

Detecting New Patterns of Attacks — Results and Applications of Large Scale Sensing Networks

Torsten Voss and Klaus-Peter Kossakowski
DFN-CERT Service GmbH, Heidenkampsweg 41, D-20097 Hamburg,
Voss | Kossakowski@DFN-CERT.de

Abstract: It is still not clear, how large scale sensing networks can be turned into useful resources of incident response teams. Recent research has shown that the work of incident response teams is clearly exposed to denial of service attacks if the handling of low number / high priority incidents is not separated from the work related to high number / low priority incidents [WK05]. This would imply that handling the magnitude of data coming from large scale sensing networks will pose concrete operational problems to any incident response team dealing with it. While there are some strategies to mitigate this problem, we believe that only selecting the 'interesting' events through filtering is not good enough and give away useful insights that are inside the data but not yet obviously visible for an unaware observer. Therefore our research objective is to identify successful strategies of how to extract useful data automatically out of large data sets. So far we have succeeded to improve a suggested algorithm and test it's application in an operational setting. This paper will outline the algorithm, any improvement made as well as the key insights in it's application.

1 Background

The creation of large scale sensing networks like eCSIRT.net or Leurre has enabled the aggregation and correlation of useful data across many organizations. While such data can be utilized directly for incident response like eCSIRT.net or analyzing new trends and attacks like Leurre, their application within early warning systems has proven to be challenging. The biggest challenge is the amount of data available, effectively denying any human operator to sieve through in search for the famous needle in a haystack. What is much needed is a strategy to assist any human operator - or analyst - in concentrating on the new information only. We expect that any tool support will allow then the analyst to dig into the available data as deep as she/he actually needs it to do - but it should not be necessary right from the start. Current approaches that have been utilized to solve this problem can be categorized into the following groups:

1. **Traditional IDS Thinking:** Define pattern that alert the analyst or define statistical schemes (thresholds, hidden markov etc.) as well as triggering functions for alerts.
2. **Attack Graph Thinking:** Based on the knowledge of attacks and events that cannot yet be categorized as full-fledged attacks (like probing and scanning) all data is assessed and evaluated

3. **Supporting the Human Thinking:** Provide interfaces and representations of data in a way to support the easy and flexible analysis of large data chunks, most often related to graphical representations
4. **Data Mining Thinking:** Based on experiences in other realms it is possible to mine large data sets for pattern that can be described by attributes like frequency or timelines.

While all of the above approaches have their tradeoffs and it seems most likely that any current system will need to deploy a combination of these approaches. We have been focusing on data mining approach in particular. Based on the documents of [CCH02] and later papers about the relationship to IDS systems [AI05] we decided to implement the algorithm and test it under real life conditions. One of the main design decisions was to not integrate the alert function directly into the algorithm. Therefore the algorithm just provides us with one (of some and potentially many) ways to aggregate the data available to us. Based on the results other functions (again we will need to focus on statistical methods or simple thresholds) can then be utilized to evaluate, if it would be necessary to alert any human analyst for further studies.

2 The Environment for our practical Tests

The setup we have utilized collects large set of IDS, netflow, network monitoring and low interactive sensor network data and provides all available data in terms of a standardized internal data representation. This internal data representation was used as the basis for our practical evaluation, of how such data mining pattern generation function would fit into the workflow of an analyst. Instead of concentrating on the application context of early warning systems only, we also had the traditional incident response process in mind.

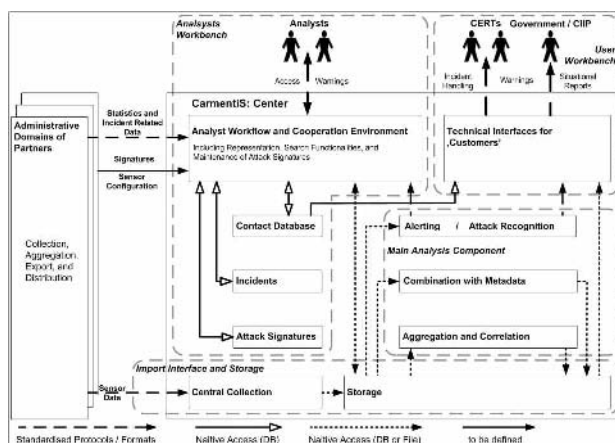


Figure 1: A complete Composition of the architecture

The global goals and architecture of the environment we were able to utilize is described in another paper of this proceedings [GMS06], therefore we will not repeat what is already outlined there. While the overall architecture in figure 1 provides a framework for all kind of functions and services, we have decided at the very start to structure all functions in such a way, that scalability and extensibility can be achieved. While originally concentrating on the data collection and storage, it was therefore possible to build on unified interfaces and building blocks to implement the algorithm as some kind of 'add-on'. To do so we needed to read data from the storage, process the data and report any findings to the analyst workbench. Figure 2 shows the components that are of importance for the integration of the algorithm.

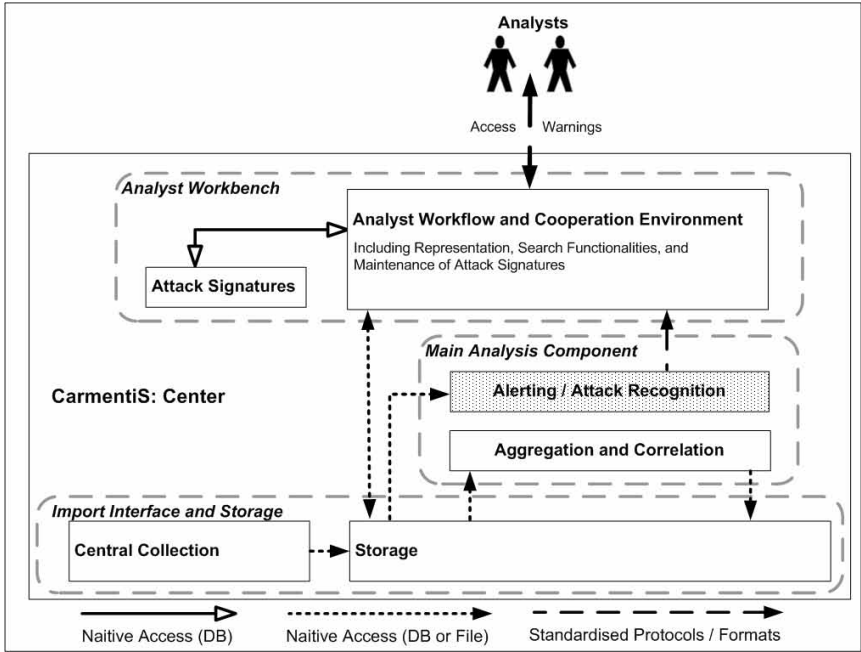


Figure 2: Association Rule Mining is part of the Backend Analysis / Alerting Engine

It is the responsibility of the analyst to decide how to process any findings further. In some cases further analysis will be necessary, in other cases the analyst will trigger an alert to the CSIRT community right away, having sufficient prove that something has gone bad. This is outside the scope of our algorithm and while this might be seen as a weakness, we believe that it is in fact one strength of the overall architecture. Our approach allows us to run several, even competing, algorithms and let the analyst deal with the overall assessment. Further streamlining and improvement over time might however develop tailored algorithms specifically for our purposes. This will pose no further problems given that only the attributes of the reported results will be changed and that more workflows - automated or not - can be added without problems.

We will now concentrate on the foundation of our work: the algorithm.

3 Algorithm

Our work - as many other similar projects - is based on the work of R. Agrawal et al. [AIS93] from the year 1993. The algorithm was evolved to mining data for association rules. Later on the algorithm was improved on the conceptual as well as on the implementational level. Association rule mining was used within the context of intrusion detection in various projects. We based on research on the modified algorithm from A. Alharby et al. [AI05] who proposes association rule mining to detect new attack pattern.

Association rule mining as defined by the so called 'Apriori' algorithm only reports continuous pattern. These are pattern which are based on consecutive fields only, not allowing any gaps or place holders for an individual field in a sequence. The work of A. Alharby proposes discontinuous pattern, an idea we advanced further also allowing place holders at the end of a pattern.

In the next sections the fundamental definition, notation and the algorithm are explained.

3.1 Definition

Before we go on we will need to define a few key terms that are used throughout the paper:

Pattern: Pattern describe the shared characteristics of any number of records. We differentiate further two types of pattern.

Continuous Patterns: These are pattern types, for which all elements are in the right order and without a gap in their sequence, e.g. the pattern 'bcd' and 'ab' are continuous pattern. Figure 3 (next page) shows all continuous pattern based on the CISCO netflow format which are considered for the algorithm.

Discontinuous Patterns: These are pattern types, that might have one or more gaps in their sequence, e.g. the pattern 'a e' with 'bcd' missing as well as 'a c e' are discontinuous pattern. Figure 4 (also next page) shows all discontinuous pattern again based on the CISCO netflow format.

Star (connotated as: '*'): This symbol represent the 'gap' as a placeholder within a sequence of element-attributes in a discontinuous pattern, e.g. we will write 'a*e' instead of 'a e' further on. A discontinuous pattern never starts or ends with a star.

All pattern, that might be considered for further processing by the algorithm are called *candidate pattern*.

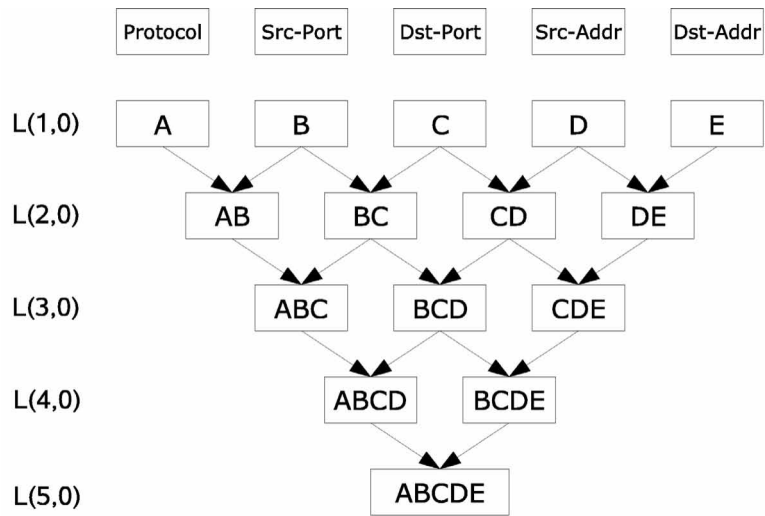


Figure 3: Continuous pattern tree (Figure from [AI05])

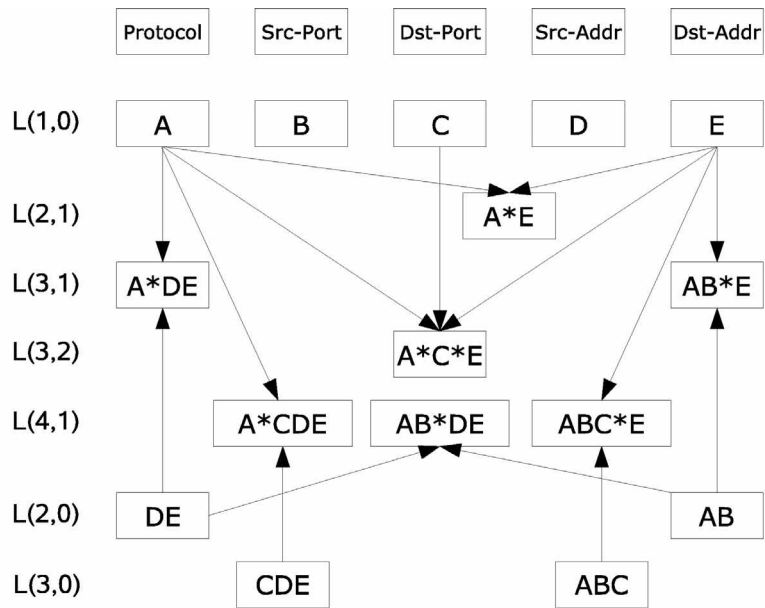


Figure 4: Discontinuous pattern tree

3.2 Notation

Some further remarks on the specific notation are necessary:

Record is a set of information that is provided as input format for this algorithms, e.g. a CISCO netflow [Cis] or a Argus flow [QoS]

Element is a field of a record, e.g. the source port or the protocol of a particular CISCO netflow

Attribute is the characteristic of a particular element, e.g. '1026' for a source port or '10.0.0.1' for a source address

Value k is the counter of element attributes that are combined into one pattern

Value l is the counter describing how many stars are in a particular pattern

Threshold is the limit value of the candidate patterns, how often the pattern need to be in the records. The threshold is set by the analyst.

$C_x(k, l)$ is the table that contains the set of candidate pattern with k elements and l stars. It is used to build $L_x(k, l)$. The 'x' in the tablename is a placeholder for the elements, e.g. $C_a(1, 0)$ contains the protocol candidate pattern, $C_b(1, 0)$ contains the source-port, $C_c(1, 0)$ the destination-ports, ... That are also combined with other elements like $C_{de}(1, 0)$ with source- and destination-address or with a 'discontinuous pattern' like $C_{a..e}(2, 1)$ with protocol and destination-address.

$L_x(k, l)$ stores all patterns which have occurred more often than the defined threshold. It contains patterns with k elements and l stars.

3.3 Processing the Records

Our current work is tested with CISCO netflow [Cis] derived for a darknet with Nepenthes honeypot sensors. All traffic going in or out of this network is considered 'malicious', no user traffic falsifies the retrieved netflows. The quantity of the netflows vary from 350.000 to over 4 million and is shown in the figure 5. The number of netflows per day for each protocol is shown in the figure 7.

All records read are processed step by step as described here:

Initializing $L_x(1, 0)$: At the very start of the algorithm every element of the data processed is already a candidate for a one element pattern. First we select all different elements from one element type and save them into a temporary table as candidates for $L_x(1, 0)$. Then we count how many times every element of the temporary table is present in the overall database and the count is saved for every element in the temporary candidate table. After counting, all elements from the temporary table are

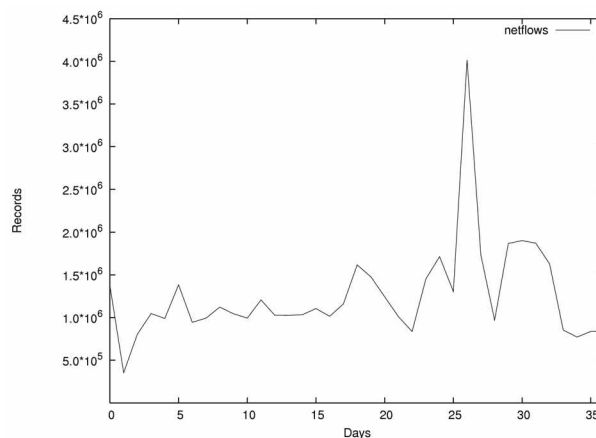


Figure 5: Number of netflows processed per day

copied into $L_x(1, 0)$ if and only if their counter is higher than the threshold value set for the algorithm. This calculation is carried out for all other remaining element types.

For example: All different elements from 'prot' (e.g. the protocols by number 1, 6 and 17) are saved into the temporary table. Then we count how often the various protocols exist in the database (e.g. 67.000 instances of protocol 1, 50.158 instances of protocol 6 and 179.894 instances of protocol 17). If the counters of the elements within the temporary table exceeds the threshold value (e.g. 60.000), than these elements are saved into the table $L_a(1, 0)$ (e.g. the elements for protocol 1 and 17 are saved).

Identify continuous pattern $L_x(k, 0)$: Continuous pattern are always build by the combination of adjacent $L_x(k - 1, 0)$ -tables which have been either created by the initialization or further processing. Therefore we can save the combinations easily into a temporary table as candidates for $L_x(k, 0)$. We again count how often every candidate describe a record in the dataset. This value is saved into the candidate table and when the number of patterns are higher than threshold value the candidate is then saved into the $L_x(k, 0)$ -table.

This step is repeated for all other remaining tables as created by the steps before ($L_{ab}(2, 0)$, $L_{bc}(2, 0)$, ..., $L_{abcde}(5, 0)$ as shown in figure 3).

Identify discontinuous pattern $L_x(k, l)$: In the next step all continuous pattern are combined with each other into new temporary tables like shown in figure 4 and then saved into temporary tables. After counting all occurrences, how much records are described by these pattern in the dataset, pattern exciding the threshold value are copied into the $L_x(k, l)$ table, l indicating the number of 'gaps' in the otherwise continuous sequence.

If all these steps have been carried out, the results can be saved for further processing. In our environment this means that the output is reviewed by an analyst to identify areas of concern. Further analysis - outside the algorithm work presented here - will then show what kind of problem might have been identified and what measures need to be taken. Our future work will concentrate on pattern that are not based on single data records like CISCO netflows.

4 Improvements and Extensions

While we have been busy with the implementation and tests in a large scale testing environment, it became obvious that improvements were necessary and unavoidable. The processing of large data sets requires attention to details, not needed in simple test environments. To make the resulting pattern more complete we extended the definition of discontinuous pattern. In addition we added useful information fields to each pattern which were originally not considered for this algorithm.

4.1 Parallelizing the Processing

If we look at the figures outlining the pattern trees (see figure 3 and figure 4) we can see that at least some part of the processing can be done in parallel, as long as the results are not depending on still pending processes. The scheduler that coordinates this parallelism allows for a much better resource utilization and improves the overall execution time. In our test environment up to four processing steps were carried out in parallel.

4.2 Process Optimisation

Most of the processing time is spend by calculating the number of combinations as candidates for $L_x(2, 0) - L_x(5, 0)$. The original algorithm from A. Alharby [AI05] is a lot of faster than the original 'Apriori' algorithm [AIS93]. This is archived by taking the lower level $L_x(k-1, 0)$ to create the higher one $L_x(k, 0)$. This improvement is in our experience in practise not enough. With the high number of records to process we were faced, we run into practical problems.

An example for our actual problem: To build the pattern $L_{bc}(2, 0)$ we need to combine all source ports elements from $L_b(1, 0)$ with the destination ports elements from $L_c(1, 0)$ and save the combinations to a temporary table as the new candidates.

Taking up the dataset of our neflows from any ordinary day we have - let's say - 15.000 source ports in the $L_b(1, 0)$ table and 16.000 destination ports in the $L_c(1, 0)$ table. If we combine all source ports with all destination ports, we have more than 240 million combinations. To save these significant number of combinations, the

server will need space on the harddisk, as the memory of any usual server will not be enough for this task.

The counting, how many times any combination exist in the dataset, costs even more cpu time. On the given day we have 4 million records and by counting every of the combinations of source and destination ports alone (240 million). This equals to over $9.6 * 10^{14}$ comparisons. Tests to process the performance have been carried out with slices of 10 million combinations. Altogether over four hours were spend for all slices, it is obvious that this will not work in any practical terms.

The problem originates through the creation of combinations, that are not in the database at all. Approx. 90 percent of the processing time is therefore spend without any benefit. To overcome this problem we extract only those combinations that are really represented in the dataset. Therefore we make a 'join' between the first pattern table, in our example $L_b(1, 0)$, and the dataset with all records. With the join we select the desired attributes (source and destination ports) and also delete all duplicates. This can be efficiently done with a single SQL statement:

```
SELECT DISTINCT b.srcport,
               b.dstport
  INTO TEMPORARY tmp_dist
  FROM l10_b a
  INNER JOIN dataset b
  ON (a.srcport=b.srcport);
```

In addition this instruction saves all extracted combinations into a new temporary table (tmp_dist) which can be used to count the occurrences of this combination in the dataset.

The results in our example: Only 270.000 combinations have been identified applying our approach. Compared to the original number of 240 million combinations the improvement is significant and allows efficient processing of huge numbers.

To join continuous pattern with three and more elements a slightly different approach is used. These patterns are created by joining two patterns, e.g. $L_{ab}(2, 0)$ with $L_{bc}(2, 0)$, with a shared common value, e.g. 'b'. This is described by Witten et al. [WF05].

4.3 Introducing new Discontinuous Pattern

Due to the definition for 'star' only discontinuous pattern are allowed that have an 'a' as the start and an 'e' as the end. As 'e' is the place holder for the destination address some useful pattern are actually missed, e.g. attacks that are carried out against some destination addresses. By allowing that discontinuous pattern can end with a 'star', we gain more pattern as shown in figure 6. To do so we need to re-define the previously given definition to read like:

Star (connotated as: '*'): This symbol represent the 'gap' as a placeholder within a sequence of element attributes in a discontinuous pattern, e.g. we will write 'a*e' instead of 'a e' further on. A discontinuous pattern never starts with a star.

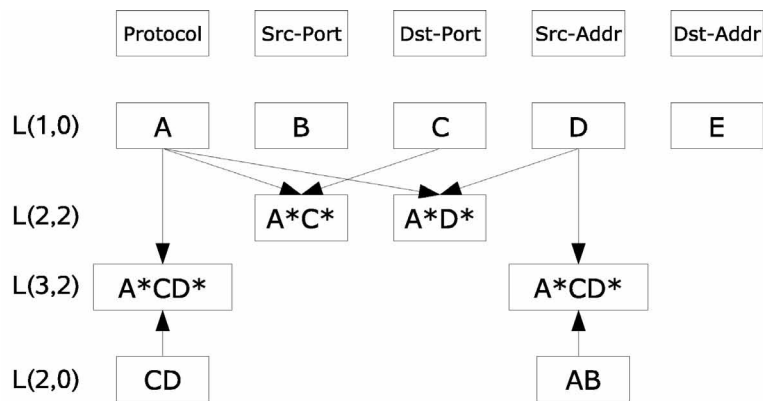


Figure 6: Additions to the Discontinuous pattern tree

4.4 Saving Counters and Timestamps

For the analyst that has to deal with the derived pattern it is of great interest how many times a pattern occurs in the dataset. This information was simply included in the results as additional output field.

An analysts is even more interested in any timestamp information that relates to any particular pattern. This will allow him later to check other resources to confirm a particular attack or advance the assessment. To achieve this goal for every pattern two additional attributes - which are not used further in the algorithm - need to be stored. One attribute (called 'first') is the timestamp of the first record that creates a particular pattern. This assumes that all records are processed in the correct time order. The second attribute (called 'last') contains the timestamp of the last record of the dataset that fits this particular pattern. The negative side effect is the necessary updates of the second attribute for every step of the algorithm.

5 Insights for the Analyst

To show the usefulness of the information provided by the algorithm we will give some short and simple examples. These examples can only provide a short overview.

1. Suspicious ICMP Peaks: Figure 7 shows two peaks for ICMP (protocol 1) on the 18th, 19th and 25th day of this test period. A further look into other information in figure 8 reveals two IP addresses that show the same behaviour.

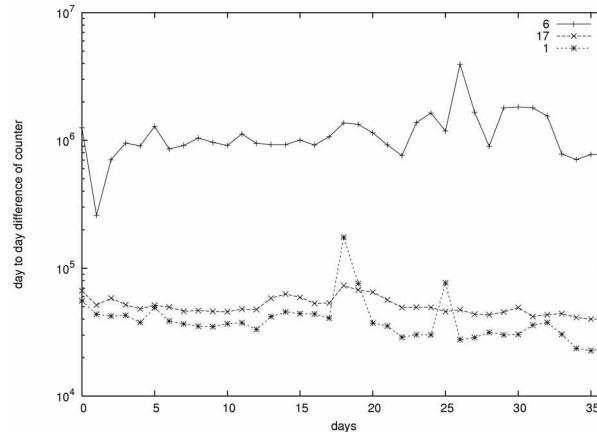


Figure 7: Example 1: Two suspicious ICMP Peaks (logarithmic scale)

2. Continuous Attack on UDP 1434: Figure 8 shows another typical behaviour. At some point in time a particular source IP address starts an attack on UDP 1434 (ms-sql-server) on day 11. This attack is continuously observed over the remaining part of this test period.

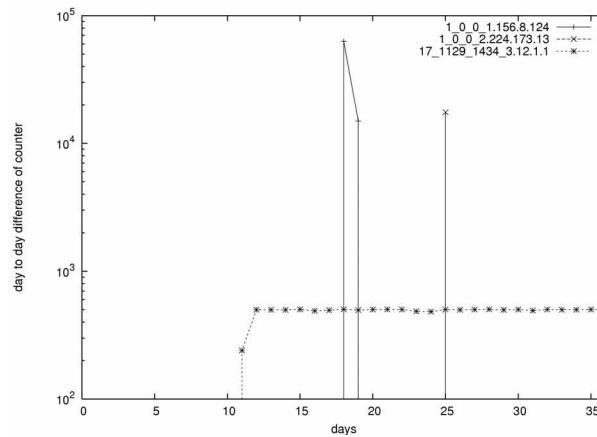


Figure 8: Example 1 and 2: Two attackers that caused the ICMP peaks; One attacker that concentrates on UDP 1434 (logarithmic scale, IP's anonymized)

3. Two Combination Attacks: Similarities between different pattern can result from combined attacks (which are not yet aggregated by our version of the algorithm). Figure 9 shows two combined attacks, each coming from one particular IP address. One attack consist of TCP connections to port 139 and 445, the second attack has additional TCP connections to port 80.

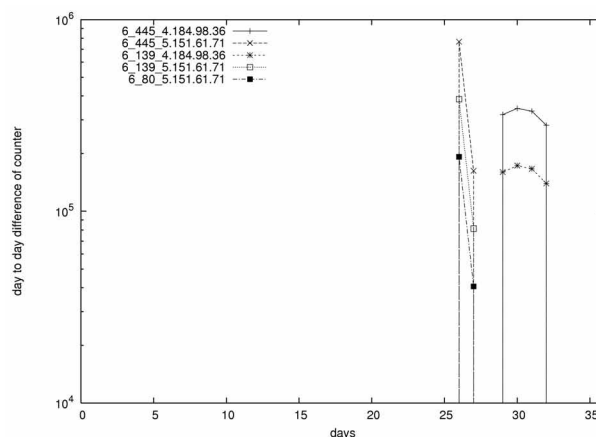


Figure 9: Example 3: Two combined Attacks (logarithmic scale, IP's anonymized)

6 Outlook

While data mining is recognized as a valuable approach to determine interesting information from large data sets it is still to be shown whether this approach fits in the overall incident response life cycle and can be applied effectively in the context of any real-time early warning systems.

With our future work we hope that we will be able to answer these questions and show the usefulness of further evaluations that take advantage of combined efforts of the data collection by many stakeholders that have a joint interest: to mitigate threats to their organizations and communities, regardless from which site they might origin.

7 Acknowledgment

We would like to thank all partners of the CarmentiS project as well as our colleagues, that have helped us to implement the algorithm in a near operational environment. Especially we are grateful for the support of Juergen Sander (PRESECURE) and Andreas Buntin (DFN-CERT) for their ongoing patience and support.

References

- [AI05] Abdulrahman Alharby and Hideki Imai. Hybrid Intrusion Detection Model Based on Ordered Sequences. In Vladimir Gorodetsky, Igor V. Kottenko, and Victor A. Skormin, editors, *Computer Network Security, Third International Workshop on Mathematical Methods, Models, and Architectures for Computer Network Security, MMM-ACNS 2005, St. Petersburg, Russia, September 25-27, 2005, Proceedings*, volume 3685 of *Lecture Notes in Computer Science*, pages 352–365. Springer, 2005.
- [AIS93] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining Association Rules between Sets of Items in Large Databases. In Peter Buneman and Sushil Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, Washington, D.C., 26–28 May 1993.
- [CCH02] Yen-Liang Chen, Shih-Sheng Chen, and Ping-Yu Hsu. Mining Hybrid Sequential Patterns and Sequential Rules. In *Information Systems*, volume 27, pages 345–362, 2002.
- [Cis] Cisco Systems, Cisco IOS NetFlow White Papers. www.cisco.com/en/US/products/ps6601/prod_white_papers_list.html.
- [GMS06] Bernd Grobauer, Jens Ingo Mehlau, and Juergen Sander. CarmentiS: A Co-Operative Approach Towards Situation Awareness and Early Warning for the Internet. In *Proceedings of IMF 2006*, 2006.
- [QoS] QoSient, Argus Documentation. www.qosient.com/argus/index.htm.
- [WF05] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, second edition, June 2005.
- [WK05] Johannes Wiik and Klaus-Peter Kossakowski. Dynamics of Incident Response. In *Annual FIRST Conference, Singapore, June 2005, Proceedings*, 2005.