. In: Proceedings of the ACM SIGCOMM workshop on models, methods and tools for reproducible network research. New York, NY, USA: ACM; 2003. p. 5–12.
RTI International. PREDICT: Protected repository for the defense of infrastructure against cyber threats, http://www.predict.org/; 2011.
Sommer R, Paxson V. Outside the closed world: on using machine learning for network intrusion detection. In: Security and privacy, IEEE Symposium on; 2010. p. 305–16.
Sommers J, Yegneswaran V, Barford P. A framework for malicious workload generation. In: Internet measurement conference; 2004. p. 82–7.
Sommers J, Yegneswaran V, Barford P. Toward comprehensive traffic generation for online IDS evaluation. Technical Report. University of Wisconsin Madison; 2005.
Tavallaee M, Stakhanova N, Ghorbani AA. Toward credible evaluation of anomaly-based intrusion detection methods. Trans Sys Man Cyber Part C 2010;40:516–24.
The Shmoo Group. Defcon, http://cctf.shmoo.com/; 2011.
University of California. KDD Cup 1999 data, http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html; 2011.
Valgenti VC, Kim MS. Simulating content in traffic for benchmarking intrusion detection systems. In: Proc of 4th international ICST conference on simulation tools and techniques SIMUTools2011; 2011.
Weigle MC, Adurthi P, Hernndez-campos F, Jeffay K, Smith FD. Tmix: a tool for generating realistic TCP application workloads in ns-2. Comput Commun Rev 2006;36:65–76.
Wright CV, Connelly C, Braje T, Rabek JC, Rossey LM, Cunningham RK. Generating client workloads and high-fidelity network traffic for controllable, repeatable experiments in computer security. In: Recent advances in intrusion detection; 2010. p. 218–37.

**Ali Shiravi** received the B.Eng. degree in Software Engineering from Tarbiat Moallem University of Tehran, Iran and the M.Sc. degree from Sharif University of Technology, Tehran, Iran. He is currently working toward the Ph.D. degree in the Information Security Centre of Excellence, University of New Brunswick, Canada. His research interests include network security, network traffic analysis, graph theory, graph clustering, network simulation, network traffic profiling, and security data visualization.

**Hadi Shiravi** received the B.Eng. degree in Software Engineering from The university of Science and Culture, Iran and the M.Sc. degree from the University of New Brunswick, Canada. He is currently working toward the Ph.D. degree in the Information Security Centre of Excellence, University of New Brunswick, Canada. His research interests include network security, security data visualization, network traffic generation, network attacks, and botnet detection.

**Mahbod Tavallaee** received his bachelor degree in Software Engineering from Tarbiat Moallem University of Tehran in 2004. Right after his graduation, he was accepted for the Masters degree in Information Technology at the University of Tehran. Having finished his Masters in 2007, he worked as a Research Associate in Iran Telecommunication Research Center for about six months before starting his Ph.D. program at the University of New Brunswick in September 2007. Currently, he is an active member of the Information Security Center of Excellence (ISCX) at the University of New Brunswick, and his main field of research is Intrusion Detection Systems.

**Ali A. Ghorbani** received the B.Sc. degree from the University of Tehran, Iran, in 1976, the M.Sc. degree from George Washington University, Washington, DC, in 1979, and the Ph.D. degree from the University of New Brunswick (UNB), Fredericton, NB, Canada, in 1995. He is currently a Professor and Dean with the UNB, where he is the Director of Information Security Center of Excellence, and is also the coordinator of the Privacy, Security and Trust network annual conference. He holds a UNB Research Scholar position and is the Coeditor-in-Chief of the Computational Intelligence: An International Journal, and an Associate Editor of the International Journal of Information Technology and Web Engineering. His current research interests include web intelligence, network security, complex adaptive systems, critical infrastructure protection, and trust and security assurance. Dr. Ghorbani is a member of the Association for Computing Machinery, the IEEE Computer Society, and the Canadian Society for Computational Studies of Intelligence.