Attack-Test and Verification Systems, Steps Towards Verifiable Anomaly Detection

Marcel Fourné, Dominique Petersen, Norbert Pohlmann

Institut für Internet-Sicherheit
Westfälische Hochschule
Neidenburger Str. 43
45897 Gelsenkirchen
{fourne,petersen,pohlmann}@internet-sicherheit.de

Abstract: Botnet, network malware and anomaly detection algorithms are hard to evaluate and compare against each other due to different data sets. In some cases over-specialization on known malware gives high detection rates due to unknown artifacts in the training data set. This may lead to new malware being unnoticed on a network, because the detection algorithm has not been optimized for this case.

Our proposal is a new and work-in-progress approach to generate parametricized and randomized testing data sets on the fly. We plan to couple this with the an automatic verification system to assess the quality of detection algorithms without internal knowledge of their working.

We hope to encourage discussion to enhance the draft of our idea and especially to go into more detail on our work in progress.

1 Introduction

The research field of Network Anomaly Detection (NAD) and more specifically network malware – mainly botnet – detection is a hot field with many approaches to find ever new algorithms. Many events can be classified as network anomalies, but to qualify as an attack, there has to be some form of intent. There is a semantic gap between anomalies and attacks, since the intent of attacks must be understood to build a good model, which then builds the basis for pattern detection of network activities related to an attack. If the scope of detection lies only in the network, intent is hard to quantify, but can be concluded from abnormal statistics created by events impeding the functions of benign network activities. A denial of service attack can be found quite easily, but algorithms which find small anomalies predating targeted attacks are harder to develop. Algorithms to find botnet activity which predates attacks from botnets are in the class of NAD algorithms, but have correlation criteria for anomalies related to botnet spreading — or, more general: attack preparation. These have been optimized to almost perfection in finding the anomalies which were known during their creation. Other algorithms have been built which need continuous training, but no clear statements to the quality of their detection ratio can be made for unforeseen network events. Some algorithms are just tested using their training

data set, so over-optimization on some anomalies in this set can lead to detection rates approaching 100% while still failing in the presence of new malware samples in the wild. To approach this herculean task in finding one algorithm – or even a mixture of algorithms – which detect new threats "best", we conceived a system to quantify algorithm detection performance. These algorithms can be anomaly detection algorithms, attack detection algorithms or both, since our system will be able to replay any kind of network event.

We like to present our approach and show our rationale behind our new ideas to get some feedback from other researchers to cover aspects which may have different relevance criteria than we have thought of so far.

2 Background

In the network anomaly detection field we have commercial, research and self-taught parties trying to find malware. We can group them based on their detection approaches in two groups:

1. NAD systems using fixed signatures of anomalous events and malware
2. NAD systems using complex algorithms and communication pattern recognition

Systems using approach number 1 find new and previously unencountered malware with a lower chance, so most research projects do not rely solely on this. In this work we will not focus on systems with approach number 1, but we will try to aid development of systems using complex algorithms like those based on machine learning and pattern recognition which need a training phase before their deployment. This does in no way prohibit the system from testing signature-based NAD systems.

The quality of network traffic captures and other sources of training data for detection algorithms is a key point which can make or break fine tuned detection algorithms.

Raw network packet captures – also known as PCAPs – are the most detailed format for analysis of network events. They are also one of the most problematic ones, due to privacy protection laws of different jurisdictions. Even unintentional reading of network traffic may compromise third party data, so using PCAPs as a data source for general detection usage poses a big risk.

On the other end of the spectrum, standardized NetFlow data as generated by many unmodified commercial routing soft- and hardware contains much less information. Up to OSI layer 4 protocol data is commonly logged together with usage statistics, grouped by each network flow. This does not solve the privacy protection problem, since IP-addresses are contained in these statistics, so there are much less data but still too much information according to privacy protection laws.

Both are common inputs for training of NAD systems, but both exhibit a certain flaw: They are stale data, containing at most network anomalies which were exhibited in the wild until the date of their capture. Any NAD system which is trained on such a data set

will not be trained against anomalies unknown until then and therefore may miss them.

An approach to fix this is to have different data sets, one for training and one for testing of an algorithm. This alleviates the problem partially, at the cost of storing more data sets and using only a part of them for training – thereby wasting training potential.

We will try to fix this by building a quality assessment framework before thinking about detection algorithms. The test should not be geared toward achieving good results with a certain algorithm, so no knowledge of internal workings of the algorithms may be used by the verification of detection results.

3 Related Work

Our network simulation approach was directly influenced by the work of Guanhua [Yan05], who developed a model to simulate networks using different levels of detail to build an abstraction over single packet injection.

Injecting network attacks from a database was an approach used by Leszczyna et al. in [LFM08], which we took as a base for our design of a high performance network simulator. Our aim is not only testing malware detection algorithms but also generating clean network data for training.

Gorecki et al. also used a different approach for their network malware activity simulator in [GFKH11] for dynamic malware analysis and a reduction in malware volume for manual analysis. They employ a simulation of the internet for local malware, going into more detail for each service, good enough to fool the malware into behaving like being connected to the real internet. From this we take the necessity to model not only local network activity, but also traffic from the internet for a more real simulated network environment.

Since the DARPA evaluation of intrusion detection systems in 1998 and 1999 [LHF+00a] there has been an ongoing discussion on the validity of modeled traffic as well as the quality of network traffic captures [LHF+00b, MC03, 20000]. From a theoretical perspective, artifacts are impossible to eliminate entirely, since the behavior of networked software is also Turing complete. The most recent collection of key points and problems in this approach by Rossow, Dietrich et al. [RDG+12] gives criteria for building prudent experimental setups to minimize statistical biases.

Instead of only simulating TCP and UDP traffic like Krishnaswamy did for his Wormulator in [Kri09] to model rapidly spreading network malware, we intend to model any type of raw network packet as long as they can be taken apart into network flows.

Detection algorithm verification is an important part in almost any development of such algorithms, commonly used data sources being either preprocessed NetFlow data from real networks or full network PCAPs. Two examples for this are the works of Bilge et al. in [BBR+12] to detect C&C Channels for the first type using heuristics and another work of Bilge et al. on finding DNS fast flux traffic in [BKKB11] using full network data to find a detailed criterion for decisions.

The Internet Analysis System (IAS) [PP06, PHBP08] of the Institute for Internet Security [IfIS] is another NAD system, taking a counting based detection approach. This system can capture as much detection relevant data as a direct network packet capture, but compresses and pseudonymizes these by only taking distinct network descriptors and counting their occurrences. By not transmitting verbatim IP-addresses or application layer payload data, the IAS has been designed to be compliant with different privacy protection laws.

We decided to use the IAS to make use of detailed network protocol data as well as its new flow data format as developed in the iAID research project [IfIS12], its aim being network anomaly detection on a smaller scale. Described in the work of Petersen and Pohlmann in [PP13] is the idea of a NAD system at an internet scale, so their work would be relevant to our own research. This chosen setup should provide enough data for our work and we could benefit from their custom data format, since we want to provide a detailed network traffic simulation for developing NAD algorithms capable of working on the network links of internet service providers.

Rossow et al. published their Sandnet [RDB+11] to build a database of malware and other anomalous traffic by simulating vulnerable hosts on virtualized hardware in a confined and monitored network environment. Their findings and data allow us to outsource the gathering of network malware traffic samples, so we do not have to duplicate others efforts.

4 Approach

The idea is to have pseudorandom traffic based on a technology mix and different user behavior, indistinguishable from real and live network traffic but known to be clean of anomalous events until chosen to contain them. Using such a data set of a size limited only by researchers choice can lead to better training results and at the same time can be used for continuous verification.

We will describe one of the ideas we have as an example to achieve this goal, if only to start discussion. The results of the DARPA IDS evaluation are of big influence to our decisions, but since this is only the draft of a system, we would like to postpone rigorous analysis of future results and only mention potential solutions to structural problems.

The task of generating good enough network traffic can be split in two parts: The initial modeling of the network traffic and the verification of it.

Our design for the Injector and Verifier together with the classic Internet Analysis System can be seen in Figure 1. Of course any NAD system can implement the interfaces, so our system is not constrained to just test the detection rate of the IAS. The box in Figure 1 can be seen as a black box as long as their reports use a standardized reporting format. Of special importance is the network bus on which internet access is possible, but not necessary for our testing setup.

All network traffic data can be simulated by the Injector from its database, including network malware activity and other attacks. We do not plan to mix the generated traffic with live network traffic for training of machine learning algorithms, since this approach has

been shown to lead to unforeseeable results.

On the same network bus the Internet Analysis System can get its input, which is taken apart into the Detection Engine and its output in turn forwarded to our Verifier. The out of band communication channel between the Injector and Verifier is used for signaling of injection times of attack events on the network traffic, so the Verifier can get exact timings for the processing times of the classic Internet Analysis System.

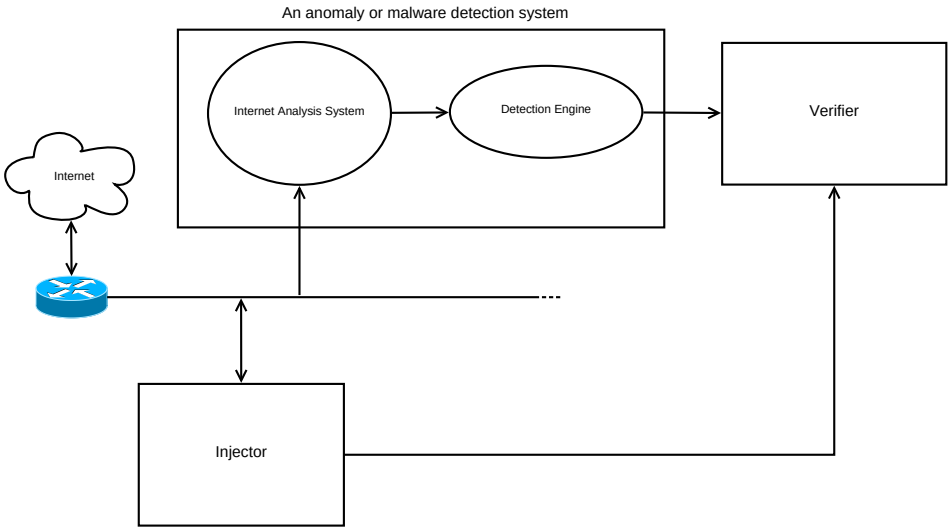

Figure 1: Interaction of the Injector, a NAD system (the IAS for example) and the Verifier, Internet access optional

First, we want to define what we mean by the term verification: When looking at general software development, unit tests have proven themselves as a common way to check for errors. During their creation, a narrow test-case is defined which should result in a specified behavior of the software. We plan to create a test-case harness for NAD algorithms by defining the exact times when such an algorithm should signal a detection.

This leaves room for error, since the behavior is not exactly specified for the time when no anomaly is injected, so our view of verification is less strict than many formal models. This is a goal of our work, since the generation of network traffic is not as perfect as real, live network captures.

1. All algorithms declare everything as an anomaly
2. Some algorithms find a number of anomalies
3. No algorithm finds any anomaly

From these cases number 1 and 3 are clearly an indicator for something being wrong in our setup. To exclude these cases, the test setup can be enhanced by two very special

algorithms: An "always report anomaly on network traffic" and a "never report anomaly on network traffic" algorithm. Both are never meant to really find anomalies and malware, but they make cases number 1 and 3 impossible. Since the detection algorithms generate results from the network events, the verifier does not directly receive network traffic.

The two constant output algorithms serve as a source of events, even when other algorithms do not generate them. This is a way of proving liveness in the NAD system without changing other algorithms.

For a more complete setup, we have to combine the answers of these two anomalous anomaly detection algorithms with the answers of the real anomaly detection algorithms. This is only an example and by no means the best approach for every NAD system, because other systems under inspection of our attack-test and verification system may use other ways to eliminate systematic errors. All our Verifier cares about are the final event reports and its own internal communication with the Injector, the other components being third parties.

We expect to see a data set with at least two outliers, all real detection algorithms in between. If the number of outliers is only 1, we have very anomalous generated network traffic, in the sense of either no anomalies or everything anomalous. With this, we can check against 3 cases of something having gone wrong:

- No network traffic, nothing reported
- Only 1 outlier, no anomalies in the generated network traffic
- Only 1 outlier, everything anomalous

This is a further step toward narrowing down on the quality of the modeled traffic, since we can now say for sure that some traffic was generated and analyzed without looking into the NAD system. Having ruled out the borderline cases, we can look at the distribution of reported events, which is more fruitful as a topic of its own. The goal is to have a ranking among the algorithms according to their fitness for detection, based on as little inside knowledge of the algorithms as possible.

As a basis, we will showcase the input side of the system first before going into more detail of the verification.

4.1 Modeling network traffic

In short recapitulation, we will be modeling network traffic based on captured traffic data and recombining individual preprocessed PCAPs based on run-time-configurable parameters. Some of the parameters and their mix can be technical in nature, like the type of browser used to surf the web, the type of target system or time-based, like the bandwidth used. The last one can depend on simulated day or night cycle, weekends, mornings... all these show significant changes in network traffic behavior. All these and many others can be implemented to test under which conditions the detection rate changes.

As a basic unit of inspection, we will not be using a single packet but a single network flow, composed of a number of packets which would not form a coherent communication stream if taken apart. For simplicity, we will take a bidirectional communication stream as two distinct but associated network flows, which of course are processed together during the preprocessing stage. This simplifies any search and replace for pseudo-randomization, since it can be done in two separate steps for each direction of the communication stream to be injected later into our data set.

All injected traffic will be preprocessed to behave like normal traffic, including corresponding addresses, sequence numbers and other attributes. Of course, this can not be guaranteed in every combinatoric detail, but we will employ manually tagged traffic and compare its mixture to real network traffic to minimize the number and impact of artifacts.

We will generate a mix of network traffic using an even higher level of abstraction, since the choice of network flows becomes unwieldy for higher data rates and it can be automated just as easily. As parameters of choice, the total network usage and the shares of it for each major class of network traffic should be enough to enable a simpler – maybe random – choice algorithm to select the network flows to inject.

The details lie in the classification of the classes of network traffic, but this can be done manually beforehand. The database of traffic flows can even be extended later on without interrupting operation.

At this point, all traffic will ideally be clean of anomalies by virtue of all sources going into the modeled traffic being manually cleaned of any conflicting events. A crucial part lies in the scope of the cleaned anomalies, since random errors will always be in real network data but only seldom signify malicious intent.

The next step is in modeling anomalies. Pure, random anomalies such as bit flips are like untargeted fuzzing at the network level, which may be a test for corner cases of network stacks but do not lead to our intended goal.

Targeted anomalies are harder to model, so we will be using a database of attacks and inject one of them as a single event, the time of it being synchronized to the later verification stage.

As our source of attacks, we will take the database from Rossow et al. as built from their Sandnet [RDB+11].

4.2 Verification of detection algorithms and network traffic

While having injected anomalies at synchronized times into our data set, we will be measuring the output – if any – and delay of detection algorithms.

The hard part is not the evaluation of the algorithms, although it is the main goal in later work. We need high quality input for machine learning based NAD algorithms not only in testing but also during their training phase. This is a core idea of the Verifier, the verification of detection algorithms as well as their surrounding environment.

To effectively monitor the detection algorithms, it is also necessary to monitor the quality of their input – "garbage in, garbage out"[1], so when implementing a verifier for the algorithms, we have indirectly built a verifier for the generated network traffic.

Our system is universal in nature, allowing flexible extensions as this is only thought as an example for implementation.

We have our set of detection algorithms and their two supporting false algorithms like described in section 4, so we can concentrate on finding a good distribution of results the detection algorithms found. An example could look like the one seen in Figure 2 for two detection algorithms, their raw and unscaled outputs without threshold values. Of course, only using one detection algorithm other than the two known erroneous ones, we have no meaningful distribution. Even if we have a bigger set of results from real detection algorithms which trigger all at the same events, we have no chance to say anything more than it was possible from the method in section 4.

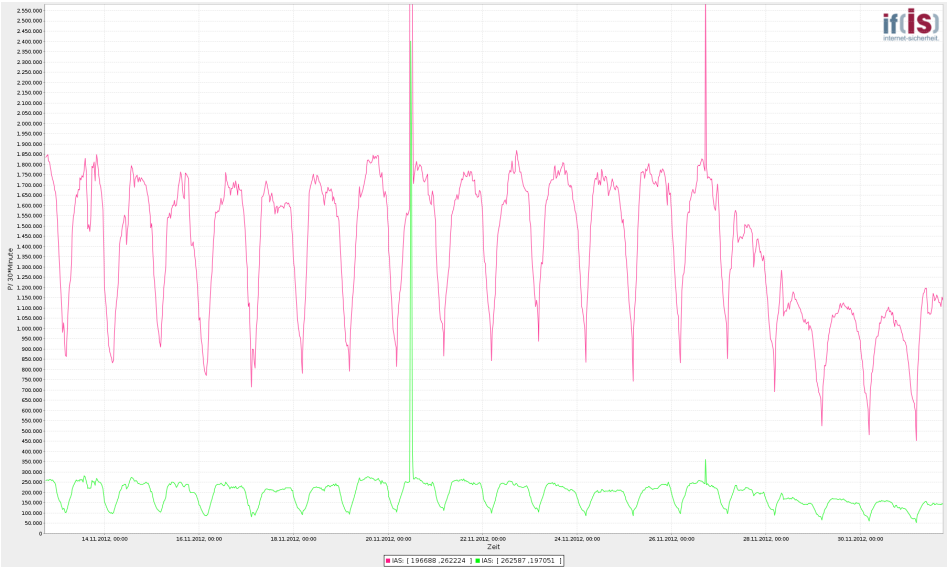

Figure 2: The current state of the IAS and its event diagrams

Assuming a subset of real network detection algorithms having in itself different behavior under input of a set of network events – with anomalies – we aim to have a ranking among the algorithms. Using the delay until detection, we have another indirect metric to their performance, the faster algorithms necessarily being less hardware intensive.

The best algorithm is of course one which detects all anomalous events using as little delay as possible at any injection rate, but any one approaching this goal should be a good

[1]This concept has been known since Charles Babbages wrote in [Bab64]: "On two occasions I have been asked, "Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?" ... I am not able rightly to apprehend the kind of confusion of ideas that could provoke such a question."

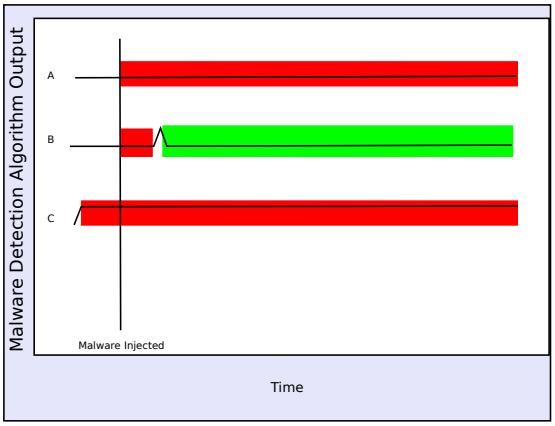

Figure 3: An idea for a possible output of the Verifier

starting point.

Erroneously labeled events can be tracked separately for each algorithm and should be weighted accordingly for final results. Depending on the usage scenario, either as a pre-filter or as a final indicator, the weight should be lower or higher than that for missed malware events, since the last scanner in a chain should be tasked to prevent false-positives. It should be a front-end task to appease the user and a back-end should report as much information as possible. The Verifier is not planned as a malware event notification framework, so we are free to choose our weighting functions without bias.

Possible output for the Verifier can be seen in Figure 3. We have three detection algorithms under supervision, aptly named A, B and C. We have plotted their detection results over time and marked the injection of a malware network event. Algorithm A does not detect anything, so after the injection we mark its results as erroneous. B is a classic successful algorithm which detects the event after a short time, so in the meantime we mark its output as red, but after its detection we have seen it to be correct, so this is changed to green. Detection algorithm C is an example for our "always detect" check. Its detection before the injection of the malware event is an error, so even after the injection we do not want to mark it as a successful algorithm.

The behavior showcased in Figure 3 is not final, since we have not modeled an acceptable failure rate, but it should be enough to see where we want to go with our ideas.

The special idea of the Verifier is that of a black box test, where no detail of the detection algorithm is needed to be known to evaluate its overall performance. This misses special criteria like detailed hardware resource usage, but these can influence later choices after their main qualification as good detection algorithms, so the separation of the Detection Engine and the Verifier – maybe even on different host systems – makes it conceptually clearer that our main aim is correct, stable and fast malware detection.

5 Conclusion

We implore others to collaborate with us on our aim to test and verify NAD systems as rigorously as possible to achieve stable network anomaly and malware detection.

We have demonstrated our new concept to model clean network traffic and inject anomalous events at chosen points in time. The design is geared towards generating a high-quality network simulation, while maintaining a high possible throughput by working on whole network flows instead of single packets, reducing processing overhead. A mixture of these can be injected according to parameters chosen by researchers for their algorithms, building a custom network utilization to model different usage scenarios.

The other aspect of our work is a new way to check the results of different network anomaly detection algorithms against previously unknown and untrained network malware. Using this, we will also be able to verify the quality of our generated network traffic using simple and later on more complete quality criteria. With data from real network dumps we are able to enhance our testing criteria for the generated network traffic and cross-check our setup to find small statistical aberrations which could influence training of machine learning algorithms.

The inherent problems around simulation artifacts are known to us and we do not claim to solve all statistical biases, but make it easier to generate a model for them.

The Injector and Verifier will make it easy for us to experiment with new detection algorithms and to change usage scenarios in our testing network before any deployment in different real networks. This is not a replacement for testing in real networks, but it enhances the research and development of detection algorithms before their deployment in these networks.

6 Future Work

We aim to get feedback on our presented ideas and to incorporate these into our design before we have finalized every aspect. Of course we will be implementing our ideas during the course of the next months. This work has already begun, so we look forward to have this system in place to aid testability for new algorithm designs and their implementations.

We will define configurable parameters to set threshold values for detection and test them over longer stretches of time. It should be possible to automate this to run over some days or longer to find optimal points in the parameter sets for each detection algorithm for each type of network traffic structure.

As our next step, we will develop new botnet detection algorithms and use the Injector and Verifier to compare their detection performance. We plan to detect malware on the network, which has not been detected beforehand due to different positioning of sensors. Our setup aims to simulate high-bandwidth internet links, so new algorithms will be developed to achieve good malware detection rates in this setting.

References

[20000] Testing Intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory. *ACM Trans. Inf. Syst. Secur.*, 3(4):262–294, November 2000.

[Bab64] Charles Babbage. *Passages from the life of a philosopher*. 1864.

[BBR+12] Leyla Bilge, Davide Balzarotti, William Robertson, Engin Kirda, and Christopher Kruegel. Disclosure: detecting botnet command and control servers through large-scale NetFlow analysis. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 129–138. ACM, 2012.

[BKKB11] Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. Exposure: Finding malicious domains using passive dns analysis. In *Proceedings of NDSS*, 2011.

[GFKH11] Christian Gorecki, Felix C Freiling, Marc Kührer, and Thorsten Holz. TRUMANBOX: improving dynamic malware analysis by emulating the internet. In *Stabilization, Safety, and Security of Distributed Systems*, pages 208–222. Springer, 2011.

[IfIS] Institute for Internet Security. http://www.internet-sicherheit.de/.

[IfIS12] Institute for Internet Security. iAID: innovative Anomaly- and Intrusion-Detection. http://www.iaid-project.org, 2012.

[Kri09] Jyotsna Krishnaswamy. *Wormulator: Simulator for Rapidly Spreading Malware*. PhD thesis, San Jose State University, 2009.

[LFM08] Rafał Leszczyna, Igor Nai Fovino, and Marcelo Masera. MAlSim: mobile agent malware simulator. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, page 35. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[LHF+00a] Richard Lippmann, Joshua W Haines, David J Fried, Jonathan Korba, and Kumar Das. The 1999 DARPA off-line intrusion detection evaluation. *Computer networks*, 34(4):579–595, 2000.

[LHF+00b] Richard Lippmann, Joshua W Haines, David J Fried, Jonathan Korba, and Kumar Das. Analysis and results of the 1999 DARPA off-line intrusion detection evaluation. In *Recent Advances in Intrusion Detection*, pages 162–182. Springer, 2000.

[MC03] Matthew V Mahoney and Philip K Chan. An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection. In *Recent Advances in Intrusion Detection*, pages 220–237. Springer, 2003.

[PHBP08] Dominique Petersen, Kilian Himmelsbach, Sascha Bastke, and Norbert Pohlmann. Measuring and warning, 2008.

[PP06] Norbert Pohlmann and Marcus Proest. Internet Early Warning System: The Global View. In *Securing Electronic Business Process*. ISSE, Vieweg, 2006.

[PP13] Dominique Petersen and Norbert Pohlmann. An ideal internet early warning system. In *Advances in IT Early Warning*, pages 9–20. Fraunhofer AISEC, 2013.

[RDB+11] Christian Rossow, Christian J Dietrich, Herbert Bos, Lorenzo Cavallaro, Maarten van Steen, Felix C Freiling, and Norbert Pohlmann. Sandnet: Network traffic analysis of malicious software. In *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, pages 78–88. ACM, 2011.

[RDG+12] Christian Rossow, Christian J Dietrich, Chris Grier, Christian Kreibich, Vern Paxson, Norbert Pohlmann, Herbert Bos, and Maarten van Steen. Prudent Practices for Designing Malware Experiments: Status Quo and Outlook. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 65–79. IEEE, 2012.

[Yan05] Guanhua Yan. *Improving Large-Scale Network Traffic Simulation with Multi-Resolution Models*. PhD thesis, Dartmouth College Hanover, New Hampshire, 2005.