BETH Dataset: Real Cybersecurity Data for Unsupervised Anomaly Detection Research

Kate Highnam*†[1,2], Kai Arulkumaran*[1,3], Zachary Hanif*[4], and Nicholas R. Jennings[5]

[1]Imperial College London
[2]The Alan Turing Institute
[3]ARAYA Inc.
[4]University of Maryland, College Park
[5]Loughborough University

Abstract

We present the BETH cybersecurity dataset for anomaly detection and out-of-distribution analysis. With real "anomalies" collected using a novel low-level tracking system, our dataset contains over eight million data points tracking 23 hosts. Each host has captured benign activity and, at most, a single attack, enabling cleaner behavioural analysis. In addition to being one of the most modern and extensive cybersecurity datasets available, BETH enables the development of anomaly detection algorithms on heterogeneously-structured real-world data, with clear downstream applications. We give details on the data collection, suggestions on pre-processing, and analysis with initial anomaly detection benchmarks on a subset of the data.

1 Introduction

When deploying machine learning (ML) models in the real world, anomalous data points and shifts in the data distribution are inevitable. From a cyber security perspective, these anomalies and dataset shifts are driven by both defensive and adversarial advancement. To withstand the cost of critical system failure, the development of robust models is therefore key to the performance, protection, and longevity of deployed defensive systems.

Current research into the robustness of ML models tends to be based on modifying common datasets, and extrapolating performance to disparate anomaly detection applications. For out-of-distribution (OoD) estimation, researchers combine pairs of existing datasets, such as MNIST-FashionMNIST, CIFAR10-CelebA, or CIFAR10-ImageNet32 [21], to mimic data distribution shift. Other data for evaluating robustness is constructed by modifying samples within datasets; for example, corrupting, perturbing, or shifting samples [6, 22] to generate anomalies. These primarily use image datasets [13, 3, 31, 10, 27, 17], and sometimes text datasets [14, 12, 7]. Such experiments are said to be motivated by domains such as security, but it is unknown exactly how well new methods—in particular, those centred around deep learning (DL)—may generalise beyond these input modalities in other real world applications. The difficulty with real datasets is that complexity and domain expertise remain a barrier to entry for ML researchers.

In this paper, we present the BPF-extended tracking honeypot (BETH) dataset[1] as the first cybersecurity dataset for robustness benchmarking in unsupervised anomaly detection. Collected

*Equal Contribution
†Corresponding Author: kwh19@ic.ac.uk
[1]Available at `https://www.kaggle.com/katehighnam/beth-dataset`

Proceedings of the Conference on Applied Machine Learning for Information Security, 2021

using a novel honeypot tracking system, our dataset has the following properties that make it attractive for the development of robust ML methods: 1) at over eight million data points, this is one of the largest cyber security datasets available; 2) it contains modern host activity and attacks; 3) it is fully labelled (for verification); 4) it contains highly structured but heterogeneous features; and 5) each host contains benign activity and at most a single malicious user, which is ideal for behavioural analysis and other research tasks. In addition to the described dataset, further data is currently being collected and analysed to add alternative attack vectors to the dataset.

There are several existing cybersecurity datasets commonly used in ML research: the KDD Cup 1999 Data [8], the 1998 DARPA Intrusion Detection Evaluation Dataset [11, 15], the ISCX IDS 2012 dataset [29], and the NSL-KDD dataset [30]. The KDD'99 data is an abstracted view of network activity from the DAPRA'98 data; the NSL-KDD dataset is the same as the KDD'99 Data after, primarily, removing duplicates. Each includes millions of records of realistic activity for enterprise applications, with labels for attacks or benign activity. The KDD1999, NSL-KDD, and ISCX datasets contain network traffic, while the DARPA1998 dataset also includes limited process calls. However, these datasets are at best almost a decade old, and are collected on in-premise servers. In contrast, BETH contains modern host activity and activity collected from cloud services, making it relevant for current real-world deployments. In addition, some datasets include artificial user activity [29] while BETH contains only real activity. BETH is also one of the few datasets to include both kernel-process and network logs, providing a holistic view of malicious behaviour. Refer to Table 1 for further comparison between BETH and these other cybersecurity datasets.

This paper begins with a description of the data collection process and the relevance of the available features. We then perform an analysis of the first set of kernel-level process logs collected, including anomaly detection benchmarks[2]. Our benchmarks include both traditional baselines [26, 28, 16], as well as a state-of-the-art deep-learning-based method [21]. In summary, the isolation forest (iForest) [16] archives the highest area under the receiver operating characteristic (AUROC) on the initial, labelled subset of our data. We believe the scale and range of attacks available in our full dataset will pose a challenge for all current anomaly detection methods.

2 The BETH Dataset

The BETH dataset currently represents 8,004,918 events collected over 23 stand-alone honeypots, running for about five non-contiguous hours on a major cloud provider. For benchmarking and discussion, we selected the initial subset of the process logs. This subset was further divided into training, validation, and testing sets with a rough 60/20/20 split based on host, quantity of logs generated, and the activity logged. Only the test set includes malicious activity, as expected when training unsupervised anomaly detection models. Table 2 provides a summary of the dataset, while Table 3 and Table 4 provide a description of the kernel-process and DNS log features, respectively.

In this section, we first detail the log collection methodology, followed by a description of the overall dataset. The final subsection discusses potential research questions that could be investigated using our dataset.

2.1 Collection Methodology

The challenge of crafting a honeypot is two-fold: make it tempting enough to infiltrate, and track activity without being detected. The former is typically done by providing "free" resources to an attacker, i.e., easily accessible computer power. Our implementation currently runs hosts

[2] `https://github.com/jinxmirror13/BETH_Dataset_Analysis`

Table 1: Comparison of our BETH dataset against other cybersecurity datasets used in ML research. These attributes signify the ideal features of a robust model benchmark and reduce the barrier to entry for ML researchers looking for a real-world dataset. The attributes were manually verified and compared with Ring et al.'s [25] table of network-based datasets.

Dataset Name	Size	Kernel Processes	Re-created Traffic	Single User Activity	Simple Network Environment	Cloud Infrastructure
DARPA1998 [11]	Not Stated, 4GB compressed	✓	✓	✖	✖	✖
KDD1999 [8]	7+ million records	✖	✓	✖	✖	✖
NSL-KDD [30]	∼ 2 million records	✖	✓	✖	✖	✖
ISCX IDS2012 [29]	∼ 2 million flows & ∼ 81.1GB compressed	✖	✓	✖	✖	✖
BETH	∼ 8 million records	✓	✖	✓	✓	✓

Table 2: General characteristics of the kernel-process logs, including our initial benchmark subset.

Dataset	Length	% of subset	# of hosts
Training	763,144	66.88%	8
Validation	188,967	16.56%	4
Testing	188,967	16.56%	1
Subset Total	1,141,078	100%	13
Total	8,004,918	-	23

Table 3: The description and type of each feature within the kernel-level process logs, tracking every create, clone, and kill process call. Starred features were included in the model baselines and converted as described in Appendix A.

Feature	Description
timestamp	Seconds since system boot
processId*	Integer label for the process spawning this log
threadId	Integer label for the thread spawning this log
parentProcessId*	Parent's integer label for the process spawning this log
userId*	Login integer ID of user spawning this log
mountNamespace*	Set mounting restrictions this process log works within
processName	String command executed
hostName	Name of host server
eventId*	ID for the event generating this log
eventName	Name of the event generating this log
argsNum*	Length of `args`
returnValue*	Value returned from this event log (usually 0)
stackAddresses	memory values relevant to the process
args	List of arguments passed to this process
sus	Binary label as a suspicious event (1 is suspicious, 0 is not)
evil	Binary label as a known malicious event (0 is benign, 1 is not)

with a single `ssh` vulnerability: any password will be accepted to login. This is enough to entrap automated, naive SSH scanning and brute force methods. In the future we plan to deploy hosts with other vulnerabilities, with which we hope to observe other attack vectors.

To log activity in real time, each host runs Ubuntu 18.04 with a Docker container [20]. This encapsulates our two-sensor monitoring system utilising the extended Berkeley Packet Filter (eBPF) [5] which runs isolated programs in a Linux kernel sandbox to directly monitor the packets that it processes. Due to various checks and restrictions placed on the design of these programs, they can be placed in a live kernel without needing to restart it. The eBPF enables tracking of any user-level activity at pre-specified points.

The first sensor is set to listen and exfiltrate relevant data packets resulting from internal activity. In particular, this sensor tracks all Linux system calls to create, clone, and kill processes. The second sensor logs network traffic, specifically DNS queries and responses from all processes on the host machine, including those processes running within the hosted Docker containers. When the desired packet appears, it is parsed out to pre-defined fields and then transmitted to a collection server.

These process and network data packets are gathered on a separate node hosting a message queue server. Sensor clients connect to this server and transmit their individual data packets as events occur, without batching. This allows the collection and re-transmission of data at an atomic level. The data is then stored locally in two correlated files with respective logs, allowing analysts to understand both local and network effects that occur jointly.

2.2 Dataset Characteristics

The dataset is composed of two sensor logs: kernel-level process calls and network traffic. As the initial benchmark subset only includes process logs, this section only covers these; a description of the network logs can be found in Table 4.

Each process call consists of 14 raw features and 2 labels, described in Table 3. These largely contain categorical features with some containing large integers, necessitating further processing. Thus, for our benchmarking, we converted several fields to binary variables based

Table 4: The description and type of each feature within the DNS logs.

Feature	Description
timestamp	Date and time in the format "YYYY-MM-DDTHH:MM:SSZ" for when the packet was sent or received
SourceIP	Source IP address of the packet
DestinationIP	Destination IP address of the packet
DnsQuery	Sent DNS query (e.g. the URL submitted - "google.com")
DnsAnswer	DNS response; can be `null`
DnsAnswerTTL	List of integers sent as strings, can be null; The time to live of the DNS answer
DnsQueryNames	Name of the requested resource
DnsQueryClass	Class code for the resource query
DnsQueryType	Type of resource record (A, AAAA, MX, TXT, etc.)
NumberOfAnswers	Number of answer headers in the packet
DnsOpCode	Header information regarding which operation this packet was sent (e.g. standard query is 0)
SensorId	Same as the hostName in the process records; Name of host server
sus	Binary label as a suspicious event (1 is suspicious, 0 is not)
evil	Binary label as a known malicious event (0 is benign, 1 is not)

on field expertise, as described in Appendix A. We note that this conversion process creates more duplicates due to the removal of critical fields, such as `processName` and `args`, which have no widely-accepted encoding scheme for any machine learning model to utilise. There are some inherent duplicates which we keep within the dataset prior to conversion for the accuracy of real logs and to not limit users of the data.

Each record in the process logs and DNS logs was manually labelled suspicious (`sus`) or `evil` to assist post-training analysis[3]. Logs marked suspicious indicate unusual activity or outliers in the data distribution, such as an external `userId` with a `systemd` process[4], infrequent daemon process calls (e.g. `acpid` or `accounts-daemon`), or calls to close processes that we did not observe as being started. Evil[5] indicates a malicious external presence not inherent to the system, such as a bash execution call to list the computer's memory information, remove other users' `ssh` access, or un-tar an added file. Events marked `evil` are considered "out of distribution," as they are generated from a data distribution not seen during training.

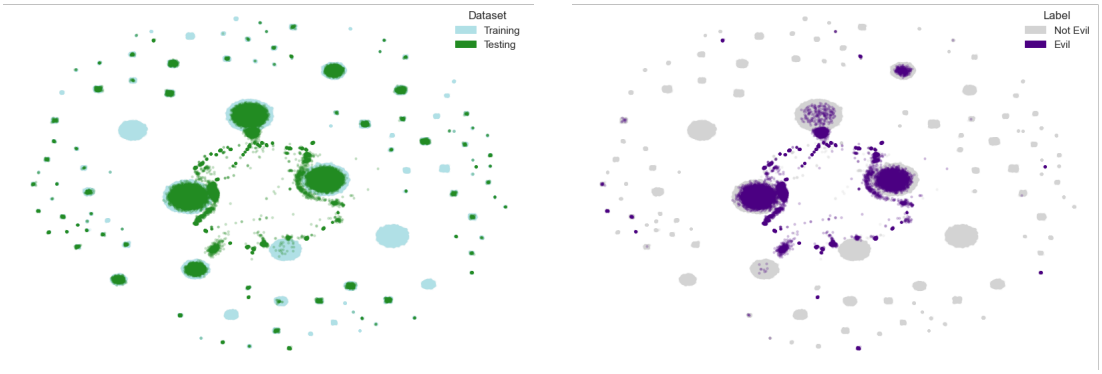
A subset of the kernel process logs were divided into a typical 60/20/20 split for training, validation, and testing, based on the amount of activity recorded, and `evil` labels from each host. As is typical in unsupervised anomaly detection, our training and validation sets are each composed of logs generated from multiple hosts which only recorded activity from the OS and cloud infrastructure management. Because there are no signs of infiltration in these process logs, activity resulting from these hosts is considered benign and the events generated by the OS and cloud infrastructure management to be "in-distribution".

Our initial testing dataset contains all activity on a single exploited host, including its

[3]These labels are provided from a single reviewer and should not be solely relied upon for industry systems.

[4]In the scope of our honeypot any external user traffic is suspicious, but some of these events were initiated by the cloud provider.

[5]We note that presence in this dataset does not constitute a "conviction", as no real damage was done.



(a) Fitted on Training and transformed Testing (b) Same as (a), coloured with `evil` labels

Figure 1: UMAP visualisations of the training and testing dataset using the pre-processed features (see Appendix A). Figure (a) shows the overlap between the training and testing dataset; Figure (b) highlights the trails of `evil` events.

OS and cloud infrastructure management. The first attack we logged is an attempt to setup a botnet; more details are available in Appendix B. The full dataset contains other malicious activity performed within our honeypots, including cryptomining and lateral movement (between servers). These various attacks may also be compared to answer alternative research questions with our data, as discussed in Subsection 2.3. As each exploited host only contains a single staged attack, with no artificial noise in the benign activity, BETH is one of the cleanest cyber security datasets available to distinguish between malicious and benign activity.

As an initial investigation of the data, we visualised the (pre-processed) training and testing datasets with uniform manifold approximation and projection (UMAP) [19]. UMAP was first fitted to the training set before being used to project the testing set into the same space. As can be seen in Figure 1, the data from both sets forms several large clusters in the centre, surrounded by many smaller clusters, with both benign and malicious activity spread across the entire space. The first image shows significant overlap between the training and testing sets. The second image shows that `evil` events appear in distinct areas. This indicates that unsupervised methods could potentially detect a large portion of the "anomalous" events.

2.3 Research Questions

The BETH dataset could answer other cyber security questions than just OoD analysis. Unlike logs within real deployed systems that contain no labels for malicious events, our BETH dataset contains (recently recorded) real data with labels. One use for this dataset would be to profile the attacker or malware's behaviour [2]. For instance, the known `evil` events could form a unique *fingerprint*, a method of uniquely identifying the tactic used by the attacker, to link an attack to its family or appropriate resolution strategy [1]. One could also use graph analysis of process relationships to find malicious cliques [4], or use time series analysis of execution sequences to profile process names (`processName`) on a modern OS. This latter topic is particularly interesting as some attackers rename malicious processes to benign process names to trick systems into running malicious code. The logs would present a benign process name, even if the arguments or events were inconsistent with normal activity.

3 Anomaly Detection Baselines

In this section, we provide anomaly detection benchmarks on our initial subset of logs. We chose both standard anomaly detection baselines [23, 32], which includes robust covariance [26], one-class support vector machine (SVM) [28] and iForest [16], as well as density of states estimation (DoSE) [21], which is based on deep generative models. As per [21], we report AUROC, using an ensemble of 5 models for each method.

Robust covariance [26] fits an "ellipsoid with the smallest volume or with the smallest covariance determinant" [24] around the central data points; the tightness is controlled assuming a given level of contamination with anomalies (which we set to 0.05). The anomalies are then scored using the Mahalanobis distance. Similarly, the one-class SVM fits a hyperplane to discriminate between the support of the in-distribution data and OoD data [28]. As kernelised SVMs scale with $O(N^2)$ and our data subsets are substantial in size, we instead utilised `scikit-learn`'s linear SVM with stochastic gradient descent, after whitening the data. In contrast to the other methods, the iForest [16] tries to characterise anomalous points in the data distribution using an ensemble of "isolation trees".

Given the scale of the dataset, we also considered DL-based OoD detection methods. In particular, DoSE uses summary statistics (such as the log-likelihood or posterior entropy), calculated over the training set by a trained generative model, in order to characterise the "typical set". In our work we train a variational autoencoder (VAE) [9], consisting of two 2-layer neural networks with 64 hidden units and ReLU activation functions for the encoder and decoder. The first layer of the encoder concatenates learned embeddings of all input features. The final layer of the decoder outputs a set of logits for categorical distributions for all features. We use a 2D latent representation. Each VAE is trained using the AdamW optimiser [18] with learning rate 0.003 and a weight decay of 0.1; early stopping was used with the validation loss. We picked the hidden size $\in \{64, 128, 256\}$, learning rate $\in \{0.003, 0.0003, 0.00003\}$, and weight decay $\in \{0, 0.01, 0.1\}$, using a grid search on the validation loss. Other than modelling the observations as a product of categorical distributions, our setup is largely the same as the original paper [21]. However, due to the size of the training set, we were only able to use DoSE with a linear one-class SVM trained using SGD (as opposed to a kernel SVM).

Table 5: OoD AUROC results.

Method	Robust Covariance	One-Class SVM	**iForest**	VAE+DoSE-SVM
AUROC	0.519	0.605	**0.850**	0.698

As seen in Table 5, the iForest performs best at differentiating `sus` events from the benign in our testing dataset. We attribute this to the small set of discrete features available and the conspicuous nature of the attack. DL-based models are less competitive on these sets of features, but have the potential to deal with more raw categorical and even text-based features, which we hope to explore in future work. Finally, we note that imbalanced labelling, summarised in Table 6, necessitates further investigation of what each model predicts is benign or not.

4 Conclusions

In this paper, we present our BETH cybersecurity dataset for anomaly detection and OoD analysis. The data was sourced from our novel honeypot tracking system recording both kernel-level process events and DNS network traffic. It contains real-world attacks in the presence of benign modern OS and cloud provider traffic, without the added complexity of noisy artificial user activity. This cleanliness is ideal for OoD analysis, such that each host in the dataset

only contains one or two data-generating distributions. We also include baselines for anomaly detection trained on a subset of the BETH dataset: robust covariance, one-class SVM, iForest, and DoSE-SVM (with a VAE).

For future work, we plan to collect and publish more attacks for alternative testing datasets. This will also allow investigations in comparing attacks or perhaps testing in a continual learning setting.

Acknowledgements

We are thankful for the many incredible academics who supported this work. Dr. Arinbjörn Kolbeinsson inspired the included UMAP visualisations. Professor Sergio Maffeis provided incredible advise on the presentation and edits to the paper. Also, thank you to the reviewers for their positive feedback.

This work was also supported by The Alan Turing Institute, during an author's participation in the Turing Enrichment Program (October 2021 - June 2022), under the EPSRC grant EP/N510129/1.

References

[1] D. Brumley, J. Caballero, Z. Liang, J. Newsome, and D. Song. Towards automatic discovery of deviations in binary implementations with applications to error detection and fingerprint generation. In *USENIX Security Symposium*, page 15, 2007.

[2] Q. Chen, S. R. Islam, H. Haswell, and R. A. Bridges. Automated ransomware behavior analysis: Pattern extraction and early detection. In *International Conference on Science of Cyber Security*, pages 199–214. Springer, 2019.

[3] G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2921–2926. IEEE, 2017.

[4] A. A. Elhadi, M. A. Maarof, and A. H. Osman. Malware detection based on hybrid signature behaviour application programming interface call graph. *American Journal of Applied Sciences*, 9(3):283, 2012.

[5] B. Gregg. *BPF Performance Tools: Linux System and Application Observability*. Addison-Wesley Professional, 1st edition, 2019. ISBN 0136554822.

[6] D. Hendrycks and T. Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *Proceedings of the International Conference on Learning Representations*, 2019.

[7] D. Hendrycks, X. Liu, E. Wallace, A. Dziedzic, R. Krishnan, and D. Song. Pretrained transformers improve out-of-distribution robustness. *arXiv preprint arXiv:2004.06100*, 2020.

[8] S. Hettich and S. Bay. The uci kdd archive [http://kdd. ics. uci. edu]. irvine, ca: University of california. *Department of Information and Computer Science*, 152, 1999.

[9] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[10] A. Krizhevsky, V. Nair, and G. Hinton. Cifar-10 (canadian institute for advanced research). URL `http://www.cs.toronto.edu/~kriz/cifar.html`.

[11] M. L. Labs. 1998 darpa intrusion detection evaluation dataset, 1998. URL `https://www.ll.mit.edu/r-d/datasets/1998-darpa-intrusion-detection-evaluation-dataset`.

[12] K. Lang. Newsweeder: Learning to filter netnews. In *Machine Learning Proceedings 1995*, pages 331–339. Elsevier, 1995.

[13] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[14] D. D. Lewis. Reuters-21578 text categorization collection data set, 1997.

[15] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham, et al. Evaluating intrusion detection systems: The 1998 darpa off-line intrusion detection evaluation. In *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*, volume 2, pages 12–26. IEEE, 2000.

[16] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation forest. In *2008 eighth ieee international conference on data mining*, pages 413–422. IEEE, 2008.

[17] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[18] I. Loshchilov and F. Hutter. Fixing weight decay regularization in adam. 2018.

[19] L. McInnes, J. Healy, and J. Melville. Umap: Uniform manifold approximation and projection for dimension reduction, 2018. URL `http://arxiv.org/abs/1802.03426`. cite arxiv:1802.03426Comment: Reference implementation available at http://github.com/lmcinnes/umap.

[20] D. Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

[21] W. Morningstar, C. Ham, A. Gallagher, B. Lakshminarayanan, A. Alemi, and J. Dillon. Density of states estimation for out of distribution detection. In *International Conference on Artificial Intelligence and Statistics*, pages 3232–3240. PMLR, 2021.

[22] Y. Ovadia, E. Fertig, J. Ren, Z. Nado, D. Sculley, S. Nowozin, J. V. Dillon, B. Lakshminarayanan, and J. Snoek. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. *arXiv preprint arXiv:1906.02530*, 2019.

[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[24] D. Peña and F. J. Prieto. Multivariate outlier detection and robust covariance matrix estimation. *Technometrics*, 43(3):286–310, 2001.

[25] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho. A survey of network-based intrusion detection data sets. *Computers & Security*, 86:147–167, 2019.

[26] P. J. Rousseeuw. Least median of squares regression. *Journal of the American statistical association*, 79(388):871–880, 1984.

[27] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[28] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–1471, 2001.

[29] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers & Security*, 31 (3):357–374, 2012. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2011.12.012. URL `https://www.sciencedirect.com/science/article/pii/S0167404811001672`.

[30] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani. A detailed analysis of the kdd cup 99 data set. In *2009 IEEE symposium on computational intelligence for security and defense applications*, pages 1–6. IEEE, 2009.

[31] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[32] S. W. Yahaya, A. Lotfi, and M. Mahmud. Towards a data-driven adaptive anomaly detection system for human activity. *Pattern Recognition Letters*, 145:200–207, 2021. ISSN 0167-8655. doi: https://doi.org/10.1016/j.patrec.2021.02.006. URL `https://www.sciencedirect.com/science/article/pii/S0167865521000611`.

A Pre-Processing

In this section, we provide more details on the raw features in the dataset, as well as pre-processing suggestions, which we used in our baselines:

`timestamp`: We left this field out to consider the dataset as a sample from a distribution rather than time series. We recommend using the values as they are or also leave them out, depending on the method chosen.

`processId`: Process IDs 0, 1, and 2 are meaningful since these are always values used by the OS, but otherwise a random number is assigned to the process upon creation. We recommend replacing `processId` with a binary variable indicating whether or not `processID` is 0, 1, or 2.

`threadId`: While this value did not appear useful in our analysis, it might suggest how to link process calls if obfuscated in the system. No conversion is recommended at this time.

`parentProcessId`: Same as `processId`, the same mapping to a binary variable should suffice.

`userId`: The default in Linux systems is to assign OS activity to some number below 1000 (typically 0). As users login, they are assigned IDs starting at 1000, incrementally. This can be altered by a user, but none of the current logs gave evidence an attacker did this. We used a binary variable to indicate `userId` < 1000 or `userId` ≥ 1000. Alternatively, one could use an ordinal mapping that buckets all `userId` < 1000 at zero and then increment upwards for each new user. Also, no more than four logins were viewed per host in our current datasets.

`mountNamespace`: This field is somewhat consistent across our hosts and determines the access a certain process has to various mount points. The most common value for this feature is 4026531840 or `0xF0000000`, which is for the `mnt/` directory where all manually mounted points are linked. It is noted that all logs with `userId` ≥ 1000 had a `mountNamespace` of 4026531840, while some OS `userId` traffic used different `mountNamespace` values. We converted this feature into a binary mapping for whether or not `mountNamespace` $= 4026531840$.

`processName`: This is a string field of variable length (ranging from one to fifteen characters). When manually analysing the data, this was a critical field in conjunction with the `eventName`.

For our baselines, we refrained from utilising this, although the model should be given an encoding of this using a hash or ability to learn a useful encoding on its own. It is noted that attackers can easily change the `processName` to override a benign one so their traffic looks regular. This was not observed within the current dataset.

`hostName`: This field is useful for grouping the dataset into related subsets of data generated from the same honeypot. The name of the host name does not transfer between the model development subsets described in this paper.

`eventId`: Linux systems assign an integer corresponding to the `eventName`. We include this field as-is for our benchmarks.

`eventName`: Event names uniquely map to `eventId`, so we drop it from training.

`argsNum`: This raw feature is included as-is, since, at this time, adequately parsing `args` requires either more sophisticated pre-processing or a more complex ML model.

`returnValue`: This is also called the exit status and can be used to determine whether a call completed successfully or not. Mappings for this can vary, as this value is decided between the parent and child process. We mapped `returnValue` into three values based on the common usage of the field: -1 when negative (error), 0 when zero (success), and 1 when positive (success and signalling something to the parent process).

`stackAddresses`: It is difficult to clearly relate this feature during manual analysis and the large values within a variable size list make processing automatically difficult without encoding or learning an extra embedding. Thus this field was dropped from training our baselines.

`args`: There are many options in this variable list of dictionaries. For simplicity, we refrain from utilising any of these values. However, more features can and should be created for future work.

Finally, BETH contains two binary, manually-labelled flags: `sus` and `evil`. Examples and the explanation of how these labels were created are detailed in Section 2.2. A breakdown of these labels within the subsets for model development is given in Table 6.

Table 6: Breakdown of `sus` and `evil` labels by training, validation, and testing subsets.

Dataset	sus=0, evil=0	sus=1, evil=0	sus=1, evil=1
Training	761875 (99.8%)	1269 (0.02%)	0 (0.00%)
Validation	188181 (99.6%)	786 (0.04%)	0 (0.00%)
Testing	17508 (9.27%)	13027 (6.89%)	158432 (83.84%)

B Testing Dataset Details

This testing dataset was extracted from a single honeypot. The overall attack appears to be instantiating a botnet node. The timeline of the events recorded is provided in Figure 2; this is the typical attack pattern. The server is initially accessed, it may run a few setup operations in the environment to send some details to its Command and Control (C2) for a customised attack, it sleeps for a while, intermittently checks in with the C2 or a clock, and then launches its attack until complete.

In this case, the honeypot is first accessed at around 411 seconds from booting. Several

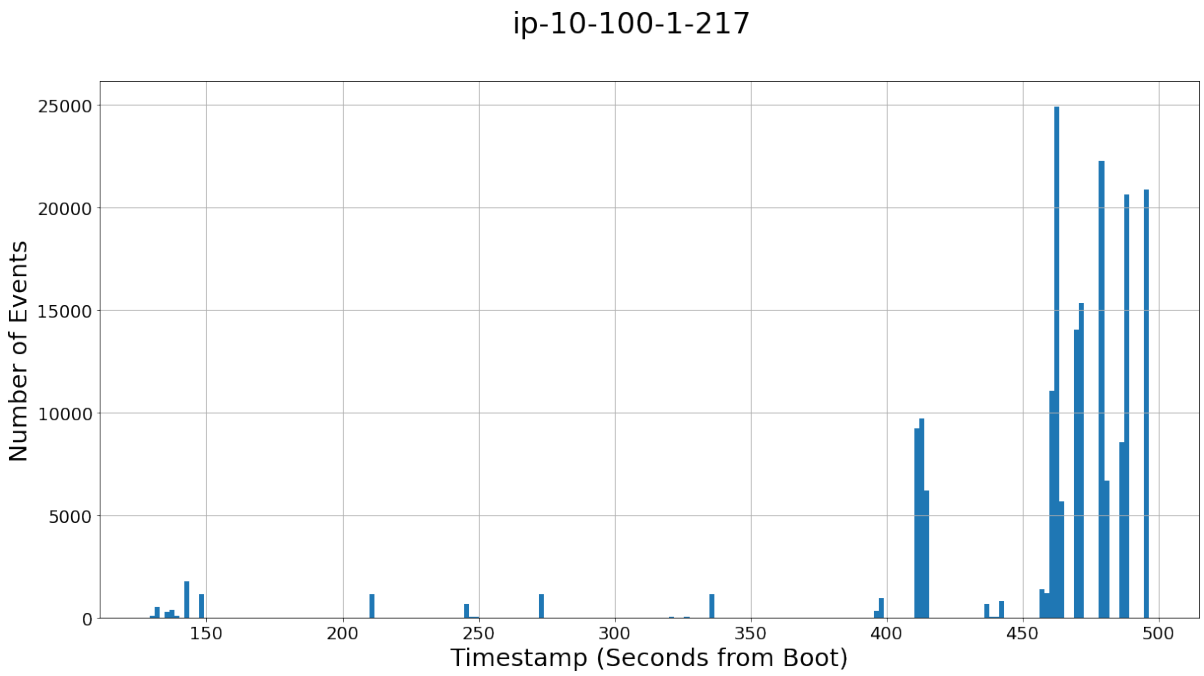


Figure 2: The timeline of the attack captured in the testing dataset is displayed as a histogram based on the number of events and seconds from the "boot" or starting up of the machine.

thousand lines are then recorded in the process logs denoting the setup of the new user profile. This happens within milliseconds; these are detailed logs of everything the OS does during the short pause before the terminal opens for user entry when `ssh`-ing into a server. This user then sleeps, pausing all user activity for some number of seconds. This appears to happen at random intervals—a more sophisticated technique than using consistent intervals—of which the latter would give a clear signature of automated activity.

After a few minutes, it sets up an SFTP server to download a file called `dota3.tar.gz` (known botnet malware) and scopes out the system using common commands such as `whoami`, `ls`, and `cat /proc/cpuinfo`. After about 7.5 minutes, it unpacks the `dota3.tar.gz` and runs over a hundred threads, all attempting to connect with different servers.