# ICIMP 2018

The Thirteenth International Conference on Internet Monitoring and Protection

ISBN: 978-1-61208-652-1

July 22 - 26, 2018

Barcelona, Spain

**ICIMP 2018 Editors**

Michael Massoth, Hochschule Darmstadt - University of Applied Sciences, Germany

# ICIMP 2018

## Foreword

The Thirteenth International Conference on Internet Monitoring and Protection (ICIMP 2018), held between July 22 - 26, 2018- Barcelona, Spain, continued a series of special events targeting security, performance, vulnerabilities in Internet, as well as disaster prevention and recovery.

The design, implementation and deployment of large distributed systems are subject to conflicting or missing requirements leading to visible and/or hidden vulnerabilities. Vulnerability specification patterns and vulnerability assessment tools are used for discovering, predicting and/or bypassing known vulnerabilities.

Vulnerability self-assessment software tools have been developed to capture and report critical vulnerabilities. Some of vulnerabilities are fixed via patches, other are simply reported, while others are self-fixed by the system itself. Despite the advances in the last years, protocol vulnerabilities, domain-specific vulnerabilities and detection of critical vulnerabilities rely on the art and experience of the operators; sometimes this is fruit of hazard discovery and difficult to be reproduced and repaired.

System diagnosis represent a series of pre-deployment or post-deployment activities to identify feature interactions, service interactions, behavior that is not captured by the specifications, or abnormal behavior with respect to system specification. As systems grow in complexity, the need for reliable testing and diagnosis grows accordingly. The design of complex systems has been facilitated by CAD/CAE tools. Unfortunately, test engineering tools have not kept pace with design tools, and test engineers are having difficulty developing reliable procedures to satisfy the test requirements of modern systems. Therefore, rather than maintaining a single candidate system diagnosis, or a small set of possible diagnoses, anticipative and proactive mechanisms have been developed and experimented. In dealing with system diagnosis data overload is a generic and tremendously difficult problem that has only grown. Cognitive system diagnosis methods have been proposed to cope with volume and complexity.

Attacks against private and public networks have had a significant spreading in the last years. With simple or sophisticated behavior, the attacks tend to damage user confidence, cause huge privacy violations and enormous economic losses.

The CYBER-FRAUD track focuses on specific aspects related to attacks and counterattacks, public information, privacy and safety on cyber-attacks information. It also targets secure mechanisms to record, retrieve, share, interpret, prevent and post-analyze of cyber-crime attacks.

Current practice for engineering carrier grade IP networks suggests n-redundancy schema. From the operational perspective, complications are involved with multiple n-box PoP. It is not guaranteed that this n-redundancy provides the desired 99.999% uptime. Two complementary solutions promote (i) high availability, which enables network-wide protection by providing fast recovery from faults that may occur in any part of the network, and (ii) non-stop routing. Theory on robustness stays behind the attempts for improving system reliability with regard to emergency services and containing the damage through disaster prevention, diagnosis and recovery.

We take here the opportunity to warmly thank all the members of the ICIMP 2018 Technical Program Committee, as well as all of the reviewers. The creation of such a high quality conference program would not have been possible without their involvement. We also kindly thank all the authors who dedicated much of their time and efforts to contribute to ICIMP 2018. We truly believe that, thanks to all these efforts, the final conference program consisted of top quality contributions.

Also, this event could not have been a reality without the support of many individuals, organizations, and sponsors. We are grateful to the members of the ICIMP 2018 organizing committee for their help in handling the logistics and for their work to make this professional meeting a success.

We hope that ICIMP 2018 was a successful international forum for the exchange of ideas and results between academia and industry and for the promotion of progress in the field of Internet monitoring and protection.

We are convinced that the participants found the event useful and communications very open. We hope that Barcelona provided a pleasant environment during the conference and everyone saved some time to enjoy the charm of the city.

**ICIMP 2018 Chairs:**

**ICIMP Steering Committee**
Sathiamoorthy Manoharan, University of Auckland, New Zealand
Terje Jensen, Telenor, Norway
Christian Callegari, University of Pisa, Italy

**ICIMP Industry/Research Advisory Committee**
Daisuke Mashima, Advanced Digital Sciences Center, Singapore
Bernhard Tellenbach, Zurich University of Applied Sciences, Switzerland
Miroslav Velev, Aries Design Automation, USA
Pethuru Raj, IBM Global Cloud Center of Excellence, India

# ICIMP 2018

## COMMITTEE

## Table of Contents

# Applying Lomb Periodogram to Round-trip Time Estimation from Unidirectional Packet Traces with Different TCP Congestion Controls

Toshihiko Kato, Xiaofan Yan, Ryo Yamamoto, and Satoshi Ohzahata
University of Electro-Communications
Tokyo, Japan
e-mail: kato@is.uec.ac.jp, yanxiaofan@net.is.uec.ac.jp, ryo_yamamoto@is.uec.ac.jp, ohzahata@is.uec.ac.jp

***Abstract*—Network operators often attempt to analyze traffic in the middle of their networks for various purposes. In such traffic analysis, the estimation of Round-Trip Time (RTT) is indispensable. Primarily, the RTT estimation is performed by consulting the relationship between a request and its response, such as a data segment and the associated ACK segment. However, in the middle of Internet, it is common that a network operator monitors traffic only in one direction. In such a case, an operator is required to estimate RTT from unidirectional packet traces. So far, several methods have been proposed for RTT estimation from unidirectional traces. In this paper, we adopt the Lomb periodogram method and apply it to various TCP traces, collected through Ethernet or wireless LAN, with different congestion control algorithms. As a result, the method can estimate RTT roughly, but the results are not accurate enough for subtle analysis, such as congestion window estimation.**

***Keywords- Unidirectional Packet trace; Round-trip Time; Lomb Periodogram; Congestion Control.***

## I. INTRODUCTION

Traffic analysis in the middle of Internet is an important issue for network operators. It can be applied the traffic classification, the traffic demand forecasting, and the malicious traffic detection. In the previous paper, we proposed a method to infer TCP congestion control algorithm from passively collected packet traces [1]. It adopts the following approaches.

(1) Focus on a specific TCP flow using source/destination IP addresses and ports.
(2) From the mapping between data segments and acknowledgment (ACK) segments, estimating Round-Trip Time (RTT) of the focused flow.
(3) Estimate a congestion window size (cwnd) from the data size transferred during one RTT.
(4) Obtain a sequence of cwnd values, and calculate a sequence of cwnd difference between adjacent cwnd values (we call ⊿cwnd).
(5) From the mapping between cwnd and ⊿cwnd, infer a congestion control algorithm for the TCP flow.

This method requires a bidirectional trace to obtain both data and ACK segments.

In actual networks, however, it is often possible that only unidirectional traces are collected in the middle of networks. In this case, the above method cannot be applied. So, in another previous paper, we tried to modify the above method to infer TCP congestion control algorithms from unidirectional traces [2]. In the modified method, a fixed time duration is used instead of RTT, and data size transferred during this duration was handled as cwnd. As a result, congestion control algorithms were estimated in some cases, but not in other cases. This is because our method depends largely on RTT value.

On the other hand, the estimation of RTT from traces has been actively studied and there are several proposals [3]-[6]. The RTT estimation methods proposed so far are classified into three categories. One is a method called Data-to-ACK-to-Data, which measures time between a data segment and the data segment sent just after the first data segment is ACKed [3]-[5]. This requires bidirectional packet traces and our first paper used it. Next is a method based on the autocorrelation [4][5]. This method counts the number of data segments in a short interval, and makes an array of counts indexed by the normalized interval. Then, it calculates the autocorrelation over the array and takes the maximum as a RTT. This method can be applied to unidirectional packet traces. The third one is use of spectral analysis [5][6]. A sequence of data segments are handled as a pulse function of time, which takes 1 when there is a data segment. Then, the frequency characteristic of this function is analyzed and the inverse of first harmonic is taken as RTT. Since the interval of data is irregular, the special analysis is performed by the Lomb periodogram [7].

For the purpose of precise RTT estimation from unidirectional traces, we adopt the third method because it can work for various type of traffic [5]. In this paper, we apply the Lomb periodogram to the RTT estimation from unidirectional packet traces, which we used in our previous papers, including different TCP congestion control algorithms, and discuss the results in detail. The rest of this paper is organized as follows. Section II explains the related work, including the problems we suffered from in our second previous paper [2] and the conventional RTT estimation methods. Section III describes a detailed scheme to estimate RTT using the Lomb periodogram. Section IV gives the results of RTT estimation for different TCP congestion control algorithms. In the end, Section V concludes this paper.

## II. RELATED WORK

### A. *Problems on congestion window size estimation from unidirectional traces*

In our previous papers [1][2], we collected packet traces in the configuration shown in Figure 1. A TCP data sender is connected with a bridge through 100 Mbps Ethernet. The bridge inserts 100 msec RTT (50 msec delay for each direction) and 0.01% packet losses. The bridge is connected with a TCP data receiver through IEEE 11g wireless LAN

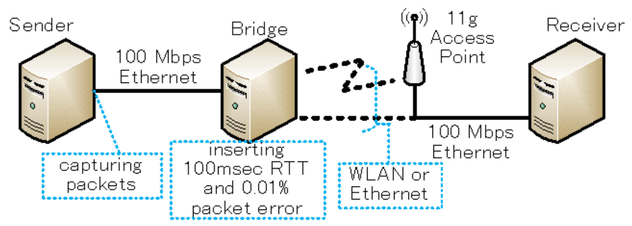


Figure 1. Experiment configuration.

(WLAN) or 100 Mbps Ethernet. The packet trace is collected at the TCP sender side. The collected traces include bidirectional ones, and in the unidirectional analysis, we picked up only data segments from the TCP sender to the TCP receiver.

Figures 2 and 3 show the results for CUBIC TCP [8] and TCP Vegas [9]. In the analysis a from bidirectional trace, cwnd and ⊿cwnd are estimated in the way described in Section I, and their relationship is given in the figures (by blue dots). In the analysis from a unidirectional trace, we assumed that RTT is 100 msec. The data size transferred during 100 msec and its difference are called sentData and ⊿sentData, respectively, and shown in the figures by orange dots. In the case of CUBIC TCP, both results show the similar graph, which is a function in the form of $\left(\sqrt[3]{cnwd}\right)^2$ with decreasing and increasing parts [1]. This result means that the unidirectional analysis works well. In the case of TCP Vegas, however, the results for bidirectional analysis and unidirectional analysis are significantly different. According to the Vegas algorithm, ⊿cwnd takes 1,460 bytes (one segment size), 0, or -1,460 bytes independently of cwnd values, which is represented by the blue dots [1]. But, in the result for unidirectional analysis, the ⊿sentData values indicated by the orange dots are unstable. So, the unidirectional analysis does not work well.

In our experiment, the trace for CUBIC TCP is collected in the configuration that uses Ethernet between the bridge and the TCP receiver, and that for TCP Vegas is collected by use of WLAN. This is one of the reasons. Figure 4 shows examples of the time variation of TCP sequence number for CUBIC TCP and TCP Vegas. In the case of CUBIC TCP, data segments are transferred in groups and there are idle time periods without any data transmissions. Therefore, in the unidirectional analysis, a sequence of data segments sent within a congestion window can be traced by use of 100 msec, which is a RTT determined tentatively. But, in the case of TCP Vegas, data segments are transmitted contiguously, and therefore, if RTT is not estimated correctly, a sentData value does not match the real cwnd value.

There considerations mean that the RTT estimation is critical for inferring TCP congestion control algorithms.

## B. Related work on RTT estimation

As described in Section I, the RTT estimation methods are classified into three categories; the Data-to-ACK-to-Data method, the autocorrelation based method, and the spectral analysis method.

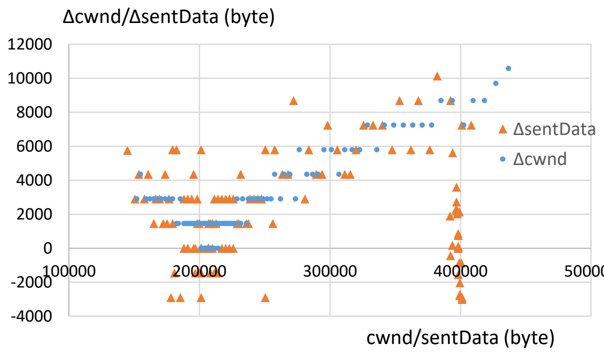


Figure 2. Result for CUBIC TCP [1][2].

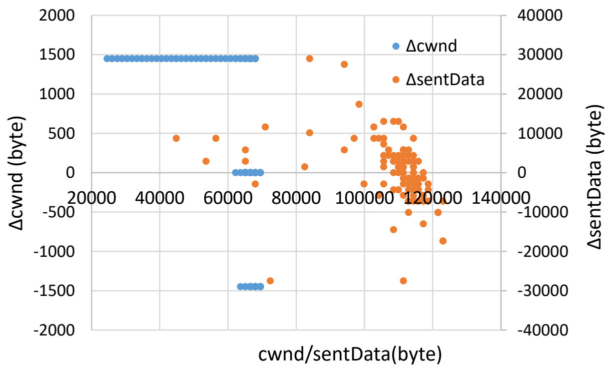
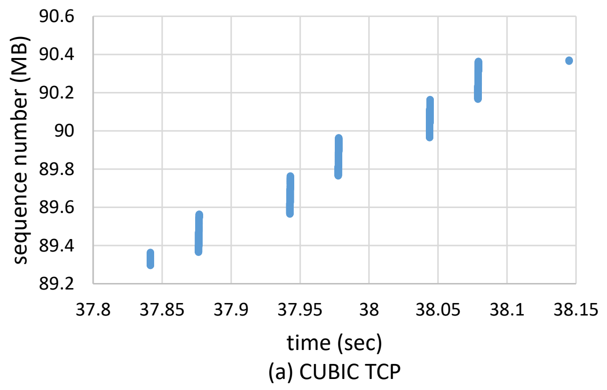


Figure 3. Result for TCP Vegas [1][2].



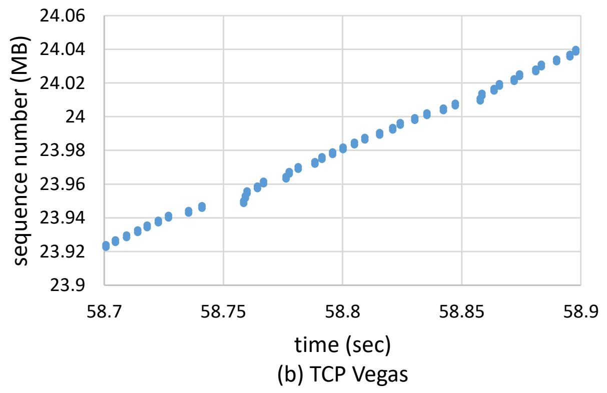


Figure 4. Sequence number vs. time.

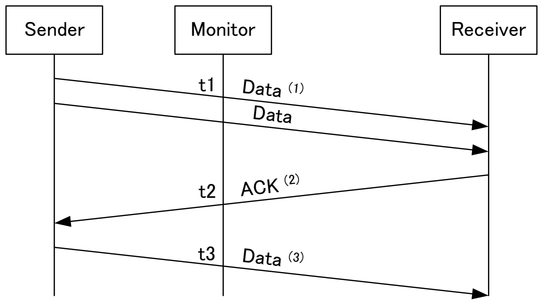


Figure 5.  Data-to-ACK-to-Data method.

The Data-to-ACK-to-Data method is illustrated in Figure 5.  Since there is some transmission delay between a TCP data sender and a monitor capturing packet traces, the following procedure is used to estimate RTT between sender and receiver.  (1) A monitor focuses on a data segment, and remembers the time (t1).  (2) A monitor catches the ACK segment that acknowledges the data segment.  (3) A monitor detects the data segment sent by the sender just after the ACK segment in (2), and remember the time (t2).  (4) t2 – t1 is a RTT for this moment.  In order to detect data segment (3), the TCP time stamp option is used.

In the autocorrelation based method, the RTT estimation is performed once per measurement interval $T$.  An array $P[n]$ maintaining the count of data segments is prepared using unit time $\Delta t$, where $n$ is ranging from 0 to ${T}/{\Delta t} - 1$.  If a data segment is detected at an interval $[start\ time + m \cdot \Delta t,$ $start\ time + (m+1) \cdot \Delta t)$, one is added to $P[m]$.  For all the data segments from *start time* to *start time* +$T$, the array $P[n]$ is arranged.  After that, the autocorrelation function is defined as

$$A(l) = \frac{1}{{T}/{\Delta t} - l} \sum_{j=1}^{{T}/{\Delta t} - l} P[j] \cdot P[j+l]. \qquad (1)$$

for lags $l = 0 \cdots {T}/{\Delta t} - 1$.  RTT is computed as $\max(A)$.  This method can be applied to the unidirectional analysis, and will work well for the cases that data segments are distributed unevenly in a trace, such as the case of CUBIC TCP in Figure 4.  However, for an evenly distributed trace, such as the case of TCP Vegas in Figure 4, it is concerned that RTT cannot be estimated correctly.

The spectral analysis method will be the most promising for RTT estimation among the three methods. Traditional spectral analysis, such as Fast Fourier Transform (FFT) assume that time domain data are regularly sampled [10]. However, in the RTT estimation, the time domain data is packet inter-arrival time of a specific flow.  This data is sampled at each data packet capturing.  This means that the time domain data in this case is irregularly sampled.  In the case of the spectral analysis for irregularly sampled data, the Lomb periodogram is commonly used [6].  The details are shown in the next section.

## III.  RTT Estimation Using Lomb Peridgram

In the RTT estimation based on the Lomb periodogram, time sequence $\{t_i\}$ $(i = 1, \cdots)$ is considered as an input, where $t_i$ corresponds to one data segment capturing time.  At a specific time $t_k$, the frequency characteristic of this time sequence is calculated using $N$ time samples $t_{k-N+1}, \cdots t_k$ in the following way $(N > k)$ [6].

- The minimum and maximum frequencies of the range for power spectrum are defined as
  $$f_k^{min} = \frac{1}{t_k - t_{k-N+1}} \text{ and } f_k^{max} = \frac{N}{2} f_k^{min}.$$
  Accordingly, the power spectrum is calculated for angular frequency
  $$\omega_i = 2\pi f_k^{min} + i\Delta_\omega \quad (i = 0, \ldots 2N-1),$$
  where $\Delta_\omega = 2\pi \frac{f_k^{max} - f_k^{min}}{2N}$.
- The power spectrum at angular frequency $\omega_i$ is defined as
  $$P_k^N(\omega_i) = \frac{1}{2\sigma_k^2} \left\{ \frac{\left[\sum_{j=0}^{N-1} (h_{k-j} - \bar{h}_k) cos\omega_i (t_{k-j} - \tau_k)\right]^2}{\sum_{j=0}^{N-1} cos^2 \omega_i (t_{k-j} - \tau_k)} + \frac{\left[\sum_{j=0}^{N-1} (h_{k-j} - \bar{h}_k) sin\omega_i (t_{k-j} - \tau_k)\right]^2}{\sum_{j=0}^{N-1} sin^2 \omega_i (t_{k-j} - \tau_k)} \right\} \qquad (2)$$
  where $\bar{h}_k$ and $\sigma_k^2$ are the mean and variance of $N$ samples of $h_k$:
  $$\bar{h}_k = \frac{1}{N} \sum_{j=0}^{N-1} h_{k-j} \qquad (3)$$
  $$\sigma_k^2 = \frac{1}{N-1} \sum_{j=0}^{N-1} h_{k-j}^2 - \frac{N}{N-1} \bar{h}_k^{\ 2}, \qquad (4)$$

and where  is the solution of:

$$tan(2\omega_i \tau_k) = \frac{\sum_{j=0}^{N-1} sin 2\omega_i t_{k-j}}{\sum_{j=0}^{N-1} cos 2\omega_i t_{k-j}}. \qquad (5)$$

From the $2N-1$power spectrum values specified in an $\omega - P(\omega)$ plane, local maximum values are calculated. Among the frequencies generating local maximum power spectrum values, the fundamental frequency $f_0$ is estimated under the condition that other frequencies generating local maximum values are multiples of $f_0$.  At last, $T = {1}/{f_0}$ is the estimated RTT.

## IV.  Results of Aplying  Peridgram to Various Congestion Control Algorithms

This section describes the results of RTT estimation for various types of TCP traces with different congestion control algorithms.  We use the packet traces used in our previous papers [1][2].  As described in Section II.A, these traces are collected at the sender side in the configuration shown in Figure 1.  Since packet losses are inserted at the bridge, we picked up a part of packet traces where no packet losses are detected, that is, where the sequence number of TCP segments keeps increasing.  The traces themselves have bidirectional packet information and only the capturing time of data segments is extracted to build unidirectional traces.  Together with the extraction, the real RTT is calculated from the mapping between data segments and ACK segments.

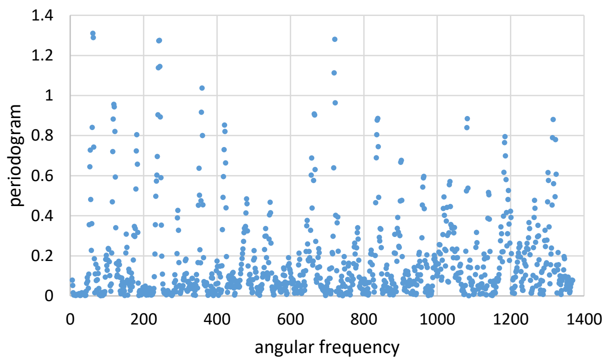
### A.  Result for traces including TCP Reno

TCP Reno is a classic congestion control method which adopts an additive increase and multiplicative decrease (AIMD) algorithm.  Here, cwnd is increased each time the TCP sender receives an ACK segment acknowledging new data.  The increase is $\frac{1}{cwnd}$ segments during the congestion avoidance phase, and as a result, cwnd is expected to be increased by one segment during one RTT.

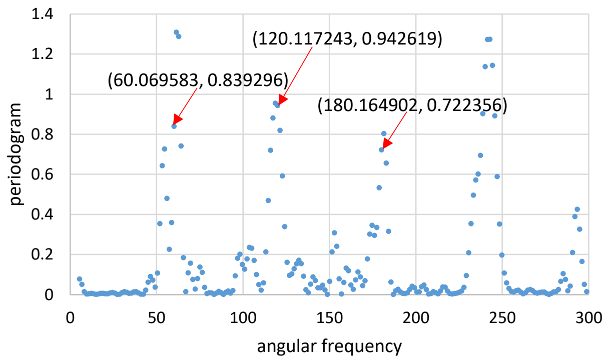
The Reno packet trace we used here is collected in the network configuration with Ethernet (see Figure 1), and we picked up a part from 27.010458 sec. to 45.99513 sec. in the trace, where there no retransmissions are detected for 7068 data segments. We used $N$ = 500 in calculating the Lomb periodogram.

Figure 6 shows a result of RTT estimation from the Reno trace. Figure 6(a) is the result for periodogram at time 28.156143 sec. The horizontal axis is an angular frequency and the vertical axis is a periodogram. This figure shows there are several peaks periodically. Figure 6(b) zooms up the low angular frequency part of Figure 6(a). It shows that there are harmonized frequencies such that there are large periodogram values at some frequencies which are integral multiple of a specific frequency (fundamental frequency $f_0$). In Figure 6(b), angular frequencies 60.069583, 120.117243, and 180.164902 are those frequencies. From this result, we can conclude that $2\pi f_0 = 60.069583$. So, we obtain $f_0 = 9.56037123$ and RTT = $^1/_{f_0}$ = 0.10459845 sec.

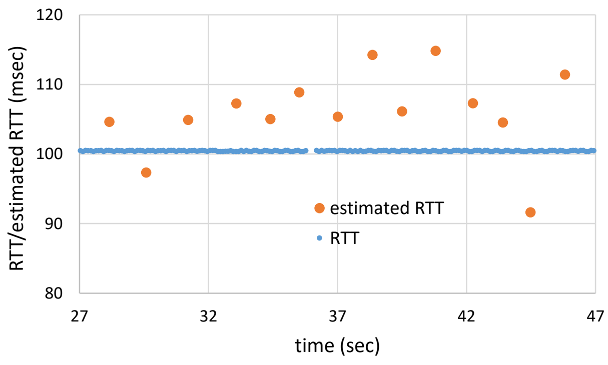
We conducted similar calculations for multiple points of time in the trace and obtained the estimated RTT as shown in Figure 6(c). This figure also gives actual RTT values obtained from data and ACK segments in the original trace information. This result says that, although the actual RTT is extremely stable at 100 msec, the estimated RTT includes some errors in the order of 10 msec. The reason that the actual RTT is stable is that this experiment is conducted through only Ethernet and that there are no large delay variations. However, the RTT estimation by use of the Lomb periodogram cannot reflect this situation.



(a) periodogram at time 27.03713 sec.



(b) zooming up low angular frequency part



(c) estimated RTT and actual RTT
Figure 6. RTT estimation from Reno trace.

## B. Result for traces including CUBIC TCP

As described in Section II.A, CUBIC TCP defines *cwnd* as a cubic function of elapsed time $T$ since the last congestion event [8]. Specifically, it defines cwnd by (6).

$$cwnd = C\left(T - \sqrt[3]{\beta \cdot \frac{cwnd_{max}}{C}}\right)^3 + cwnd_{max} \qquad (6)$$

Here, $C$ is a predefined constant, $\beta$ is the decrease parameter, and $cwnd_{max}$ is the value of *cwnd* just before the loss detection in the last congestion event. Comparing with TCP Reno, cwnd increases faster in CUBIC TCP.

We estimated RTT from the unidirectional packet trace including only data segments with CUBIC TCP. The trace is collected in the configuration using only Ethernet. We picked up a part in the trace from 23.483123 sec. to 38.348383 sec. for the RTT estimation. By applying the Lomb periodogram similarly with the case of Reno, we obtained estimated RTT as shown in Figure 7. This figure also gives actual RTT values.

The results show that the actual RTT is stable at 100 msec. and, on the other hand, the estimated RTT changes a lot between 90 msec. and 140 msec. The fluctuation is larger for CUBIC than TCP Reno. Especially, the difference between the estimated RTT and the actual RTT becomes large when the time is between 36 sec. and 38 sec. During this period, the cwnd value itself becomes large and the large cwnd value may give some bad influence to the RTT estimation.

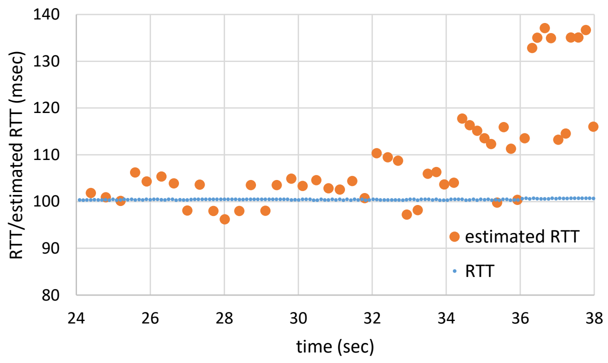


Figure 7. RTT estimation from CUBIC trace.

## C. Result for traces including TCP Vegas

TCP Vegas estimates the bottleneck buffer size using the current values of *cwnd* and RTT, and the minimal RTT for the TCP connection, according to (7) [9].

$$BufferSize = cwnd \times \frac{RTT - RTT_{min}}{RTT} \qquad (7)$$

At every RTT interval, Vegas uses this *BufferSize* to control *cwnd* in the congestion avoidance phase in the following way.

$$\varDelta cwnd = \begin{cases} 1 & (BufferSize < A) \\ 0 & (A \leqq BufferSize \leqq B) \\ -1 & (BufferSize > B) \end{cases} \qquad (8)$$

Here, A = 2 and B = 4 (in unit of segment) are used in the Linux operating system.

We estimated RTT from the unidirectional packet trace including only data segments with TCP Vegas. In this case, in contrast with the above cases, the trace is collected in the configuration using WLAN. We picked up a part in the trace from 37.988347 sec. to 59.699611 sec. for the RTT estimation. By applying the Lomb periodogram to this time sequence, we obtained estimated RTT as shown in Figure 8, with actual RTT values.

In this case, the estimated RTT is stable around 100 msec, and on the other hand, the actual RTT values are scattered between 100 msec. and 140 msec. That is, although the actual RTT is changing, the RTT estimated by the Lomb periodogram does not follow the fluctuation. As we indicated in Section II.A, the timing of capturing data segments is almost uniformly distributed in this case. As a result, it is considered that the Lomb periodogram method cannot detect the actual RTT.

## D. Result for traces including TCP Veno

TCP Veno (Vegas and ReNO) is an example of hybrid type congestion control method, considering packet losses and delay. It uses the *BufferSize* in (7) to adjust the growth of *cwnd* in the congestion avoidance phase as follows. If $BufferSize > B$ (*B* is the Vegas parameter *B*), cwnd grows by 1/cwnd for every other new ACK segment, and otherwise, it grows in the same manner with TCP Reno. That is, when the congestion status is heavy, i.e., the bottleneck buffer size is large, the increasing rate of cwnd is halved.

We estimated RTT from the unidirectional Veno trace captured in the WLAN configuration in Figure 1. We picked up a part in the trace from 37.684643 sec. to 52.653736 sec. including 23,360 data segments. By applying the Lomb periodogram to this sequence, we obtained estimated RTT as in Figure 9, which also gives the actual RTT.

Similarly with the case of TCP Vegas, the estimated RTT is rather stable around 100 msec., which is different from the actual RTT spreading in the rage between 100 msec. and 130 msec.

## V. CONCLUSIONS

This paper described the results of applying the Lomb periodogram method to estimating RTT from unidirectional packet traces including TCP segments with different congestion control algorithms, TCP Reno, CUBIC TCP, TCP Vegas, and TCP Veno. Among them, the packet traces for TCP Reno and CUBIC TCP are collected in the network configuration using only Ethernet, and those for TCP Vegas and TCP Veno are from WLAN configuration. The performance evaluation gave the following results.

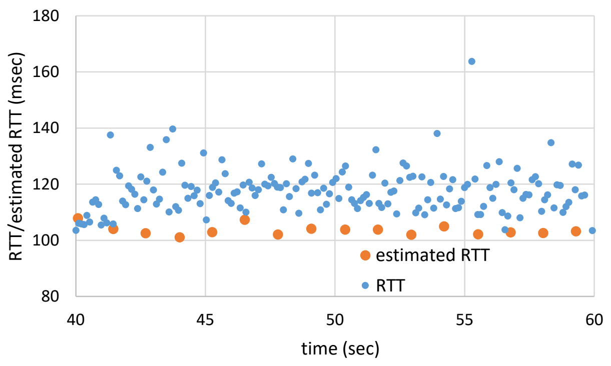


Figure 8. RTT estimation from Vegas trace.

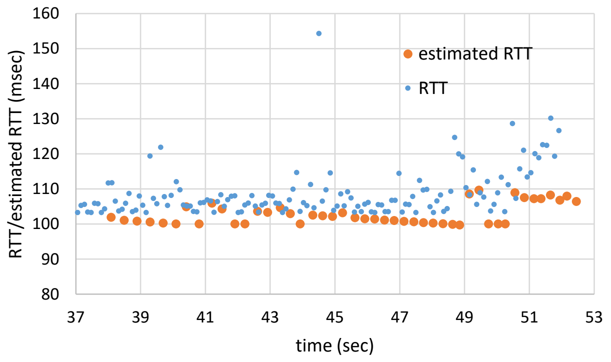


Figure 9. RTT estimation from VENO trace.

First of all, the Lomb periodogram method was possible to estimate an approximate RTT values from unidirectional packet traces. Strictly speaking, however, the estimated RTT values have some errors and they are not tolerable for the approaches that require accurate RTT estimation, such as our method to infer the TCP congestion algorithms from unidirectional packet traces [2]. Moreover, although the experiments adopted here added a fix delay, actual TCP communications suffer from variable delay like Bufferbloat [12]. So, the accurate estimation will be more difficult in real environments.

The second point is that the estimation is affected largely by the network configuration, such as with Ethernet or with WLAN. It is also affected somehow by the congestion control used in packet traces. In our experiment, the traces of TCP Reno and CUBIC TCP were collected in an Ethernet configuration. In this case, the actual RTT was stable and the estimated RTT was fluctuated. In the CUBIC TCP trace, where the congestion control is more aggressive, the errors of the estimated RTT increased. On the other hand, the traces of TCP Vegas and TCP Veno were collected in a WLAN configuration. In this case, while the actual RTT was fluctuated, the Lomb periodogram method could not estimate this fluctuation and the estimated RTT was stable.

Since it is difficult to estimate RTT correctly from unidirectional packet traces, we need to develop a new method to infer TCP congestion control algorithms from unidirectional traces.

## REFERENCES

[1] T. Kato, A. Oda, C. Wu, and S. Ohzahata, “Comparing TCP Congestion Control Algorithms Based on Passively Collected Packet Traces,” IARIA ICSNC 2015, pp. 135-141, Nov. 2015.

[2] T. Kato, L. Yongxialee, R. Yamamoto, and S. Ohzahata, “How to Characterize TCP Congestion Control Algorithms from Unidirectional Packet Traces,” IARIA ICIMP 2016, pp. 23-28, May 2016.

[3] H. Jiang and C. Dovrolis, “Passive Estimation of TCP Round-Trip Times,” ACM SIGCOMM Comp. Commun. Rev. vol. 32, issue 3, pp. 75-88, Jul. 2002.

[4] B. Veal, K. Li, and D. Lowenthal, “New Methods for Passive Estimation of TCP Round-Trip Times,” Passive and Active Nework Measurement, PAM 2005, LNCS, vol. 3431, pp. 121-134.

[5] R. Lance and I. Frommer, “Round-Trip Time Inference Via Pasive Monitoring,” ACM SIGMETRICS Perf. Eval. Rev., vol. 33, issue 3, pp. 32-38, Dec. 2005.

[6] D. Carra et al., “Passive Online RTT Estimation for Flow-Aware Routers Using One-Way Traffic,” NETWORKING 2010 LNCS6091, pp. 109-121, 2010.

[7] J. Scargle, “Statistical aspects of spacial analysis of unevently spaced data,” J. Astrophysics, vol 263, pp. 835-853, Dec. 1982.

[8] S. Ha, I. Rhee, and L. Xu, “CUBIC: A New TCP-Friendly High-Speed TCP Variant,” ACM SIGOPS Op. Syst. Review, vol. 42, issue 5, pp. 64-74, Jul. 2008.

[9] L. Brakmo and L. Peterson, “TCP Vegas: End to End Congestion Avoidance on a Global Internet,” IEEE J. Sel. Areas Commun., vol. 13, no. 8, pp. 1465-1480, Oct. 1995.

[10] S. Kay and S. Marple, “Spectrum analysis; A modern perspective,” Proc. of the IEEE, vol. 69, issue 11, pp. 1380-1419, Nov. 1981.

[11] C Fu and C. Liew, “TCP Veno: TCP enhancement for transmission over wireless access networks,” IEEE J. Sel. Areas Commun., vol. 21, no. 2, Feb. 2003.

[12] S. Strowes, “Passively Measuring TCP Round-trip Times,” ACM Queue, vol. 11, issue 8, pp. 1-12, Aug. 2013.

# Network Visibility-aware Blacklist Generation

Pierre Edouard Fabre*†, Jouni Viinikka*, Hervé Debar† and Gregory Blanc†
*6cure
g4200 Herouville-Saint-Clair, France
Email: {pef, jvi}@6cure.com
† Institut Mines-Telecom, Telecom SudParis
CNRS Samovar UMR 5157, 91011 Evry , France
Email: {herve.debar,gregory.blanc}@telecom-sudparis.eu

***Abstract*—Volumetric Distributed Denial of Service (DDoS) attacks have become a major concern for network operators, as they endanger the network stability by causing severe congestion. Access Control Lists (ACLs), and especially blacklists, have been widely studied as a way of distributing filtering mechanisms at network entry points to alleviate the effect of DDoS attacks. Different blacklist generation approaches, as proposed in the literature, are dependent on the information available on the network traffic. Nonetheless, the collection of traffic information comes at a cost that increases with the level of detail. To study the impact of the level of detail available, we formulate three scenarios. Each scenario describes a typical collection granularity used by operators. We then define blacklist generation algorithms corresponding to each granularity. Scenarios are evaluated with a mix of real legitimate and generated attack traffic. The evaluation shows that the amount of information does have an impact on the attack filtering results, and that one should choose the blacklist generation algorithms in regard of the available level of detail. Experiments also show that having more information does not always translate to more efficient filtering.**

***Keywords*—volumetric DDoS; network monitoring; ACLs; blacklists.**

## I. INTRODUCTION

The volume of bandwidth-depleting Distributed Denial of Service (DDoS) attacks has repeatedly reached new records in the recent years. In addition to disrupting the targeted service, these attacks can cause congestion at different points upstream from the actual target, creating wider perturbations.

Any mitigation solution downstream from a choke point will be ineffective [1], [2], as the saturation of an upstream link causes losses of legitimate traffic as well before it reaches the mitigation solution.

A distributed deployment of mitigation solutions could allow them to act before the funneling effect of attack traffic converging towards the target becomes too important. Although researchers have proposed distributed deployment strategies [3]–[5], the financial cost associated with the large number of nodes to deploy is often prohibitive.

Another option would be to use existing, widely deployed equipment, e.g., routers, for mitigation. Routers, for example, implement different mechanisms, such as FlowSpec and Access Control List (*ACL*), that can be used to drop potentially a large part of a volumetric DDoS attack, depending on the attack characteristics and thus to alleviate the congestion. The remaining part of the attack traffic may then be filtered with a more precise, dedicated solution [6]. The coarse granularity of filtering mechanisms available in network equipment is likely to cause collateral damage, i.e., legitimate traffic being filtered. Researchers have already worked on blacklist generation algorithms to create efficient filtering lists aiming at reducing collateral damage while maximizing the attack traffic filtering [7]. These algorithms typically take as input information on legitimate and malicious traffic, including lists of legitimate client and attacker IPs.

In this paper, we focus on the impact of *network visibility* on the blacklist generation problem. By network visibility we mean the availability of traffic information, and to the best of our knowledge, this impact has not been studied yet. We define and examine several scenarios reflecting the different levels of information a network operator has access to. We study the efficiency of blacklists deployed on a single node - distributed and/or collaborative filtering schemes are not considered. Finally, we provide means to find a trade-off between the level of visibility - increased visibility comes with increased cost - and efficiency of filtering.

The rest of the paper is organized as follows, Section II provides definition which our work is based on and the generic blacklist assumption. The Section III is an overview of the literature in the traffic filtering area. Section IV lays the fundamental problem of information availability to generate blacklists and formulate scenarios depicting levels of network visibility. In Section V, we detail blacklist generation scheme designed to fit in these scenarios. Section VI describes experiments and discusses their results. Finally, Section VII concludes the paper.

## II. BACKGROUND

Our study is focused on the mitigation of bandwidth-depleting DDoS attacks. This section provides definitions used in the remainder of this paper, describes the threat landscape, and states our underlying the assumptions.

### A. Definitions

We will be using the following definitions in this paper.

*Aggregate* is a network address prefix aggregate as used in route aggregation.

*Detection system* is any system capable detecting volumetric DDoS attack and reporting the source and destination addresses participating in the attack. A detection system can be external to the network being monitored.

*Monitoring system* is any system providing network telemetry for the monitored system. For our needs, we expect the telemetry to include at least traffic volumes between source and destination addresses, eventually at some level of aggregation.

*IP flow* (IPf) is a stream of packets, sharing the tuple $<$ source IP, destination IP $>$. Defined this way, the flow includes only one direction of traffic and for example a TCP connection will result in two IP flows.

*Malicious aggregate* (MALagg) is an aggregate of traffic, defined as a set of one or more IPf that, according to a detection mechanism, contains malicious traffic. It should be noted that due to the coarse granularity of its definition, such an IPf can also contain legitimate traffic.

*Monitored aggregate* (MONagg) is a set if one or more IPf as observed by a monitoring system. Depending on the monitoring system's configuration, it reports MONaggs with a particular granularity, eventually aggregating source and/or destination addresses. In other words, MONagg are defined by the source and destination network addresses (both using CIDR), where the source and destination netmasks are fixed.

*Rule* denotes an aggregate of source IPs, one destination IP, and an action for the matching traffic. In our case, the action is always deny, i.e., packets matching a rule are dropped. We call the tuple $<$ network source prefix, destination IP $>$ of a rule a filtered aggregate (FILagg). Note that we use source prefix aggregation as explained in Section V-A.

*Access Control List* (*ACL*) is a set of rules against which traffic is matched by the filtering mechanism. Network equipment often implement ACLs in hardware [8], for performance reasons. On the other hand, hardware implementation becomes with size constraints, and we denote the maximal number of rules in an ACL with $N$.

## B. Threat Landscape

Volumetric DDoS (i.e., bandwidth-depleting DDoS) attacks aim at disrupting a service by consuming the incoming bandwidth and causing congestion at the target, or upstream from the final target.

From the victim (i.e., the final target or a congested network) point of view, the attack sources can appear either as spoofed or not. This means that the malicious traffic's source addresses are faked or real. In fact an attacker can, in some cases, falsify the source IPs of the traffic. In this work, we only consider non spoofed attacks. ghis is a reasonable statement for at least two reasons.

Considering amplification DDoS attacks, which represent a large portion of volumetric DDoS attack [9], massive part of traffic (i.e., from amplifiers towards target) is unspoofed. In fact, the source IP addresses match the sources of traffic, i.e., the amplifier's IPs. Consequently, the number of sources seen in the attack is limited by the number of amplifiers the attacker can find and abuse.

In addition, current direct attacks using Internet of Things (IoT) botnets pave the way to the use of protocols that required non spoofed IP addresses. That is the case of the attack against the Krebsonsecurity website [10] for which attackers made use of the GRE protocol. Remarkably, some direct massive attacks do not make use spoofed traffic, such as the one that hit OVH [11]. The accumulated volume of malicious traffic at each of its network entry points reached around 1Tbps. More than 145k simultaneous non spoofed sources (particularly IoT devices) have been identified as participant of this attack.

## C. Assumptions

A prerequisite for blacklisting is the identification of the items to block. In networking, an item refers to network traffic, which can be identified with header fields such as IP addresses, layer 4 protocol and ports. While the detection of the attack is not in the scope of this paper, we expect to obtain alerts containing an exhaustive list of IPfs, i.e., $<$ source IP, target IP $>$ tuples associated with the attack. We consider that this IPf's granularity is a trade-off between the network requirements and mitigation capabilities. In fact, it is coarse enough to be reasonable assumption for the majority of network operator. Besides, it can be regarded as acceptable, in regard to the mitigation, as Pack et al. [6] stated that ACLs can be used as a coarse pre-filter in combination with a finer grained mitigation, such as a middle-box. The middle-box could then trigger an alert using DOTS [12] or IDMEF [13] formats, so that it will include the identification of MALaggs.

## III. State of The Art

Filtering traffic is an essential function in a network to mitigate attacks with distributed sources. While some researchers build workarounds to network equipment limitations, the network industry improves the implementation of Access Control Lists in off-the-shelf equipments. This section first provides a review of traffic filtering methods and then we detail the use of ACLs from academic and industrial points of view.

## A. Traffic Filtering

Middle-boxes, as proposed for example by Tan et al. [14], aim at providing traffic filtering functions that routers do not implement. Generally, these functions allow a finer-grained filtering and/or are dedicated to mitigate a particular threat. However, the use of a middle-boxe against volumetric DDoS attacks often shifts the bottleneck from the target to the middle-box. Indeed, the attack traffic converging towards the middle-box is likely to cause saturation on the box's upstream link. Qazi et al. [3] studied the deployment of such middle-boxes to address this particular drawback and to dynamically manage the mitigation resources. However, multiplying middle-boxes within the network turns out to be costly.

The use of existing, already in place network equipments to achieve a distributed first line of defense has also been proposed by the industry. Blackholing, such as described by Cisco [15], provides a simple and resource-efficient method [16] to drop a collection of packets based on their destination or source prefix. A destination-based blackhole would, however, disrupt the service by entirely dropping the traffic routed towards it. ACLs can be used for more precise filters, compared to the coarse granularity of blackholing. On routers, ACLs may match on IP header fields, for example source and destination IPs, or the transport layer protocol. Formerly, the major drawback of ACLs was the performance of large ACLs tables. Vendors have fixed this performance issue by implementing filtering in hardware instead of in the router software [17]. The major drawback of the hardware implementation in filtering lists is the limitation of the size of the ACLs [18].

### B. Blacklists Implementation and Usage

The use of list of filtering rules, i.e., either whitelists, blacklists or a mix of both, within a constrained environment has been widely studied in the literature. In fact, filtering lists have to be optimized to fit equipment constraints. Industry attempted to solve the CPU consumption issue of software-based filters by implementing them in hardware [17]. However, filtering lists are stored in a fast but expensive memory (TCAM) which is size-limited [6]. Maccari et al. [19] propose the use of a memory efficient structure (Bloom Filter [20]) to reduce the size occupied by a whitelist. [6], [8], [21], [22] considered the memory limitation and aimed at reducing the size of lists using source prefix aggregation.

Several aggregation schemes have been proposed in the literature. Network Aware Clusters [23] identify topologically-closed sources using the BGP routing table. The Hierarchical Heavy Hitters [24] algorithm produces aggregates with approximately equal rates (throughput, bandwidth, etc.) of legitimate traffic. Pack et al. [6] study the capability of filtering lists (whitelists, blacklists and a combination of both) to filter malicious traffic while preserving legitimate traffic. Aggregates are computed using a comparison between a baseline period of traffic and the last period of traffic. Goldstein et al. [21] also propose a history-based algorithm to generate filtering rules using Bayesian decision theory. Soldo et al. [8] develop a framework to build optimal ACLs, where the definition of an optimum depends on the filtering goal, e.g., blocking all sources, some sources, preserving bandwidth, etc. The aggregate computation and selection is driven by weights (i.e., scores) assigned to each source. Although, they evoked a different method to assign scores to sources, the importance of these weights has not been assessed. In this paper, we evaluate the impact of scores based on either flow count or volume, depending on the amount of information about traffic we can retrieve.

## IV. Blacklist Generation Problem Characterization

Literature proposes to generate blacklists using the information about the threat and network traffic. However, the attack details depends on the equipment that detect it. Yet, detection mechanisms do not provide equal level of details. Similarly, the visibility that the operator has on his network (e.g., amount of information, level of detail) is highly dependent on the monitoring policy, equipment, etc. We therefore, define three scenarios describing different network visibility levels and illustrate them in Table I.

TABLE I. Information availability-driven scenario

| **Scenario** | **Description** | **MALagg identification** | **MALagg telemetries** | **MONagg telemetries** |
|---|---|---|---|---|
| 1 | Minimum requirement | Yes | No | No |
| 2 | Enhanced detection | Yes | Yes | No |
| 3 | Full network visibility | Yes | Yes | Yes |

The *minimum requirement* scenario describes the minimum information required to generate a rule-based (cf. Section II-A) blacklist, i.e., a list of malicious aggregates (MALaggs), cf. Section II-C.

The *enhanced detection* scenario describes the context where an operator has access to a more detailed information about malicious traffic than solely the identification. He may then be able to retrieve metrics for MALaggs, for example from the detection mechanism. These metrics are collected at the same granularity as the detection, i.e., the tuple $<$ source IP, destination IP $>$.

Our *full network visibility* scenario, evoked in Table I, depicts the use of monitoring information to reduce the amount of collateral damages. Monitoring information is provided for monitored aggregates and, as such it does not differentiate legitimate from malicious aggregates. Off-the-shelf mechanisms, such as NetFlow [25], sFlow [26] or IPFIX [27], are able to provide metrics such as volumetries for traffic aggregates (i.e., MONagg). However, because the granularity of MALaggs is defined by the detection system, and since the MONagg granularity depends on the monitoring system configuration, both aggregate granularities are not always the same. We will study the impact of information availability on the blacklist efficiency in view of these scenarios.

## V. Proposition

We propose a filtering scheme that deals with the problems raised in Section II-B and the context exposed in Section IV. As the number of rules in a blacklist is limited, the capability of the ACL to filter malicious traffic depends on the ability of the generation process to aggregate malicious flows, so that the amount of filtered attack traffic is maximized. A workaround would be to increase the amount of traffic to be filtering among the whole traffic, e.g., by shortening the length of the rules source prefix. However, this also probably induces more collateral damage. Consequently, the ACL generation should also tend to minimize the false positives, i.e., legitimate traffic that is being included by the ACL. For example, a DNS server used by the target may also be abused in an amplification attack.

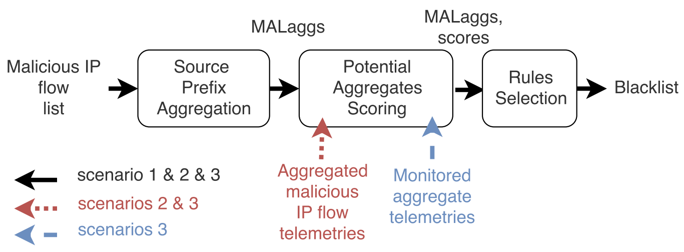

MALaggs
MALaggs, scores
Malicious IP flow list
Source Prefix Aggregation
Potential Aggregates Scoring
Rules Selection
Blacklist
scenario 1 & 2 & 3
scenarios 2 & 3
scenarios 3
Aggregated malicious IP flow telemetries
Monitored aggregate telemetries

Fig. 1. Workflow of ACL generation

The ACL generation process is depicted in Figure 1. First, we compute all possible malicious aggregates (MALaggs) for IP flows IPf included in the alert (cf. Section V-A). Traffic to each destination IP address is treated separately, so that regardless of the filtering granularity, i.e., either based on source IP or both source and destination IPs, only the source IP of malicious flows is aggregated. Second, a score is computed for each of the MALagg (cf. Section V-B) by using aggregated malicious IPf telemetries if they are available (scenario 2 and 3) and monitored aggregates (MONagg) telemetries in scenario 3. Third, the top $N$ MALagg are selected as rules to form the blacklist (cf. Section V-C). Then, the scores are regularly recomputed to maintain up-to-date blacklists, for example every time an alert is received from the detection system.

### A. Source Prefix Aggregation

As widely approved by the literature ( [7], [21], [24], [28], [29]), we reduce the number of ACL rules using aggregation of source IP addresses for a given destination IP. We thus maintain a separate list of sources for each target.

We define the *aggregation limit* ($AL$) as the minimal source prefix length of potential aggregates, so that aggregates have a source prefix length between the $AL$ and 32. Figure 2 shows an example of computed source aggregates for a given destination IP. Considering an $AL$ of 26, an alert that contains the following source IPs [ 1.66.180.12, 1.66.180.13, 1.66.180.50, 1.66.180.60, 1.66.180.201 ] for a single target results in the following list of possible source aggregates [ 1.66.180.12/32, 1.66.180.13/32, 1.66.180.50/32, 1.66.180.60/32, 1.66.180.201/32, 1.66.180.12/31, 1.66.180.48/28, 1.66.180.0/26], shown in green in Figure 2. The aggregate 1.66.180.0/24 is not included in the MALaggs as the netmask length exceeds the $AL$.

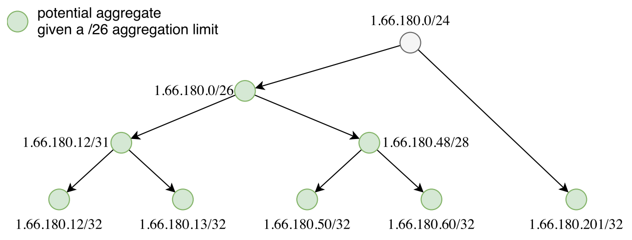


Fig. 2. Example of source aggregation tree for a given destination

### B. Malicious Aggregates Scoring

We define three main strategies, for scoring MALaggs that aim at dealing with scenarios that only include information about malicious traffic (cf. scenarios 1 and 2, Table I). A fourth strategy, concerns the last scenario that includes monitoring telemetries. For all strategies, aggregates with high scores are more likely to be added to the ACL. We do not claim that these simple strategies are better than the state of the art. They aims at reflecting how network information can be used and how level of information impacts the filtering.

*Scenario 1.a* aggregate scoring is relevant to scenario 1 where network operators can only retrieve a list of MALaggs. The generation scheme scores possible aggregates by only taking into account the length of the aggregate's source IP prefix $p$, as shown in (1). Aggregates with a shorter source IP prefix length get a higher score and are more likely to be inserted in the ACL, such that scores are narrowed between 0 and 32.

$$score_{1.a}(p) = 32 - length(p) \tag{1}$$

*Scenario 1.b* aggregate scoring also focuses on scenario 1 where operators only get a list of malicious contributors. The score is equal to the ratio between the number of malicious sources ($\mathcal{MS}$) within an aggregate and the complement to 32 of the aggregate's source netmask length, to which has been added 1 so that /32 prefixes does not result in a division by 0. In that case potential aggregates which include larger number of malicious sources and/or whose source prefix is small get a higher score to be put first in the blacklist, so that we try to minimize collateral damages. As a result, score rated between 0 and $2^{32} - 1$ is expressed in (2). In fact, as scored prefixes always contain malicious traffic null score is never reached.

$$score_{1.b}(p) = \frac{|\mathcal{MS} \cap p|}{(32 - length(p)) + 1} \tag{2}$$

*Scenario 2* aggregate scoring also takes the malicious IPf telemetries as input. Since we aim at mitigating volumetric attacks and their congestion effect on the network, we consider the volumetry as the ground metric to assess the impact of aggregates. The aggregate score - expressed in bytes in (3) - refers to the volume of malicious traffic towards the target, which source IPs ($\mathcal{MS}$) are included in the aggregate source prefix $p$. The $malVol(ip)$ function depicts the byte sum of malicious traffic from $ip$ towards the target reported during the last period. As such, the score ranges from 0 to the total volume of attack traffic.

$$score_2(p) = \sum_{ip \in \mathcal{MS} \cap p} malVol(ip) \tag{3}$$

*Scenario 3* aggregate scoring depicts the use of MONagg telemetries. MALagg's scores - also expressed in bytes and ranged from 0 to sum of volumes of all malicious IPfs - are obtained by multiplying the score obtained in scenario 2 and the ratio between the volume of malicious traffic and an estimation of the overall traffic within the aggregate $p$ ($overallVol(p)$), as expressed in (4).

$$score_3(p) = score_2(p) \times \frac{score_2(p)}{overallVol(p)} \tag{4}$$

This is an estimation because the length of source prefix of potential rules may not always be equal to the prefix length of source prefix of monitored aggregates . In fact, the length of source prefix ($p$) of the potential aggregates to filter varies between the aggregation limit and 32, while the source prefix ($p_m$) length of MONagg is fixed by the monitoring system configuration (cf. Section IV). The estimation depends on the value of the source prefixes' length as can be seen in (5).

$$overallVol = \begin{cases} \sum_{p_m \subset p} monVol(p_m) & \text{for } length(p) < length(p_m) \\ monVol(p) & \text{for } length(p) = length(p_m) \\ (monVol(p_m) - malVol(p)) & \\ \times \frac{nbHosts(p)}{nbHosts(p_m)} + malVol(p) & \text{otherwise} \end{cases} \tag{5}$$

If the prefix of a MONagg is larger than the prefix to filter, the estimation of is equal to the sum of the volume of all monitoring aggregates ($monVol$) included in the source prefix to filter $p$, The estimation is equal to the volume of the monitoring aggregate when the length of the aggregate to filter is equal to configured prefix length of monitoring aggregates. Otherwise, we estimate the volume of legitimate traffic within the source prefix $p$ towards a given destination. We first assume that remaining traffic volume (i.e. $monVol(p_m) - malVol(p)$) is evenly distributed on the highest number of hosts in a subnet of size $length(p_m)$ expressed in (6). Then, we add the volume of these sources included in the prefix $p$ to the volume of malicious traffic for this aggregate.

$$nbHosts(p_m) = 2^{32-length(p_m)} \tag{6}$$

### C. Rules Selection

Finally, we select the top $N$ rules among all scored potential aggregates to form the blacklist. The process is depicted in Figure 3. The potential aggregates are sorted according to their scores in descending order. In the example, scores between parentheses have been set arbitrarily. However, it is possible that the aggregate $1.66.180.12/32$ has a higher score than one of its parent aggregate, e.g., $1.66.180.12/30$. A legitimate client (e.g., $1.66.180.14$) with a large volumetry may reduce the $1.66.180.12/30$ aggregate score. Then, the top $N$ ($N = 3$ in Figure 3) are used to generate the ACL. Considering the aggregation mechanism, an overlap is possible only if an aggregate is included another. Consequently, to avoid wasting rules, an ACL has to be exclusive. Then, when we try to insert in the top $N$ an aggregate that includes or is included in an already inserted aggregate, we keep the aggregate with the smallest prefix length and remove the other. In the example, $1.66.180.50/32$ and $1.66.180.48/28$ are both in the top 3 aggregates. However, as the second aggregate includes the first one, only $1.66.180.48/28$ is kept in the final blacklist.

## VI. Experiments

Blacklists are widely used to mitigate DDoS attacks. However, while literature has proposed algorithms to generate such filters with a realistic number of rules, they do not evaluate the efficiency of their approach in regard of amount of information on the network available. These proposed algorithms, however, are not applicable in all networks due to the requirements in terms of information availability. In this paper, we formulated three scenarios describing different network visibility levels and proposed basic blacklist generation strategies for each scenario. We conduct simulation in which we generate blacklists using scenario related strategies and apply resulting filters on traffic captures. We then compare results for each scenario-driven strategy. It allows us to study the impact of the levels of available information on the filtering efficiency, i.e., the ability to drop malicious traffic while preserving legitimate flows.

### A. Metrics and Variables

We rely on two commonly used metrics when dealing with filtering, the *true positive rate* ($TP_r$, also known as sensitivity or recall) and the *false positive rate* ($FP_r$) in order to assess the scoring strategies. The $TP_r$ evaluates the proportion of malicious traffic that is being filtered by the mechanism, how the filter is able to correctly drop malicious traffic. The generation strategies have been designed to maximize this percentage. Conversely, the $FP_r$ measures the proportion of collateral damages. This allows validating the use of monitoring information to reduce the collateral damages. Both metrics are then well fitted to assess the twofold definition of the efficiency.

The $TP_r$ is obtained as the ratio between the number of filtered malicious IPf and the total amount of malicious flows. Correspondingly, the false positive rate ($FP_r$) is the ratio between the number of filtered legitimate flows and the sum of legitimate flows.

The blacklist construction is tuned using two parameters, the maximal number of rules in a filter ($N$) and the aggregation

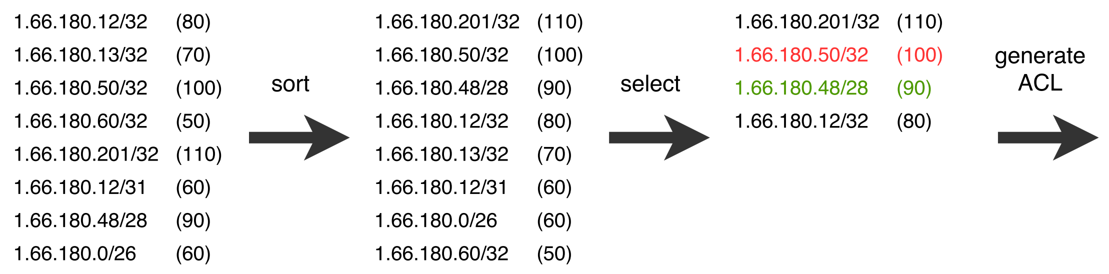


Fig. 3. Selection of top 3 rules among potential source aggregates for a given destination

limit ($AL$). Soldo et al. [8] used from few hundreds to few thousands rules. We then execute experiments with different values of $N$ between 10 and 500 maximum filtered aggregates in the filter. Aggregation limit is fixed to either /24 or /8 to study its impact with a short and a long filtered aggregates prefix length. An $AL$ of /0 and /32 have not been considered here, as a filtered aggregate /0 will result in dropping the whole traffic. Conversely, filtering with the whole IP (/32 prefix) instead of an aggregate, with at most 500 rules cause at most $0.25\%$ of malicious IPf to be filtered which we can consider as pointless. In fact, this depends on how aggressive are these IPfs. However, for reasons of clarity, we decided not to include /32 prefixes.

We consider two configurations for the MONagg's granularity reported by the monitoring system. The first one define records for destination IPs, i.e. traffic metrics are reported for $<$ /0 source prefix, /32 destination prefix $>$ aggregates. This kind of monitoring configuration may be used for networks where each customer is identified by the destination IP such as data centers. The second finer granularity is defined by the tuple $<$ /24 source prefix, /32 destination prefix $>$. That can be used, for example, by ISPs, so that records match the largest common inter-AS BGP prefixes advertisements [30].

## B. Results

The behavior of ACL-based filtering, as described in Section V-B, is studied for each scenario defined in Section IV. We use real legitimate traffic from the MAWI data set [31] as legitimate traffic superimposed with generated attack traffic. Traffic has been captured on February 2017 during 15 minutes on a transit link and has been cleaned from attack traffic [1]. The inbound part of this capture has an average packet rate of 51,000 packet per second (295 Mbps). In parallel, we generate 10 different attack traffics.

In order to consistently run experiments with the MAWI capture and one of the 10 attack traffic, we follow the procedure below. We select 1000 legitimate sources from the MAWI capture that will also send attack traffic. The remaining malicious sources are randomly chosen such that they are not seen in the legitimate capture. In total, 200,000 malicious sources are selected. We generated a constant bit rate attack traffic with a bandwidth of 1.3Gb/s. While the overall number of attack sources is realistic [10], [11], the overall volume fall short of the most massive current attacks due to computational constraints. Each of the MALagg has also a constant throughput throughout the attack, which is randomly chosen between 0.6 and 1.4 the average per flow bit rate. More realistic source dynamics will be considered in future works. Scores are regularly re-computed (e.g. every 60 seconds in Table II) to update the variation of legitimate traffic. Figure 4 shows the average true and false positive rates ($TP_r$ and $FP_r$) for each scoring function with multiple $AL$s (depicted in columns) and varying values of $N$ (x-axis).

[1]MAWI capture is available at http://www.fukuda-lab.org/mawilab/v1.1/2017/02/03/20170203.html

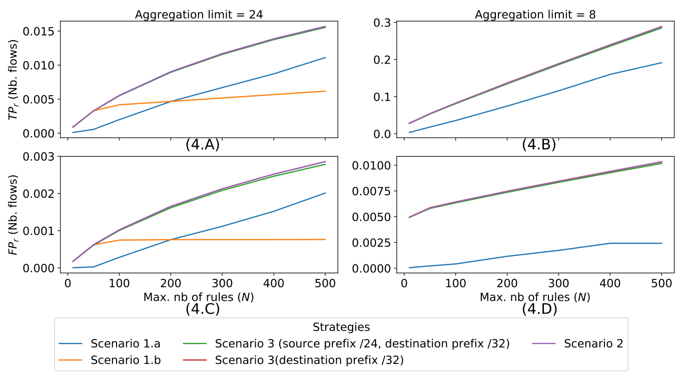


Fig. 4. Comparison of scoring functions

1) *Scenario 1 - Minimum Requirements:* For the *Scenario 1.a* score function, the $TP_r$ shows odd trends. For example, in Figure C, the $TP_r$ has a small increase for $N$ varying from 10 to 50. This growth increases for a number of rules greater than 50. In fact the strategy does not succeed in selecting the top $N$ rules. This is due to the fact that a lot of filtered aggregates (more than 190,000) have the same score. However, the score function has to select the top $N$, where $N$ is less than 500. There is therefore no rational method to select the top $N$ rules. This results in a pseudo random selection of filtered aggregates .

The *Scenario 1.b* scoring function (in orange) grows linearly from 10 to 50 rules in the filter for each sub-figure. In fact, the number of malicious IPf added per rule linearly decreases considering a maximum rule count greater than 50. However, the $TP_r$ of *Scenario 1.b* score function shows a much larger increase for $N = 10$ to 50. This is also due to the distribution of traffic, where the top $N$ rules contain very dense malicious aggregates with significantly high volumetry. The *Scenario 1.b* scoring function is then correctly choosing the top 10 to 50 rules. This large increase for small $N$ values does not appear in sub-figure 4.B, as malicious sources are more evenly distributed among the source aggregates for small prefixes. In other words, in Figure B, traffic aggregated in /24 source prefixes emphasizes some aggregates with high impact. However, these /24 aggregates with high score are diluted in /8 aggregates.

## C. Scenario 2 - Enhanced Detection

The $TP_r$ and $FP_r$ of the *Scenario 2* scoring function (in violet) coincides with the *Scenario 1.b* from $N = 10$ to 50 in Figures A and C. Using malicious IPf telemetries provide no added value given a small number of rules in a filter. Conversely, from $N = 50$ to 500 in Figure A, the $TP_r$ of the *Scenario 2* grows faster than for the baseline scenario. For example, given $N = 500$ (cf. Table II), the number of dropped malicious flows in *Scenario 2* is just over twice for the *Scenario 1.b*, and the same proportion applies considering the filtered volumetry. The drawback is that the $FP_r$ also grows faster (cf. Figure C), resulting in an increase of the collateral damages. Table II shows that, for $N = 500$, the legitimate dropped traffic in *Scenario 2* is around 4 and 5 times the *Scenario 1.b* statistics expressed respectively in terms of number of flows and volumetry. Both behavior are mostly due to the fact that the scores of *Scenario 1.b* are devalued when the source prefix of the MALagg grows, that also induces an increase of the probability to include legitimate traffic. The *Scenario 2* does not try to reduce the collateral damages. However, the chosen scenario scoring functions is not so efficient with an $AL$ of /24, as only 1.50% of malicious traffic is filtered.

When we shorten the $AL$, e.g., from Fig A to B, the *Scenario 2* and *1.b* display similar trends. In fact, a large part of malicious IPf is quite dispersed, so that long source prefixes only aggregate a small part of malicious traffic. As a consequence, in Figure A, the *Scenario 1.b* scoring function favor malicious flows over malicious aggregates. In contrast, shorter source prefix (i.e. up to /8) aggregates more malicious traffic, so that they get by the *Scenario 1.b* a higher score than longer prefix aggregates. As a consequence, filtered aggregates selected by the *Scenario 1.b* scoring function matches the ones selected by the *Scenario 2* and the $FP_r$ curves coincide, cf. Figure B. Both *Scenario 1.b* and *2* scoring functions allow filtering around 30% of malicious traffic (in terms of number of flows and volumetry), using solely 500 rules with an average of 1% of filtered legitimate traffic (Table II).

*1) Scenario 3:* The curves of *Scenario 3* scoring function configured with a /32 destination prefix granularity (depicted in red) coincide for all sub-figures with the results of the *Scenario 2*. Considering an $AL$ of /24 (Figure A), this is due to the fact that very few filtered aggregates contain both legitimate and malicious traffic, so that introducing monitoring information is not been able to provide much value-addition. For an $AL$ of /8, cf. Figure B, the reason is that legitimate and malicious traffics are highly distributed among filtered aggregates , so that scores get similar results for the rules. This also explains the fact that the $FP_r$ trends of the *Scenario 3* configured with $<$ /24 source prefix, /32 destination prefix $>$ granularity (shown in green) results almost similar to the *Scenario 2* false positive rate. While this means that it does not reduce the efficiency of the blacklist in terms of malicious traffic filtering, this does not help in preserving legitimate traffic.

The *Scenario 3* that uses$<$ /24 source prefix, /32 destination prefix $>$ MONagg granularity, depicts an improvement of the $FP_r$ compared to the *Scenario 2* in Figure C. Although this seems small when expressed in terms of number of flows (i.e., 0.01% of legitimate flows preserved, cf. Table II), results are a little more significant when the $FP_r$ is expressed in terms of volume (0.2%).

## D. Discussion

Experiments with attack traffic without variations show the basic efficiency and behavior of scoring functions for the considered scenarios. First, considering the hardware limitations of the number of rules in a router, the *Scenario 1.a* scoring function is irrelevant. In fact, the strategy does not allow choosing correctly the top $N$ rules, as more than $N$ rules obtain the best score. In order to use it effectively, routers would require at least around 190k rules for an $AL$ of /24. This minimum number of rules is dependent on the malicious traffic distribution, i.e., poorly distributed malicious traffic would require fewer rules to be aggregated. The *Scenario 2* reaches the highest efficiency when it comes to only filtering malicious traffic. However, as we increase the $AL$, the *Scenario 1.b* scores results similar to the *Scenario 2*. In other words, the optimal efficiency in the scenario 1 is at the same level as the efficiency of a scenario with a higher level of detail (scenario 2), assessed in terms of the number of dropped flows.

Our evaluation is based on the assumption that the malicious traffic is not spoofed, as explained in Section II-B. If it were not the case, DDoS mitigation with blacklists could cause more collateral damage, as spoofed attack traffic could overlap more easily with legitimate traffic and increase the number of MALagg that also contain legitimate traffic. We configured an IPf-level overlap of 1,000 over 200,000, which seems in most cases far above reality. For example, considering amplification attacks, this means that the target legitimately connect with 1,000 amplifiers (DNS servers, ...). This may be the case when the target is a proxy or a NAT gateway. However the overlap is exacerbated using source aggregation.

In our approach, we also supposed that the list of malicious aggregates, e.g., contained in the alert, is exhaustive. This is not true in all cases, as the detection mechanisms are not perfect and/or do not report attack sources exhaustively due to the potentially large number of sources in DDoS attacks.

TABLE II. AVERAGE STATISTICS OF DROPPED TRAFFIC FOR EACH SCENARIO CONSIDERING A BLACKLIST GENERATION EACH 60S ($N = 500$)

| | | | Scenario 1.a | Scenario 1.b | Scenario 2 | Scenario 3 (source prefix /24, destination prefix /32) | Scenario 3 (destination prefix /32) |
|---|---|---|---|---|---|---|---|
| *AL* = /24 | malicious traffic | #IPfs | 2222 (1.11%) | 1234 (0.62%) | 3136 (1.57%) | 3107 (1.55%) | 3136 (1.57%) |
| | | std | 50 | 55 | 25 | 28 | 25 |
| | | MBytes | 108.5 (1.11%) | 60.28 (0.62%) | 154.89 (1.59%) | 153.68 (1.58%) | 154.89 (1.59%) |
| | | std | 2.63 | 2.81 | 1.38 | 1.47 | 1.38 |
| | legitimate traffic | #IPfs | 1222 (0.2%) | 462 (0.08%) | 1730 (0.29%) | 1687 (0.28%) | 1729 (0.29%) |
| | | std | 814 | 266 | 1010 | 2012 | 1010 |
| | | MBytes | 8.06 (0.37%) | 2.89 (0.13%) | 14.68 (0.67%) | 10.45 (0.48%) | 14.67 (0.67%) |
| | | std | 1.73 | 0.91 | 3.42 | 2.69 | 3.42 |
| *AL* = /8 | malicious traffic | #IPfs | 38291 (19.15%) | 57822 (28.91%) | 57722 (28.86%) | 57020 (28.51%) | 57713 (28.86%) |
| | | std | 164 | 168 | 172 | 226 | 171 |
| | | MBytes | 1866.1 (19.14%) | 2819.38 (28.92%) | 2824.54 (28.97%) | 2790.21 (28.62%) | 2824.02 (28.96%) |
| | | std | 8.34 | 8.67 | 8.48 | 11.18 | 8.46 |
| | legitimate traffic | #IPfs | 1437 (0.24%) | 6195 (1.03%) | 6191 (1.03%) | 6096 (1.02%) | 6182 (1.03%) |
| | | std | 29 | 1657 | 1656 | 1665 | 1657 |
| | | MBytes | 0.22 (0.01%) | 32.16 (1.47%) | 32.16 (1.47%) | 32.02 (1.46%) | 31.62 (1.45%) |
| | | std | 0.01 | 5.25 | 5.23 | 5.03 | 5.27 |

However, to be efficient, a detection mechanism is likely to provide top malicious aggregates, i.e., the aggregates with most impact, e.g., the aggregates with the highest data rate as we deal with volumetric DDoS. The malicious aggregates not included in the alert are thus likely have only a small effect in the network congestion.

We focused in this paper continuous per IPf throughput. The impact of more dynamic attack traffic will be studied later. We expect that the efficiency would be affected by new parameters such as the monitoring records collection period, and the blacklist refresh period. Moreover, history-based algorithm should be studied, such as Exponentially Weighted Moving Average (EWMA) used in [7], [32], to generate blacklists which handle temporal trends of malicious IPf contributions.

## VII. CONCLUSION

We presented an evaluation of simple blacklisting algorithms, from the perspective of an operational constraint, i.e., level of information an operator can retrieve from the network. The assessment scenarios considered the fact that operators do not have equal level visibility on their networks, depending on the functionality and configuration of the monitoring system. The minimal requirement for blacklist generation is the identification of source IPs of attack traffic. From there on, additional traffic information, such as volumetries of attack and legitimate traffic, can be used to improve the efficiency of the blacklists. Experiments highlighted that a generation algorithm does not fit well in all scenarios and it should be carefully chosen in regard of the available network information. Furthermore, in some situations providing more detailed information improved the filtering results only up to a given point, suggesting that the algorithms' behaviors should be evaluated in the context in which they are to be used.

We also considered in this paper the aggregation of monitored traffic (i.e., generation of monitored aggregates) as a possible optimization of the monitoring system, although it will degrade the quality of flow reporting. We acknowledge that this is not the only possible configuration parameter a network operator is able to leverage to optimize network monitoring system, for example flow sampling is another widely used optimization. While the impact of flow sampling on the attack detection [33], [34] has been studied, its effect could also be studied in regard of the scenario 3. As a consequence, efficiency of filtering can be assessed in the light of the cost of monitoring collection, in term of storage, flow records bandwidth consumption.

## REFERENCES

[1] M. Casado, P. Cao, A. Akella, and N. Provos, "Flow-Cookies: Using Bandwidth Amplification to Defend Against DDoS Flooding Attacks," in *Proc. IWQoS*, jun 2006, pp. 286–287.
[2] A. Greenhalgh, M. Handley, and F. Huici, "Using Routing and Tunneling to Combat DoS Attacks." in *SRUTI*, 2005, pp. 1–7.
[3] Z. A. e. a. Qazi, "SIMPLE-fying Middlebox Policy Enforcement Using SDN," in *ACM SIGCOMM*, 2013, pp. 27–38.
[4] A. Mahimkar, J. Dange, V. Shmatikov, H. M. Vin, and Y. Zhang, "dFence: Transparent Network-based Denial of Service Mitigation." in *4th USENIX Symposium on Networked Systems Design & Implementation (NSDI 07)*, 2007, pp. 327–340.
[5] A. Abujoda and P. Papadimitriou, "MIDAS: Middlebox discovery and selection for on-path flow processing." in *COMSNETS*, 2015, pp. 1–8.
[6] G. Pack, J. Yoon, E. Collins, and C. Estan, "On Filtering of DDoS Attacks Based on Source Address Prefixes," in *Securecomm*, 2006, pp. 1–12.
[7] F. Soldo, A. Le, and A. Markopoulou, "Predictive blacklisting as an implicit recommendation system," in *2010 Proceedings IEEE INFOCOM*, March 2010, pp. 1–9.
[8] F. Soldo, K. Argyraki, and A. Markopoulou, "Optimal Source-Based Filtering of Malicious Traffic," *IEEE/ACM Transactions on Networking*, vol. 20, no. 2, pp. 381–395, apr 2012.
[9] "Verisign Distributed Denial of Service Trends Report - 3rd Quarter 2017," Tech. Rep. 3, 2017.
[10] B. Krebs, "Krebsonsecurity hit with record ddos," 2016.
[11] OVH, "The DDoS that didn't break the camel's VAC," 2016.
[12] N. Teague, "Open threat signaling using rpc api over https and ipfix," Internet-Draft, July 2015.
[13] H. Debar, D. Curry, and B. Feinstein, "The Intrusion Detection Message Exchange Format (IDMEF)," RFC 4765 (Experimental), IETF, Mar. 2007.
[14] T. H. Tan, C. Y. Ooi, and M. N. Marsono, "rrBox: A Remote Dynamically Reconfigurable Middlebox for Network Protection." in *CANDAR*, 2014, pp. 219–225.
[15] "Remotely Triggered Black Hole Filtering - Destination Based and Source Based," Cisco Systems, Tech. Rep., 2005.

[16] I. Vordos, “Mitigating Distributed Denial of Service Attacks with Multi-Protocol Label Switching-Traffic Engineering (MPLS-TE),” Ph.D. dissertation, Naval Postgraduate School, 2009.

[17] “Understanding ACL on catalyst 6500 series switches,” Cisco, Tech. Rep.

[18] M. Xu, S. Yang, D. Wang, F. Li, and J. Wu, “Source address filtering for large scale networks,” *Computer Communications*, pp. 64–76, 2014.

[19] L. Maccari, R. Fantacci, P. Neira, and R. M. Gasca, “Mesh Network Firewalling with Bloom Filters,” in *ICC*, 2007, pp. 1546–1551.

[20] B. H. Bloom, “Space/Time Trade-offs in Hash Coding with Allowable Errors.” *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[21] M. Goldstein, C. Lampert, M. Reif, A. Stahl, and T. Breuel, “Bayes Optimal DDoS Mitigation by Adaptive History-Based IP Filtering,” in *ICN*, 2008, pp. 174–179.

[22] A. Kalliola, T. Aura, and S. Šćepanović, “Denial-of-Service Mitigation for Internet Services,” in *Proc. NordSec*, 2014, pp. 213–228.

[23] B. Krishnamurthy, J. Wang, B. Krishnamurthy, and J. Wang, “On network-aware clustering of Web clients,” *ACM SIGCOMM Computer Communication Review*, vol. 30, no. 4, pp. 97–110, oct 2000.

[24] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava, “Finding Hierarchical Heavy Hitters in Data Streams,” in *VLDB*, vol. 29, 2003, pp. 464–475.

[25] “NetFlow,” Cisco, Tech. Rep., 2008.

[26] “Traffic Monitoring using sFlow,” sFlow.org, Tech. Rep., 2003.

[27] G. Sadasivan, N. Brownlee, B. Claise, and J. Quittek, “Architecture for IP Flow Information Export,” RFC 5470, IETF, Mar. 2009.

[28] A. Kalliola, K. Lee, H. Lee, and T. Aura, “Flooding DDoS mitigation and traffic management with software defined networking,” in *Proc. CloudNet*, 2015, pp. 248–254.

[29] M. Collins and M. Reiter, “An Empirical Analysis of Target-resident DoS Filters,” in *Proc. SECPRI*, 2004, pp. 103–114.

[30] D. Bayer, “Visibility of Prefix Lengths in IPv4 and IPv6,” 2010. [Online]. Available: https://labs.ripe.net/Members/dbayer/visibility-of-prefix-lengths

[31] R. Fontugne, P. Borgnat, P. Abry, and K. Fukuda, “MAWILab: combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking.” in *CoNEXT*, 2010, p. 8.

[32] J. Freudiger, E. De Cristofaro, and A. E. Brito, “Controlled data sharing for collaborative predictive blacklisting,” in *Detection of Intrusions and Malware, and Vulnerability Assessment*, M. Almgren, V. Gulisano, and F. Maggi, Eds. Cham: Springer International Publishing, 2015, pp. 327–349.

[33] D. Brauckhoff, B. Tellenbach, A. Wagner, M. May, and A. Lakhina, “Impact of packet sampling on anomaly detection metrics.” in *Internet Measurement Conference*. ACM, 2006, pp. 159–164.

[34] A. Pescape, D. Rossi, D. Tammaro, and S. Valenti, “On the impact of sampling on traffic monitoring and analysis,” in *ITC*, 2010, pp. 1–8.

# Cascading Three Step Approach for Anomaly Detection in Unsupervised Communication Meta Data of IP-based Internet of Things Devices

Andreas Schäfer and Michael Massoth
Department of Computer Science
Hochschule Darmstadt (h_da) – University of Applied Sciences Darmstadt,
and Center for Research in Security and Privacy (CRISP), Darmstadt, Germany
E-mail: andreas.schaefer@stud.h-da.de; michael.massoth@h-da.de

*Abstract*— **Internet of Things (IoT) are globally connected devices which are able to collect and exchange information. The increasing usage of IoT-devices in industrial and private environments result in the need for higher security and constant surveillance of such devices. Since 2016 novel botnets, consisting only of IoT-devices, where observed to execute major Distributed Denial of Service (DDoS) attacks. Due to the autonomous nature of these IoT devices, a compromised device might never be detected by system administrators. This creates the need for continuous monitoring of IoT network traffic. A possible solution for this problem is the permanent monitoring of anomalies within the network traffic of the IoT devices. Anomaly Detection Systems (ADS) monitor the behavior of a system and flag significant deviations from the normal activity as anomalies. This paper presents a new three step approach for anomaly detection in unsupervised communication meta data by cascading X-means clustering, decision tree, and statistical analysis, in order to monitor and protect IoT networks.**

***Keywords-anomaly detection; internet of things; unsupervised machine learning; intrusion detection and prevention***

## I. INTRODUCTION

The Internet of Things (IoT) could be defined as a huge set of sensors and actuators, embedded in physical objects, which are linked through wired and/or wireless networks, often using the same Internet Protocol (IP), that connects the Internet [6]. The basic idea is that devices (things = sensors and actuators) perform tasks independently from human interaction and are connected to the Internet [7]. IoT devices work to a large extent with information from their immediate surroundings. This is to support people in their everyday life or in their work. As a rule, IoT devices behave as predictable as possible, since they are based on fixed implemented algorithms, and human influence on the devices is therefore minimal to non-existent.

The use of IoT is becoming increasingly widespread worldwide. It is estimated that by 2020 more than 30 billion devices of this kind will be in use around the world. The rapidly developing market and the high price pressure on manufacturers often result in insufficient investment in the security features of the IoT devices. Software is usually updated only rarely or not at all. This and the easy accessibility of the IoT devices via the Internet increases the risk of possible compromise by attackers. At the end of 2016, IoT botnets were first widely used in Distributed Denial of Service (DDos) attacks around the world. One of these attacks on the provider Dyn reached a bandwidth of 1.2 Terabits per second (TBps) and temporarily crippled platforms like Twitter, Amazon, CNN, PayPal and many other sites. Therefore, the vulnerability of these IoT devices has therefore often been criticized by IT security experts and researchers in the past. However, the lack of security of IoT devices is not only a danger for potential targets of botnets, the owners of the devices also have a great interest in anticipating a compromise. Especially in industrial environments, IoT devices are playing an increasingly important role in business processes, for example as production machines, part of the infrastructure or as sensors. These devices could also be sabotaged or misused by attackers for espionage purposes. As the number of IoT devices in use increases, so does the threat to businesses and the global threat posed by botnets. It is therefore essential for companies to monitor their IoT devices to detect possible compromises. Anomaly Detection Systems (ADS) [8] monitor the behavior of the IoT system and flag significant deviations from the normal activity as anomalies [11]. This paper presents a new approach for anomaly detection in unsupervised communication meta data of IP-based IoT devices by cascading X-means clustering, decision tree, statistical analysis, and the computation and monitoring of trust values for the monitored individual IoT devices.

The paper is structured as follows. In Section II, the focus of work, field of application and validity, as well as a short overview of the requirement specifications and the restrictions of the new anomaly detection approach is given. Section III presents some related work. In Section IV, the concept and used methods are discussed in more detail, including the following topics: Anomaly-based intrusion detection and prevention, unsupervised machine learning, the collection of meta data, the communication model and cascading three step approach, as well as computational trust. The overall conclusion of the new approach for anomaly detection in unsupervised communication meta data by cascading X-means clustering, decision tree, statistical analysis, is given in Section V. The paper ends with an outlook on future work, and points to an additional planned paper.

## II. FOCUS OF WORK AND PREREQUISITES

Due to the wide variety of IoT devices and corresponding communications protocols, the focus on application and range of use of the new anomaly detection monitor for IoT devices will be specified in detail. In addition, all prerequisites and restrictions of the concept will be discussed. First of all it should be mentioned, that the new approach focuses on the monitoring of internet protocol-based (IP-based) IoT devices only. Other network communication protocols will be neglected and are not taken into account.

A second very important prerequisite and restriction of the new approach is the focus on request/response application protocols only. The two most important request/response application protocols are the Hypertext Transfer Protocol (HTTP) and the Constrained Application Protocol (CoAP) [12]. Other application layer protocols, which use, e.g., the Publish/Subscribe principle, will be neglected and are not taken into account for the new approach.

The Constrained Application Protocol (CoAP) [12] is a specialized web transfer protocol, made for communication with constrained nodes and constrained networks in the Internet of Things. The CoAP is mainly designed for machine-to-machine (M2M) [5] applications, such as smart energy, intelligent buildings, home automation, smart grid, and smart factory applications. CoAP was developed as an Internet Standards Document, RFC 7252. CoAP is designed to use minimal resources, both on the device and on the network. Instead of a complex transport stack, CoAP use the User Datagram Protocol (UDP) over IP. Like HTTP, CoAP is based on the wildly successful Representational State Transfer (REST) model: Which means, servers make resources available under a Uniform Resource Locator (URL), and clients access these IoT resources using methods such as GET, PUT, POST, and DELETE. Since HTTP and CoAP share the REST model, both application protocols can easily be connected by using application-agnostic cross-protocol proxies. A Web client may not even notice that it just accessed a IP-based sensor resource. CoAP and HTTP can carry different types of payloads, and can identify which payload type is being used. CoAP and HTTP integrates with Extensible Markup Language (XML), JavaScript Object Notation (JSON), and many other data formats.

Additional assumptions: It will be assumed that the IP-based IoT devices show a normal communication behavior and behave predictably. Furthermore, it will be assumed that no tagged training meta data of the normal behavior of the individual IoT devices are available. In practice, it is very often the case, that no tagged trainings data are available for the particular IoT devices. The most important assumption is the following one: It will be assumed that the capable normal communication behavior of an individual IoT device can be observed and monitored, without compromising by malware, during a certain training time period.

As short overview, the new anomaly detection approach and concept has the following prerequisites and requirements:

- The IoT devices, to be monitored, communicate via the internet protocol (IP) and an IP network.
- The anomaly detection approach shall have a good performance and may require little storage space.
- The anomaly detection system should synchronize recognized connections with signatures.
- A compromise of an IoT device should be detected by anomaly detection.
- Payload of the data packets is ignored.
- The training data are not labeled.
- It is assumed that the monitored IoT devices work without human interaction and their behavior is largely predictable.
- For the monitoring and detection of anomalies, the metadata of the individual connections between end devices is considered only.
- Network traffic should be monitored both online and offline.
- A separate communication model is trained for each new IP-based IoT device type.
- Any anomalies that occur are saved and can be viewed by the system administrator.
- A computational trust value shall be displayed to the managing system administrator for each device, which is to serve as an indicator of the trustworthiness of this particular IP-based IoT device.

## III. RELATED WORK

Gaddam et al. [1] developed a two-stage algorithm that uses K-means and Iterative Dichotomiser 3 (ID3) decision tree algorithm. For this, they used labeled training data where each data set is marked as attack or normal traffic. In the first phase, the test data is divided into k clusters. Based on the data vectors assigned to a cluster, a decision tree is then trained via ID3. This avoids two main disadvantages of K-means:

(1) forced assignment: If the value for k is smaller than there are real groups, dissimilar data vectors are assigned to the same cluster.
(2) class dominance occurs when a cluster contains a large part of the data vectors.

In theory, this two-step process should result in the trained tree recognizing more subclasses in the clusters. In order to decide whether a new connection represents an anomaly when applying the model, two rules are applied to the new data vector: First the nearest neighbor (cluster) is found and then the most similar rule of the corresponding ID3 tree is searched. Thereby, especially the decision tree algorithm ID3 needs labeled training data. The authors of this method state that an accuracy of 96.24 percent in detection of attacks was achieved with a network anomaly data test (Network Anomaly Data NAD-1998 dataset) with a false positive rate of 0.03 percent.

Meidan, Y. et. al. [14] presented how supervised machine learning can be applied to analyze network traffic data in order to detect unauthorized IoT devices by manual labeling.

The new cascading three step approach discussed in the following sections would like to overcome the restriction of supervised learning and labeled training data.

## IV. CONCEPT AND USED METHODS

Intrusion detection is the monitoring of networks with the aim of detecting security-relevant events. These can be breaches of security rules or malware transmitted over the network. Such rules, also called security policy, are rules laid down by system administrators to which users of a network must adhere. They are designed to prevent users from unknowingly making their devices vulnerable to malware or trying to obtain rights that they are not entitled to.

An Intrusion Detection System (IDS) [3] automates the process of this intrusion detection by permanently monitoring network traffic for communication typical of such actions. An Intrusion Prevention System (IPS) has the same capabilities as an IDS and can still prevent detected events [4].

The main tasks of an IDS are:

- Recording information about discovered events.
- Inform system administrators about events.
- Creating reports.

While IDS focuses on detecting suspicious events, one of the tasks of an IPS is to additionally prevent certain events.

This can be done in various ways:

- Stopping the attack.
- Customize the security configuration.
- Manipulating the attack.

During anomaly-based intrusion detection, all network traffic is synchronized with a communication model that represents normal network traffic. If a significant deviation from the model is found, an intrusion report is triggered. This model can be configured for a network, user or computer. Statistical methods are often used for the comparison with the model. The advantage of Anomaly-Based Intrusion Detection is the ability to detect previously unknown attacks that could not be detected using a signature.

A model representing the normal traffic of a network, user or computer is configured either manually or automatically during a training period. Training times can be static or dynamic. A static model remains unchanged during its useful life and can only be replaced with a new model.

Dynamic models adjust their configuration during runtime. Both methods can cause problems over time, because networks change over time, and so the communication behavior of the network, computer or user can change. Static models therefore generate more false positive over time. Dynamic models do not have this problem, but are more susceptible to slow takeover by an attacker. This could only start with a small number of compromising network requests and increase them over time. A dynamic model would adapt to the behavior without triggering alarms.

## A. *Unsupervised Machine Learning*

Unlike supervised machine learning, unsupervised machine learning does not require labeled training data. In the context of anomaly detection, these algorithms are based on two assumptions: First, that the entire network traffic is normal, and only a small proportion of traffic are attacks. Second, abnormal traffic differs from normal traffic based on statistical data. Widespread Unsupervised Machine Learning algorithms are, e.g., the K-Means, k-Nearest Neighbor (k-NN) algorithm.

Many unsupervised machine learning algorithms are based on clustering. During clustering, data vectors are grouped together based on their similarities. It is used in exploratory data mining, so it is an exploratory measure that examines and clustered the similarity of the data vectors. Depending on the algorithm used, outliers can also be detected. Outliers are individual data vectors that cannot be assigned to an existing cluster. Outliers could be anomalies in our context here.

K-means clustering is one of the most commonly used clustering algorithms. K-means divides all data vectors into k clusters and guarantees that data within a cluster are similar. Here, the center of a cluster is determined, a "centroid", to which the distances of the individual data vectors are calculated. The distance function is the Euclidean distance. This method determines related data vectors. K-Means groups N data vectors into k clusters, where $k < N$ must apply.

The standard k-Means algorithm can only be used with numerical data, but has better performance than other comparable clustering algorithms. However, the parameter k must be set manually. Pelleg and Morre [9] therefore developed the "X-Means" method, in which the k-means algorithm is tested with different k's and the resulting clusters are tested using a density function. The parameter k, which generates the clusters with the highest density, is then recommended for cluster training. With given training data, the optimal value for k is therefore found with this algorithm without having to be configured manually in advance. The X-Means algorithm will be used for the unsupervised machine learning and clustering of related data vectors into centroids.

## B. *Collection of Meta Data*

Network metadata will be collected in the area of network administration for analysis purposes. For this purpose, the entire traffic of a network is observed and metadata about individual connections is recorded. In most cases, therefore, this is done at a central communication point, for example at a gateway.

For the collection of meta data the Bro Network Security Monitor (IDS) [13] will be used. Bro IDS [15] is an open source network monitoring framework based on Unix, which has been extended by some IDS-typical functions. Bro IDS processes observed network traffic in two steps:

(1) Event Engine: The Event Engine analyzes the observed network traffic live and detects events. The trigger for these events is defined in an associated enhancement. It is possible to collect further information about this event using these enhancements. Bro comes preinstalled with some extensions, such as signature and connection logs, but also application protocol events, like , e.g., Domain Name Service (DNS) and Dynamic Host Configuration Protocol (DHCP).

(2) Policy Script Engine: Uses events triggered by the Event Engine to perform custom actions. Here further analyses of the generated data can be carried out and dependent behavior can be implemented.

The Event Engine will be used to create connection logs. This information contains metadata for each detected connection. They consist of a mixture of numerical and categorical data, such as the number of transmitted packets, the contacted port, the IP addresses involved and much more.

In Bro IDS it is also possible to define signatures. If a recognized connection corresponds to the behavior of a defined signature, a corresponding event is triggered and a signature log is created.

Bro IDS is able to monitor a network live at a central communication point, for example as a gateway. However, there is also the option to analyze recorded network traffic. This would fulfill the requirement to be able to operate both online and offline.

### C. *Communication Model and Cascading 3 Step Approach*

A communication model is the basis on which an ADS or IDS analyzes observed network traffic and detects anomalies. Based on the above requirements, a categorizing communication model is designed for the new anomaly detection approach. This means that all recognized connection metadata will be divided into categories. During the training phase, these categories are learned automatically using the procedure described below. If the assignment of new connection metadata to a category is not possible after the end of the training, during the application phase, the communication behavior is not known and therefore represents an anomaly. The new anomaly detection approach in unsupervised communication meta data of IP-based IoT devices is characterized by cascading X-means clustering, decision tree, and statistical analysis. The cascading three step approach works like this:

Step (1): X-means clustering (see Figure 1): In the first step, unsupervised training meta data will be collected and categorized based on the numerical data it contains about clustering. Therefore selected numerical data $t_n$ from the connection metadata are used to categorize the training data into k clusters using k-means clustering. For the k-means algorithm, the parameter k, which determines the number of clusters to be formed, must be set. However, since the number of clusters is not known in advance, k must first be found. X-means clustering is used for this. Additionally, the system administrators should be given the option to set the k parameter manually.

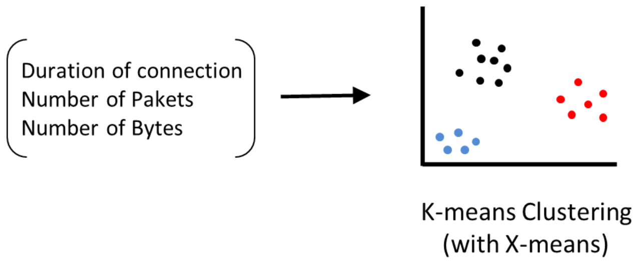


Figure 1. X-means Clustering

After the clustering of the training data has been completed, only the centroids of the clusters are stored persistently. For the later classification of metadata on during the application phase, only the k centroids are required to determine the affiliation of a data vector to a cluster. Since a trained model should not change, moving the centroids as in the k-means algorithm is no longer necessary. The $t_n$ data vectors are therefore not needed for future use and can be deleted.

Step (2): Generation of decision trees (see Figure 2): For all data within a cluster, categorical values of the connection metadata will be stored in a decision tree structure to divide the observed connections into further subcategories. As a result, the recognized categories are further subdivided by generating trees from the categorical data. Proven in practice and used in the approach are the following decision tree structure (from root to leaf): Transmission Protocol => Service => Port Number => IP Address.

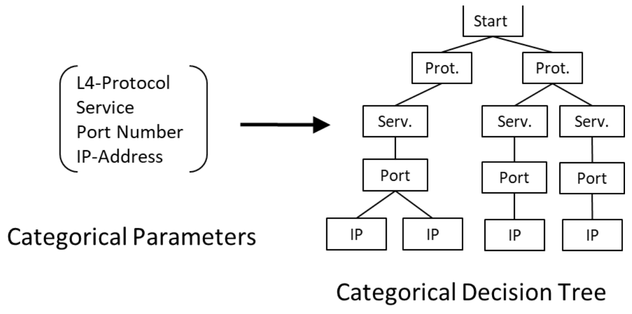


Figure 2. Generation of Decision Trees by Categorical Parameters

Step (3): Statistical evaluation (see Figure 3): The allocation of the connection metadata, collected during the training time to the individual categories, will be calculated proportionately in order to enable a statistical analysis later. As result of the statistical analysis, the distribution of the individual categories of the clusters and tree paths are calculated proportionately.

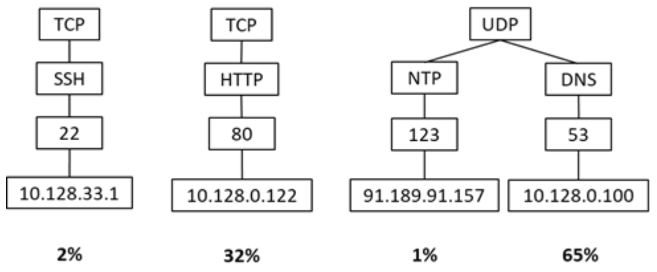


Figure 3. Statistical Distribution of the individual subcategories of the generated clusters and tree paths (an arbitrary example)

### D. *Training data, parameters, and analysis*

The whole of all training data for a particular machine is referred to in this section as $T_i$, where i represents a particular machine. $T_i$ consists of a series of data tuples t, each of which represents a connect log and contains all metadata of a connection. The objects in t consist of different data types: numeric (numbers) and categorical (characters and strings).

Not all metadata provided by logging frameworks are relevant for training communication models. Therefore, a series of data are selected from t, which are used for model creation. The numerical data is referred to as $t_n$ and the categorical data as $t_k$. When the course is finished, the system switches to the application phase. New metadata is recognized online or offline and applied to the communication model. Such a Conn log to be checked has the same design as $t \in T_i$ and is called a $\in A_i$. $A_i$ is the set of recognized connection metadata during the application phase of a device identified by i. The communication model trained with the training data $T_i$ is used to check the tuples $a \in A_i$. An subset of a contains the same selected numerical data as $t_n$, the same applies to $a_k$ subset of a and $t_k$.

In order to guarantee the accuracy of the model, different methods suitable for the properties of the data types are used for learning the subcategories.

Numerical parameter: Numeric values are used to rank the scope of a connection. For this purpose, five elements from t are used, which are summarized in a new tuple $t_n$:
1. Duration: duration of the connection. 2. OrigPkts: Number of sent packets of the connection initiator. 3. RespPkts: Number of packets received by the connection initiator. 4. OrigBytes: number of bytes sent. 5. RespBytes: number of received bytes.

Categorical parameter: In contrast to numerical parameters, categorical parameters can only take a defined number of possible values. In this approach, the following four objects are transferred from t to $t_k$:
1. Transmission protocol: The used Layer 4 protocol (UDP or TCP). 2. Used Service: If detected, the application protocol used (like , e.g., DNS or DHCP) 3. Target port: The contacted target port. 4. Internet protocol destination address: The contacted IP address.

Since it can be assumed that IoT devices behave predictably, the observed categorical values can be regarded as complete in all $t_k$ with sufficiently long training time. This means that all possible combinations of categorical values occur at least once during the training phase. Using the exploratory measure in the previous step, k categories for link metadata were found. Each cluster contains at least one $t \in T_{ix}$, where $T_{ix}$ represents a certain subcategory. Whose categorical values can be learned further in order to find further subcategories within a cluster. These categorical values must now be learned in such a way that it can be efficiently checked whether there is a matching $t_k$ for a given $a_k$ where $a_k = t_k$. Tree structures are suitable for this. All $t_k$ are grouped in a tree $b_j \in B_i$. $B_i$ contains all the trees of a communication model, where i represents a specific device. Where j stands for the cluster for which this tree is trained. Each element of the tuple $t_k$ corresponds to a node in tree $b_j$. The depth of the tree corresponds to $| t_k |$. Please note that the tree levels are selected in such a way that an optimal tree structure is generated.

For the categorical values selected above, this order is (from root to leaf): Transmission Protocol (TCP or UDP) => Service => Port Number => IP Address. The reason for this is that successive values must always be in a 1:n relationship.

A transmission protocol such as TCP or UDP can be used by several services, one service can use several ports and several IP addresses can be addressed via the same port. Saving the different combinations of categorical values in trees removes redundant elements and simplifies matching with new data sets. Trees are generated for each communication model. Each leaf of the trees represents its own communication category.

### *E. Computational Trust*

The anomaly detection generates positive and negative experiences. These are summarized in a trust model and used to calculate a computational trust value for every individual IoT device in the network. This value allows system administrators to evaluate the behavior of an IoT device at a glance.

The concept of trust in a social context is well known from everyday life. Trust allows people to delegate tasks and assess whether information should be shared with another individual. Trust also enables us to evaluate information shared with us. Computational Trust raises the social construct of trust in the field of computer science to build trust or distrust between agents (devices or people). A practicable definition of trust has been specified by the sociologist Diego Gambetta [2]: “Trust (or, symmetrically, distrust) is a particular level of the subjective probability with which an agent assesses that another agent or group of agents will perform a particular action, both before he can monitor such action (or independently of his capacity ever to be able to monitor it) and in a context in which it affects his own action.” So trust is treated as something subjective, which is always associated with some form of prediction and expectation of behavior. Another important element of trust is reputation.

During the application phase, the communication model generates positive and negative experiences based on the observed connection metadata and the results of anomaly detection. If it is possible to assign the received Conn log (a) to a category, a positive experience is saved. If this is not possible or if a distribution anomaly is triggered, a negative experience is stored. These experiences are persistently stored and used by a computational trust model, to calculate a trust value for each device.

This value is the subjective probability value E, which indicates the probability with which future experiences with a device can be rated positive. The higher this value, the more trustworthy is an IoT device. This allows system administrators to control the behavior of an IoT device by checking a simple numerical value $x \in [0, . . . 1]$ to evaluate. If the confidence value of a device decreases unexpectedly quickly, it can be assumed that the device will be compromised because the device triggers a high number of anomalies.

However, not all triggered anomalies actually indicate a compromise of the IP-based IoT device. The particularly rigid communication model can lead to an increase in false positives. However, by calculating trusted information based on experience, these individual false positives do not have a strong impact on trustworthiness, which does not unnecessarily upset the system administrator. Only with a compromise of the IoT device can be expected with an increased number of anomalies, whereby the trust value decreases permanently. Computational Trust is therefore used to find and eliminate the false positive anomalies triggered by anomaly detection.

As trust model the proof-of-concept implementation will use Certain Trust, developed by Sebastian Ries [10], which allows to represent trust for ubiquitous computing and P2P systems in a way, which can be interpreted and updated by software agents as well as by users. A key feature of Certain Trust is that it is capable of expressing the certainty of a trust opinion, depending on the context of use.

## V. CONCLUSION AND OUTLOOK

This paper has presented and discussed a new cascading three step approach for anomaly detection in unsupervised communication meta data of IP-based Internet of Things devices. The new approach cascades X-means clustering, decision tree, and statistical analysis (see Figure 4 below), in order to monitor and protect IoT networks. The new approach is restricted to IP-based IoT devices, and request/response application protocols. The cascading three step approach is designed for anomaly detection in unsupervised CoAP and HTTP communication meta data. Additionally a trust model has discussed in order to allow system administrators to control the behavior of an IoT device by simply checking the trust value of this particular IoT device.

It is planned to write an additional research paper on the real proof-of-concept implementation of the presented three step anomaly detection approach for the next International Conference on Cyber-Technologies and Cyber-Systems. Additionally, a detailed evaluation of CoAP and HTTP communication meta data of IP-based IoT devices, including a verification of the false positive rate, shall be done.

This work was supported by the Center for Research in Security and Privacy (CRISP), Darmstadt, Germany.

## REFERENCES

[1] Shekhar R. Gaddam; Vir V. Phoha; and Kiran S. Balagani, “K-Means+ ID3: A novel method for supervised anomaly detection by cascading K-Means clustering and ID3 decision tree learning methods,” In IEEE Transactions on Knowledge and Data Engineering, vol. 19 (2007), no. 3, pp. 345–354, 2007. http://dx.doi.org/10.1109/TKDE.2007.44. – DOI 10.1109/ TKDE. 2007.44

[2] Kapitel Can we Trust Trust? In: Diego Gambetta, “Trust: Making and Breaking Cooperative Relations,” electronic edition, pp. 213–237, 2000.

[3] (CSD), NIST Computer Security D.: NIST SP 800-94, Guide to Intrusion Detection and Prevention Systems (IDPS), 2007.

[4] Ali A. Ghorbani, Wei Lu, and Mahbod Tavallaee, Chapter 4 “Theoretical Foundation of Detection” in “Network Intrusion Detection and Prevention,” Advances in Information Security vol. 47, Springer Science, 2010.

[5] Christian Karasiewicz, “Why HTTP is not enough for the Internet of Things,” https://www.ibm.com/developerworks/ community/blogs/mobileblog/entry/why_http_is_not_enough _for_the_internet_of_things?lang=en. September 2013.

[6] Knud L. Lueth, “Why the Internet of Things is called Internet of Things: Definition, history, disambiguation”, https://iot-analytics.com/internet-of-things-definition/. Dezember 2014

[7] Luigi Atzori, Antonio Iera, and Giacomo Morabi “The Internet of Things: A survey,” In Computer Networks, Volume 54, Issue 15, pp. 2787-2805, 28 October 2010, http://dx.doi.org/10.1016/j.comnet.2010.05.010. – DOI 10.1016 j.comnet.2010.05.010

[8] Monowar H. Bhuyan ; D. K. Bhattacharyya ; and J. K. Kalita, “Network Anomaly Detection: Methods, Systems and Tools,” In IEEE Communications Surveys & Tutorials, Volume: 16, Issue: 1, pp. 303 - 336, 2014.

[9] Dau Pelleg, and Andrew Moore, “X-means: Extending K-means with Efficient Estimation of the Number of Clusters,” In Proceedings of the 17th International Conf. on Machine Learning, Morgan Kaufmann, pp. 727–734, 2000.

[10] Sebastian Ries, “Certain Trust: a trust model for users and agents,” In Proceedings of the ACM symposium on Applied computing (SAC 2007), pp. 1599-1604, 2007. DOI=http://dx.doi.org/10.1145/1244002.1244342

[11] Salima Omar, Asri Ngadi, and Hamid H. Jebur, “Machine Learning Techniques for Anomaly Detection: An Overview,” In: International Journal of Computer Applications (0975 – 8887), Volume 79, No. 2, October 2013.

[12] Zach Shelby; Klaus Hartke; and Carsten Bormann, “The constrained application protocol (CoAP),” https://tools.ietf.org/html/rfc7252. Version: 2014

[13] Robin Sommer, “Bro: An Open Source Network Intrusion Detection System,” In: DFN Arbeitstagung über Kommunikationsnetze, pp. 273-288, 2003.

[14] Meidan, Y., Bohadana, M., Shabtai, A., Ochoa, M., Tippenhauer, N.O., Guarnizo, J.D., and Elovici, Y. „Detection of Unauthorized IoT Devices Using Machine Learning Techniques”, 2017, https://arxiv.org/pdf/1709.04647.pdf

[15] The Bro Network Security Monitor - https://www.bro.org

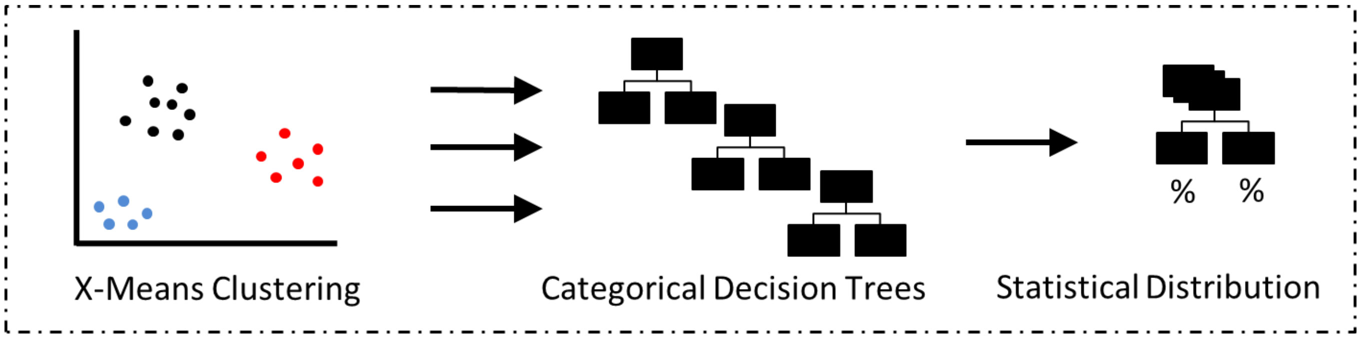


Figure 4. Cascading Three Step Approach for Anomaly Detection in Unsupervised Communication Meta Data of IP-based Internet of Things Devices

# Exploiting the Potential of Web Application Vulnerability Scanning

Damiano Esposito, Marc Rennhard

School of Engineering
Zurich University of Applied Sciences
Winterthur, Switzerland
Email: `espo,rema@zhaw.ch`

Lukas Ruf, Arno Wagner

Consecom AG
Zurich, Switzerland
Email: `Lukas.Ruf,Arno.Wagner@consecom.com`

***Abstract*—Using automated web application vulnerability scanners so that they truly live up to their potential is difficult. Two of the main reasons for this are limitations with respect to crawling capabilities and problems to perform authenticated scans. In this paper, we present JARVIS, which provides technical solutions that can be applied to a wide range of vulnerability scanners to overcome these limitations. Our evaluation shows that by using JARVIS, the vulnerability detection performance of five freely available scanners can be improved by more than 100% compared to using them in their basic configuration. As the configuration effort to use JARVIS is small and the configurations are scanner-independent, JARVIS also allows to use multiple scanners in parallel in an efficient way. In an additional evaluation, we therefore analyzed the potential and limitations of using multiple scanners in parallel. This revealed that using multiple scanners in a reasonable way is indeed beneficial as it increases the number of detected vulnerabilities without a significant negative impact on the reported false positives.**

***Keywords*–Web Application Security; Vulnerability Scanning; Vulnerability Detection Performance.**

## I. INTRODUCTION

Security testing is important to achieve security and trustworthiness of software and systems. Security testing can be performed in different ways, ranging from completely manual methods (e.g., manual source code analysis), to semi-automated methods (e.g., analyzing a web application using an interceptor proxy), to completely automated ways (e.g., analyzing a web service using a vulnerability scanner).

Ideally, at least parts of security testing should be automated. One reason for this is that it increases the efficiency of a security test and frees resources for those parts of a security test that cannot be easily automated. This includes, e.g., access control tests, which cannot really be automated as a testing tool doesn't have an understanding of which users or roles are allowed to perform what functions. Another reason is that automating security tests allows to perform continuous and reproducible security tests, which is getting more and more important in light of short software development cycles.

There are different options how to perform automated security testing. The most popular approaches include static and dynamic code analysis and vulnerability scanning. Vulnerability scanners test a running system "from the outside" by sending specifically crafted data to the system and by analyzing the received response. Among vulnerability scanners, web application vulnerability scanners are most popular, as web applications are very prevalent, are often vulnerable and are frequently attacked [1]. Note also that web applications are not only used to provide typical services such as information portals, e-shops or access to social networks, but they are also very prevalent to configure all kinds of devices attached to the Internet, which includes, e.g., switches, routers and IoT devices. This further undermines the importance of web application security testing.

At first glance, using web application vulnerability scanners seems to be easy as they claim to uncover many vulnerabilities with little configuration effort – as a minimum, they only require the base URL of the application to test as an input. However, their effective application in practice is far from trivial. The following list summarizes some of the limitations:

1) The detection capability of a scanner is directly dependent on its crawling performance: If a scanner can't find a specific resource in a web application, it can't test it and won't find vulnerabilities associated with this resource. Previous work shows that the crawling performance of different scanners varies significantly [2], [3].
2) To test areas of a web application that are only reachable after successful user authentication, the scanners must authenticate themselves during crawling and testing. While most scanners can be configured so they can perform logins, they typically do not support all authentication methods used by different web applications. Also, scanners sometimes log out themselves (e.g., by following a logout link) during testing and sometimes have problems to detect whether an authenticated session has been invalidated. Overall, this makes authenticated scans unreliable or even impossible in some cases.
3) To cope with these limitations, scanners usually provide configuration options, which can increase the number of detected vulnerabilities [4]. This includes, e.g., specifying additional URLs that can be used by the crawler as entry points, manually crawling the application while using the scanner as a proxy so it can learn the URLs, and specifying an authenticated session ID that can be used by the scanner to reach access-protected areas of the application if the authentication method used by the web application is not supported. However, using these options complicate the usage of the scanners and still do not always deliver the desired results.
4) With respect to the number and types of the reported findings, different vulnerability scanners perform dif-

ferently depending on the application under test [5]. Therefore, when testing a specific web application, it's reasonable to use multiple scanners in parallel and combine their findings. However, the limitations described above make this cumbersome and difficult, as each scanner has to be configured and optimized differently.

The goal of this paper is to overcome these limitations and to evaluate how much this improves the vulnerability detection performance of web application vulnerability scanners. To achieve this goal, we developed JARVIS, which provides technical solutions to overcome limitations 1 and 2 in the list above. Using JARVIS requires only minimal configuration, which overcomes limitation 3. And finally, JARVIS and its usage are independent of specific vulnerability scanners and can be applied to a wide range of scanners available today, which overcomes limitation 4 and which provides an important basis to use multiple scanners in parallel in an efficient way.

JARVIS was then applied to several vulnerability scanners to evaluate its effectiveness and to learn more about the potential and limitations of combining multiple scanners. In this analysis, the five freely available scanners listed in Table I were used.

TABLE I. ANALYZED WEB APPLICATION VULNERABILITY SCANNERS

| *Scanner* | *Version/Commit* | *URL* |
|---|---|---|
| Arachni | 1.5-0.5.11 | http://www.arachni-scanner.com |
| OWASP ZAP | 2.5.0 | https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project |
| Skipfish | 2.10b | https://code.google.com/archive/p/skipfish/ |
| Wapiti | r365 | http://wapiti.sourceforge.net |
| w3af | cb8e91af9 | https://github.com/andresriancho/w3af |

The choice for using freely available scanners was mainly driven by the goal to evaluate the performance of using multiple scanners in parallel. This is a much more realistic scenario with freely available scanners as commercial ones often have a hefty price tag. Also, previous work concluded that freely available scanners do not perform worse than commercial scanners [2], [3]. Arguments for using the scanners in Table I instead of using others include our previous experience with these scanners, that these scanners are among the most popular used scanners in practice, and that they perform well in general according to [3].

The main contributions of this paper are the following:

- Technical solutions to improve the crawling coverage and the reliability of authenticated scans of web application vulnerability scanners. In contrast to previous work (see Section IV), our solutions cover both aspects, can easily be applied to a wide range of scanners available today, and require only minimal, scanner-independent configuration.
- An evaluation that demonstrates how much the vulnerability detection performance of five different scanners is improved when using these technical solutions.
- An evaluation that demonstrates the benefits and limitations when using multiple scanners in parallel.

The remainder of this paper is organized as follows: Section II introduces the technical solutions to overcome the limitations of today's scanners and Section III contains the evaluation results. Related work is covered in Section IV and Section V concludes this work.

## II. TECHNICAL APPROACH OF JARVIS

One way to improve the vulnerability detection performance of scanners is to directly adapt one or more current scanners. However, the main disadvantage of this approach is that this would only benefit one or a small set of scanners and would be restricted to scanners that are provided as open source software. Therefore, a proxy-based approach was chosen that is independent of any specific scanner, that does not require adaptation of current scanners, and that can be used with many scanners that are available today and most likely also with scanners that will appear in the future. The basic idea is illustrated in Figure 1.

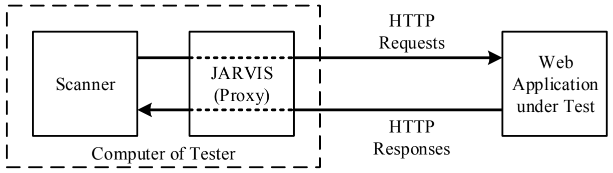


Figure 1. Proxy-based Approach of JARVIS.

A proxy-based approach means that JARVIS, which provides the technical solutions to overcome the limitations of the scanners, acts as a proxy between the scanner and the web application under test. This gives JARVIS access to all HTTP requests and responses exchanged between scanner and web application, which allows to control the entire crawling and scanning process and to adapt requests or responses as needed. This proxy-based approach is possible because most scanners are proxy-aware, i.e., they allow to configure a proxy through which communication with the web application takes place. Note that JARVIS can basically be located on any reachable host, but the typical scenario is using JARVIS on the same computer as the scanner (e.g., on the computer of the tester).

As a basis for JARVIS, the community edition version 1.7.19 of Burp Suite [6] is used. Burp Suite is a tool to support web application security testing that allows to record, intercept, analyze, modify and replay HTTP requests and responses. Therefore, Burp Suite already provides many basic functions that are required to implement JARVIS. In addition, Burp Suite provides an application programming interface (API) so it can be extended and JARVIS makes use of this API.

JARVIS consist of two main components. The first is described in Section II-A and aims at improving the test coverage of scanners. This component should especially help scanners that have a poor crawling performance. The second component, described in Section II-B, aims at improving the reliability of authenticated scans and should assist scanners that have limitations in this area. Finally, Section II-C gives a configuration example when using JARVIS to demonstrate that the configuration effort is small.

### A. *Improving Test Coverage*

Improving test coverage could be done by replacing the existing crawler components of the scanners with a better one (see, e.g., [7]–[9]). While this may be helpful for some scanners, it may actually be harmful for others, in particular if the integrated crawler works well. Therefore, an approach was chosen that does not replace but that assists the crawling components that are integrated in the different scanners. The idea is to supplement the crawlers with additional URLs (beyond the base URL) of the web application under test. These additional URLs are named *seeds* as they are used to seed the crawler components of the scanners. Intuitively, this should significantly improve crawling coverage, in particular if the integrated crawler is not very effective. To get the additional URLs of a web application, two different approaches are used: endpoint extraction from the source code of web applications and using the detected URLs of the best available crawler(s).

Endpoint extraction means searching the source code (including configuration files) of the web application under test for URLs and parameters. The important benefits of this approach are that it can detect URLs that are hard to find by any crawler and that it can uncover hidden parameters of requests (e.g., debugging parameters). To extract the endpoints, ThreadFix endpoint CLI [10] was used, which supports many common web application frameworks (e.g., JSP, Ruby on Rails, Spring MVC, Struts, .NET MVC and ASP.NET Web-Forms). In addition, further potential endpoints are constructed by appending all directories and files under the root directory of the source code to the base URL that is used by the web application under test. This is particularly effective when scanning web applications based on PHP.

Obviously, endpoint extraction is only possible if the source code of the application under test is available. If that's not the case, the second approach comes into play. The idea here is to use the best available crawler(s) to gather additional URLs. As will be shown later, Arachni provides good crawling performance in general, so Arachni is a good starting point as a tool for this task. Of course, it's also possible to combine both approaches to determine the seeds: extract the endpoints from the source code (if available) *and* get URLs with the best available crawler(s).

Once the seeds have been derived, they must be injected into the crawler component of the scanners. To do this, most scanners provide a configuration option. However, this approach has its limitations as such an option is not always available and usually only supports GET requests but no POST requests. Therefore, the seeds are injected by JARVIS. To do this, four different approaches were implemented based on *robots.txt*, *sitemap.xml*, a landing page, and the index page.

Using *robots.txt* and *sitemap.xml* is straightforward. These files are intended to provide search engine crawlers with information about the target web site and are also evaluated by most crawler components of scanners. When the crawler component of a scanner requests such a file, JARVIS supplements the original file received from the web application with the seeds (or generates a new file with the seeds in case the web application does not contain the file at all). Both approaches work well but are limited to GET request.

The other two approaches are more powerful as they also support POST request. The landing page-based approach places all seeds as links or forms into a separate web page (named *landing.page*) and the scanner is configured to use this page as the base URL of the web application under test (e.g., http://www.example.site/landing.page instead of http://www.example.site). When the crawler requests the page, JARVIS delivers the landing page, from which the crawler learns all the seeds and uses them during the remainder of the crawling process. One limitation of this approach is that the altered base URL is sometimes interpreted as a directory by the crawler component of the scanners, which means the crawler does not request the landing page itself but tries to fetch resources below it. This is where the fourth approach comes into play. The index page-based approach injects seeds directly into the first page received from the web application (e.g., just before the </body> tag of the page *index.html*). Overall, these four approaches allowed to successfully seed all scanners in Table I when used to test the web applications in the test set (see Section III-A).

As an example, the effectiveness of the landing page-based approach is demonstrated. To do this, WIVET version 4 [11] is used, which is a benchmarking project to assess crawling coverage. Table II shows the crawling coverage that can be achieved with OWASP ZAP (in headless mode) and Wapiti when they are seeded with the crawling results of Arachni via a landing page.

TABLE II. CRAWLING COVERAGE

| *Scanner* | *Raw coverage* | *Coverage when seeded with the crawling results of Arachni* |
|---|---|---|
| Arachni | 92.86% | |
| OWASP ZAP | 14.29% | 96.43% |
| Wapiti | 48.21% | 96.43% |

Table II shows that the raw crawling coverage of Arachni is already very good (92.86%), while Wapiti only finds about half of all resources and OWASP ZAP only a small fraction. By seeding OWASP ZAP and Wapiti with the crawling results of Arachni, their coverage can be improved drastically to 96.43%. This demonstrates that seeding via a landing page indeed works very well.

## B. Improving Authenticated Scans

Performing authenticated scans in a reliable way is challenging for multiple reasons. This includes coping with various authentication methods, prevention of logouts during the scans, and performing re-authentication when this is needed (e.g., when a web application with integrated protection mechanisms invalidates the authenticated session when being scanned) to name a few. It is therefore not surprising that many scanners have difficulties to perform authenticated scans reliably.

To deal with these challenges, several modules were implemented in JARVIS. The first one serves to handle various authentication methods, including modern methods based on HTTP headers (e.g., OAuth 2.0). The module provides a wizard to configure authentication requests, can submit the corresponding requests, stores the authenticated cookies received from the web applications, and injects them into subsequent requests from the scanner to make sure the requests are interpreted as authenticated requests by the web application. The main advantages of this module are that it enables authenticated scans even if a scanner does not support the authentication method and that it provides a consistent way to configure authentication independent of a particular scanner.

Furthermore, a logout prevention module was implemented to make sure a scanner is not doing a logout by following links or performing actions which most likely invalidate the current session (e.g., change password or logout links). This is configured by specifying a set of corresponding URLs that should be avoided during the scan. When the proxy detects such a request, it blocks the request and generates a response with HTTP status code 200 and an empty message body. In addition, a flexible re-authentication module was developed. Re-authentication is triggered based on matches of configurable literal strings or regular expressions with HTTP response headers (e.g., the *location* header in a redirection response) or with the message body of an HTTP response (e.g., the occurrence of a keyword such as *login*).

## C. Configuration Example

To give an impression of the configuration effort needed when using JARVIS, Table III lists the parameters that must be configured when scanning the test application BodgeIt (see Section III-A). In this example, the seeds are extracted from the source code.

TABLE III. EXAMPLE CONFIGURATION WHEN SCANNING BODGEIT

| *Parameter* | *Value(s)* |
|---|---|
| Base URL | http://bodgeit/ |
| Source code | ~/bodgeit/ |
| Authentication mode | POST |
| Authentication URL | http://bodgeit/login.jsp |
| Authentication parameters | password=password<br>username=test@test.test |
| Out of scope | http://bodgeit/password.jsp<br>http://bodgeit/register.jsp<br>http://bodgeit/logout.jsp |
| Re-auth. search scope | HTTP response body |
| Re-auth. keywords | Login, Guest, user |
| Re-auth. keyword interpretation | Literal string(s) |
| Re-auth. case-sensitive | True |
| Re-auth. match indicates | Invalid session |
| Seeding approach(es) | Landing page, robots.txt, sitemap.xml |

The entries in Table III are self-explanatory and show that the configuration effort is rather small. In particular, the configuration is independent of the actual scanner, which implies that when using multiple scanners in parallel (see Section III-D), this configuration must only be done once and not once per scanner.

## III. EVALUATION

This section starts with a description of the evaluation setting. Then, the results of the evaluation of the vulnerability detection performance is presented when the scanners are used with and without the improvements described in Section II. In the final step, the benefits and limitations of using multiple scanners in parallel is evaluated.

## A. Evaluation Setting

Table IV lists the web applications that were used to evaluate the scanners (Cyclone Transfers and WackoPicko do not use explicit versioning).

All these applications are deliberately insecure and well suited for security training and to test vulnerability scanners. The main reason why the applications in Table IV were chosen is because they cover various technologies, including Java, PHP, Node.js and Ruby on Rails.

TABLE IV. WEB APPLICATIONS USED FOR THE EVALUATION

| *Application* | *Version* | *URL* |
|---|---|---|
| BodgeIt | 1.4.0 | https://github.com/psiinon/bodgeit |
| Cyclone Transfers | – | https://github.com/thedeadrobots/bwa_cyclone_transfers |
| InsecureWebApp | 1.0 | https://www.owasp.org/index.php/Category:OWASP_Insecure_Web_App_Project |
| Juice Shop | 2.17.0 | https://github.com/bkimminich/juice-shop |
| NodeGoat | 1.1 | https://github.com/OWASP/NodeGoat |
| Peruggia | 1.2 | https://sourceforge.net/projects/peruggia/ |
| WackoPicko | – | https://github.com/adamdoupe/WackoPicko |

The evaluation uses four different configurations that are listed in Table V.

TABLE V. CONFIGURATIONS USED DURING THE EVALUATION

| *Config.* | *The scans are executed...* |
|---|---|
| *-/-* | ...without seeding and non-authenticated (i.e., using the basic configuration of the scanners by setting only the base URL) |
| *S/-* | ...with seeding and non-authenticated (i.e., using the technical solution described in Section II-A) |
| *-/A* | ...without any seeding and authenticated (i.e., using the technical solution described in Section II-B) |
| *S/A* | ...with seeding and authenticated (i.e., using both technical solutions described in Sections II-A and II-B) |

As the source code of all these applications is available, the endpoint extraction approach described in Section II-A is used for seeding in configurations *S/-* and *S/A*.

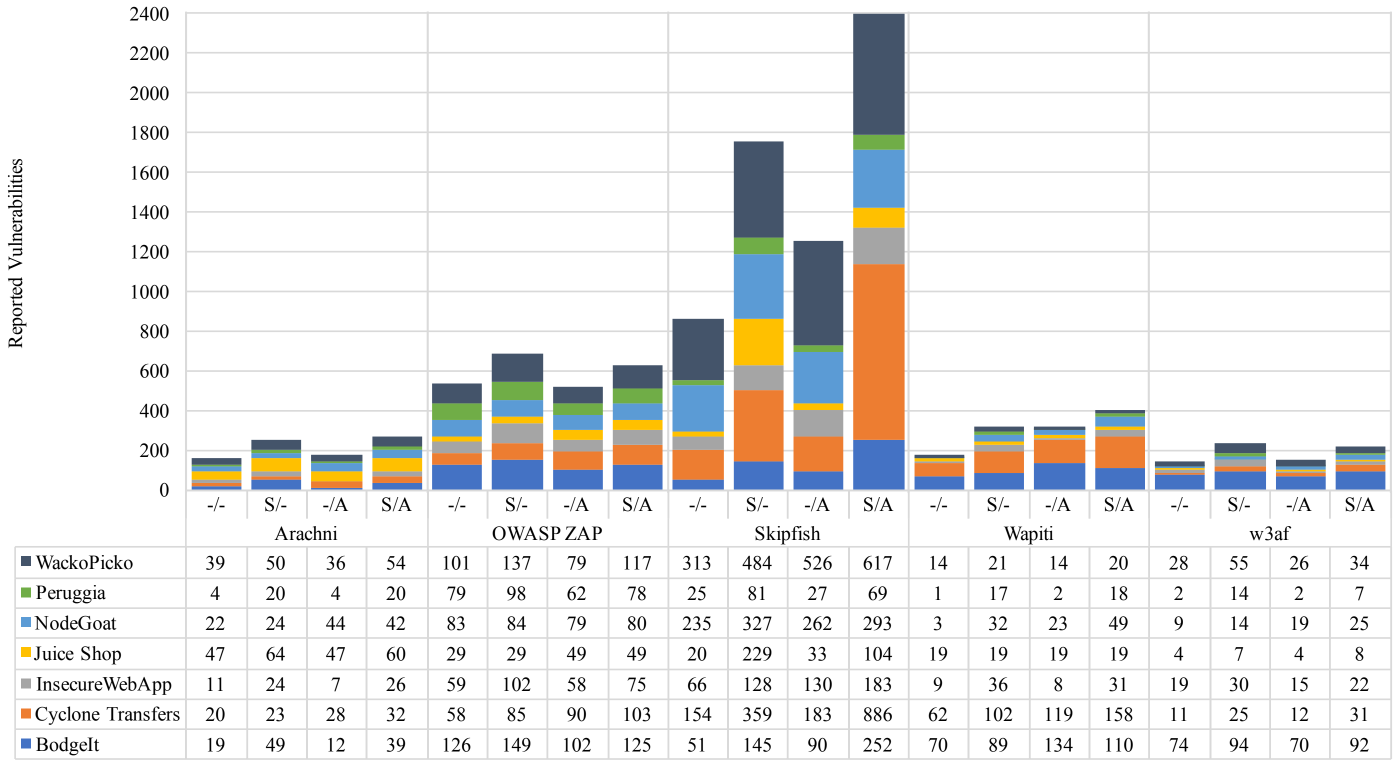
The test applications were run in a virtual environment that was reset to its initial state before each test run to make sure that every run is done under the same conditions and is not influenced by any of the other scans.

## B. Overall Evaluation

The first evaluation analyzes the overall number of vulnerabilities that are reported by the scanners when using the four different configurations described in Table V. Figure 2 illustrates the evaluation results.

The first observation when looking at Figure 2 is that some scanners identify many more vulnerabilities than others. For example, Skipfish reports about ten times as many findings as Arachni or w3af. However, this doesn't mean that Skipfish is the best scanner, because Figure 2 depicts the "raw number of vulnerabilities" reported by the scanners and does not take into account false positives, duplicate findings, or the criticality of the findings. For instance, about 80% of the vulnerabilities reported by Skipfish are rated as *info* or *low* (meaning they have only little security impact in practice) while the other scanners report a much smaller fraction of such findings.

More importantly, Figure 2 shows that the technical solution to improve test coverage works well with all scanners and all test applications included in the evaluation. The number of vulnerabilities reported when seeding is used is nearly always greater than without seeding. For instance, Arachni reports 64 vulnerabilities in Juice Shop in configuration *S/-* compared to 47 in configuration *-/-*. Similarly, when using authenticated scans, Arachni reports 39 findings in BodgeIt in configuration *S/A* compared to 12 in configuration *-/A*. The same is true when adding up the vulnerabilities of a specific scanner over all test applications: Configuration *-/-* always reports fewer findings than configuration *S/-* (e.g., 162 vs. 254 with Arachni) and configuration *-/A* always reports



| | Arachni | | | | OWASP ZAP | | | | Skipfish | | | | Wapiti | | | | w3af | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | -/- | S/- | -/A | S/A | -/- | S/- | -/A | S/A | -/- | S/- | -/A | S/A | -/- | S/- | -/A | S/A | -/- | S/- | -/A | S/A |
| ■ WackoPicko | 39 | 50 | 36 | 54 | 101 | 137 | 79 | 117 | 313 | 484 | 526 | 617 | 14 | 21 | 14 | 20 | 28 | 55 | 26 | 34 |
| ■ Peruggia | 4 | 20 | 4 | 20 | 79 | 98 | 62 | 78 | 25 | 81 | 27 | 69 | 1 | 17 | 2 | 18 | 2 | 14 | 2 | 7 |
| ■ NodeGoat | 22 | 24 | 44 | 42 | 83 | 84 | 79 | 80 | 235 | 327 | 262 | 293 | 3 | 32 | 23 | 49 | 9 | 14 | 19 | 25 |
| ■ Juice Shop | 47 | 64 | 47 | 60 | 29 | 29 | 49 | 49 | 20 | 229 | 33 | 104 | 19 | 19 | 19 | 19 | 4 | 7 | 4 | 8 |
| ■ InsecureWebApp | 11 | 24 | 7 | 26 | 59 | 102 | 58 | 75 | 66 | 128 | 130 | 183 | 9 | 36 | 8 | 31 | 19 | 30 | 15 | 22 |
| ■ Cyclone Transfers | 20 | 23 | 28 | 32 | 58 | 85 | 90 | 103 | 154 | 359 | 183 | 886 | 62 | 102 | 119 | 158 | 11 | 25 | 12 | 31 |
| ■ BodgeIt | 19 | 49 | 12 | 39 | 126 | 149 | 102 | 125 | 51 | 145 | 90 | 252 | 70 | 89 | 134 | 110 | 74 | 94 | 70 | 92 |

Figure 2. Reported Vulnerabilities per Scanner and Test Application.

fewer findings than configuration *S/A* (e.g., 178 vs. 273 with Arachni).

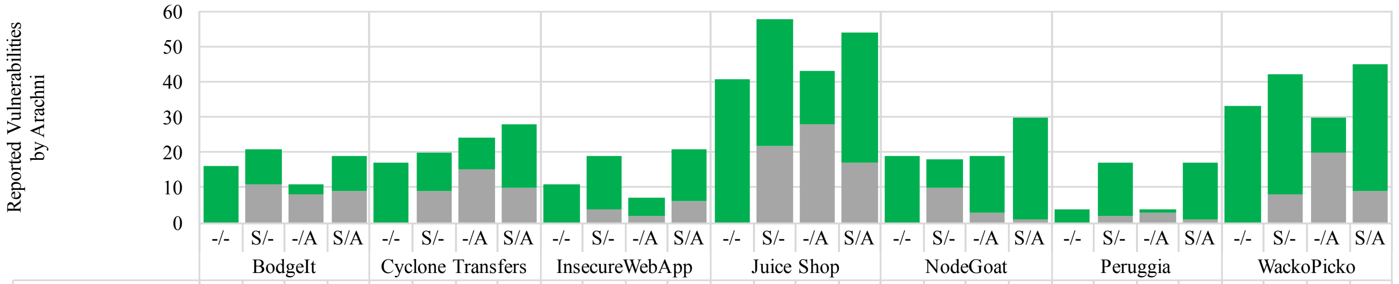
Likewise, Figure 2 demonstrates that the technical solution to improve authenticated scans also works well. For instance, when scanning Cyclone Transfer, Wapiti reports 62 findings in configuration *-/-* and 119 findings in configuration *-/A*. Also, scanning in configuration *S/-* delivers 102 vulnerabilities, which can be increased to 158 in configuration *S/A*. And finally, this also holds true over all applications, as for most scanners, the bars in Figure 2 are higher in configuration *-/A* compared to *-/-* and in configuration *S/A* compared to *S/-*. Note that to make sure that authenticated scans were carried out reliably, the involved requests and responses were analyzed after each scan. This showed that it was indeed possible to maintain authentication during the entire scan, which further undermines that the technical approach is sound.

Intuitively, additionally seeding a scanner or performing authenticated scans should always also report all vulnerabilities that are detected when scanning without additional seeding or without authentication. However, this is not the case. To demonstrate this, it was analyzed how many of the vulnerabilities reported in the basic configuration are also found when scanning in other configurations. To do this, the reports of the scanners were first processed with ThreadFix [12]. ThreadFix allows to normalize reports of different scanners, to eliminate duplicates, and to compare the results of different scanners or different runs by the same scanner. Figure 3 shows the results of the analysis for the findings reported by Arachni. Note that because of the processing with ThreadFix and in contrast to Figure 2, the bars now represent the number of unique findings that were reported.

First of all, Figure 3 undermines what was observed above: Additional seeding and authenticated scans result in a greater number of reported findings, as can be seen by comparing the heights of the bars. In addition, as the bars represent unique vulnerabilities, the additional findings are not just duplicates of already detected findings, but they are truly new findings. Beyond this, Figure 3 confirms that when using additional seeding and/or authenticated scans, not all vulnerabilities that are reported in the basic configuration *-/-* are detected again. For example, considering BodgeIt, Arachni reports 16 findings in configuration *-/-*. When using configuration *S/-*, then 21 findings are reported in total, of which 10 are “new” findings compared to *-/-* (indicated by the green part of the bar). However, only 11 of the 16 vulnerabilities reported in configuration *-/-* are detected again (“old” findings, indicated by the gray part of the bar) and 5 are missing. The same can be observed with the other configurations and with all test applications, which means that in general, additional seeding and/or authenticated scans deliver a significant number of new findings, but also misses several of the findings that are reported in the basic configuration. Note that the same behavior can be observed with all scanners, but only the results of Arachni are included due to space restrictions.

Determining the exact reasons for this behavior would require a detailed analysis of the crawling components of the scanners and the web applications in the test set, which is beyond the scope of this work. Therefore, only a few arguments are given that show the observed behavior is reasonable:

- Providing the crawler component of a scanner with additional seeds has a direct impact on the order in which the pages are requested. A different order



| | BodgeIt | | | | Cyclone Transfers | | | | InsecureWebApp | | | | Juice Shop | | | | NodeGoat | | | | Peruggia | | | | WackoPicko | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | -/- | S/- | -/A | S/A | -/- | S/- | -/A | S/A | -/- | S/- | -/A | S/A | -/- | S/- | -/A | S/A | -/- | S/- | -/A | S/A | -/- | S/- | -/A | S/A | -/- | S/- | -/A | S/A |
| 🟩 New Vulnerabilities | 16 | 10 | 3 | 10 | 17 | 11 | 9 | 18 | 11 | 15 | 5 | 15 | 41 | 36 | 15 | 37 | 19 | 8 | 16 | 29 | 4 | 15 | 1 | 16 | 33 | 34 | 10 | 36 |
| ▨ Old Vulnerabilities | 0 | 11 | 8 | 9 | 0 | 9 | 15 | 10 | 0 | 4 | 2 | 6 | 0 | 22 | 28 | 17 | 0 | 10 | 3 | 1 | 0 | 2 | 3 | 1 | 0 | 8 | 20 | 9 |

Figure 3. Reported Unique Vulnerabilities by Arachni and per Test Application.

implies different internal state changes within the web application under test [7], which typically leads to a different behavior of the web application and therefore to different findings.

- When doing authenticated scans, some of the resources that do not require authentication are often no longer reachable, e.g., registration, login and forgotten password pages. As deliberately insecure web applications often use these resources to place common vulnerabilities, this has a major impact in this test setting.

The consequence of this observation is that when scanning a web application, the scanners should be used in all four configurations to maximize the number of reported findings. And obviously, although this was not analyzed in detail, when an application provides different protected areas for different roles, scanning should be done with users of all roles.

## C. Detailed Evaluation

The evaluation in Section III-B demonstrates that when considering just the number of reported vulnerabilities, JARVIS works well. However, its still unclear whether there's a true benefit in practice because it may be that the additionally found vulnerabilities are mainly false positives or non-critical issues.

To get a better understanding, a more detailed analysis focusing on SQL injection (SQLi) and cross-site scripting (XSS) vulnerabilities was done. To do this, all reported vulnerabilities of these types were manually verified to identify them as either true or false positives. This required a lot of effort, which is the main reason why the focus was set on these two types. Nevertheless, this serves well to evaluate the true potential of JARVIS as both vulnerabilities are highly relevant in practice and highly security-critical. In addition, the test applications contain several of them, which means SQLi and XSS vulnerabilities represent a meaningful sample size. Figure 4 shows the results of this analysis. Just like in Figure 3, the bars represent the number of unique findings that were reported.

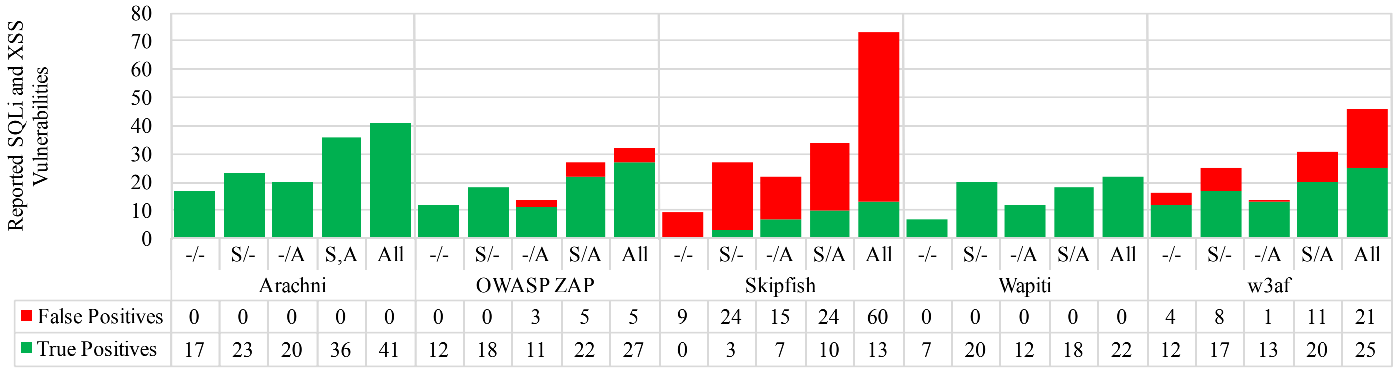
Looking only at the true positives (green bars), Figure 4 confirms that JARVIS indeed works well in the sense that using additional seeding and authenticated scans allows the scanners to detect highly relevant and security-critical vulnerabilities that are not reported in the basic configuration, which is true for all scanners. The results also undermine that it's important to perform scans in all four configurations (named configuration *All*), as the sums of the detected vulnerabilities (bars labeled with *All*) are always greater than the vulnerabilities detected in any of the other configurations. Furthermore, the results demonstrate that for each of the five scanners, combining the results of all configurations yields more than twice as many vulnerabilities (true positives) as when performing scans only in the basic configuration -/-, so JARVIS results in an improvement of over 100%.

In addition, Figure 4 shows that scanners that tend towards reporting false positives (red bars) do so also in the advanced configurations, but overall, the fraction of false positives remains more or less constant independent of the configuration. That's an important results as it demonstrates that the technical improvements result in more true findings without an increased percentage of false positives. And finally, Figure 4 allows to compare the scanners. In particular, based on the test applications and focusing on SQLi and XSS vulnerabilities, it shows that Arachni performs best (without producing a single false positive) and Skipfish performs quite poorly, especially with respect to false positives. This also puts into perspective the results of the first evaluation (see Figure 2), where Skipfish reported many more vulnerabilities than the other scanners.

## D. Combining Multiple Scanners

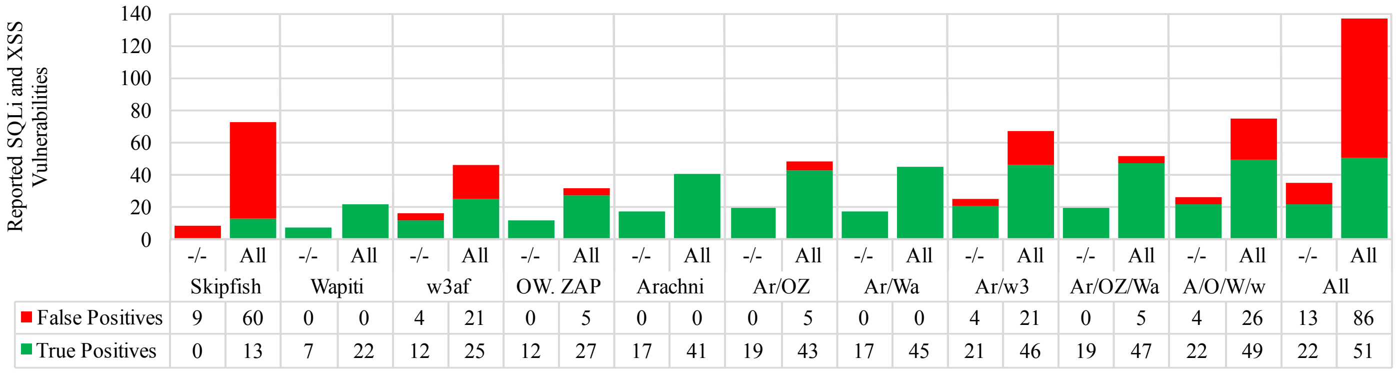
In the final evaluation, the benefits of using multiple scanners in parallel are analyzed. Figure 5 shows the combined unique true and false positives when using individual scanners and different combinations thereof and when using the scanners in the basic configuration -/- or in configuration *All*. The results are ranked from left to right according to the number of true positives that are identified in configuration *All*.

Looking at the results in configuration *All*, the rightmost bar combines the results of all five scanners, which obviously delivers most true positives (51), but which also delivers most false positives (86). The results also show that in this test setting, Arachni performs very well on its own, as it finds 41 true positives (without a single false positive), which means that the other four scanners combined can only detect 10 true positives that are not found by Arachni. Looking at combinations of scanners, then Arachni & Wapiti (Ar/Wa) perform well and identify 45 of the 51 true positives without any false positive. Combining Arachni, OWASP ZAP & Wapiti (Ar/OZ/Wa) is also a good choice as it finds 47 true positives with only a few false positives. This demonstrates



| | Arachni | | | | | OWASP ZAP | | | | | Skipfish | | | | | Wapiti | | | | | w3af | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | -/- | S/- | -/A | S,A | All | -/- | S/- | -/A | S/A | All | -/- | S/- | -/A | S/A | All | -/- | S/- | -/A | S/A | All | -/- | S/- | -/A | S/A | All |
| 🟥 False Positives | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 5 | 5 | 9 | 24 | 15 | 24 | 60 | 0 | 0 | 0 | 0 | 0 | 4 | 8 | 1 | 11 | 21 |
| 🟩 True Positives | 17 | 23 | 20 | 36 | 41 | 12 | 18 | 11 | 22 | 27 | 0 | 3 | 7 | 10 | 13 | 7 | 20 | 12 | 18 | 22 | 12 | 17 | 13 | 20 | 25 |

Figure 4. Reported Unique SQLi and XSS Vulnerabilities per Scanner, over all Test Applications.



| | Skipfish | | Wapiti | | w3af | | OW. ZAP | | Arachni | | Ar/OZ | | Ar/Wa | | Ar/w3 | | Ar/OZ/Wa | | A/O/W/w | | All | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | -/- | All | -/- | All | -/- | All | -/- | All | -/- | All | -/- | All | -/- | All | -/- | All | -/- | All | -/- | All | -/- | All |
| 🟥 False Positives | 9 | 60 | 0 | 0 | 4 | 21 | 0 | 5 | 0 | 0 | 0 | 5 | 0 | 0 | 4 | 21 | 0 | 5 | 4 | 26 | 13 | 86 |
| 🟩 True Positives | 0 | 13 | 7 | 22 | 12 | 25 | 12 | 27 | 17 | 41 | 19 | 43 | 17 | 45 | 21 | 46 | 19 | 47 | 22 | 49 | 22 | 51 |

Figure 5. Reported Unique SQLi and XSS Vulnerabilities using different Scanner Combinations, over all Test Applications.

that combining multiple scanners is beneficial to increase the number of detected true positives without a significant negative impact on the number of reported false positives. However, blindly combining as many scanners as possible (e.g., all five scanners used here) is not a good idea in general as although this results in most true positives, it also combines all false positives. Finally, comparing the results in configuration *All* with the ones in configuration *-/-* demonstrates that even when combining multiple scanners, configuration *All* increases the number of detected true positives always by more than 100%, which again undermines the benefits of JARVIS.

Note that since seven test web applications that cover several technologies are used, the results are at least an indication that the combinations of scanners proposed above should perform well in many scenarios. However, this is certainly no proof and it may be that other combinations of scanners are better suited depending on the web application under test. This means that in practice, one has to experiment with different combinations to determine the one that is best suited in a specific scenario.

## IV. Related Work

Several work has been published on the crawling coverage and detection performance of web application vulnerability scanners. In [2], more than ten scanners were compared, with the main results that good crawling coverage is paramount to detect many vulnerabilities and that freely available scanners perform as well as commercial ones. The same is confirmed by [3], which covers more than 50 free and commercial scanners and which is updated regularly. In [4], Suto concludes that when carefully training or configuring a scanner, detection performance is improved, but this also significantly increases the complexity and time effort needed to use a scanner. Furthermore, Bau et al. demonstrate that the eight scanners they used in their analysis have different strengths, i.e. they find different vulnerabilities [5].

Other work specifically aimed at improving the coverage of vulnerability scanning. In [7], it is demonstrated that by taking into account the state changes of a web application when it processes requests, crawling and therefore scanning performance can be improved. In [8], van Deursen et al. present a Selenium WebDriver-based crawler called Crawljax, which improves crawling of Ajax-based web applications. The same is achieved by Pellegrino et al. by dynamically analyzing JavaScript code in web pages [9].

Our work presented in this paper builds upon this previous work as it delivers practical and effective technical solutions to overcome the limitations and exploit the potential identified by others. What sets our approach apart from other work is that it addresses not only crawling coverage but also the reliability of authenticated scans, that it is scanner-independent, and that it can easily be applied to most vulnerability scanners available today. In addition, we provide a detailed evaluation using several scanners and several test applications that truly demonstrates the benefits and practicability of our technical solutions.

## V. Conclusion

In this paper, we presented JARVIS, which provides technical solutions to overcome some of the limitations – notably crawling coverage and reliability of authenticated scans – of web application vulnerability scanners. As JARVIS is independent of specific scanners and implemented as a proxy, it can be applied to a wide range of existing vulnerability scanners. The evaluation based on five freely available scanners and seven test web applications covering various technologies demonstrates that JARVIS works well in practice and that the vulnerability detection rate (true positives) of the scanners can by improved by more than 100% compared to using the scanners in their basic configuration.

The configuration effort to use JARVIS is small and the configurations are scanner-independent. Therefore, JARVIS also provides an important basis to use multiple scanners in parallel in an efficient way. The provided analysis shows that combining multiple scanners is indeed beneficial as it increases the number of true positives, which is not surprising as different scanners detect different vulnerabilities. However, it was also demonstrated that blindly combining as many scanners as possible is not a good idea in general because although this results in most true positives, it also delivers the sum of all false positives reported by the scanners. In the evaluation, the combination of Arachni & Wapiti or Arachni, OWASP ZAP & Wapiti yielded the best compromise between a high rate of true positives and a low rate of false positives. As a representative set of web application technologies was used in the evaluation, it can be expected that these combinations work well in many scenarios, but this is no proof and in practice, one has to experiment with different combinations to determine the one that is best suited in a specific scenario.

## Acknowledgement

This work was partly funded by the Swiss Confederation's innovation promotion agency CTI (project 18876.1 PFES-ES).

## References

[1] WhiteHat Security, "2017 application security statistics report," Tech. Rep., 2017, URL: https://www.whitehatsec.com/resources-category/premium-content/web-application-stats-report-2017/ [accessed: 2018-05-15].

[2] A. Doupé, M. Cova, and G. Vigna, "Why johnny can't pentest: An analysis of black-box web vulnerability scanners," in Proceedings of the 7th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, ser. DIMVA'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 111–131.

[3] S. Chen, "SECTOOL market," 2016, URL: http://www.sectoolmarket.com/price-and-feature-comparison-of-web-application-scanners-unified-list.html [accessed: 2018-05-15].

[4] L. Suto, "Analyzing the accuracy and time costs of web application security scanners," Tech. Rep., 2010, URL: https://www.beyondtrust.com/wp-content/uploads/Analyzing-the-Accuracy-and-Time-Costs-of-Web-Application-Security-Scanners.pdf [accessed: 2018-05-15].

[5] J. Bau, E. Bursztein, D. Gupta, and J. Mitchell, "State of the art: Automated black-box web application vulnerability testing," in Proceedings of the 2010 IEEE Symposium on Security and Privacy, 2010, pp. 332–345.

[6] PortSwigger, "Burp Suite," URL: https://portswigger.net/burp [accessed: 2018-05-15].

[7] A. Doupé, L. Cavedon, C. Kruegel, and G. Vigna, "Enemy of the state: A state-aware black-box web vulnerability scanner," in Proceedings of the 21st USENIX Security Symposium (USENIX Security 12). Bellevue, WA: USENIX, 2012, pp. 523–538.

[8] A. v. Deursen, A. Mesbah, and A. Nederlof, "Crawl-based analysis of web applications: Prospects and challenges," Science of Computer Programming, vol. 97, 2015, pp. 173 – 180.

[9] G. Pellegrino, C. Tschürtz, E. Bodden, and C. Rossow, "jäk: Using dynamic analysis to crawl and test modern web applications," in Research in Attacks, Intrusions, and Defenses, H. Bos, F. Monrose, and G. Blanc, Eds. Cham: Springer International Publishing, 2015, pp. 295–316.

[10] ThreadFix, "ThreadFix endpoint CLI," URL: https://github.com/denimgroup/threadfix/tree/master/threadfix-cli-endpoints [accessed: 2018-05-15].

[11] B. Urgun, "WIVET: Web input vector extractor teaser," URL: https://github.com/bedirhan/wivet [accessed: 2018-05-15].

[12] ThreadFix, "Application vulnerability correlation with ThreadFix," URL: https://threadfix.it [accessed: 2018-05-15].

# A Hybrid Approach for Enhancing Android Sandbox Analysis

Ngoc-Tu Chau*, Jaehyeon Yoon† and Souhwan Jung*
*School of Electronic Engineering
Soongsil University
Seoul, Korea 06978
Email: chaungoctu@soongsil.ac.kr, souhwanj@ssu.ac.kr
† Department of Software Convergence
Soongsil University
Seoul, Korea 06978
Email: yjh7593@naver.com

***Abstract*—Dynamic analysis solutions are applied to prevent malicious applications from bypassing Android sandbox using dynamic payload techniques. However, such dynamic analysis methods are vulnerable to malware that use Anti-Analysis and Anti-Emulator techniques. Malicious applications use Anti-Emulation techniques to archive sensitive information that can be used to distinguish between sandbox and real device. Upon identifying sandbox environment, malicious applications may implement several of evasion techniques to avoid from being analyzed. The main problem, however, is that it can be easy for even a novice user to get sensitive information provided by a sandbox with just a little effort. Although there are work-around solutions for solving the problem by directly updating the sensitive information before building the sandbox, they are still containing some limitations in practice. Firstly, it is inconvenient to change the sensitive information after the sandbox or instrumentation module are deployed. Secondly, the updated information can be inconsistent and illogical. To provide a flexible approach for these issues, this paper proposes a dynamic approach that updates the sensitive information based on Sensitive Information Provider server that is located outside the sandbox. The Sensitive Information Provider (SIP) could be a collector that retrieves and processes sensitive information from one or more seeder mobile devices or could be a set of mobile devices. Because of the device-based information, the proposed approach provides a consistence and logic output when it is compared with other solutions. Furthermore, since the proposed solution separates the source of sensitive information from the sandbox, it is possible to update the sensitive information even after the sandbox was deployed. However, the proposed approach sacrifices performance to flexibility and thus it is only suitable to specific environments. The implementation section also analyzes the use-cases which are suitable to apply the proposed solution.**

***Keywords*–Android Analysis; Sensitive Information Provider; Anti-Analysis**

## I. INTRODUCTION

There are three main analysis methods that have been used by Android sandboxes to analyze an application: 1) Static analysis; 2) Dynamic analysis and 3) Hybrid analysis [1]. Static analysis (also known as Source code analysis) technique works by extracting and analyzing information based on the given android application package (APK) file. AndroidManifest.xml, resources and Dalvik bytecode are the most useful information for analysis since they contain the structure of the application, permissions for the application, and behavior of the application (through the byte-code). Representatives for static analysis approach are FlowDroid [2], Droid Intent Data Flow Analysis for Information Leakage (DIDFAIL), AndroSimilar [3]. Unlike static method that performs analysis through śtatic(or non-running) source code, dynamic approach performs application analysis by running the application inside a customized sandbox. Behaviors of an analyzed application are recorded and inspected to check for malicious activity. A hybrid solution is the combination of both static and dynamic methods. On the other hand, Android malware families also evolve themselves in order to avoid being scanned by the analyzer and to bypass the sandbox system. Dynamic payload techniques are usually used by an Android malware to deal with static analysis. With dynamic payload techniques, a malicious code can be encrypted or obfuscated within the Android package to go undetected by the analyzer. However, such dynamic payload techniques are futile against dynamic analysis sandbox since the analyzed application is installed and run directly on the sandbox environment. To handle with dynamic analysis sandbox, cybercriminals usually perform anti-analysis techniques to avoid detection [4]. There are workaround solutions for solving the problem of sensitive information. Sandbox provider can modify the sensitive information before building the sandbox or takes advantage of dynamic instrumentation tools like Xposed and Frida to provide a **fixed manipulation scenario**. However, it is troublesome to rebuild the whole source code in order to change the manipulation scenario. Moreover, the function does not always work since the return of sensitive value sometimes needs to be **logic** and **consistent**. In order to archive better flexibility of sensitive information, this paper introduces a dynamic approach to separate between the sensitive information provider and sandbox environment. The proposed model is aimed at increasing the flexibility of sensitive information inside analysis sandboxes. The rest of the paper is organized as follows. Section 2 provides more information and examples about the analysis methods, anti-emulation techniques. Section 3 introduces proposed approach. Section 4 shows the implementation result with analysis of the new proposed model. Last section summaries the contribution and future research for the research topic.

## II. RELATED WORKS

In this section, in order for the reader to easily reach to the subject mentioned in the proposed models, we introduce the basic knowledge related to the analysis. In addition, solutions related to anti-analysis are also mentioned, along with related articles for providing knowledge about traditional methods of checking sandbox environment.

### A. Analysis Methods

Analysis methods aim at assessing the application vulnerability as well as analysis an application for malicious code. The analysis methods consists of: 1) Static analysis; 2) Dynamic analysis and 3) Hybrid analysis. The following subsections describe detail about each approach.

*1) Static Approach:* Static method analyzes an application without actually executing the APK file. Most of static analysis methods are usually depend on the decompilation technique to repackage the APK file. There are various types of static analysis methods including: Resources-based and Bytecode-based analysis. Basically, resources-based analysis extracts the configuration and resources files for detecting abnormal information. An application are considered abnormal either when it contains patterns that match with a specific malware signature or when it requests a combination of sensitive permissions [5]–[8]. Bytecode-based analysis extracts classes, methods or instructions and applies control-flow analysis to check for malicious actions. Taint analysis that uses to keep track the data that propagate from a source of sensitive information to sink [9] [10]. Data propagation tracking method usually applied in bytecode-based analysis to detect privacy leakage. Since there is no requirement to deploy and execute application, the performance of static analysis is quite fast. However, static analysis can be easily bypassed by using dynamic payload technique like code encryption, dynamic loading, or reflection technique. There are many applicants for static analysis including Androguard, AndroSimilar, APKInspector, Drozer (also known as Mercury).

*2) Dynamic Approach:* In dynamic analysis, an application is installed and run within a customized sandbox. In order to collect behaviors of an application, the sandbox or android framework running inside sandbox has to be modified. In order to interact with the applications, tools that generate events have been used. One of the example for generating random events are monkey tool that is provided together with the Android SDK. There are various methods that applied to the sandbox to check for malicious application, but they are useless if the application refuses to execute all of its code. Because of that reason, the most challenge point in deploying a dynamic analysis sandbox is to prevent an application from performing Anti-Emulation techniques. Because an application needs to be run and inspected inside the sandbox, dynamic analysis is a trade-off between performance and efficiency. The representatives for dynamic analysis are Andromaly, Bouncer, TaintDroid, Droidbox and various of research topics about dynamic sandbox [5], [11]–[15].

*3) Hybrid Approach:* Hybrid approaches take advantage of both static and dynamic analysis. [1] has proposed a hybrid approach for Android malware analysis where both static analysis and dynamic analysis are performed and outputs are analyzed to check for suspicion behaviors. Although there are not many representatives for the hybrid approach, this idea could be a new direction for anti-malware researchers.

### B. Anti-Emulation Methods

Anti-Emulation methods are used by cybercriminals to check for the execution environment and to avoid being scanned by the sandbox. This paper divides Anti-Emulation techniques into two type of methods: 1) Sandbox Evasion and 2) Sandbox Detection. The following subsections describe detail about each approach.

Sandbox evasion technique is the method of hiding a part of source code until one or more conditions are met [16] [17]. An evasion technique that requires human interactions can be solved by tools that generate random events like Monkey tool [18]. It is note that not all applications that use evasion technique are malicious, some applications that related to financial or banking environment usually use evasion technique to avoid being run on the rooted device. Some applications that contain Easter egg, which is a hidden message or feature, also use evasion technique for hidden features. On the other side, malicious applications also depend on the evasion technique to hide their malicious code. Since both benign and malicious applications sometimes use the same evasion techniques, it is difficult for the sandbox to determine between the benign and malicious application. There are various type of evasion technique. One of the simplest methods is to configure the execute time. In the time configuration method, a malicious code will be executed whenever a certain time condition is met. The time configuration technique is effective for sandboxes that only spend fix amount of time to do analysis. The other evasion technique is human behaviour configuration. In human behaviour configuration, a specific code will be execute only if a specific human action is detected, for example: touch or scroll onto the screen. Sandbox evasion technique can be solved by simply satisfying the condition given by application. For example, a time configuration method can be circumvented by updating the sleep duration using repackaging technique or manipulating the clock of sandbox. Repackaging technique allows sandbox to decode and make modification before rebuild the source code. An evasion technique that requires human interactions can be solved by tools that generate random events like Monkey tool [18]

A more aggressive use of evasion techniques is actively detection of analysis sandbox. Sandbox detection techniques are based on the fact that sandbox is not a real mobile device [4], [19]. This paper calls the information that used to distinguish between a sandbox and real device as sensitive information. A sandbox is made by various system layers including: 1) Application layer and 2) Virtualization layer. Because of that reason, there are various type of sensitive information that can be achieved through those sandbox layers. In application layer, users (both malicious and benign) can get sensitive information through API call provided by Android framework. For example: getDeviceId call from TelephonyManager object return a device registration number of a mobile device, but it is return null in sandbox like Android Virtual Device. In Virtualization layer, sensitive information are information that related to different of network information, process difference, or caching.

## III. Proposed Model

This section describes the hybrid approach as a work-around solution for sandbox detection technique. The word *hybrid* means a combination between sandbox and mobile device. The main motivation is to separate between sandbox and sensitive information source. Figure 1 illustrates the design for our approach.

The proposed approach creates an interceptor module called Sensitive Information Interceptor (SI Interceptor) that

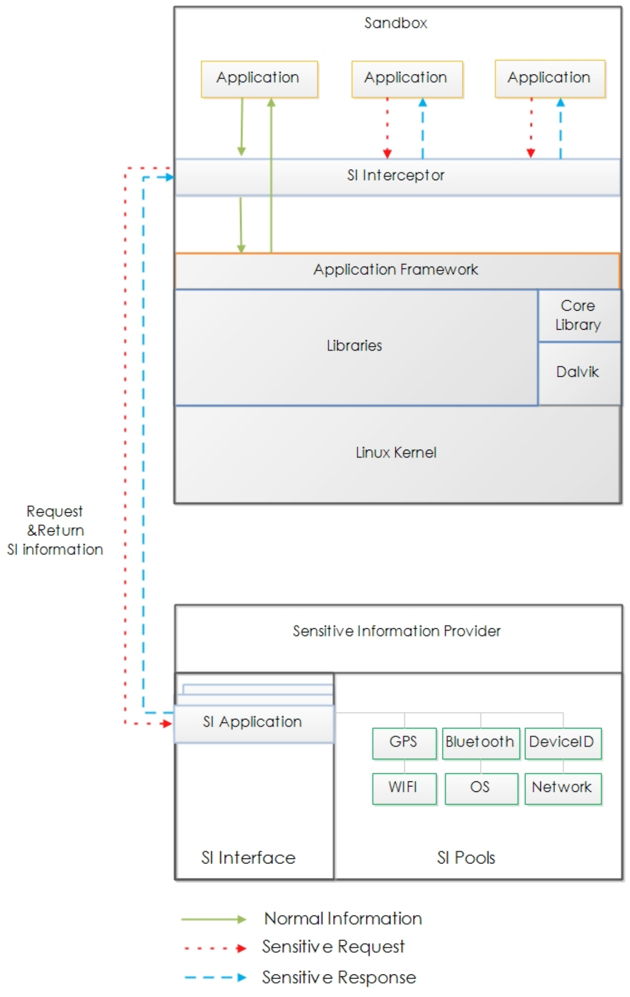


Figure 1. Hybrid Sandbox Architecture

stays between application and application framework. The SI Interceptor will check all request for information of sandbox and intercept sensitive requests. Sensitive request is a request that is expected to return a sensitive data. The sensitive request is forwarded to Sensitive Information Provider (SIP). Upon receiving the request to communicate, the SIP create an SI Application to handle the request. After get the request information, an SI Application processes and queries to the Sensitive Information Pools (SI Pools) for the related sensitive information. After getting the sensitive information, the SI Application send sensitive response to SI Interceptor that will return the data back to the requested application.

There are some points that must be considered for this architecture: 1) Feasibility, 2) Performance, and 3) Security. Because new interceptor module is added into the sandbox, it is obvious that the performance will be reduced. There are some factors that are related to performance problems including the cost of interception, network, and SI request. The SI request cost is depending on the difference between SIP side and sandbox. However, since the SIP server can provide information for many applications as well as sandboxes at the same time, SIP server can apply caching technique to some or all sensitive information. Network cost can be reduced by setup the sandbox and SIP server in the same gateway. The interception cost is depending on the intercept method that is applied to the system.

The security also needs to be considered. Since the sensitive request will be sent out to the network, it should be protected in a way that it can not be manipulated by Android applications. However, in case if the request for network information is considered as sensitive, the Android application will receive information from SIP server, which will not exposed the sandbox network information. The second security consideration is the security of SIP server. If SIP server is not a server that collect mobile information but a mobile device, it could be harm by the malicious sensitive request. In this case, the SI Interceptor should wisely decide which request could be considered as sensitive information.

TABLE I. LIST OF HYBRID API

| Type | API name | Return Type |
|---|---|---|
| TelephonyManager | getDeviceId | String |
| TelephonyManager | getLine1Number | String |
| TelephonyManager | getSubscriberId | String |
| TelephonyManager | getSimCountryIso | String |
| TelephonyManager | getNetworkCountryIso | String |
| TelephonyManager | getSimSerialNumber | String |
| TelephonyManager | getSimState | Integer |
| TelephonyManager | getNetworkType | Integer |
| LocationManager | getLastKnownLocation | Location |
| ConnectivityManager | getNetworkInfo | NetworkInfo |

## IV. IMPLEMENTATION

This paper analyses the effects of hybrid architecture on 28 malware samples that are known to include anti emulator techniques that check for sensitive information. The samples are provided by VirusShare. We decided to use only a small samples set since the main purpose of this implementation is to demonstrate the possibility of our design. Furthermore, since the approach is only at the prototype stage, the API covered by this implementation is also limited. The SIP server is a mobile device with Universal Subscriber Identity Module (USIM) setup. The authors have chosen 3 android API packages that are usually used by malware to check the sensitive information. These packages are 1) android.telephony.TelephonyManager, 2) android.location.LocationManager, and 3) android.net.ConnectivityManager. List of hybrid API is shown in the Table I.

Table II shows the result between Log API and Hybrid API when executing all apps in android VM. Log API means that the authors only log the API call and do nothing with the result. Some apps need to run with UI tool to simulate user behaviours. The result shows hybrid solution could reveal more information called by the malware in some cases.

### A. *Performance Problem*

In the second implementation, the author chose one method to be intercepted is "getDeviceId" that class by the object of TelephonyManager class. This function will return the device ID of a mobile device. In case of the sandbox environment, the "getDeviceId" method will return a null value or 00000000000000 since the value is fixed before sandbox

TABLE II. LOG API AND HYBRID API

| Number | Process name | Log Only | Hybrid |
|---|---|---|---|
| 1 | nang.dv (with UI tool) | 8 | 8 |
| 2 | com.googleapi.cover | 1 | 1 |
| 3 | com.software.application | 1 | 1 |
| 4 | com.hou.jokescreen | 7 | 9 |
| 5 | com.mobi.screensaver.fzllove1 | 1 | 1 |
| 6 | com.liuwei.XiaopinClub | 17 | 33 |
| 7 | net.xfok.info.liujialing | 2 | 4 |
| 8 | com.readnovel.book_32415 | 6 | 11 |
| 9 | com.xiaoyangrenworkroom.facerecognize | 15 | 25 |
| 10 | ru.android.apps | 3 | 3 |
| 11 | Jk7H.PwcD | 5 | 5 |
| 12 | com.bratolubzet.stlstart | 2 | 2 |
| 13 | ngjvnpslnp.iplhmk | 8 | 8 |
| 14 | com.zhuaz.bugaishipdq | 0 | 0 |
| 15 | com.soft.install | 1 | 1 |
| 16 | install.app (with UI tool) | 0 | 7 |
| 17 | com.android.mmreader739 | 5 | 5 |
| 18 | com.googleapi | 2 | 2 |
| 19 | com.hdc.bookmark1566 | 1 | 2 |
| 20 | com.outfit7.talkinggingerfree | 9 | 10 |
| 21 | com.sinosoft.duanxinwzw | 39 | 62 |
| 22 | com.android.system | 0 | 6 |
| 23 | com.tencent.token | 12 | 12 |
| 24 | com.baoyi.meijiaba | 19 | 43 |
| 25 | com.unitepower.mcd33305 | 6 | 16 |
| 26 | azbc88881.jingdian10 | 6 | 25 |
| 27 | com.android.kmax.tie | 13 | 22 |
| 28 | com.androidbox.lz3net2 | 1 | 1 |

TABLE III. 1ST TIME REQUEST BETWEEN HYBRID AND NO HYBRID

| Number of 1st Times Request | Hybrid | No Hybrid |
|---|---|---|
| 1 | 210 | 0 |
| 2 | 205 | 0 |
| 3 | 403 | 1 |
| 4 | 31 | 0 |
| 5 | 149 | 0 |
| 6 | 103 | 0 |
| 7 | 79 | 0 |
| 8 | 96 | 1 |
| 9 | 130 | 0 |
| 10 | 104 | 0 |

is built. On the other hand, if the method is called by an application in a mobile phone, the return value is the device ID of that mobile. This demo involves the mobile phone as SIP server and a Virtual Machine (VM) run Android OS. Both VM and mobile phone connect to the same gateway. Also, a simple application is installed inside Android OS.

After the implementation, with 20 requests sent to the SIP server, the average time is only 7 milliseconds with caching from SIP server. The application shows a very slow response from the first request. Table III shows the cost (in milliseconds) for each 1st time request (by clearing the cache of SIP server for each try).

It is easy to notice that only the first request cost much performance, about 151 milliseconds for each request. Because of that reason, system with distributed SIP applications may reduce the average time more closely to the performance of non-intercepted sandbox.

Based on the implementation result, the effectiveness for one application with one method in proposed model could be calculated as follows:

$$\delta = \begin{cases} T_I + T_N + T_R & \text{if } 1^{st} \text{ request} \\ T_I + T_N + T_C & \text{if not } 1^{st} \text{ request} \end{cases}$$

Where $\delta$ is the average performance per request. $T_I$, $T_N$, $T_R$, and $T_C$ are the performance cost for interception, networking, request of SI information (in the SIP server), and cost for getting information from cached.

And the effectiveness for one application with n methods will be calculated as follows:

$$\delta = \frac{T(1)_I + T(1)_N + T(1)_R + \sum_{i=2}^{n} (T(i)_I + T(i)_N + T(i)_C)}{n}$$

At last, the effective of m applications with n methods is:

$$\delta = \frac{T(1)_I + T(1)_N + T(1)_R + m * \sum_{i=2}^{n} (T(i)_I + T(i)_N + T(i)_C)}{n * m}$$

As m goes larger, the SI request time will get smaller. In this case, the performance of proposed approach will depend on the time cost for intercept a method and cost for request transmit to the network. In a LAN network (SIP and sandbox have same gateway), the cost of request transmit could be very small. In an idea condition, the different between performance of proposed approach and non-intercepted approach is only depending on the interception algorithm.

## B. Pre-initialize Solution for SIP server

The main problem of proposed approach is that it takes the first sensitive request a long time to response. Since an application may usually call a method for one time only during its life-cycle, the performance will be very slow if the sensitive result have not been cached. In order to solve the problem, the SIP server could run pre-initiate function that caches common and high frequency sensitive methods before establishing communication channel with any SI interceptor. By doing the pre-initiate method, the effective of common sensitive methods could be improved into:

$$\delta = \frac{\sum_{i=1}^{n} (T(i)_I + T(i)_N + T(i)_C)}{n}$$

## C. Discussions Of The Approach

The solution can be applied as an additional module for supporting dynamic analysis. The experiments focus on feasibility and performance overhead of the method before further development. Since it is only at the prototype stage, a small number of dataset were applied. According to the performance test result, there is a delay for the first request of sensitive information in which a malicious app can use as a fingerprint for sandbox detection. However, for the second time or if there is another app already request the same information, the delay is the same as provided by existing instrumentation method.

## V. Summary and Future Work

The existing solutions for manipulating sensitive information are through instrumentations and modification of Android source code. However, those existing approaches often provide fixed manipulation scenario with illogical information. Because of the above problem, this paper proposed an instrumentation-based approach for a sandbox to improve quality of sensitive information. Basically, the proposed model provides an intercept-based module for handling the request for sensitive information and forward to a remote Sensitive Information Provider (SIP) server. The SIP server has the responsibility to process and returns the value that is similar to a sensitive information of the mobile device. The performance result shows a close to non-intercepted from the second request of the same method. In the future, more research will be done in order to provide depth analysis of security problems and to optimize the architecture.

## Acknowledgment

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2018-2017-0-01633) supervised by the IITP(Institute for Information & communications Technology Promotion) and Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No.2016-0-00078, Cloud based Security Intelligence Technology Development for the Customized Security Service Provisioning)

## References

[1] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M. S. Gaur, M. Conti, and et.al., “Android security: a survey of issues, malware penetration, and defenses,” IEEE communications surveys & tutorials, vol. 17, no. 2, 2015, pp. 998–1022

[2] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, and et.al., “Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps,” Acm Sigplan Notices, vol. 49, no. 6, 2014, pp. 259–269.

[3] P. Faruki, V. Ganmoor, V. Laxmi, M. S. Gaur, and A. Bharmal, “Androsimilar: robust statistical feature signature for android malware detection,” in Proceedings of the 6th International Conference on Security of Information and Networks. ACM, 2013, pp. 152–159.

[4] T. Petsas, G. Voyatzis, E. Athanasopoulos, M. Polychronakis, and S. Ioannidis, “Rage against the virtual machine: hindering dynamic analysis of android malware,” in Proceedings of the Seventh European Workshop on System Security. ACM, 2014, p. 5.

[5] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, “andromaly: a behavioral malware detection framework for android devices,” Journal of Intelligent Information Systems, vol. 38, no. 1, 2012, pp. 161–190.

[6] Y. Feng, S. Anand, I. Dillig, and A. Aiken, “Apposcopy: Semantics-based detection of android malware through static analysis,” in Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2014, pp. 576–587.

[7] G. K. Chin Erika, Felt Adrienne Porter and W. David, “Analyzing inter-application communication in android,” in Proceedings of the 9th international conference on Mobile systems, applications, and services. ACM, 2011, pp. 239–252.

[8] B. Sanz, I. Santos, C. Laorden, Ugarte-Pedrero, and et.al., “Puma: Permission usage to detect malware in android,” in International Joint Conference CISIS’12-ICEUTE 12-SOCO 12 Special Sessions. Springer, 2013, pp. 289–298.

[9] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, “Hey, you, get off of my market: detecting malicious apps in official and alternative android markets.” in NDSS, vol. 25, no. 4, 2012, pp. 50–52.

[10] J. Kim, Y. Yoon, K. Yi, J. Shin, and S. Center, “Scandal: Static analyzer for detecting privacy leaks in android applications,” MoST, vol. 12, 2012.

[11] A. Reina, A. Fattori, and L. Cavallaro, “A system call-centric analysis and stimulation technique to automatically reconstruct android malware behaviors,” EuroSec, April, 2013.

[12] D. Damopoulos, G. Kambourakis, and G. Portokalidis, “The best of both worlds: a framework for the synergistic operation of host and cloud anomaly-based ids for smartphones,” in Proceedings of the Seventh European Workshop on System Security. ACM, 2014, p. 6.

[13] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, and et.al., “Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones,” ACM Transactions on Computer Systems (TOCS), vol. 32, no. 2, 2014, p. 5.

[14] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, “Crowdroid: behavior-based malware detection system for android,” in Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices. ACM, 2011, pp. 15–26.

[15] J. Huang, X. Zhang, L. Tan, P. Wang, and B. Liang, “Asdroid: Detecting stealthy behaviors in android applications by user interface and program behavior contradiction,” in Proceedings of the 36th International Conference on Software Engineering. ACM, 2014, pp. 1036–1046.

[16] J. Oberheide and C. Miller, “Dissecting the android bouncer,” SummerCon2012, New York, 2012.

[17] T. Vidas and N. Christin, “Evading android runtime analysis via sandbox detection,” in Proceedings of the 9th ACM symposium on Information, computer and communications security. ACM, 2014, pp. 447–458.

[18] Google, “Ui/application exerciser monkey. online: http://developer.android.com/tools/help/uiautomator/index.html.”

[19] D. Maier, T. Müller, and M. Protsenko, “Divide-and-conquer: Why android malware cannot be stopped,” in Availability, Reliability and Security (ARES), 2014 Ninth International Conference on. IEEE, 2014, pp. 30–39.

# A New Approach of Network Simulation for Data Generation in Evaluating Security Products

Pierre-Marie Bajan

University of Paris-Saclay
and IRT SystemX
France
Email: {first.last}@irt-systemx.fr

Christophe Kiennert
and Herve Debar

Telecom SudParis
France
Email: {first.last}@telecom-sudparis.eu

***Abstract*—Evaluating a security product requires the ability to conduct tests to assert that the product reacts as expected, both in terms of scalability and semantics. However, the production of evaluation data at a large scale with a high semantic is very costly with current methods. Load tests are semantically poor and semantic tests require a testbed environment to be deployed at a large scale. Evaluation data from real world activity need to be anonymized and a compromise must be made between the request of the evaluator and the interest of the real world organization. Moreover, to evaluate the full scope of a security product, the evaluator needs multiple test methods. In this paper, we describe a new methodology to produce evaluation data with a customizable level of realism and the possibility to be deployed at a large scale with lower resource requirements for a network support than a testbed environment. Our prototype relies on this method to generate realistic activity for up to 250 simulated users interacting with a real-world webmail server.**

***Keywords–cybersecurity; simulation; evaluation.***

## I. Introduction

Security products can be defined as all services and products designed to protect a service, machine or network against attacks. Like other products, they must be tested to guarantee adherence to specifications. Tests can be divided into two categories: *semantic tests* – tests of capability that require data with a high-level of semantic; and *load tests* – tests that subject the product to a large amount of data.

With current testing methods [1], *load tests* are semantically poor, thus not realistic. Meanwhile, *semantic tests* either require vast amount of resources to reach large scales (e.g., testbed environments), or rely on real life captures with their own set of challenges (e.g., elaboration of the ground truth and privacy concerns). Moreover, a complete evaluation of a security product tests several properties of the product and the evaluator needs to select different methods with the right granularities. The granularity of interactions of the data corresponds to the level of control or precision of the data. For an evaluator, the right granularity for a testing method is a granularity that is fine enough to test specific vulnerabilities or properties. Rather than relying on several methods with different granularities, we aim to elaborate a method to produce data with a customizable granularity and the possibility to achieve large scale generation with appropriate semantic.

In this paper, we present a methodology to produce simulated evaluation data with different granularities independently of the network support. To achieve variable granularity of our model, we formally define two concepts. First, a *Data reproducing function* represents the level of realism of the simulation (the property of the data to reproduce) and decides the level of control over the data. Second, *elementary actions* correspond to the most atomic actions the evaluator can simulate, the basis upon which the experimental scenario will be built. We also develop a prototype of our methodology and validate our approach with a series of experiments.

The remainder of this paper is organized as follows. Section II reviews the related work on the production of evaluation data and their limits. Section III defines the concepts of our methodology and uses those concepts to introduce our model. Section IV explains the different choices we made for the implementation of our prototype and shows the experiment results to validate our model. Finally, we conclude our work in Section V.

## II. Related work

### A. Semantic tests

Semantic tests generate evaluation data with high semantic value. Their goal is to generate realistic workloads to produce real-life reactions of the security product or to test specific functionalities and vulnerabilities of the product. One of the approaches to obtain data with the highest semantics is to use real world data, which can come from several sources: provided by organizations doing real world productions, or obtained from honeypots where actors from the real world were tricked into interacting with a recording system to learn about the current trends (ex: generation of intrusion detection signatures using a honeypot [2]).

However, the evaluator does not have a complete knowledge of the content of the data. Some of it can be misidentified or the intent behind some actions misinterpreted. Moreover, real world data are difficult to obtain. Organizations are reluctant to provide data that can damage their activity, and data anonymization has the drawback of deleting relevant information (e.g., challenges of anonymization [3] and desanonymization techniques [4]). As for honeypots, the evaluator can never know beforehand how many data he can obtain or what kind of data he will gather.

Another way to obtain high semantic data is to generate them according to a defined scenario, relying on tools and scripts to produce specific and calibrated data. Those scripts can be homegrown scripts, exploits, or software testing scripts that try every function of a software to validate its specifications. Manually generating the data (e.g., video transcoding [5], file copy operations [6], compiling the Linux kernel [7], etc.) offers the greatest control over the interactions inside the data, but the automation of the activity generated through scripts with tools like exploit databases (Metasploit [8], Nikto [9], w3af [10], Nessus [11]) also offers good control.

However, those methods are quite time-consuming or require in-depth knowledge of the evaluated product. Moreover, the available tools are not necessarily appropriate for customized generation.

## B. Load tests

Load tests create stress on the tested product [12]. The most common tests use workload drivers like SPEC CPU2000 [7], ApacheBench [13] [14], iozone [14], LMBench [15] [7], etc. They produce a customizable workload with a specific intensity. The evaluator can also manually start tasks or processes known to stimulate particular resources (e.g., kernel compilation [13] [15], files download [16], or execution of Linux commands [16]). Those methods are designed to test particular resources of a system (like I/O, CPU and memory consumption) or produce large amount of workload of a specific protocol. For example, SPEC CPU 2017 generates CPU-intensive workloads while ApacheBench generates intensive HTTP workloads. However, the semantics of the workloads are low: the generated data are characteristic of the driver used and do not closely resemble real life data.

## C. Deployment of semantic tests at a large scale

Evaluators prefer tests that are both intensive and with a high semantic, as the performance of security products like intrusion detection systems often deteriorate at high levels of activity [17]. Methods for semantic tests are deployed on a large scale network support like a testbed environment where a large amount of resources and contributors are gathered to create a large-scale test. Evaluators must either have access to a testbed environment with enough resources to deploy large-scale experiments or use the results of other organizations that conducted large-scale experiments and made their data publicly available for the scientific community (DARPA/KDD-99 [18], CAIDA [19], DEFCON [20], MawiLab [21], etc.).

However, publicly available datasets, on top of often containing errors [1], are not designed for the specific needs of each evaluator. The evaluator needs to have an in-depth knowledge of the characteristics of the activities recorded in the dataset to avoid having an incorrect interpretation of the results of studies using those datasets. Finally, those large-scale experiments produce one-time datasets that can be quickly outdated.

## III. Our evaluation data production method

Our goal is to generate evaluation data at a large scale with a customizable level of realism and semantic richness. The main weakness of large scale semantic test's method lies in the testbed environment. A testbed environment requires large resources and a lot of contributors to set up, use, maintain and return to a previous state. The virtual machines used to support the data generation methods are costly and the light virtual machines are currently too limited for the requirements of semantic methods. We propose a new production method that generates controlled activity data from short traces independently of the network support. It can be implemented on a testbed environment or on a network support with lower requirements like a lower end network simulator.

We also want our method to meet the need of evaluators to generate tests with a rich variety (different systems, properties of the data, etc.) and to devise hybrid tests, both semantic and load oriented. In our methodology, the simulated data is not produced by the execution of activity functions of the host system but by a generic *Data generating function* that represents a level of realism required of the simulated data. A simulated activity is comprised of a single *Data generating function* that is provided with a set of *Model data* extracted from the execution of specific *Elementary actions* and used to create a *Script* of the activity. In the following section, we define concepts on which we build a formal description of our methodology.

## A. Concepts and definitions

*1) Elementary action:* In our methodology, we distinguish real activity – not issued from our simulation method – ($\mathcal{R}$) and simulated activity – issued from our simulation method – ($\mathcal{S}$) in *Elementary actions*. We call *Elementary action* ($A$) a short ordered set of interactions that represents an action between two actors of the activity. Those actors are a *Host* – a source of generated data – or a *Service* – a set of functionalities available to a *Host*. A *Service* can be an external server or an internal service.

For each *Elementary action*, we acquire *Model data* that are the captured data of the execution of this *Elementary action* during real activity. The goal of *Elementary actions* is to divide the activity we simulate in individual actions that correspond to an entry of the ground truth, such as "connection to the web interface of a webmail server". The ground truth is an exact representation of the activity generated. So a finer set of *Elementary actions* for an activity means a finer representation of the simulated activity and a finer control of the activity model for the evaluator.

*Model data* take different forms (traces, logs, values, etc.), and to label the ground truth, *Model data* are classified by the evaluator. The evaluator can use that classification to label the resulting *Simulation data*. For example, the evaluator can create two classes of *Model data* to represent malicious activity and benign activity, respectively. In other contexts, such as generating activity for the evaluation of administration tools of a network, the actors to consider are different (security: attacker/user, administration: admin/user/client) and the evaluator will have to define classes of data accordingly.

After capturing *Model data* for every *Elementary action* relevant for the evaluation, the resulting set of *Model data* is then given to a *Data generating function*.

*2) Data generating function:* A *Data generating function* ($f$) creates *Simulation data* from *Model data*. *Simulation data* ($d^{simulation}$) is created from the execution by a *Host* ($H$) of an *Elementary action* ($A$) during a simulated activity. It is the output of a *Data generating function* and we express our demands for *Simulation data* as *Equivalence*.

We call *Equivalence* ($\sim$) the fact that two activity data have the same properties.

$$d_A^{activity} \sim d'^{\,activity}_A$$
$$\Longleftrightarrow$$
$$Properties(d_A^{activity}) = Properties(d'^{\,activity}_A)$$

The properties of the data are of different forms: acknowledgement of the data by the *Service*, size of sent packets, value of a measure, etc. and they represent the level of realism chosen by the evaluator. The evaluator selects a set of *Elementary actions* to decide the finesse of control over the simulation and he chooses a *Data generating function* to reproduce the properties of the data he requires. If the *Data generating function* that produces *Simulation data* from a dataset of *Model data* cannot produce data with the same properties, it is useless for the evaluator. Thus, we define the following verification property of *Data generating functions*:

*Property 1:* a *Data generating function* $f$ is said to be **useful to a set of *Model data*** $\mathcal{D}$ if all *Simulation data* generated by $f$ from any *Model data* that belong to $\mathcal{D}$ is equivalent to the data used as model.

$$\forall\ d \in \mathcal{D}\ and\ f\ /f(d) = d^{simulation}$$
$$\Rightarrow f\ is\ useful\ to\ \mathcal{D},\ if\ \forall d \in \mathcal{D}, d \sim d^{simulation}$$

The evaluator can select the *Data generating function* that is useful to his *Model data* with *Simulation parameters* ($p^{simulation}$) and provide the *Data generating function* with additional parameters called *Elementary action parameters* ($p^A$). *Elementary action parameters* allow the evaluator to modify the behavior of the *Data generating function* to match a larger dataset of *Model data* (e.g. services accepting the same credentials but with different identifiers, such as ”id=” and ”_id=”) or provide a finer control (e.g. possibility to change the credentials in the submit form).

*Data generating functions* are selected with *Simulation parameters* by the evaluator for the properties they preserve and the evaluator adapts or controls the *Simulation data* with *Elementary action parameters*. The data exchanged by the program that controls the simulation and the *Host* that runs a *Data generating function* is the *Control data* ($d^{control}$) and is essentially the ground truth of the simulation. The compilation of the *Control data* informs us of all the actions taken during the simulated activity.

*3) Scenario and Scripts:* We finally define a *Script*, which is the representation of a realistic behavior of a *Host*. A *Script* is an ordered set of actions coupled with *Elementary action parameters*. These actions can be *Elementary actions* or actions that do not generate activity data (e.g. ”wait X seconds”). A *Script* ($Script_H$) is defined for each individual *Host* and describes the activity it must generate during the simulation. The set of defined *Scripts* is called the *Scenario* ($Sce$) of the simulation.

A *Script* can be represented as a graph of actions, as illustrated in Figure1.

*4) Our model:* Figure 2 is a representation of our model. In that figure, the evaluator provides the simulation control program with the *Simulation parameters* ($p^{simulation}$) and the *Scenario* ($Sce$):

$$Sce = \{Script_{H_0}, Script_{H_1}\} = \{([A, p^A], \dots), ([A, p'^{\,A}], etc.)\}$$

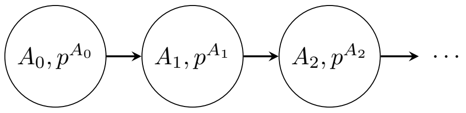


Figure 1. Example of a *Script*

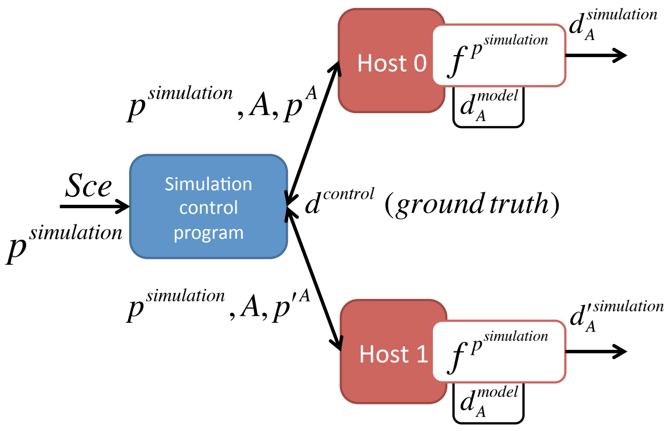


Figure 2. Generation of simulated activity from short traces

The simulation control program interprets the *Scenario* and the *Simulation parameters* and deduces the number of *Hosts* in the current simulation. It instructs the *Hosts* $H_0$ and $H_1$ to reproduce the *Elementary action* ($A$) with the parameters $p^{simulation}$ and $p^A$. Then, each *Host* retrieves the *Model data* associated to the *Elementary action* and executes the *Data generating function* ($f$) selected in the *Simulation parameters*. That function produces *Simulation data*, which is sent to a *Service*. The use of different *Elementary action parameters* by $H_0$ and $H_1$ results in the generation of different *Simulation data* even when the *Data generating function* and the *Model Data* are the same:

$$\left.\begin{array}{l} d_A^{simulation} = f^{p^{simulation}}(d_A^{model}, p^A) \\ d'^{\,simulation}_A = f^{p^{simulation}}(d_A^{model}, p'^{\,A}) \end{array}\right\}$$
$$\not\Rightarrow$$
$$d_A^{simulation} = d'^{\,simulation}_A$$

After the *Hosts* inform the simulation control program that they finished simulating the *Elementary action* $A$, they await the next simulation orders from the simulation control program.

The model we presented is the situation where all the *Hosts* are simulated and the *Services* are real services. If some *Hosts* also acted as *Services*, they could also initiate the generation of *Simulation data* according to requests received from other *Hosts* in the form of other *Simulation data*.

In our model, the ground truth is built upon the *Control data* of *Hosts* simulated by our model. Therefore, no data from *Hosts* unrelated to the simulation are processed.

Lastly, we must present one of the major issues of our model: the parameterization of the *Elementary actions*. Indeed, in accordance with the required level of realism of the simulation, the parameterization must allow the *Data generating function* to preserve various data properties. The higher the level of realism, the more complex the reproduction of *Elementary actions* becomes. Therefore, designing a *Data* *generating function* for a highly realistic simulation, where not only packet size is preserved but also data acknowlegement, requires to consider three main aspects:

- **typing**: identification and generation of short-lived data like tokens, identifiers of session, etc.
- **semantics**: modification of inputs with a high semantic value in the *Model data*: credentials, mail selection, mail content, etc.
- **scalability**: a large scale execution of the *Data generation function* can have consequences on the previous aspects and requires additional changes (e.g., creation of multiple user accounts in the *Service* database).

These three aspects are integrated to the *Elementary action parameters*. However, a few in-depth issues still require further consideration and development in order to elaborate a model able to adapt to various test situations without the intervention of the evaluator. The typing issue can be solved with methods based on machine learning, but others may require specific methodologies according to the context of the evaluation. For example, in the case of real-life network reproduction, the semantic and scalability issues can be solved by identifying and using inputs with a high semantic value recorded during a long *Model data* acquisition period.

## IV. Implementation of the Proposed Method

In this section, we describe the implemention of a prototype that follows the requirements of our model, and we show that simulations based on that prototype are both scalable and realistic.

This prototype uses Mininet [22] as the network support of our simulation. Mininet is an open-source network simulator that deploys lightweight virtual machines to create virtual networks, and able to create hundreds of lightweight virtual machines in a short amount of time.

### A. Model of the prototype

Our prototype contains several *Data generating functions* that conserve each of these properties: execution time, packet size, acknowledgement of the data by the *Service*. Based on these *Data generating functions* we simulate the activity of 50 to 200 *Hosts* representing regular employees of a small company interacting with the *Service* of a webmail server Roundcube on a Postfix mail server. A simulation control program follows the *Script* described in Figure 3 for all the *Hosts* of the simulation. In Figure 3, the *Elementary actions* are in italics while actions that do not generate activity data are in a regular font. The *Host* can simulate two different series of *Elementary actions* after a waiting period of $X$ seconds each time. The intensity of the *Script* can be modulated by modifying the value of $X$.

To make sure that our method improves the existing methods, it must meet these two requirements:

- **Scalability:** ability to generate evaluation data proportionally to the scale of the simulation, up to a few hundreds of hosts.
- **Realism:** closeness of the generated data to the the referential *Model data*.
  Our *Data generating functions* are designed to preserve specific data properties (cf. Property 1), thus matching different levels of realism. The following experiments aim at proving that these properties are still preserved in the context of a *Scenario* with an increasing number of hosts.

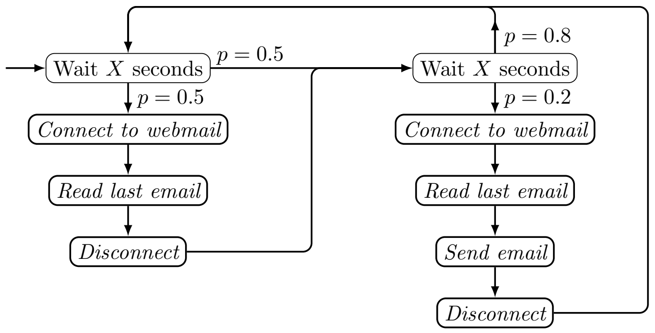


Figure 3. Generation of simulated activity from short traces

### B. Experiments on the prototype

The validation of our prototype is conducted with two separate experiments, which aim to prove that our model leads to both scalable and realistic data generation.

The first experiment is a control experiment. We deploy 5 virtual machines on the network simulator Hynesim [23] and make them generate the activity of our simulation. We script the *Elementary actions* of the *Script* described in Figure 3 with the web driver Selenium [24] and make the virtual machines use their browser to interact with the webmail server. This experiment provides referential values for our second experiment. To prove the scalability of our prototype, we expect proportionality between these values and the results of our simulation, with respect to the number of *Hosts*.

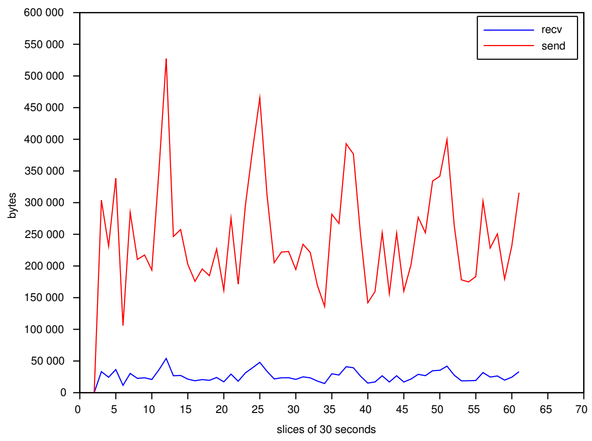


Figure 4. Network traffic of the webmail server for a single experiment (50 *Hosts*)

In the second experiment, we simulate different number of *Hosts* (5, 50, 100, 150, 200 and 250) and make them generate the activity of regular users using a webmail service for 30 minutes. We measure the activity at three different points: the webmail server, the network simulator Mininet and the

TABLE I. NUMBER OF LINES IN THE WEBMAIL LOG FILES.

| | 5 VMs | | 5 Hosts | | 50 Hosts | | 100 Hosts | | 150 Hosts | | 200 Hosts | | 250 Hosts | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Filenames** | **avg** | **stdev** | **avg** | **stdev** | **avg** | **stdev** | **avg** | **stdev** | **avg** | **stdev** | **avg** | **stdev** | **avg** | **stdev** |
| **userlogins** | 90 | 9 | 112 | 10 | 1032 | 36 | 2084 | 45 | 3085 | 52 | 4121 | 53 | 5118 | 74 |
| **imap** | 43245 | 5070 | 57775 | 5306 | 487883 | 22742 | 984642 | 28820 | 1450507 | 27792 | 1933823 | 21117 | 274825 | 235985 |
| **sql** | 4955 | 525 | 6703 | 563 | 56081 | 1886 | 113031 | 2452 | 167138 | 2964 | 223427 | 2906 | 265354 | 4688 |

server hosting the simulation. Every 30 seconds, we measure four parameters: CPU usage, memory usage, network I/O, and disk I/O. Figure 4 is an example of the measured activity. It represents the network traffic received and sent by the webmail server with 50 simulated *Hosts*. Each *Host* follows the *Script* described in Figure 3, with $X = 30$.

We also retrieve the logs produced by the webmail server during both experiments. The quantity and content of the logs is analyzed in Table I and Table II.

In the second experiment, we use the *Data generating function* with the highest level of realism: the adapted replay. That *Data generating function* preserves the data acknowledgement by the *Service* and allows *Elementary action parameters* to modify the inputs of submitted forms. Concretely, it means that a server cannot distinguish the adapted replay from an interaction with a real user. Also, with the help of *Elementary action parameters*, the evaluator can freely change the credentials replayed to the webmail server.

The analysis of the results aims at proving that the data generated in the second experiment are consistent with those obtained from the first experiment, in terms of both quantity and semantics.

Table I represents the quantity of logs produced by the webmail server during both experiments. We express the average number of lines in the log files of the webmail and their standard deviation. The first column is the name of the main log files produced by the server: ”userlogins” logs every connection (successful or not), ”imap” logs every instruction from the server that uses the IMAP protocol, and ”sql” logs every interaction between the server and its database. The entries under the name ”5 VMs” correspond to the results of the control experiment while the other entries are the results of the simulation experiment.

The number of lines in ”userlogins” represents the number of connections during the experiments (one line per connection) and can be used to calculate the number of sessions created during both experiments. Figure 5 shows the average number of sessions created during the second experiment and its standard deviation according to the number of simulated *Hosts*. We also estimate the average number of sessions inferred from the results of the control experiment, based on proportionality ($avg(\text{”5 VMs”}) \times \frac{number\ of\ Hosts}{5}$).

We observe that the number of sessions created during the second experiment is close to our estimation. Our simulation produces more sessions than expected but it can be explained by the fact that our *Data generating function* reproduces the *Model data* of an *Elementary action* faster than the browser of the virtual machines. Hence, in a period of 30 minutes, the simulated activity has gone through more cycles of the *Script* than the control experiment. A projection of the number of lines of the other log files (”imap” and ”sql”) displays similar results.

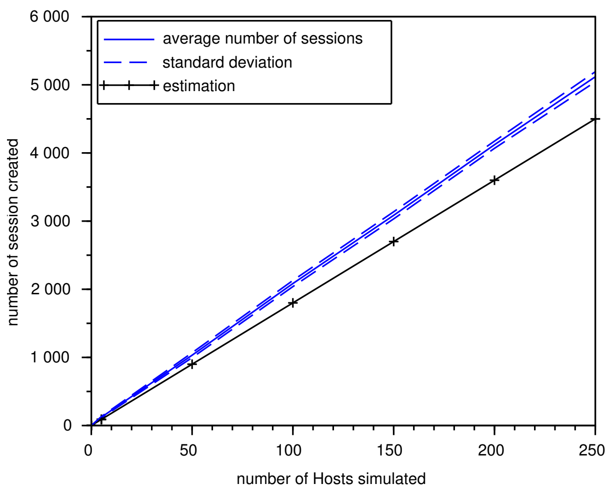


Figure 5. Number of sessions created during simulation (blue) compared to estimation (black)

These results establish that the simulated activity produces a consistent amount of logs. In Figure 6, we examine the network traffic produced by our simulated activity. The blue and red parts represent the average number of bytes, respectively, received and sent by the webmail server every 30 seconds, along with the standard deviation. For comparison, the black lines correspond to the estimation of the expected results based on the control experiment. As before, the results of the second experiment are close to our estimation. The deviation can be justified with the same explanation regarding the activity speed difference. This deviation is also partly due to the cached data. Since these data are stored on the host after the first connection, the amount of exchanged data during the first connection is higher than during subsequent sessions.

However, our *Data generating function* does not take cached data into account. Therefore, our simulated connections request more data from the webmail server than estimated. This observation is part of the parametrization issues of the *Data generating function* raised at the end of Section III. Adding *Elementary action parameters* to modify the behavior of the function can solve this issue as we did for previous typing and semantic issues. However, the addition of new *Elementary action parameters* is made from empiric observation and could be improved by adding new methods to our model like machine learning.

Despite those issues, we have shown that the simulated activity of the second experiment generated a large network activity proportionally to the number of simulated *Hosts*, as expected. We now focus on proving that the activity semantics was also preserved.

TABLE II. SIGNATURE LOG ENTRIES.

| | | 5 VMs | | 5 Hosts | | 50 Hosts | | 100 Hosts | | 150 Hosts | | 200 Hosts | | 250 Hosts | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Signatures** | **Actions** | **avg** | **stdev** | **avg** | **stdev** | **avg** | **stdev** | **avg** | **stdev** | **avg** | **stdev** | **avg** | **stdev** | **avg** | **stdev** |
| **imap.sign1** | connect | 90 | 9 | 122 | 10 | 1032 | 36 | 2079 | 44 | 3075 | 55 | 4118 | 54 | 4874 | 87 |
| **imap.sign2** | connect | 90 | 9 | 122 | 10 | 1032 | 36 | 2079 | 44 | 3075 | 55 | 4118 | 54 | 4874 | 87 |
| **imap.sign3** | connect | 90 | 9 | 122 | 10 | 1032 | 36 | 2079 | 44 | 3075 | 55 | 4118 | 54 | 4874 | 87 |
| **imap.sign4** | connect | 90 | 9 | 122 | 10 | 1032 | 36 | 2079 | 44 | 3075 | 55 | 4118 | 54 | 4874 | 87 |
| **imap.sign5** | connect | 90 | 9 | 122 | 10 | 1032 | 36 | 2079 | 44 | 3075 | 55 | 4118 | 54 | 4874 | 87 |
| **imap.sign6** | connect | 90 | 9 | 122 | 10 | 1032 | 36 | 2079 | 44 | 3075 | 55 | 4118 | 54 | 4874 | 87 |
| **imap.sign7** | connect | 90 | 9 | 122 | 10 | 1032 | 36 | 2079 | 44 | 3075 | 55 | 4118 | 54 | 4874 | 87 |
| **imap.sign8** | connect | 90 | 9 | 122 | 10 | 1032 | 36 | 2079 | 44 | 3075 | 55 | 4118 | 54 | 4874 | 87 |
| **imap.sign9** | connect | 90 | 9 | 122 | 10 | 1032 | 36 | 2079 | 44 | 3075 | 55 | 4118 | 54 | 4873 | 88 |
| **imap.sign10** | connect | 90 | 9 | 122 | 10 | 1032 | 36 | 2079 | 44 | 3075 | 55 | 4118 | 54 | 4873 | 88 |
| **sql.sign1** | connect | 90 | 9 | 122 | 10 | 1032 | 36 | 2079 | 44 | 3075 | 55 | 4118 | 54 | 4874 | 87 |
| **sql.sign2** | disconnect | 90 | 9 | 122 | 10 | 1032 | 36 | 2079 | 44 | 3085 | 52 | 4121 | 53 | 5118 | 74 |
| **imap.sign11** | open | 89 | 9 | 122 | 10 | 1028 | 36 | 2069 | 44 | 3059 | 52 | 4090 | 53 | 4808 | 91 |

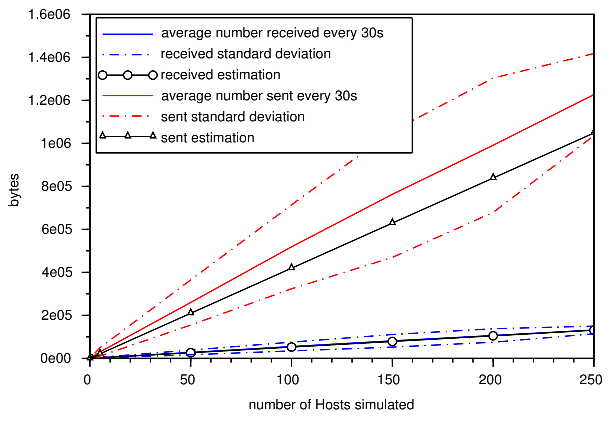


Figure 6. Network traffic of the webmail server

For each *Elementary action* of the activity *Script*, we look for log entries that could act as signatures for the action. By comparing these signatures in both experiments, we obtain the results displayed in Table II.

From Table II, the following observations can be made:

- the number of signatures for the ”connect” *Elementary action* is slightly inferior to the number of sessions (the number of lines from ”userlogins”) observed for 150 *Hosts* and above. It is explained by the fact that the signatures correspond to the number of successful connections to the webmail server. If we remove the number of lines in the ”userlogins” file that correspond to failed connections, we find the exact number of signatures for the ”connect” *Elementary action*.
- the number of signature for the ”disconnect” *Elementary action* corresponds to the exact number of sessions observed in Table I.
- the number of signatures for the ”open” *Elementary action* is slightly inferior to the number of signatures for the ”connect” *Elementary action* for 50 *Hosts* and above. It is likely due to the experiment ending before the last *Script* cycle ended for a few *Hosts*.
- no characteristic entry for the ”send an email” *Elementary action* could be found in the ”userlogins”, ”imap” and ”sql” log files.

The failure of several connections in our simulation may also be due to the parameterization of the *Data generating function*. The adapted replay *Data generating function* was designed to modify short-lived information from the *Model data* like the token or the session identifier according to the server reply from the requests. However, such modification was not included in the first request. The webmail server possibly refused some connections because they contained the same information at the same time. Therefore, an improvement of the typing of the adapted replay *Data generating function* should raise the number of successful connections with a high number of simulated *Hosts*. Table II shows that for each successful session in our simulated activity, the webmail server correctly interpreted the *Elementary actions*.

To sum up the results analysis, our prototype generates a simulated activity that produces a realistic amount of network traffic and logs from the webmail server. Moreover, the webmail server produces the appropriate number of logs reflecting the correct semantics. Therefore, our prototype succeeds in providing scalable and realistic data generation, thus validating our model.

## V. CONCLUSION

In this paper, we establish a new methodology to generate realistic evaluation data on a network support (Mininet) with far fewer requirements than the common network testbeds. This methodology takes into consideration the need for an evaluator to test different properties and evaluate different vulnerabilities in a security product. Therefore, an evaluator can select the *Data generation function* that matches the properties of the product that need to be tested. The evaluator also has a control on the granularity of the activity *Elementary actions*. The finesse of the simulated activity can be improved by introducing new *Elementary actions* or adding *Elementary action parameters* to the *Data generation function*.

We validate our model with a prototype able to generate realistic activity up to 250 users interacting with a webmail server. The traffic can be customized in terms of *Hosts* numbers as well as *Scripts* content. Therefore, it will be possible to use this prototype to develop more complex activity scenarios dedicated to the evaluation of security products such as intrusion detection systems.

However, our prototype still has a few limitations. The existing *Data generating functions* mostly focus on the creation of network activity and does not generate system activity for host-based security products. The parametrization for more realistic *Data generating functions* also raises additional issues that need to be addressed with further work. Finally, our prototype is currently limited to the simulation of *Hosts*. In parallel with the testing of network-based intrusion detection systems based on our prototype, the next steps of our work will focus on extending our prototype to include the simulation of *Services* and develop new *Data generating functions* that focus on the generation of system data rather than network data.

## REFERENCES

[1] A. Milenkoski, M. Vieira, S. Kounev, A. Avritzer, and B. D. Payne, "Evaluating computer intrusion detection systems: A survey of common practices," ACM Computing Surveys (CSUR), vol. 48, no. 1, 2015, p. 12.

[2] C. Kreibich and J. Crowcroft, "Honeycomb: creating intrusion detection signatures using honeypots," ACM SIGCOMM computer communication review, vol. 34, no. 1, 2004, pp. 51–56.

[3] V. E. Seeberg and S. Petrovic, "A new classification scheme for anonymization of real data used in ids benchmarking," in Availability, Reliability and Security, 2007. ARES 2007. The Second International Conference on. IEEE, 2007, pp. 385–390.

[4] S. E. Coull et al., "Playing devil's advocate: Inferring sensitive information from anonymized network traces." in NDSS, vol. 7, 2007, pp. 35–47.

[5] A. Srivastava, K. Singh, and J. Giffin, "Secure observation of kernel behavior," Georgia Institute of Technology, Tech. Rep., 2008.

[6] F. Lombardi and R. Di Pietro, "Secure virtualization for cloud computing," Journal of Network and Computer Applications, vol. 34, no. 4, 2011, pp. 1113–1122.

[7] J. Reeves, A. Ramaswamy, M. Locasto, S. Bratus, and S. Smith, "Intrusion detection for resource-constrained embedded control systems in the power grid," International Journal of Critical Infrastructure Protection, vol. 5, no. 2, 2012, pp. 74–83.

[8] K. Nasr, A. Abou-El Kalam, and C. Fraboul, "Performance analysis of wireless intrusion detection systems," in International Conference on Internet and Distributed Computing Systems. Springer, 2012, pp. 238–252.

[9] K. Ma, R. Sun, and A. Abraham, "Toward a lightweight framework for monitoring public clouds," in Computational Aspects of Social Networks (CASoN), 2012 Fourth International Conference on. IEEE, 2012, pp. 361–365.

[10] J.-K. Ke, C.-H. Yang, and T.-N. Ahn, "Using w3af to achieve automated penetration testing by live dvd/live usb," in Proceedings of the 2009 International Conference on Hybrid Information Technology. ACM, 2009, pp. 460–464.

[11] F. Massicotte, M. Couture, Y. Labiche, and L. Briand, "Context-based intrusion detection using snort, nessus and bugtraq databases." in PST, 2005.

[12] N. J. Puketza, K. Zhang, M. Chung, B. Mukherjee, and R. A. Olsson, "A methodology for testing intrusion detection systems," IEEE Transactions on Software Engineering, vol. 22, no. 10, 1996, pp. 719–729.

[13] R. Riley, X. Jiang, and D. Xu, "Guest-transparent prevention of kernel rootkits with vmm-based memory shadowing," in International Workshop on Recent Advances in Intrusion Detection. Springer, 2008, pp. 1–20.

[14] H. Jin et al., "A vmm-based intrusion prevention system in cloud computing environment," The Journal of Supercomputing, vol. 66, no. 3, 2013, pp. 1133–1151.

[15] J. Morris, S. Smalley, and G. Kroah-Hartman, "Linux security modules: General security support for the linux kernel," in USENIX Security Symposium, 2002, pp. 17–31.

[16] M. Laureano, C. Maziero, and E. Jamhour, "Protecting host-based intrusion detectors through virtual machines," Computer Networks, vol. 51, no. 5, 2007, pp. 1275–1283.

[17] P. Mell, V. Hu, R. Lippmann, J. Haines, and M. Zissman, "An overview of issues in testing intrusion detection systems," NIST Interagency, Tech. Rep., 2003.

[18] R. K. Cunningham, R. P. Lippmann, D. J. Fried, S. L. Garfinkel, I. Graf, K. R. Kendall, S. E. Webster, D. Wyschogrod, and M. A. Zissman, "Evaluating intrusion detection systems without attacking your friends: The 1998 darpa intrusion detection evaluation," Massachusetts Inst. of Tech. Lexington Lincoln Lab, Tech. Rep., 1999.

[19] T. V. Phan, N. K. Bao, and M. Park, "Distributed-som: A novel performance bottleneck handler for large-sized software-defined networks under flooding attacks," Journal of Network and Computer Applications, vol. 91, 2017, pp. 14–25.

[20] C. Cowan, S. Arnold, S. Beattie, C. Wright, and J. Viega, "Defcon capture the flag: Defending vulnerable code from intense attack," in DARPA Information Survivability Conference and Exposition, 2003. Proceedings, vol. 1. IEEE, 2003, pp. 120–129.

[21] R. Fontugne, P. Borgnat, P. Abry, and K. Fukuda, "Mawilab: combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking," in Proceedings of the 6th International COnference. ACM, 2010, p. 8.

[22] R. L. S. De Oliveira, A. A. Shinoda, C. M. Schweitzer, and L. R. Prete, "Using mininet for emulation and prototyping software-defined networks," in Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on. IEEE, 2014, pp. 1–6.

[23] "Homepage of Hynesim," 2018, URL: https://www.hynesim.org [accessed: 2018-04-09].

[24] R. A. Razak and F. R. Fahrurazi, "Agile testing with selenium," in Software Engineering (MySEC), 2011 5th Malaysian Conference in. IEEE, 2011, pp. 217–219.