UNIVERSITY OF AMSTERDAM

FINAL REPORT

Anomaly Detection on Log Files Based on Simplicity Theory

February 9, 2020

Mar Badias Simó
mbadiassimo@os3.nl
UvA ID: 12838268

Giacomo Casoni
giacomo.casoni@os3.nl
UvA ID: 12533750

Lecturer:
Cees de Laat

Tutor:
Giovanni Sileno

Course:
Research Project 1 - #43

Abstract

In this project we applied Simplicity Theory concepts to anomaly detection, and built a domain-independent tool capable of detecting anomalies in a series of events in a way similar to what humans supposedly do.
Simplicity Theory defines probability in cognitive terms rather than mathematical set-based ones. The theory states that a given event is unexpected, in the eyes of an observer, when it is hard to generate and/or easy to describe. Quantifying the generation and description complexity of a given event can, therefore, lead to an understanding of how said event is unexpected in the eyes of a human observer. We defined a generic framework to compute generation and description complexity for any given event represented as a combination of a certain number of features. We also provided a basic definition of how to set a context and, therefore, a specific point of view to observe events from.
We built a proof-of-concept tool that applies our framework and tested it against the 1999 DARPA IDS dataset.
The tool proved to be very accurate in predicting network based attacks, although not very precise. We hypothesised, however, several reasons to why the precision could be low, opening many possibilities for possible extensions and future work.

Contents

1 Introduction

As the Internet has become one of the fundamental aspects of our modern society, the hazard of criminal activities has grown accordingly. Intrusions in systems have threatened and do threat data security and distort network services. In this context, intruders need to be detected to prevent their misbehaviour.

Humans use common sense to detect exceptional or abnormal events and whether these can have a (positive or negative) impact on their desires or interests. This project has focused on developing a system that leverages the relevance of an event from a human-cognitive perspective by detecting its unexpectedness. Using this method we aim to provide anomaly detection and adaptive monitoring in a system.

Before presenting the contents of our research, we start this section with a brief introduction to anomaly detection and the Simplicity Theory.

1.1 Anomaly Detection

Intrusion detection techniques can be classified into two categories: signature-based and anomaly detection-based. Signature-based systems are designed to detect predefined attacks using their known signatures. These systems require regular updates and are not able to detect unknown attacks. This project will not focus on signature-based systems.

In contrast, anomaly detection systems model the normal behaviour of a target system and report the abnormal activities, which are analysed as possible intrusions. With an anomaly detection system, no previous understanding of the attacks is needed so it can deal with unknown attacks like zero-days or masquerading[8].

1.2 Simplicity Theory

The Simplicity Theory[2][3] is a framework used to define cognitive probability in terms of complexity and simplicity, rather than standard mathematical, set-based, terms.

The theory leverages the concepts of generation complexity and description complexity to define unexpectedness, which is tightly correlated with relevance[1].
Generation complexity of a state s ($C_w(s)$) is defined as the length of the shortest program that a given environment must execute to achieve a given state[1]. In simpler words, $C_w(s)$ is a measurements of the events that need to happen for state s to occur.
Description complexity of a state s ($C(s)$) is defined as the shortest possible description of s that an observer can produce to determine s without ambiguity. This description matches the Kolmogorov description of complexity, given a subjective and non-universal machine that is the observer, who can resort to specifically its previous knowledge and computing abilities.[1]

Unexpectedness of a situation s ($U(s)$) is then defined as follows

$$U(s) = C_w(s) - C(s) \tag{1}$$

This suggests that a situation is unexpected, in the eyes of an observer, when it is hard to generate (high $\mathrm{C_w(s)}$) and/or easy to describe (low $C(s)$).

An example of this could be a lottery draw. Given a lottery, each combination of numbers has the same chance of being drawn, therefore the same generation complexity is the same. However if the numbers 1-2-3-4-5-6 are drawn (maybe even in this order), a human observer will find the occurrence odd. This is explained by the Simplicity Theory, as that particular sequence of numbers has a very low description cost (for example "1 to 6"). Formula (1) will therefore yield a high unexpectedness $U(s)$ result.

1.3 Our contribution

In this project, we have applied Simplicity Theory concepts to anomaly detection, and build a tool capable of detecting anomalies in a series of events in a way similar to what a person would cognitively. This framework is not bound to a set-based paradigm, which makes it much more suitable for open-ended uses.

To achieve this we had to:

- **Set a context**. The description complexity used in the Simplicity Theory is bounded to an observer. That results in different concepts of normality being observed by different observers. While the system can be considered as a global observer machine, it is also possible to create more specific points of view, which could allow for a more fine-grained analysis of events at end. Ideally, as pointed out in [7], the only way to make sure all phenomena are observed, is to analyse a system at different scales.
  We will call these contexts we defined "object prototypes". In a n-dimensional conceptual space, where n is the number of features used to describe an event, we can visualize an object prototype as a cluster of events, in terms of which new observations can be described.
  An object prototype is defined in a given number of dimensions, depending on the number of features used to describe an event. Each dimension can assume different values. We will call these possible values "feature prototypes".

- **Quantify Simplicity Theory components for each feature**. Once features are decided, their generation and description complexity must be quantified. This step requires an ad hoc analysis of a given metric, to capture the meaning of variations in the final unexpectedness levels calculations.
  It is important that these quantification are representative of the event observed, in order to be meaningful. Because these two values both contribute to the same result, it is also important to achieve a comparable quantification of these values.

- **Define features to observe**. Each event presents some metrics, like time of occurrence, agent, action... It is important to understand and pick which ones of these metrics can reveal information regarding possible misuse[8]. Moreover, metrics can be combined in more complex features, which can themselves reveal more and more regarding the state of a system.

- **Implement and test the tool**. Following the previous considerations, we need to build a tool capable of learning what the normal behaviour of a system is, and detect deviation from normality.
  The tool should, ideally, not require any networking specific knowledge, but rather it should simply apply the Simplicity Theory, using the results obtained in the previous part of the research.

1.4 Related Work

The main work regarding Simplicity Theory has been carried out by Jean-Luis Dessalles in his papers [1] and [2], where relevance is defined and Simplicity Theory presented.

Also in [1], the computation of Generation Complexity and Description Complexity is discussed and their conclusions had been considered when quantifying the Simplicity Theory components for each feature in this project.

Anomaly detection has been deeply investigated in recent years. We can find examples in [14] where the authors propose an iterative system to take into account spatial information. Also in [13] anomaly detection methods had been reviewed and analysed for its usage in Intrusion Detection Systems (IDSs). A system for detecting TCP SYN flood attacks is designed in [15] using novel anomaly detection strategies and the 1999 DARPA dataset is used to verify its efficiency. In this research, the same dataset will be used too as it provides network traffic without and with known attacks.

Research [8] applies machine learning techniques to intrusion detection and incorporates the feedback of an expert in its computations. Moreover, it also employs a classification of data, and elaboration of interesting features to observe. Mainly for us, they provide with a way to measure each class of features.

1.5 Research Question

Given the previous definitions of anomaly detection and Simplicity Theory, our research questions can be formulated as follows:

1. "How can a tool that based on Simplicity Theory for anomaly detection be designed and implemented?"
2. "How effective said tool can be in detecting anomalies in network logs in a system?"

2 Observation subject definition

2.1 Dataset and features definition

We focused, as out subject of observations, on the DARPA 1999 IDS dataset[12].
The dataset contains five weeks worth of network captures. The first and third weeks are attack free and can be used to train anomaly detection tools. The fourth and fifth weeks contains documented attacks, and can be used to test the performance of detection programs.
For each day, captures are taken for both internal and external network traffic. The dataset contains PCAP network captures; however, we decided to work on a Comma Separated Values (CSV) file Wireshark dissection of it. The motivation is text-based input seems, to us, more suitable for open-ended future uses.
The CSV entries we took into account are the following:

- Source IP: the source IP address of the packet.
- Destination IP: the destination IP of the packet.
- Protocol: the protocol that generated the packet.
- Length: the length of the packet.
- Info: a textual representation of the content of the packet.

The "Info" entry was transformed as follows.
For each protocol, templates of possible packet textual representations were found. This was done with Spell[9], used for instance in [10] to prepare system log data for anomaly detection.
After that, the "Info" entry of each CSV line was compared against each of the templates found for the protocol of the same line.
The comparison consisted of the computation of the Levenshtein string distance[11] between the template and the entry. Levenshtein string distance is a very common tool to measure the similarity between two strings. Every protocol has a limited number of templates in its information field, so the string distance allows us to measure how different is the analysed "Info" from the known template.

The minimum result was kept and substituted to the textual information.
Following are a few lines from the dataset after all modifications (Wireshark dissection and template matching) were performed, as an example.
The first two values, number and time, were not used.

```
1,0.000000,Cisco_38:46:33,Cisco_38:46:33,LOOP,60,2
2,0.096519,172.16.112.20,192.168.1.10,DNS,78,26
3,0.101814,192.168.1.10,172.16.112.20,DNS,134,8
4,0.106695,172.16.112.194,196.37.75.158,TCP,60,28
5,0.111396,196.37.75.158,172.16.112.194,TCP,60,37
```

```
6,0.111587,172.16.112.194,196.37.75.158,TCP,60,24
7,0.275928,192.168.1.10,172.16.112.20,DNS,87,35
8,0.276578,172.16.112.20,192.168.1.10,DNS,176,72
9,0.278723,192.168.1.10,172.16.112.20,DNS,79,27
10,0.279158,172.16.112.20,192.168.1.10,DNS,144,49
```

2.2 Prototypes definition

Object prototypes can be set as the observational point of view that the Simplicity Theory operates on.
In our research, object prototypes are strictly bound to a categorical value. That will be the packet protocol, in our case (but could have been the source or destination IP), meaning that all the events for a given protocol will belong to the same object prototype.
That is, however, not necessary. Some alternatives we considered were as follows:

- Object prototypes could have been based on a combination of categorical features, for example protocols and source IP. This would have given an even more specific point of view on the events occurring in the system.
- Object prototypes could have been based on no features. This would have effectively led to the most generic and high-level view possible on the system.
- A clustering algorithm such as K-Prototypes could have been employed. This could have given more information regarding the level of correlation of different features[19].

Each object prototype is characterized by as many dimensions as there are features, four in our case. For each dimension, given a certain feature prototype, the distance from the centre of the prototype is representative of how likely an event is to occur.
Note that from this point forward we will use the terms ”feature” and ”dimension” interchangeably.
In our particular case, the four dimensions would enable to capture the following:

- Source IP: for a given protocol, which IP address generates a lot of traffic, and which ones do not.
- Source IP: for a given protocol, which IP address receive a lot of traffic, and which ones do not.
- Length: for a given protocol, what packet length is usual and what is not.
- Info: for a given protocol, which kind of events happen often, and which ones do not.

Figure 1, then, represents the concepts of object prototypes, dimensions, and feature prototypes as utilized in our research.

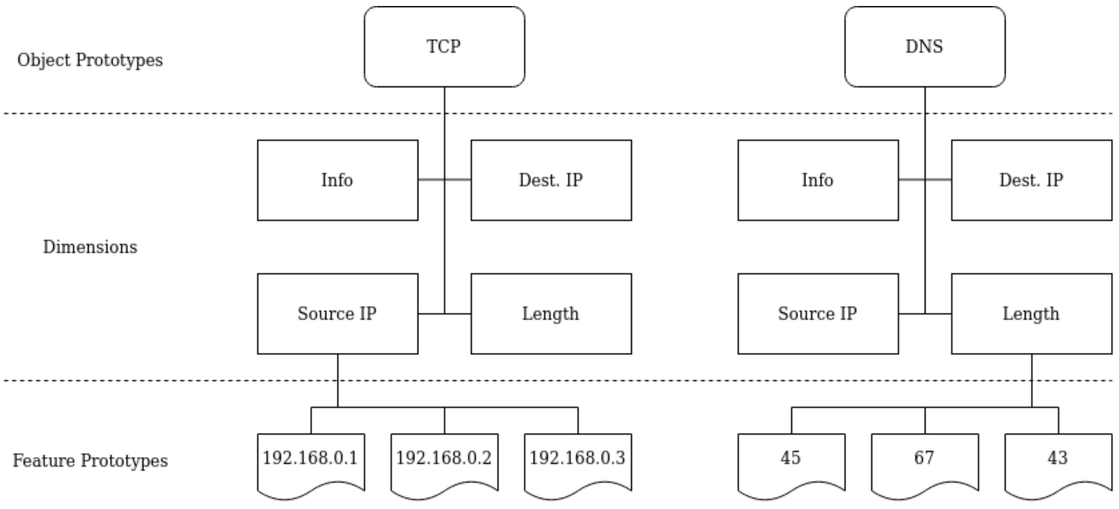


Figure 1: A representation of the concept of object prototypes, dimensions, and feature prototypes.

3 Quantification of values

The biggest challenge of applying the Simplicity Theory to a practical context resided in quantifying the generation and description complexity in a practical, computable way.
Moreover, great emphasis was also attributed to keeping these values as domain-independent as possible, so that the final result could be used for more open-ended uses than only anomaly detection in information systems.

Recall that the definition of the generation complexity is as follows:
"The length of the shortest program that a given environment must execute to achieve a given state"[1].
And the definition for description complexity is:
"The shortest possible description of s that an observer can produce to determine s without ambiguity"[1].

To generate these two values, we employ two different data structures.

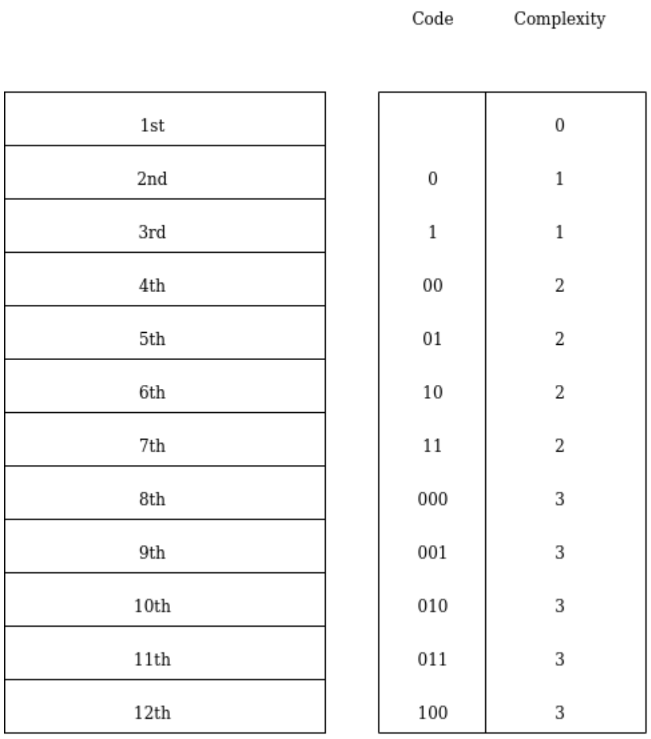
- A binary tree into which feature prototype data is organized. Nodes represent possible feature prototypes, and edges represent a binary choice (0 or 1) downwards in the tree. The more a feature prototype is characteristic of a dimension for an object prototype, the closer it will be to the root, with the root node being the most characteristic feature prototype. The complexity of a specific state, for a dimension, is given by the depth of the feature prototype in the tree.
- A stack is used to represent the memory of the machine. Observations of possible feature prototypes for a given dimension are added on top of the stack as the program runs. Whenever there is a new observation, the stack pointer always points to the top of the stack. Complexity is given by the number of bits needed to represent the number of downward movements the stack pointer needs to do before reaching a given entry in the stack.

For a given dimension, the generation complexity of a given feature prototype is constant, and represented by the depth of the prototype in the binary tree.
Description complexity, on the other hand, can be computed either by looking in the binary tree or the memory stack. The least expensive option is picked. This entails that, for a given dimension, different observations of the same prototype could yield different complexities. Moreover, given the possibility of using two separate data structures, an extra bit is needed to discriminate between the two.

The nature of the features also needs to be taken into account when dealing with quantification. Most importantly, features can be categorical or numerical. These features must be treated differently.

3.1 Categorical features

To deal with categorical features, a ranking of the possible feature prototypes must first be created. In this ranking, the most occurring prototype is placed first, and the least occurring one is placed last. A binary code is given to each feature prototype, representing the tree path from the root to the node. The length of the code, then, represents the generation complexity, as Figure 2 shows.



	Code	Complexity
1st		0
2nd	0	1
3rd	1	1
4th	00	2
5th	01	2
6th	10	2
7th	11	2
8th	000	3
9th	001	3
10th	010	3
11th	011	3
12th	100	3

Figure 2: An example of states ranking for a categorical feature.

Comparing with Figure 3 it can be noted how, indeed, the codes represent the series of binary choices to a given node, and the length represents the depth of the node (i.e. the complexity).

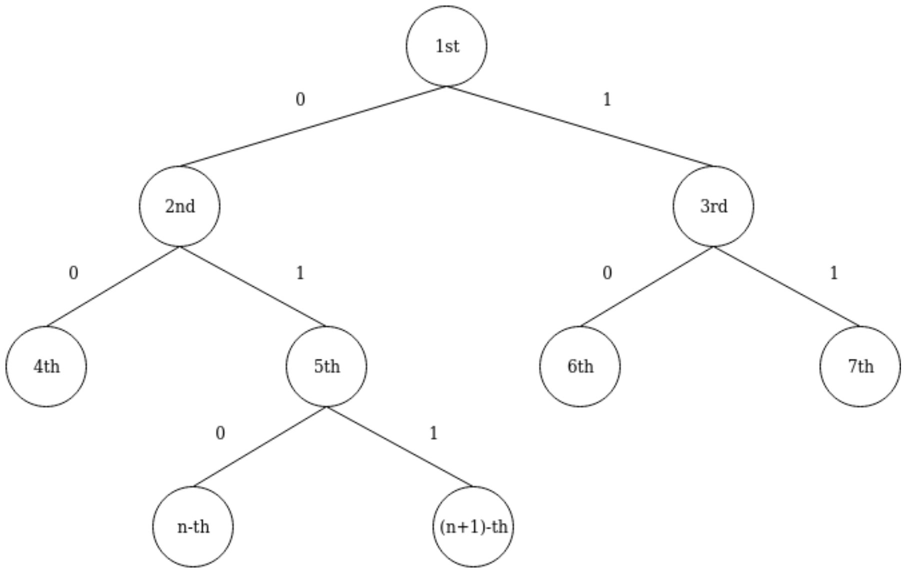


Figure 3: A tree representation of possible states for a categorical feature.

Description complexity, as mentioned earlier, can either be computed by looking into the binary tree or the memory stack.
As Figure 4 shows, the complexity of describing a feature prototype in terms of previous observations in the memory stack is simply the length of the minimum binary representation of the number of downward movements the stack pointer has to do to reach the desired observation.

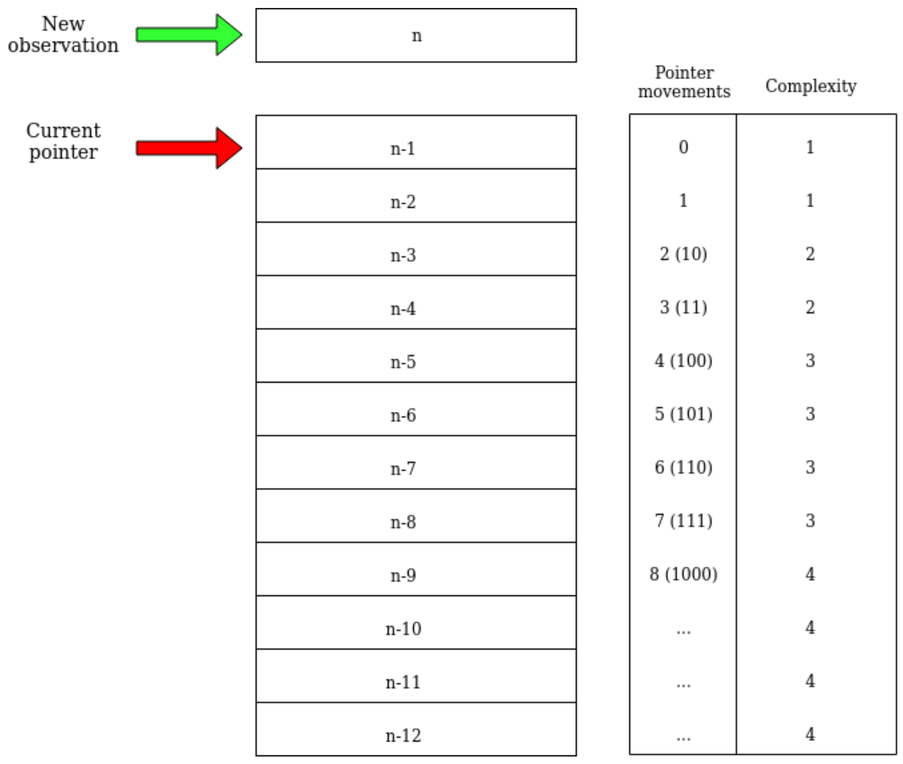


Figure 4: A representation of the memory stack of a program.

If describing a feature prototype in terms of depth in the binary tree is the cheapest option, then complexity is derived the same way as the generation complexity.

3.2 Numerical features

To compute complexities for numerical features, it is first necessary to find a way to deal with their continuous nature.

To use the same data structures defined for categorical features we focused on converting numerical features into categorical ones. That is done by computing mean and standard deviation (σ) for all the feature prototype data entries for a dimension, and then considering new observation as belonging to a category defined as an integer number of $m \times \sigma$ away from the mean, where m is an arbitrary number.
In our experimentation, we defined $m = 0.25$, because it yielded the best overall results for the case at hand, but no claims are made regarding the optimality of this decision.

A ranking of more frequently occurring feature prototypes is generated as per categorical features. States closer to the mean are considered to be occurring more frequently.
Also, just like for categorical features, a binary code is assigned to each $m \times \sigma$ range.

The generation complexity, as for categorical features, is defined as the length of the assigned code. Figure 5 shows the described concept.

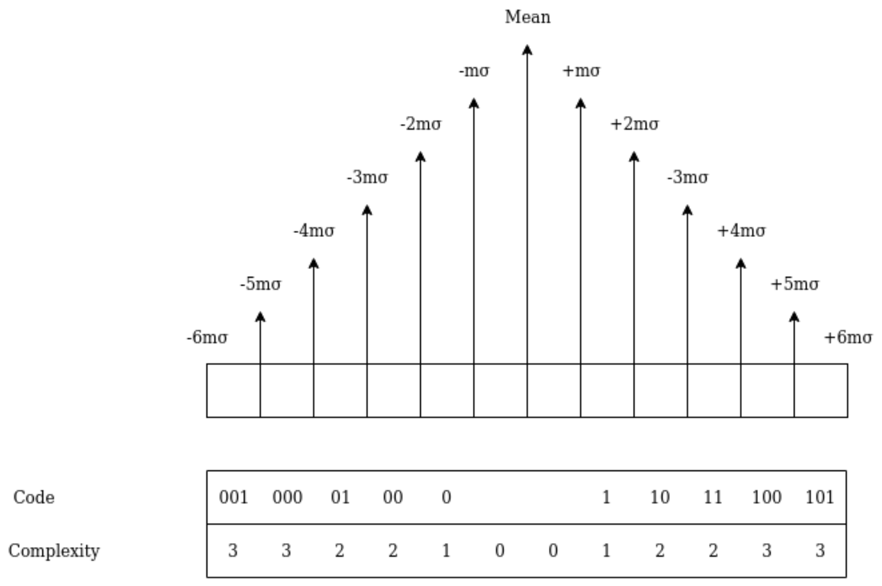
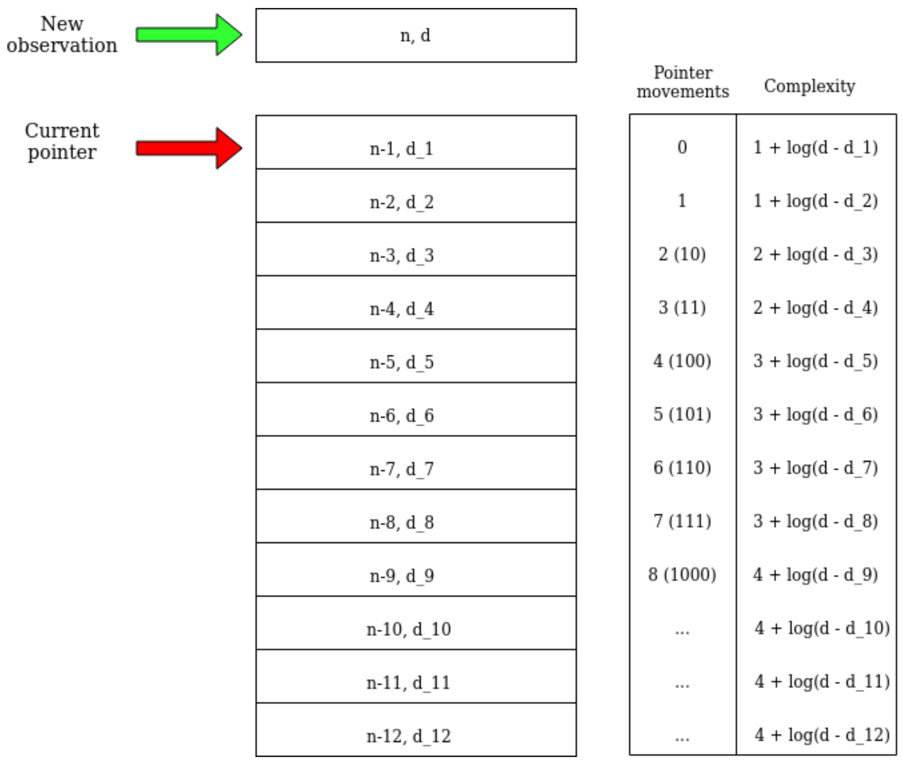


Figure 5: A representation of the coding for numerical features.

The memory stack, in the case of numerical features, contains the distance of previous observations to the mean (or values from which it can be derived). When a new observation is made, the description complexity, in terms of comparison with previous observations in the stack, is given by the minimum number of bits needed to describe the number of downward movements the stack pointer needs to reach the desired value plus the number of $m \times \sigma$ ranges the old observation is distant from the new one.
Figure 6 describes the concept.



Pointer movements	Complexity
0	1 + log(d - d_1)
1	1 + log(d - d_2)
2 (10)	2 + log(d - d_3)
3 (11)	2 + log(d - d_4)
4 (100)	3 + log(d - d_5)
5 (101)	3 + log(d - d_6)
6 (110)	3 + log(d - d_7)
7 (111)	3 + log(d - d_8)
8 (1000)	4 + log(d - d_9)
...	4 + log(d - d_10)
...	4 + log(d - d_11)
...	4 + log(d - d_12)

Figure 6: A representation of the complexity given by the memory stack of numerical features.

4 Implementation and Testing

4.1 The tool

A proof-of-concept implementation of the tool, available at https://github.com/giacomo270197/RP1, was made in Python.

The tool is completely domain-independent. It accepts CSV files as an input, and the following configuration is needed:

- Categorical features: which values, in the CSV line, represent categorical features. In the case of the DARPA dataset, this would be the source IP and the destination IP.
- Numerical features: which values, in the CSV line, represent numerical features. In the case of the DARPA dataset, this would be the length and the info.
- Object prototype focus: which value will be used to decide which prototype an entry belongs to. In our specific case that is the protocol, but it could have been any other categorical value. Of course this assumes the decision of bounding object prototypes to a single categorical dimension.

The prototype definitions must be passed as arguments in a JSON format file. Tools for generating prototypes from CSV files are also provided and are also domain-independent.
It is also possible, however, not to pass any prototype, and let the tool learn what is normal and what is not as it executes. While this saves the time needed to generate prototypes, it will also cause poor performance until a certain maturity is reached.

The tool works by analysing each line of the input sequentially and computing the unexpectedness value for each feature. The value is computed by quantifying generation complexity and description complexity as described above in Section 2, and applying Formula 1.
After that, the unexpectedness values are summed, and, if the result reaches a user-defined threshold, an alert is generated.

If a feature prototype for a dimension is observed that cannot be found in the binary tree for that dimension (i.e. a new source IP address for a protocol), the generation complexity is set to infinity, and the description complexity to zero, as the observation is infinitely hard to generate (like, for example, a lottery draw containing a letters instead of numbers). The new feature prototype is then added to the tree.
However when a new object prototype is observed (i.e. a new protocol) no action is taken other than providing a message stating that manual intervention is needed.

While the tool is domain-independent, domain-specific knowledge could improve performance in some circumstances. For example, a new source IP address for a common protocol like TCP is no cause of concern. However, the tool (and Simplicity Theory) will see it as a state infinitely hard to generate. Therefore, infinitely unexpected.
Because of these kinds of situations, we added the possibility of creating situation-specific plugins, which are allowed to modify the final unexpectedness results. A user could, for example, set the unexpectedness value for the observation of a new source IP address to zero.

4.2 Testing and Results

To test our proof-of-concept we focused on the internal network captures of the DARPA dataset.
Prototypes were generated by the observation of the in captures internal week one and three (attack free). Testing was carried out on the internal captures of the fourth week.
To evaluate the results, we considered an alert as a true positive, and an attack as detected, if it was given during a time were an attack was taking place.

On average, the tool finds 96.4% of the attacks.
Out of all the alerts, 80.6% are true positives.

While the percentage of attacks detected is satisfactory, the number of false positives is relatively high, with one alert out of five being a misfire. However, an explanation for this behaviour can be hypothesised.

1. The Simplicity Theory aims at describing expectedness, hence normality. While an argument can be made that attacks cause anomalies, and should be detected, that does not imply that anomalous states can only be generated by attacks. For example, many new IP addresses were used in the fourth week, that could not be observed in the previous week. By incorporating these addresses at the bottom of the binary tree, an assumption is made that those addresses are somewhat rare. But that is of course not always the case.
2. The tool build is meant to discover abnormality in a stream of events, rather than on a single event basis. Behaviours, more than single lines, can be perceived as abnormal. It is therefore hard to pinpoint exactly which alerts refer to which attack. A simple temporal approach, the one we took, does not take into account the fact that an attacker could cause abnormality to be detected even after the attack is concluded.
3. Again, no domain-specific knowledge is used. This takes a toll on true to false positives ratio since there is no specialised knowledge to tell which abnormal events should be no cause of concern.

One more argument that could be made against the results is that many attacks were carried out each day of week four and five in the DARPA dataset.
However, Figure 7, alerts generated per minute on Monday, shows that the alert generation pattern closely resembles the attack pattern for that specific day.
Specifically, we see no alerts before 08:18, which is when the first attack is carried out. Moreover, flat zones can be observed when no attacks are carried out:

- From 09:39 to 11:15,
- From 16:32 to 18:24,
- From 18:27 to 19:50,
- From 20:03 to 21:34.

It is also worth noticing how, although in Figure 7 the number of alerts is capped to 2000, for plotting purposes, the number would be much higher at 09:08 and 21:34 (6242 and 55050 respectively). This makes sense, since two denial of service attacks were carried out at those times, and therefore many entries were flagged as anomalous.

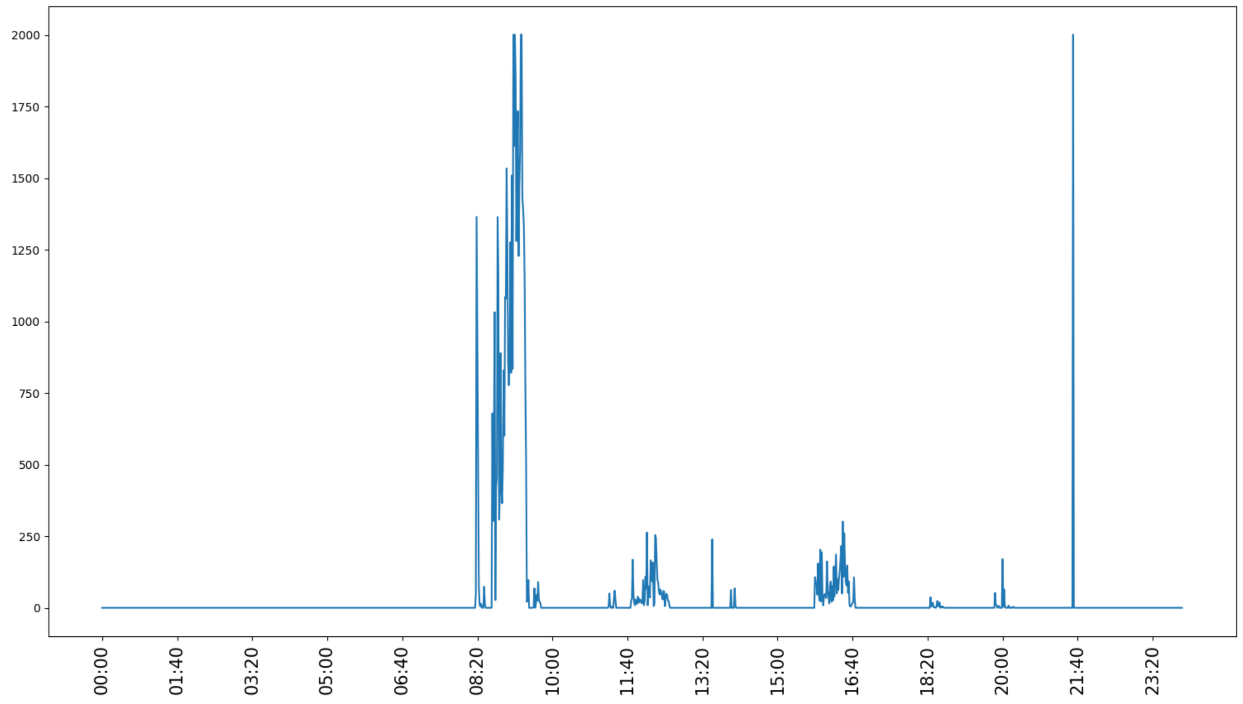


Figure 7: Number of alerts per minute.

5 Conclusions and future work

We wanted to asses how anomaly detection done via Simplicity Theory compares to other artificial intelligence methods. We compared the result of our proof-of-concept with [17], where Kohonen self-organizing map (K-Map) is used, with [18], where the authors apply Robust Support Vector Machines (RSVMs) and finally with [16] where n-gram analysis is used. All these papers use the DARPA dataset.

Our accuracy (96.4%) is comparable with the aforementioned papers, which settle on average around 90% to 94%.
The number of false positives, however, is much lower when using artificial intelligence methodologies, usually between 1% and 3%.

We believe the followings to be the major causes of our high ratio of false positives.

- We detect anomaly at a behaviour level, rather than on a per-event basis. Of course, anomalous behaviours can not only necessarily be caused by attacks, but our tool would still recognize them.
- Given the fact that many attacks occur in a relatively small amount of time, the concept of normality could be erroneously shifting as events are analysed.
- No networking knowledge is used to filter out non-worrisome network traffic anomalies.
- Little attention was paid to which features were observed, and how information was extracted from the dataset.

We believe that future work addressing these shortcomings could significantly improve the performance of our proof-of-concept.

References

[1] J.-L. Dessalles, "Algorithmic Simplicity and Relevance", *Algorithmic Probability and Friends. Bayesian Prediction and Artificial Intelligence Lecture Notes in Computer Science*, pp. 119–130, 2013.

[2] Dessalles, J.-L. (2008), "La pertinence et ses origines cognitives - Nouvelles théories", *Hermes-Science Publications, Paris*, http://pertinence.dessalles.fr.

[3] Schmidhuber, J, "Simple Algorithmic Theory of Subjective Beauty, Novelty, Surprise, Interestingness, Attention, Curiosity, Creativity, Art, Science, Music, Jokes", *Journal of SICE 48(1), 21–32 (2009)*, http://www.idsia.ch/ juergen/sice2009.pdf

[4] Dessalles, J-L., "Emotion in Good Luck and Bad Luck: Predictions from Simplicity Theory", *Proceedings of the 32nd Annual Conference of the Cognitive Science Society*, Portland, OR: Cognitive Science Society.

[5] N. Wang, J. Han, and J. Fang, "An Anomaly Detection Algorithm Based on Lossless Compression", *International Conference on Networking, Architecture, and Storage*, 2012

[6] Aric Hagberg and Alex Kent and Nathan Lemons and Joshua Neil, "Credential hopping in authentication graphs", *in 2014 International Conference on Signal-Image Technology Internet-Based Systems (SITIS). IEEE Computer Society*, Nov. 2014

[7] E. Bonabeau and J.-L. Dessalles, "Detection and emergence", *Intellectica. Revue de lAssociation pour la Recherche Cognitive*, vol. 25, no. 2, pp. 85–94, 1997.

[8] K. Veeramachaneni, I. Arnaldo, V. Korrapati, C. Bassias, and K. Li, "AI^2: Training a Big Data Machine to Defend", *2016 IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS)*, 2016.

[9] Min Du and Feifei Li, "Spell: Streaming Parsing of System Event Logs", *Proc. IEEE International Conference on Data Mining (ICDM). 859–864*, 2016

[10] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog", *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security - CCS 17*, 2017.

[11] V. I. Levenshtein, "Binary codes capable of correcting deletions insertions and reversals" *Soviet Physics Doklady* pp. 707-710, 1966.

[12] J. W. Haines, R. P. Lippmann, D. J. Fried, M. Zissman, E. Tran, "1999 DARPA Intrusion Detection Evaluation: Design and procedures" *MIT, Lexington Lab Tech. Rep.,* 2001.

[13] C. Callegari, S. Giordano, M. Pagano, "Anomaly detection: An overview of selected methods" *2017 International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON)* 2017

[14] Y. Wang, B. Xue, L. Wang, H. Li, L. Lee, C. Yu, M. Song, S. Li, C. Chang, "Iterative anomaly detection" *2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*,

[15] F. Harrou, B. Bouyeddou, Y. Sun, B. Kadri, "Detecting cyber-attacks using a CRPS-based monitoring approach" *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*

[16] Wang, Ke, and Salvatore J. Stolfo. "Anomalous Payload-based Network Intrusion Detection." *Lecture Notes in Computer Science Recent Advances in Intrusion Detection*, 2004, pp. 203–222., doi:10.1007/978-3-540-30143-1_11. doi:10.1007/978-3-540-30143-1_11.

[17] Sarasamma, S.t., et al. "Hierarchical Kohonenen Net for Anomaly Detection in Network Security." *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, vol. 35, no. 2, 2005, pp. 302–312., doi:10.1109/tsmcb.2005.843274.

[18] Wenjie Hu et al. "Robust Support Vector Machines for Anomaly Detection in Computer Security." *ICMLA, ,* 2003.

[19] Huang, Z. "Clustering large data sets with mixed numeric and categorical values", *Proceedings of the First Pacific Asia Knowledge Discovery and Data Mining Conference*, Singapore, pp. 21-34, 1997.