PeekKernelFlows: Peeking into IP flows

Cynthia Wagner
University of Luxembourg
FSTC, Campus Kirchberg
L-1359 Luxembourg, Luxembourg
+352 46 66 44 1
cynthia.wagner@uni.lu

Gérard Wagener
University of Luxembourg - SNT
FSTC, Campus Kirchberg
L-1359 Luxembourg, Luxembourg
+352 46 66 44 1
gerard.wagener@uni.lu

Radu State
University of Luxembourg
FSTC, Campus Kirchberg
L-1359 Luxembourg, Luxembourg
+352 46 66 44 1
radu.state@uni.lu

Alexandre Dulaunoy
SES S.A.
Château de Betzdorf
L-6815 Betzdorf, Luxembourg
+352 71 07 25 1
a@foo.be

Thomas Engel
University of Luxembourg
FSTC, Campus Kirchberg
L-1359 Luxembourg, Luxembourg
+352 46 66 44 1
thomas.engel@uni.lu

ABSTRACT

This paper introduces a new method for getting insights into IP related data flows based on a simple visualization technique that leverages kernel functions defined over spatial and temporal aggregated IP flows. This approach was implemented in a visualization tool called PeekKernelFlows. This tool simplifies the identification of anomalous patterns over a time period. An intuitive adapting image allows network operators to detect attacks. We validated our method on a real use-case scenario, where we inspected traffic of a high-interaction honeypot.

General Terms

[C.2.3] Network Monitoring and Management, [H.3.3] Information search and retrieval, [D.4.6] Security and Protection, [H.5.2] User interfaces.

Keywords

IP flow visualisation; Honeypot monitoring; Machine Learning with kernel methods.

1. INTRODUCTION

Network monitoring is an essential activity for network operators and not only consists by passively supervising activities of users, but also to dig deeper into available monitored data to detect if networks are in good health. Network monitoring per se is confronted to many problems, as the different natures of incidents or the storage of monitored data. Over the last years, network monitoring itself has already been studied in extends in both, academia and industrial domain.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

VizSec '10, September 14, 2010, Ottawa, Ontario, Canada.

Copyright 2010 ACM 978-1-4503-0013-1/10/09…$10.00.

The available data records on the network border in most cases are Netflow[1] records. This represents manageable quantity of data. Most commercially available routers are capable of exporting this information. Compared to full-packet captures, Netflow records can be described as chronological IP traffic sequences representing packet summaries sent between two entities. Handling huge data volumes as ~60 000 flows/second is quite common which requires prompt and immediate analysis because long-lasting offline analysis is impractical. A solution to this constraint is to store condensed forms or to perform near real-time evaluations. These evaluations are mostly complex and require strong interpretation skills of network operators.

Even more complex than normal traffic analysis, is the inspection of high-interaction honeypot traffic [11], because these entities are designed to be under attack and all traffic towards and originated from a honeypot is by default considered suspicious. After having compromised a honeypot, attackers often install customized and protected tools without log information. Hence, network monitoring and especially early pattern recognition in this kind of traffic is crucial. State of the art technologies, like connection throttling or bandwidth limiting, are difficult to set up because real attacks can hardly be predicted. Intrusion detection, like scanning activity detection has also some limits because zero day attacks can be launched against honeypots. Comparing attack patterns of two different honeypots is also difficult due to the amount of noise in traffic. In such cases, early pattern recognition and pattern comparison can be supported by visualisation techniques.

In this paper, we describe a new approach for online and offline processing of aggregated Netflow records and full-packet captures. The objective is to detect attacks and anomalies by referring to temporal and spatial aggregated IP flows, respecting

[1] Netflow records RFC3954, http://tools.ietf.org/html/rfc3954

the CIDR[2] concept. For this purpose we leverage a kernel method evaluation. Our method represents flows in a simple and active representation highlighting anomalies in networks like for instance important network topology changes.

In this work, we use an aggregation-based traffic-profiling tool called Aguri ([1, 7]) for the network-monitoring task. This tool provides flow summaries over time and space and produces different profiles pointing out flow correlations. A fine-tuned configuration of Aguri influences the aggregation granularity. Global aggregation provides an overview of the network topology, however, neglects its evolution. The Aguri profiles are used to perform a kernel calculus. We have designed a new kernel method able to capture differences between the topological and traffic volume changes between spatial and aggregated flow data without referring to a manual profile comparison. The gained differences are then graphically represented by our implemented tool, which refers to adaptive colour gradients.

Our paper is structured as follows: in section 2 we describe the theoretical part of our model. We first present the tool Aguri, then we briefly give background information how our kernel function for the processing and representation of flows works and we describe our new visualisation mechanism. In section 3 we present the implementation of our tool, by explaining its different modules necessary for representing data. In section 4 we present the experimental results. In section 5 we discuss research work related to our topic and in section 6 we present our conclusions with future work.

2. MODEL DESCRIPTION

In the following sections we describe the different parts for our visualisation model, present the necessary tools we referred to for the monitoring part and describe the metrics used for the visualisation task.

2.1 Spatial and Temporal Flow Aggregation

We have leveraged a flow-monitoring tool supporting IPv4 and IPv6 traffic, called Aguri. IP flows are monitored in near real time and then get spatially aggregated. Aggregation is particular useful to give an overview at subnet layer instead of considering each individual flow. Spatial-aggregation is realized with a special process where small flow entities are aggregated into larger prefix based trees. An advantage of this tool is that when performing aggregation, *n* nodes in a tree can be looked up in near real time within O(log *n*) time complexity. From these temporal and spatial aggregations of network traffic, the tool generates four different profiles reflecting summaries for the observed network traffic. The profiles reflect information on both host sides in a tree-like structure. One profile reflects the source addresses, the second one is responsible for the destination addresses, the third profile captures the source protocols and finally one last profile is used for the destination protocols. Temporal and spatial aggregations have a different purpose. Temporal aggregation is more coarse-grained and similar to a summary of profiles, whereas spatial aggregation performs better for real-time monitoring. A traffic profile generated by Aguri can be seen in Figure 1.

[2] CIDR: "Classless-InterDomain Routing" standard scheme for IP-address allocation and packet routing (see RFC1519, http://www.faqs.org/rfcs/rfc1519.html)

```
%!AGURI-1.0
%%StartTime: Tue Dec 01 13:54:12 (2009/12/01  13:54:12
%%EndTime:  Tue Dec 01 13:54:44 (2009/12/01  13:54:44
%AvgRate: 323.40Kbps

[src address] 1293591  (100.00)%
0.0.0.0/5 7351 (0.58%/99.22%)
          10.0.0.0/9     13545  (1.05%/30.79%)
                  10.4.0.13        237599  (18.37%)
              10.91.0.0/24  19625  (1.52%/10.09%)
                  10.91.0.22       100920  (8.57%)
                  10.91.1.4        16664   (1.29%)
       72.0.0.0/5 21618  (1.67%/37.09%)
                  74.125.79.91     202791  (15.68%)
                  74.125.79.93     214301  (16.57%)
                  74.125.79.99     27396  (2.12%)
                  74.125.79.104    13649   (1.06%)
                  83.231.205.49    324379   (25.08%)
                  83.231.205.50    73506  (5.68%)
::/0 10067  (0.78%/0.78%)
%LRU hits: 95.52%  (1790/1874)   reclaimed: 0
```

Figure 1: Aguri profile representation

It shows an extract of an Aguri profile for a source traffic profile, summarizing traffic of 5 seconds. The profile is composed of a four-line header followed by the monitored network traffic in tree-like structure that contains IP addresses, prefix lengths and the accumulation of bytes transferred and the volume compared to its sub tree expressed in percent. The structures of the other profiles are similar.

We have defined a similarity metric that can compare two Aguri profiles in order to detect traffic and structural driven similarities between two profiles. Our metric takes into account the layout of IP addresses as well as the aggregated traffic volume.

2.2 The Kernel Function

Evaluating high dimensional data, as in our case tree-like profiles with IP-related data, can be simplified by referring to Machine Learning techniques [12], like the application of kernel methods. A kernel function K can be defined as a mapping $K: X \times X \rightarrow [0,\infty]$ from a high dimensional input space X to a similarity score $K(x, y) = \Sigma_i \phi_i(x)\phi_i(y) = \phi(x) \bullet \phi(y)$, with $\phi_i(x)$ being a feature function over a data sample x. We define a new tree kernel function that we apply on the profiles generated by Aguri, in order to detect similarities between two profile trees.

$$K(T_1,T_2) = \sum_{i \in N_{T_1}, j \in N_{T_2}} s(a_i,b_j) \times v(a_i,b_j)$$

Equation 1: Kernel function

The kernel function is composed of two different feature functions. The first part is $s(a_i, b_j)$ which is the similarity measure for the topological changes in our network by comparing suffix length of nodes, defined as

$$s(a_i,b_j) = \begin{cases} \frac{2^{suffixlength_j}}{2^{suffixlength_i}} & \text{if } prefix_i \text{ is prefix of } prefix_j \\ \frac{2^{suffixlength_i}}{2^{suffixlength_j}} & \text{if } prefix_i \text{ is prefix of } prefix_j \\ 0 & \text{otherwise} \end{cases}$$

Equation 2: Topology distance measure

The second part of our kernel function is the volume part $v(a_i, b_j)$ that deals with the volume changes at different time intervals by looking at the percentage of traffic in a node, defined as Gaussian kernel (with σ being the width scaling parameter),

$$v(a_i,b_j) = \exp(-\frac{|vol\%_i - vol\%_j|^2}{\sigma^2})$$

Equation 3: Volume in percent measure

Our kernel function defined in Equation 1 takes as input two Aguri trees and returns a numerical value and has the purpose to identify similarities in the network traffic. The higher this value is, the more similar are two Aguri trees. The lower the kernel value is the more dissimilar the input is.

2.3 Visualizing Aguri Trees

In this section we describe the visualisation mechanism for processed Aguri trees by our kernel method.

For this, we refer to the values of our kernel function for Aguri trees, denoted *K*, which are put into a vector *v* that describes the traffic evolution in a sliding window manner. This vector *v* is then mapped to a rectangle that is sequentially put into an image. The rectangle is then filled with a colour derived from the kernel value *K* itself. The colour of the rectangle describes the intensity of the evolution. We define an image as a two-dimensional space having an *x*- and *y*-axis. The discrete time is defined by t = (t_0, t_1, t_2, ..., t_n). The time step denoted α, corresponds to an interval of α seconds where Aguri trees are exported such that from time t_i to t_{i+1}, α seconds have elapsed.

The first rectangle on the top left in our image represents the kernel value *K* for the first time period and is defined by the coordinate (x_0, y_0). For representing the next received kernel value at time step t_1 we have the coordinates (x_i + r, y_j), with r being the rectangle width. When reaching the end of a line, we set a line break by resetting x_i to 0 and incrementing y_i by the rectangle height r.

The freshness Γ of a picture is defined in Equation 4. The size of the data window has an impact on the freshness of the images. A short window means a smaller image and so a fresher image, whereas a larger one means more outdated the visualized traffic is, but at the same time the resulting image is more farsightedly. In our tool the freshness-parameter can be specified by the honeypot-operator. The graphical representation used in our work is somehow similar to a Self Organizing Maps (SOM) [8] with the difference that no learning process and no training set is needed for the network traffic analysis and only serves as representation of fresh traffic.

$$\Gamma = \alpha \cdot width \cdot height$$

Equation 4: Freshness equation

We use the similarities of our Aguri input obtained from the kernel function (defined in Equation 1) and map them to a colour space defined by the Red – Green – Blue model (RGB) [2]. Intuitively the colour 'black' represents the network traffic noise and relevant patterns by more intensive colours.

We are particularly interested to detect whether a given host is scanning other systems or to track dominant and long lasting TCP sessions. A dominant TCP session is a high bandwidth consuming TCP session initiated by ssh brute-force attacks or IRC bouncing. Another interesting insight is the amount of traffic targeting a given host. By focusing on a window of observed kernels we can normalize the kernels between 0 and 1. Then we multiply each K_i-value with 2^{24} aiming to explore the RGB colour space (represented in Equation 5). Additionally, we added a brightness factor denoted B, considering a higher decimal precision of the kernel values. The more an intensity factor denoted I, was added to linearly shift kernel values in the RGB space.

$$k_i' = \frac{k_i \cdot B}{\sum(k_i \cdot B)} \cdot 2^{24} + I$$

Equation 5: Mapping to RGB format

A simplified RGB colour is composed of 3 bytes. Each byte is used to represent the colours red, green and blue respectively and can be obtained by using a logical AND operation with respective bit masks. The lower bits of K_i' are represented by the colour 'blue', the next bits are used for modeling the 'green' colour part and the higher bits are mapped to the colour 'red'.

This means when having high fluctuations, the similarity between two trees is quite low and all bits are low which ends in a black colour. Small similarities are displayed in bluish colours and high similarities in erythroid colours. When all the bits are high (very high similarities) the colour tends to 'white'.

3. IMPLEMENTATION

PeekKernelFlow is the outcome of early prototyping. An overview of PeekKernelFlow is presented in Figure 2.

The current network topology of the honeypot is recovered by the Aguri-tool [1, 7]. In the same time, all network traffic is captured with the tool tcpdump[3]. If no full-packet capture is available, we use nfdump[4] with Netflow records as input for the NetflowToAguri module. The honeypot operator configures the Aguri-tool to periodically export Aguri trees, which are then processed with the AguriProcessor module.

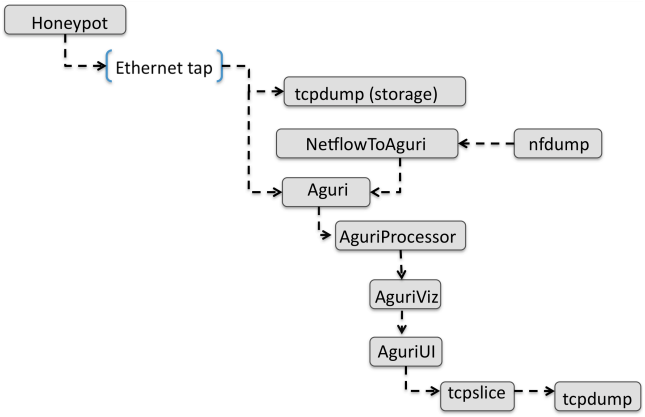

Figure 2: System Architecture

The AguriProcessor computes the kernel value *K* of two successive trees. Then the AguriViz-module reads these kernels and presents them in a two-dimensional space and is responsible for the correct visualisation of the Aguri tree kernel functions. The network operator uses the PeekKernelFlows User Interface, called AguriUI. If an interesting pattern in the picture is observed, the partial network traffic can be extracted with tcpslice which output is piped to the tool tcpdump, which is this time used to transform the peeked network packets into a human readable form.

[3] tcpdump: http://www.tcpdump.org/

[4] nfdump: http://nfdump.sourceforge.net

3.1 Component Description

3.1.1 Tcpdump[3]

Tcpdump is a popular network forensic tool and is implemented by van Jacobson et al. It is able to put a network interface in promiscuous mode meaning that all packets are intercepted including the packets that are not dedicated for the machine operating tcpdump. Then these packets are buffered and can be summarized in text form or stored to a file. Tcpdump can also read prior captured traffic. The tool tcpdump is based on the library libpcap capable of receiving packets in a binary form.

3.1.2 Nfdump[4]

The tool nfdump by P. Haag et al. reads Netflow records captured by a Netflow collector and displays them in a human-readable form and can easily be parsed.

3.1.3 Aguri

The tool Aguri [1], also uses the library libpcap using the functionality to acquire network packets. From these packets TCP flows are inspected and then stored in Patricia trees taking into account their network prefixes.

3.1.4 NetflowToAguri

In this module, we have implemented an adapter that converts captured Netflows into Aguri format such that the Aguri tool can process them.

3.1.5 AguriProcessor

The script AguriProcessor is written by us and takes as input the output trees of Aguri, waits for at least two successive Aguri trees and computes the kernel-value *K*, defined in Equation 1 between these trees. The computed *K*-values are then used in AguriViz for further processing.

3.1.6 AguriViz

Our implemented module called AguriViz processes the kernel-values for the visualisation purpose and maps them into RGB format by referring to the equations and algorithms presented in section 2.3.

3.1.7 AguriUI

The AguriUI module is our implemented visual user interface for the network operator. With this interface he can adjust the monitoring setting and the parameters (image height, width, rectangle size, freshness parameter, brightness, intensity) for the generated images. Figure 3 shows the visual interface of our tool. It shows on one hand the analysis of the source profile and on the other hand all information about the destination profile analysis. The more general information about the analyses can be obtained in the side box.

3.1.8 Tcpslice[5]

Tcpslice developed by V.Paxson is a program for extracting proportions of packet-trace files according to time stamps.

[5] tcpslice: ftp://ftp.ee.lbl.gov/tcpslice.taz.gz

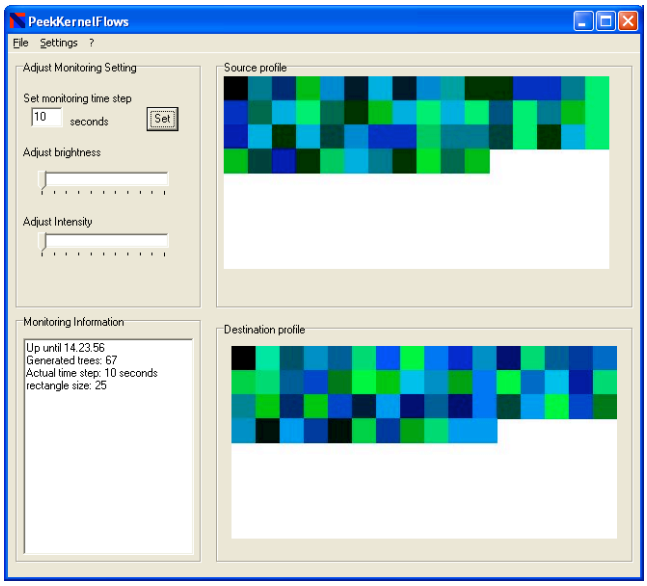

Figure 3: AguriUI module - User Interface

4. EXPERIMENTAL RESULTS

4.1 Statistical Information

For the experimental part, we have evaluated our tool PeekKernelFlow on flows from honeypot traffic. A High interaction honeypot, exposing vulnerable ssh-server, was operated for 24 hours on one public IP-address. All traffic related to this host is by definition suspicious and recorded. The honeypot interacted with 47 523 different external addresses. We used the default time parameter of Aguri (α = 5 seconds) also called freshness parameter for the experiments and represented the visualisation in a 24-bit color space. Choosing a larger freshness parameter α leads to more outdated observations. Table 1 summarizes statistical information about the used data sets.

Table 1: Dataset information

	Honeypot
Operation time	24 hours
Number of addresses	47 523
Used bandwidth	64Kbit/s
Exchanged TCP packets	1 183 419
α (seconds)	5
Colours (bit)	24

4.2 Visualisation Results

In this section we give interpretations for generated images by using PeekKernelFlows.

In Figure 4 and Figure 5 we present generated pictures by applying PeekKernelFlows to a 24-hour honeypot traffic capture. Figure 4 presents the outcomes for the analysis of the source profile whereas Figure 5 presents the destination profiles. The figures have a resolution of 1 200 x 1 000 pixels. This means with a rectangle size of r = 20 and a freshness parameter α = 5 seconds, we have represented 3 000 Aguri trees in one picture, corresponding to approximately 4 hours of traffic.

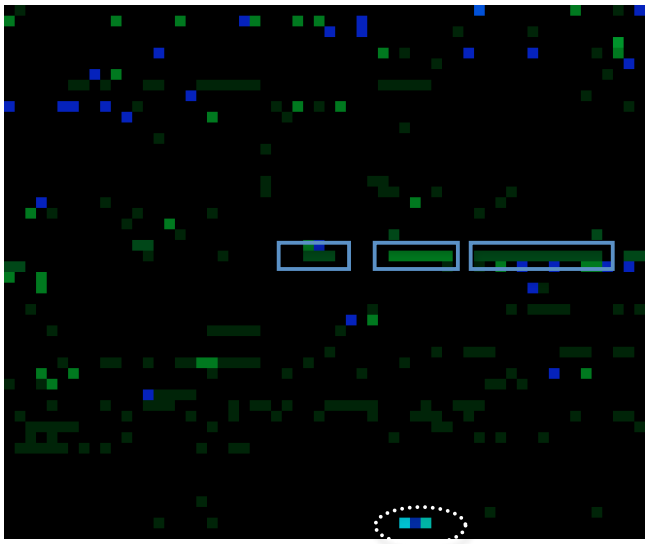

Figure 4: Visualisation of source profiles by PeekKernelFlows

We manually investigated some interesting patterns and noticed a minor design problem in the Aguri tool. According to the user manual of Aguri, the s-switch is used to output a summary every α seconds. However, by analyzing the Aguri trees we recognized that the interval is not constant.

After an investigation of the Aguri source code, we noticed that the starting and end time are taking from the captured packets. This has as consequence that moments of silence, where no packets are transmitted, are not been taken under consideration and so, we detected that the time intervals varied by $\alpha+\tau$.

An active honeypot is continuously under most varying attacks. Some attackers launch brute-force attacks against the honeypot, other attackers having already compromised the system scan or control other targets. Both kinds of attackers generate a lot of network traffic-noise that is hard to manually investigate.

In both images the background network traffic noise is represented by the colour 'black', which means that Aguri trees are completely different. If the colour tends to 'white' following the RGB-model, the more similar the successive Aguri trees are.

In Figure 4, four relevant patterns can be observed. Three successive lines in 'green' colour framed by the rectangles can be spotted. After a manual investigation of the recorded network traffic, we observed that these ' green' lines (framed by the rectangles) represents ssh brute-force attacks, whereas the 'coloured' line (framed by the dashed ellipse) in the right bottom of the image represents scanning activities of our honeypot towards other victims.

In the observed scanning activities the attackers nearly used the entire bandwidth of the honeypot and continuously scanned entire sub-networks. This induces similar successive Aguri trees. The colours of scanning activities are more 'light-coloured' than dominant TCP sessions, which are mostly in 'dark' colours. This can be explained by the kernel function K, where the topological part $s(a_i,b_j)$ is primary on the volume part $v(a_i,b_j)$.

In the source profile, we are not focused on the exact target the honeypot attacks. To achieve more fine-grained information about the targets, the destination profile can be used, represented in Figure 5.

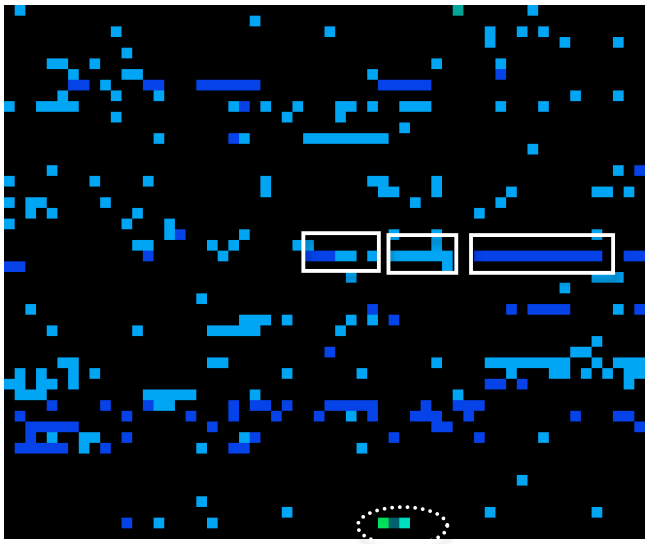

Figure 5: Visualisation of destination profiles

In the destination profile analysis we can observe more patterns represented as segments due to the focus on destinations. We can observe how long attackers stayed at a dedicated target and how much traffic was exchanged.

5. RELATED WORK

Analyzing network traffic is a tedious and error prone task. In the field of network intrusion detection, visualisation is a popular complementary instrument. As an example, Foresti et al. [3] use visualisation for representing network alerts. Visualisation can also be used at lower levels. In article Goodall et al. [6] visualize network flows. As input the authors refer to a Netflow repository that is transformed and stored in a database. A network operator can then choose between different visualisation techniques, like various histograms or flows, which are represented as a graph in a circular manner. Goodall et al. [6] presents a tool called FlowViz that refers to a similar coloured rectangular representation for showing the usage of ports. However in our work, we map kernel values obtained by the computation of Aguri trees to coloured rectangles, where the colour is a function of the kernel value. Gonzalez-Arevalo et al. [5] propose 'mice and elephants' plots. A flow is modeled as set of packets and a segment in the image represents a single connection. They also use a two-dimensional space such that the x-axis is the time. In order to avoid collisions of parallel connections, different randomly chosen heights are used. We explicitly use line breaks with the aim to increase the time space and thus increment the y-axis in a linear fashion. Patole et al. [10] refer to Self-Organizing Maps (SOM) and Mansmann et al. [9] refer to TreeMaps for dynamic intrusion detection. The graphical representation of a SOM, first presented by Kohonen [8], looks quite similar to the visualisation of PeekKernelFlows, with the crucial difference that PeekKernelFlows takes into account the time and does not need learning with training/testing phase. Glanfield et al. [4] used concentric circles to display flow relationships by respecting network hierarchies. PeekKernelFlows also respects flow hierarchies due to Aguri but focuses more on differences between successive trees over time. Glanfield et al. [4] referred to a similar problem statement in the context of network administration. The authors assume that an administrator has previously determined the subset of data that should be visualized.

The determination of such a subset is often challenging due to the noise generated by attackers. PeekKernelFlows provides a method to support honeypot administrators to quickly scan honeypot traffic with the purpose to determine interesting events in background noise. Hence, we are using a kernel function to line up similar events that we defined as anomalies. Kaizaki et al. [7] claim to detect Denial-of-Service attacks like flooding attacks only by studying the monitored profiles. An Aguri tree can be represented as a graph. One graph represents the monitored profiles in a given time interval. PeekKernelFlows can be configured such that Aguri trees can be exported into infinitesimal small intervals and tree differences are mapped into the RGB colour space.

6. CONCLUSIONS AND FUTURE WORK

We have described in this paper a visualisation tool for temporal and spatially aggregated flows. The main idea is to track changes in the topology and volume on a network between successive time intervals in order to detect anomalous behaviours. Flows are captured for a given time interval in a special tree like structure (Aguri tree). We have introduced a similarity metric that leverages kernel functions defined over such tree structures and assessed its efficiency a scenario of traffic originated from a high-interaction honeypot. A limitation of our approach is that an experienced attacker can poison our visualisation technique by generating additional noise, targeting the black colour and by this staying undetected. We plan to improve the tool by increasing the Human-Machine-Interaction, as for example adding zoom features to the user interface and integrated analysis and decisional components. Another future work feature is the implementation of 'image-transparency' for better tracking long-term evolution.

7. ACKNOWLEDGMENTS

This project is partially supported by the EFIPSANS EU-Project, CSC-SECAN-Lab and SNT. The more we want to acknowledge the National Research Fund Luxembourg, S.E.S.-Astra and RESTENA Luxembourg.

8. REFERENCES

[1] K. Cho, R. Kaizaki and A. Kato, Aguri: An aggregation-based traffic profiler, QofIS2001, LNCS 2156, pp.222-242, Springer Verlag, 2001.

[2] M.F.Cowlishaw, Fundamental Requirements for picture presentation, In Proc. of the Society for picture presentation, vol.26 no.2, pp.101-107, 1985.

[3] S. Foresti, J. Agutter, Y. Livnat, Yarden, S. Moon, R. Erbacher, Visual Correlation of Network Alerts, IEEE Comput. Graph. Appl., vol. 26 n. 2, pp. 48-59, IEEE Computer Society Press, Los Alamitos, CA, USA, 2009.

[4] J. Glanfield, S. Brooks, T. Taylor, D. Paterson, C Smith, C. Gates, J. McHugh, OverFlow: An Overview Visualization for Network Analysis, 6th International Workshop on Visualization for Cyber Security. Atlantic City, NJ., 2009

[5] B. Gonzalez-Arevalo, F. Hernandez-Campos, J.S. Marron, C. Park, Visualization Challenges in Internet Traffic Research, Graphics of Large Data Sets: Visualizing a Million, Ed. A. Unwin, M. Theus, H. Hofmann, Springer, New York, pp. 203-226, 2006.

[6] J. R. Goodall and D. R. Tesone, Visual Analytics for Network Flow Analysis, Conference For Homeland Security, Cybersecurity Applications & Technology, pp.199-204, IEEE Computer Society, Los Alamitos, CA, USA, 2009.

[7] R. Kaizaki, O. Nakamura and J. Maurai, Characteristics of Denial of Service Attacks on Internet using Aguri, ICOIN 2003, LNCS 2662, pp.849-857, Springer Verlag, 2003.

[8] T. Kohonen, Self-Organizing Maps, 3rd Edition, Springer Verlag, 2001.

[9] F.Mansmann, F. Fischer, D.A. Keim and S.C. North, Visual support for analyzing network traffic and intrusion detection events using TreeMap and graph representations, CHiMiT'09: ACM Proceedings of the Symposium on Computer Human Interaction for the Management of Information Technology, pp.19-28, Baltimore, Maryland, 2009.

[10] V. A. Patole, V. K. Pachghare, P.Kulkarni, Self Organizing Maps to build Intrusion Detection System, International Journal of Computer Applications, 0975-8887, vol.1 n.8, 2010.

[11] L.Spitzner, Honeypots: Tracking Hackers, Addison-Wesley Professional, 2002.

[12] V. Vapnik, Statistical Learning Theory, Wiley, 1998.