Computers & Security 148 (2025) 104154

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

MIDAS: Multi-layered attack detection architecture with decision optimisation

Kieran Rendall [a], Alexios Mylonas [a,*], Stilianos Vidalis [a], Dimitris Gritzalis [b]

[a] Cybersecurity and Computing Systems Research Group, Department of Computer Science, University of Hertfordshire, College Lane, Hatfield, AL10 9AB, UK
[b] Department of Informatics, Athens University of Economics and Business (AUEB), 76 Patission Ave., Athens GR-10434, Greece

ARTICLE INFO

Keywords:
Multi-armed bandits
Attack detection
Machine learning
Efficiency
Decision optimisation

ABSTRACT

The proliferation of cyber attacks has led to the use of data-driven detection countermeasures, in an effort to mitigate this threat. Machine learning techniques, such as the use of neural networks, have become mainstream and proven effective in attack detection. However, these data-driven solutions are limited by: *a)* high computational overhead associated with data pre-processing and inference cost, *b)* inability to scale beyond a centralised deployment to cope with environmental variances, and c) requirement to use multiple bespoke detection models for effective attack detection coverage across the cyber kill chain. In this context, this paper introduces MIDAS, a cost-effective framework for attack detection, which introduces a dynamic decision boundary that is used in a multi-layered detection architecture. This is achieved by modelling the decision confidence of the participating detection models and judging its benefits using a novel reward policy. Specifically, a reward is assigned to a set of available actions, corresponding to a decision boundary, based on its cost-to-performance, where an *overall* cost-saving is prioritised. We evaluate our approach on two widely used datasets representing two of the most common threats today, *i.e.,* phishing and malware. MIDAS shows that it effectively reduces the expenditure on detection inference and processing costs by controlling the frequency of expensive detection operations. This is achieved without significant sacrifice of attack detection performance.

1. Introduction

In the current threat landscape, phishing, a significant online threat, replaced malware as the biggest web-based threat (Zhang et al., 2022). Threat actors have been active with credential phishing methodologies to gain access to target email accounts, in particular since the Ukraine conflict (CrowdStrike, 2023). At the same time malware (binaries or scripts) still remain a significant threat in the current threat landscape. Malware is often detected when it resides on the disk with the use of static or dynamic analysis. As such, to evade detection a significant surge in popularity of fileless malware has been used by threat actors in recent years (Liu et al., 2024). This leads to more expensive detection techniques being used, such as sandbox dynamic analysis.

Defending against such threats has always been an ongoing area of research, but more importantly, machine learning techniques are now using richer, larger, and more complex datasets and models (Rookard and Khojandi, 2024; Zhong et al., 2024). Machine learning differs from traditional static techniques to identify such threats, such as the use of denylists. It does not entirely rely on up-to-date indicators of compromise (IoC) to be effective and, instead, generalises to handle unseen data. Notably, machine learning-based systems are computationally expensive by design, so that sophisticated attack techniques can be discovered from sets of complex patterns.

At the same time, defenders endure a number of challenges that stem from a multipart information space in their infrastructure. Some of these parts can be more heterogeneous than others, *e.g.,* email attachments compared to custom application logs. As a result, detection responsible for an attack surface needs to cope with its respective complexity. Even though the practicalities of capturing several dozens of attack techniques to form effective attack datasets has been explored in the literature (Nisioti et al., 2018; Yang et al., 2022), dealing with the deployment cost of such complex detection system is limited. Equally, large numbers of devices will contribute to an increase in data transfer, processing, and storage, prior to any inference, which can hinder the detection speed and increase investment costs.

While work has considered the use of a *"multi-layered"* approach to account for these challenges, the decision threshold to progress to subsequent layers is not particularly optimised (Doshi et al., 2023; Gupta

* Corresponding author.
E-mail addresses: ka20aci@herts.ac.uk (K. Rendall), a.mylonas@herts.ac.uk (A. Mylonas), s.vidalis@herts.ac.uk (S. Vidalis), dgrit@aueb.gr (D. Gritzalis).

https://doi.org/10.1016/j.cose.2024.104154
Received 3 July 2024; Received in revised form 4 September 2024; Accepted 8 October 2024
Available online 16 October 2024

and Singh, 2024; Rendall et al., 2020). For instance, a recent work by (Gupta and Singh, 2024) separates detection from attribution in order to reduce processing time and enhance detection accuracy. However, one could argue that the default decision boundary for a classifier (*i.e.,* 0.5), as in previous literature (Rendall et al., 2020), is not adequate. In cases where model training is not regular, one could consider the use of a dynamic decision boundary to help maintain the detection performance.

In this context, this paper introduces MIDAS (MultI-layered attack Detection Architecture with decision optimiSation), that provides a dynamic decision boundary in multi-layered attack detection architectures. MIDAS as a decision optimisation framework, proposes a *value estimate model* to reduce the cost spent on attack detection. To this end it dynamically adjusts decision boundaries using the models' probability distribution and historical environmental conditions.

MIDAS is a generic framework that can be integrated with existing data pipelines and data-driven detection engines that utilise additional, often more expensive, capabilities. Specifically, MIDAS ingests the outputs from detection models (either batch or real-time) and can work as a coordinator to optimise overall detection and thus offer cost saving. A use case of MIDAS could be a cloud-based detection service, similar to Microsoft Defender. In such a multi-layered detection architecture, each detection capability offered via the cloud environment would act as a more expensive layer in MIDAS, which complements detection that happens locally on the device. The benefits here would be to minimise costs across an estate, where costs are based on the usage of the expensive cloud based detection engine.

This paper makes the following contributions:

- It proposes MIDAS, a cost-effective framework for attack detection that is based on supervised machine learning to detect threats. MIDAS dynamically finds the most optimal set of decision boundaries in a multi-layered detection architecture. It can steer the boundary based on the environment to handle conditions like evolving model confidence across datasets.
- We utilise *Multi-armed bandits* and propose novel reward policies to model the decision confidence of the participating detection models. They were designed to implicitly balance the exploration and exploitation dilemma, while minimising cost and maximising detection performance. Our reward policies were compared to those in the state-of-the-art and were found to outperform them both in terms of cost saving and detection performance.
- We evaluated MIDAS with two datasets consisting of phishing and malware attacks, which are prevalent in the current threat landscape. We analyse the decision boundaries response to changes in parameters, reward policies and selection algorithms to identify the best performance of MIDAS in terms of cost saving and detection performance.

The rest of this paper is organised as follows: Section II provides background and related work. Section III presents the methodology and Section IV presents the evaluation of our approach. Section V provides the discussion. Finally, Section VI concludes the paper with suggestions for future work.

2. Background

This section introduces multi-armed bandits, followed by the components that form a model. It then describes the search algorithms that are commonly used and key to implementing our approach.

2.1. Multi-armed bandits

Multi-armed bandits, also known as MAB, is a classical reinforcement learning framework that enables algorithms to make decisions under uncertainty (Slivkins, 2019). The objective in MAB is to make decisions but find a balance between exploiting existing knowledge or exploring new decisions, also known as the exploitation versus exploration dilemma. The key components in MAB include the: *1)* selection algorithm, *2)* reward function, and *3)* set of actions.

As an example of use of MAB, consider a router device that needs to route traffic, but the Internet has seasonal or fluctuations in communications latency across different routes. One could use MAB, where the decision (*i.e.,* action) is to select a specific route from a collection of routes (*i.e.,* available actions), and the outcome (*i.e.,* reward) is the total latency. Here, the goal is to select actions using an appropriate algorithm that alternates between selecting the known routes with low latency, or to select infrequent routes, in order to find the most optimal under the current conditions, in this case high latency.

In this work, we explore the following selection algorithms:

1) *Boltzmann Exploration (Softmax):* The Boltzmann exploration, a Softmax method (Kuleshov and Precup, 2000), aims to select actions with the greatest expected value, and uses a parameter to control the degree of exploration. First, Softmax creates a probability distribution of all the available actions based on their expected value as, $P(a_i) = \frac{e^{V(a_i)/\tau}}{\sum_{j=1}^{n} e^{V(a_j)/\tau}}$, where $P(a_i)$ is the probability of selecting action a_i, and τ is the exploration term (*i.e.,* the degree of exploration). Note $\sum_{i=1}^{n} P(a_i) = 1$. Then, a number between 0 and 1 is chosen at random which is used to select an action. The actions with the highest probabilities are more likely to be chosen, although a higher τ will distribute the probabilities of each action more evenly, thus lead to greater chances of exploring actions with lower value estimates.
2) *Upper Confidence Bound (UCB):* UCB is a deterministic algorithm that uses an uncertainty factor to prioritise action exploration. First, at each step, the uncertainty of each action is calculated based on the number of times each action has been selected and the total number of steps, where $r_t(a) = \sqrt{\frac{2 \log t}{n_t(a)}}$ (Auer et al., 2002). Next, the average reward of each action is combined with its uncertainty value to give the final expected value. Under these conditions the greater $n(a)$ relative to t, the greater its certainty. Thus, those actions with a low selection rate are more likely to be selected, as their uncertainty factor will be higher. Here, each action will certainly be used and evaluated, meaning that an optimal set of actions will be discovered as soon as knowledge develops over time.
3) *Thomson Sampling:* This algorithm, defined as Algorithm 1, uses the Bayesian framework by modelling the reward distribution, then sample from the distribution formed over each step, to use the action that will return the greatest expected value. In our case, we chose to model an action distribution as a beta distribution where the parameters α and β are the cumulative summations of positive and negative rewards, respectively. Once an action is chosen based on the maximum sampled expected value, either parameter (α/β) is updated based on the actions effect in the environment.

2.2. Related work

This sub-section discusses the relevant literature with a focus on *i)* threat detection, and *ii)* the constraints posed by production systems.

Threat detection. Threat detection has become increasingly more complex and resource intensive. For instance, the authors in (B. Wang et al., 2023) proposed a framework to detect collusion-based attacks, *e.g.,* those using inter-app communication. They were able to combine a novel taint and static analysis in a three-stage process, namely: *i)* metadata extraction, *ii)* static analysis, and *iii)* taint analysis. Whilst this deep analysis is a more costly approach, it did improve detection performance by five percent over the state of the art.

The work in (Fang et al., 2023) proposed an Android malware detection method to increase the detection rates on highly evolving samples. They used features, such as API calls and permissions, to train a Convolutional Neural Network (CNN) and employ genetic evolution on

Algorithm 1
Thompson Sampling.

1: **Input:** Actions A, time horizon T
2: **Initialization:** Positive S_i and Negative F_a reward to 0 for each action a
3: **for** $t = 1 \ldots T$ **do**
4: For each arm a, Sample $\theta_a(t)$ from Beta($S_a + \alpha$, $F_a + \beta$)
5: $a_t \leftarrow \arg\max_a \theta_a(t)$
6: Select action a_t and observe the reward R_at
7: **if** reward $R_at > 0$ **then**
8: $S_at \leftarrow S_at + R_at$
9: **else**
10: $F_at \leftarrow F_at + R_at$
11: **end if**
12: **end for**

existing malware samples, outperforming traditional classifiers, (*e.g.,* SVM, RF). Work proposed by (Yang et al., 2024) used API semantics that involved clustering to identify malware similarity. Similarly, the work in (Chen et al., 2024) proposed an approach that factors in API calls, but instead as a sequence with their respective parameters, outperforming existing sequence methods. The use of byte sequences in order to create signatures has effectively identified malware (Saha et al., 2024). Other work has explored Android malware API/system calls and employed classification as an ensemble architecture (Bhat et al., 2023). They suggest a conventional static analysis approach fails to identify behaviour that would otherwise be present during runtime, such as dynamic code loading.

The authors of (Bertrand Van Ouytsel et al., 2024) formulated a series of malware packing detection experiments using 119 features over nine detection algorithms to find the most economical approach. They found that static features, such as API calls, had significant impact on the decision-making process and were more cost-effective.

The work in (Dodia et al., 2022) proposed novel features to identify malware that leverages anonymisation services for its communications. Each binary is characterised based on its network behaviour, which is derived from network-level features. The cost to compute the aggregation of the connection-based statistical features was not mentioned, which has been recognised by (Xin et al., 2021) as a process with major associated cost. Nonetheless, this approach did achieve high detection performance.

Threat detection architectures, which are known to be predominately centralised, have posed significant constraints on *data storage, computation,* and *data transmission* (Dong et al., 2023). The authors address these challenges by proposing a lightweight client-side distributed APT detection system. Authors in (Fang et al., 2023) employed a federated approach where each client used a local model and regularly updated a centralised global model, an efficient use of computational resources.

The authors in (Birman et al., 2022) recognise the expensive cost of multiple classifiers, so they propose a reinforcement learning method that activates classifiers based on a trained model. In this case, a neural network is trained on the confidence distribution of the classifiers and computational costings. A shortcoming with this approach, is that there is no defined sequence of detectors and so the model is prone to detector repetition. The authors attempted to discourage the model from this behaviour using an immense negative reward. However, this will only discourage the current sequence of detectors, when there is arguably an infinite number of sequence due to an undefined sequence length. Our approach considers this challenge, by using a multi-layered architecture, thus removing complexity of the model learning a sequence of effectiveness under conditions.

The authors in (Rendall et al., 2020) presented a two-layered detection architecture, where each layer represented a classifier. They introduced a decision boundary that only permits classifications from subsequent classifiers if the first probability estimate is within a boundary value. Nonetheless, this work is limited as the architecture used a static value for the decision boundary. A similar approach by (Doshi et al., 2023) used a two-layered system where the classification category 'hybrid' from layer-1 would trigger layer-2 classification, specifically to solve the class imbalance problem. Work by (van Geest et al., 2024) proposed a hybrid framework that combines the predictions from multiple models using a stacking function. Here a range of functions were assessed and identified logistic regression as the most effective.

Other focus areas of threat detection are the optimisation of features, notably the selection of features that will provide most prediction value. That is, reducing the feature dimension as much as possible without compromising on detection performance. Existing studies have proposed a range of techniques for this (Fatima et al., 2019; S. Wang et al., 2020), including the use of reinforcement learning (Wu et al., 2023).

Multi-armed bandits *(MAB)* have been used in past literature for threat detection. *MAB* is recognised to automatically handle optimisation during times of uncertainty. The authors in (Heartfield et al., 2021) used *MAB* to select the most optimal decision boundary for unsupervised anomaly detection model. Each trial on a given snapshot of data was used to recursively explore and reward a classifier's hyperparameter (a contamination value) to find the optimum threshold value. The benefits of using MAB in this context is its ability to optimise detection in changing environments.

The authors in (Dekel et al., 2023) propose the use of *MAB* to enhance the level of automation during threat hunting. They model the investigation process so that each selected action collects some artifacts associated with an attack, at a collection cost. To achieve this a knowledge base of attacks and their respective artifacts were used as search space. The iterations are bounded by a budget, so each step of the investigation is optimal, which is usually a difficult and costly process for analysts.

The authors in (Paya et al., 2024) developed a defense framework to protect against adversarial models from learning the strategy of the detection model. Here, they use *MAB* to select the most optimal set of classifiers to make a prediction on a network traffic flow, based on the classifiers' ability to provide the highest level of accuracy. Although the reward signal was unclear, a distribution for each classifier was formed over significant time, using the *Thomson Sampling* algorithm to select actions. The work in (Sagi and Rokach, 2018) argues that increasing the number of classifiers can alleviate machine learning challenges as algorithm diversity with varying features can reduce dataset imbalance, concept drift and dimensionality.

The authors in (Shen et al., 2023) proposed a *MAB* method to defend against multistage APT attacks that utilise running processes on a host as paths of attack. The authors use a provenance graph to model process dependencies for selective monitoring under limited resources. They replaced the exploration term, *i.e.,* mean reward value, in the UCB algorithm with a malevolence value. To obtain the value, an LSTM network was used on historical temporal information of paths. Therefore, the greater the term, the greater the chances of selection and in this case the monitoring of the process path.

Constraints from production environments. Recent literature seeks to understand the complexity and challenges with production ML (Xin et al., 2021). The authors in (Xin et al., 2021) present a large-scale analysis of ML pipelines at Google. They found that ~22 % of compute costs arise from data ingest processes, a significant percentage relative to a ~23 % cost for model training, which is usually recognised as the most expensive pipeline phase. The work in (Paleyes et al., 2023) presents challenges with resource-constrained environments, suggesting that one should consider the practicalities of deploying deep learning to networked systems that have very limited energy, memory, and data transmission. The authors in (Xin et al., 2021) and (Paleyes et al., 2023) also emphasise the challenges with ML deployments. Little work in the literature takes into consideration all of the aforementioned points.

Despite model deployment constraints on enterprise network estates, mobile devices are no exception. The pervasiveness of machine learning-based models, such as Deep Neural Networks (DNN), on mobile devices likely contributed to vendors focusing on hardware improvements, such

as GPU accelerated components and battery efficiency. The authors in (Deng et al., 2022) analysed 62,583 applications on the app store and found 960 on-device DNN models between 568 apps. These models concern the authors as they represent new targets for attackers due to the cost-saving for those intending to steal the pre-trained parameters. A federated-based framework was proposed by (Chen et al., 2024), in order to accelerate model training/updates while protecting the privacy of the model and datasets. While our work focuses on cost-saving, one could consider reducing the load and security concerns on the end-user's device by leveraging the edge for more complex and expensive models, using our proposed approach.

Concept drift is recognised as a significant concern that causes adverse effects on deployed model performance (Paleyes et al., 2023). As such, any decision function that is used to optimise on cost should be mindful of becoming dependent on model output, *e.g.,* a probability distribution. Increasing the number of detections can alleviate machine learning challenges as algorithm diversity with varying feature sets can reduce the impact of dataset imbalance, concept drift and dimensionality (Sagi and Rokach, 2018). This can introduce computational overhead as training time and memory extend by the number of detections. However, use of multiple models has been shown to increase overall detection performance (Gupta et al., 2022). For instance, the work by (Tseng and Chang, 2023) demonstrated that multiple binary classifiers yielded better performance than a multi-classifier. These challenges and advantages underpin our decision optimisation described in Section III.

A concern by (Jacobs et al., 2022) is the 'black-box' nature of decision-making models which limits the explainability to analysts. They propose a decision framework to increase the interpretability of models used to detect complex patterns in networks. The model complexity in this instance is driven by the size of the decision tree, and so our approach to divide features into subsets will benefit from such explainability.

With regards to threat detection, computational cost (*e.g.,* energy, time, and memory) is usually overlooked. However, while a robust detection system can have high performance, it can introduce computational overhead as training time and memory extend by the number of classifiers.

Work by (Rao et al., 2019) dealt with processing costs associated with mobile phishing detection by only using URL-based features, achieving an average time of 621 ms to complete the detection process. In this case, the real-time feature extraction timings of lexical and content demonstrated that legitimate processing took on average 5.85 times longer than phishing. Although, this work is limited by excluding page load times, which add seconds to the overall processing time. The work proposed by (Rendall et al., 2020) and (Das et al., 2020) both recognised that DNS-based feature collection is more costly than static features due to dependence on network capture. Similarly, other costly features are available that can contribute to phishing detection, such as using features derived from PKI infrastructure (Gritzalis, 2004; Iliadis et al., 2000).

The authors (Bahnsen et al., 2017) took a different approach by determining the throughput rate of phishing detection and identified that LSTM network took longer to evaluate URLs (280 p/s), when compared with RF (942 p/s). Also, while the LSTM took 80 times longer to train, the memory consumption was comparably lower (581 KB) than Random Forests (288 MB).

The work in (Xu et al., 2021) highlighted the importance of feature processing and extraction times when scaling for billions of users, and so leveraged transfer learning to reduce cost and improve generalisation.

3. MIDAS framework

This section introduces MIDAS (MultI-layered attack Detection Architecture with decision optimiSation), which provides novel decision optimisation for a multi-layered detection architecture, similar to (Rendall et al., 2020). This work formally defines the problem as the real-time selection of a decision boundary's position to (a) reduce unnecessary computational resources, stemming from unnecessary activation of a subsequent classifier c_j, and (b) maintain high classification performance levels. MIDAS comprises of three main components, namely a set of dynamic decision boundaries, a reward function, and an action selection function. As summarised in Fig. 1 and described in the rest of the section, MIDAS ingests the outputs from multiple detection models and can work as a coordinator to optimise overall detection and thus offer cost saving.

3.1. Model formulation

This section models the process of finding the most optimal decision boundary for attack detection as a MAB problem. The symbols used in the paper are summarised in Table 1.

The goal of an attack detection system is to: *i*) correctly identify attacks based on their characteristics and behaviour and *ii*) be practical for real-world adoption in a production environment by minimising its cost, *e.g.,* reducing network latency, for its users. To achieve this a set of detection models must be sufficiently trained on qualitative and representative data, collected from recent and realistic attacks.

A standalone attack detection model can prove ineffective against multi-variate attack stages, due to the challenges with generalising across the attack space. We assume that an ensemble detection architecture is used due to its ability to leverage multiple detection models, a popular method that increases the probability of high detection performance (Al-Sarem et al., 2021; Chohra et al., 2022; Gao et al., 2019; Maniriho et al., 2024; Otoum et al., 2020). While multi-classification detectors can provide more fine-grained predictions, such as the attack type, we consider them out of scope for this work. Here, we define the detection system as a sequence of binary classification models $(c_i)_{i=2}^{n}$ where n is the number of classifiers bespoke in detection, *i.e.,* feature-set and algorithm. Therefore, detection is divided into attack types using a subset of features of all available features F.

Let $F_i \subset F$ denote the feature set used by a classification model c_i, and $y_{i,t}$ be the prediction class at time t. We assume that the cost to extract features varies across the feature space, as some need capturing in real-time, *e.g.,* network traffic, whereas some are less resource intensive, *e.g.,* standardised logs. Thus, the problem can be formulated using subsets of features based on their scope and ordered by their computational cost.

The proposed system receives an incoming observation X_t at time step t and through the activation of some its classifiers predicts the class of the observation. For this purpose, the system produces a probability vector O_t, which represents the classification probabilities of each classifier. Let $O_t = P_1, P_{2, \ldots,} P_n$ denote the classification probability P_i of each model c_i at each time step t. The aim is to reduce unnecessary computational expenditure, so subsequent models $c_{i+1, \ldots,} c_n$ are only activated to perform inference when P_i falls within L_i or U_i. We refer to this subsequent activation process as *triaging*. In this case $L_i < P_i < U_i$, and L_i and U_i are the upper and lower uncertainty values, respectively that are defined in the sequel. For instance, if $L_i = 0.32$ and $U_i = 0.68$, a prediction within the range, *i.e.,* $L_i < P_i < U_i$, will be sent to c_{i+1} for subsequent classification, otherwise the prediction is the final output. Assuming the classifiers follow a hierarchy or linearity based on cost, the implications of reducing the number of subsequent classifications is a reduction in the overall detection performance.

To give the decision process flexibility, the traditional single and symmetrical decision boundary is forked to serve dual movement across the decision environment. Since the uncertainty between each class is independent per classifier, the discrete class distributions of c_i given P_i is modelled to derive boundary values L_i and U_i, where $i < n - 1$.

3.2. Action selection

As mentioned previously, the classification boundary consists of a

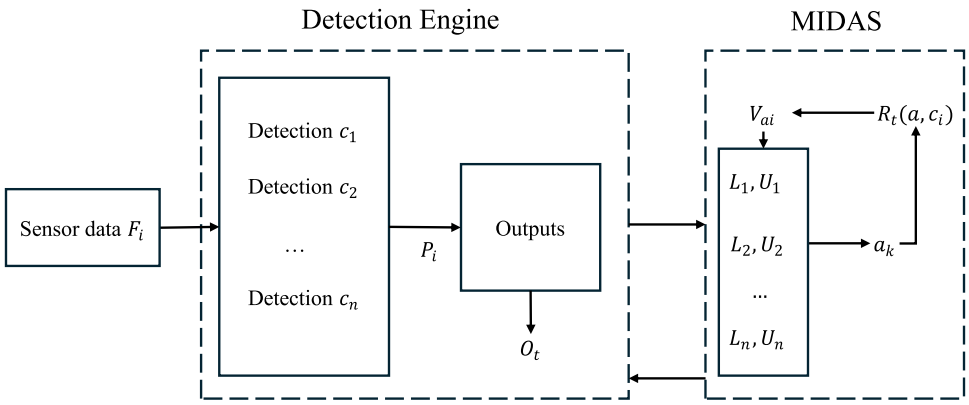

Fig. 1. Overview of MIDAS.

Table 1
List of symbols and description.

Symbol	Description
O_t	vector of classification probabilities for observation O at time t
C	list of available classifiers
c_i	classifier i
A	set of all possible actions
a	selected action value
k	contextual parameter
a_k	action at k index
V_{ai}	value estimate vector of action a for classifier c_i
U_i	upper decision boundary of classifier c_i
L_i	lower decision boundary of classifier c_i
$V_{a_i}^{(U \text{ or } L)}$	value estimate vector of action a for boundary U_i or L_i
$R_{t(a)}$	reward for action a at time step t
F	feature vector
F_i	subset of feature vector F used by c_i
P_i	probability estimate of classifier c_i
m	size of reward in bytes
$N_{t(a)}$	the number of rewards for action a up to time t
X_t	observation at time t
y_t	ground truth class label for observation X_t
$y_{i,t}$	prediction class for classifier c_i at time step t
d_{pa}	distance between P and a
D_P	minority dataset class
D_N	majority dataset class
ρ	imbalanced ratio
μ	action increment value
ν	total number of actions in set A
T	vector of distances between action a and prediction P
Z	Initial boundary value

lower and an upper boundary L_i and U_i, as each class (*i.e.,* benign, and malicious) have discrete prediction confidence, and thus have independent probability distributions. We also define the initial decision boundary Z=0.5, which is naturally the symmetry point between the high confidence density points of each class. For instance, strong negative confidence is closer to 0 and strong positive is closer to 1.

As part of the continuous learning of the system, after each prediction for each observation X_t the boundary may need to be adjusted based on the accuracy of the prediction. Let $y_t \in \{0,\ 1\}$ denote the label of the observation X_t at time t, where $y_t = 0$ is benign, and $y_t = 1$ is malicious. Based on the predicted class $y_{i,t}$ of classifier c_i for the same observation at time t, an action a takes place either for L_i if $y_t = 0$, or for U_i, if $y_t = 1$. Thus, let A represent a set of actions $A = \{a_1, a_2, \ldots, a_\nu\}$, where $a \in [0,\ 1]$, ν is the total number of available actions and each action represents the movement of the boundary. In our proposed system, each decision boundary moves either, *i)* up, *ii)* down, or, *iii)* remains the same based on the chosen action α. Thus, if $P_i \geq Z$, then $U = Z + a_k$, otherwise if $P_i \leq Z$ then $L = Z - a_k$.

In this work, three commonly used MAB algorithms have been evaluated for the action selection function, namely: *Softmax, Upper Confidence Bound (UCB)*, and *Thomson*, which were defined in Section II. A. In this work we also extended UCB, as *UCBe,* by using the *update rule* that is defined by Eq. (1), instead of using a running average for storing the value estimate of each action.

Each algorithm has its own technique to select actions, which all rely on knowledge that is captured over time from interacting with the environment. A preliminary experiment is used to find best parameters on a small scale (50 iterations), before a comprehensive evaluation with 10,000 iterations was to be conducted across the several proposed reward policies. All three algorithms require a reward policy to operate and thus a reward value and value estimate for each action.

For this reason, we define $R_t(a)$ as the reward for action a, which is derived from the reward function (further explained in Section IV.C). The reward function is designed to observe the effects in the decision environment to guide the future of each decision boundary to an optimal position and maximise the cost benefits. The selection function determines the decision boundary using an expected value derived from historical rewards.

The value estimate V_{a,c_i} is used to reflect the expected value of selecting action a for classifier c_i, based on previous recorded performances of the decision boundary (L_i or U_i). The reward is initially assigned to V by collecting the evaluation result of how well the boundary performed at time t. The value estimates are updated to reflect the effect of the action in the environment, so over time the expected value becomes more accurate. Accurate estimates translate to better action selection and thus, observations are triaged more effectively.

The value estimates for actions across the classifiers are treated independently since the actions can vary by performance under different conditions, so V_{a,c_i} extends to $V_{a,c_i}^{(L)}$ and $V_{a,c_i}^{(U)}$, respectively, such that $V_{a,c_i}^{(L)} \neq V_{a,c_i}^{(U)}$. The sum of rewards for a given action $R_1(a, c_i) + R_2(a, c_i) + \ldots R_{N_t}(a, c_i)$ where N_t is the number of times the action has been selected up to step t, needs to be efficiently reflected as a value estimate. This is because the model is designed to run continuously over time, which will eventually lead to large amounts of memory being used to store the rewards. So, in order to reduce the resources needed to store the rewards, and thus value of an action, the following function used to calculate the expected value for each action is:

$$V_{a,c_i,t+1} = V_{a,c_i,t} + \gamma \left[R_t(a, c_i) - V_{a,c_i,t}\right], \tag{1}$$

where γ is the decay constant. Note $\gamma \in (0,\ 1]$. When γ is closer to 1, the immediate rewards are of greater significance, whereas closer to 0 leads to a lower significance. An empirical evaluation is used to decide the parameter.

To assess the selection algorithms with different parameters, two metrics were used to measure the performance, namely cumulative regret and total number of observations triaged. The regret is the

cumulative difference between the chosen action and the action with the highest expected gain. An observation is considered triaged when it undergoes subsequent inference. Here, the goal is to minimise actions that move the decision boundary in areas where a high-density of true-positive and true-negative detection predictions reside. This is because they have been correctly classified, and resources would otherwise be spent on subsequent classifications that do not alter the overall prediction score. The three main selection algorithms used are described in Section II.B.

3.3. Reward policy

We evaluated MIDAS with different reward policies to find the most performant under the conditions of the selection functions and decision environment described above. A reward is either positive or negative feedback value that can be derived from the environment. It should be designed to guide the selection algorithm to select the best action at the current step. In this work, a two-step process is used decide the reward at step *t, a)* the reward value *R*, and *b)* the reward sign (+ or -). The reward policies explored in this work are: *i) static cost, ii) imbalanced cost, iii) contextual index, and iv) distance*. Section IV.B describes the metrics used to evaluate the reward policies.

Static cost reward function. This is a simplistic reward policy where the reward value is either 1 or −1. However, this approach treats all eventualities equally, either positive or negative. For instance, an action that has a negative outcome, but close to a positive outcome will be negatively affected to the same extent as strongly negative outcome. The use of a static value would depend on identifying and manually setting an optimal cost value. In this work we considered a range of reward signals with a constant static value (*i.e.,* 0.9, 0.8, 0.7, and 0.6).

Contextual Index. As previously mentioned, in our environment, the use of a simplistic reward $(1/-1)$ will treat all eventualities equally, which is sub-optimal. Therefore, this work proposes a contextual index. As previously defined a_k denotes the action at k index, where $1 \leq k \leq \nu$, from all the available actions in A. This reward function uses index k as the contextual parameter to adjust the reward value. Actions A are ascending in order by value, so a_k is highest when k its closest to ν, and as such symmetrically closest to high confidence prediction density. The boundary's goal is to learn a policy that minimises the number of selected actions located near high confidence density, since its more costly to perform inference when the overall predictive class $y_{i,t}$ is unlikely to change. To handle the value scale, the *logarithmic* transformation is used so $R_t(a_k, c_i) = log(k)$.

Imbalanced Cost. A common challenge for attack detection is training on an imbalanced dataset, which is representative of the detection environment, *i.e.,* less malicious observations than benign. This results to a "biased" reward function that does not treat equally TPs and TNs. To overcome this, our work uses a component of the reward function proposed in (Lin et al., 2020), which considers the ratio of each class in the dataset. Specifically, the minority class D_P that weighted higher than that of the majority D_N, were used to define an imbalanced ratio $\rho = \frac{|D_P|}{|D_N|}$, which was shown to improve the detection performance.

Distance. An alternative contextual reward is defined as $R_t(a, c_i) = d_{P,a,t}$, where $d_{P,a,t}$ is the distance $\mid P_i - a_t \mid$ between the current prediction P_i and selected action a at step t for either L_i or U_i. It is assumed the distance is like an error, so we treat the error range $0.5 \leq d_{P,a,t} \leq 1$ since the initial decision boundary Z is 0.5 and the best prediction achievable is 1 for the positive (benign) class and 0 for the negative (malicious) class. The distance parameter is designed to ignore the wider environment by only considering P_i and a, so it should localise the reward for the given contextual position a.

Reward Sign. The reward for moving the decision boundary is two-fold. First, the predicted value from the classifier is checked against the label. Second, the predicted value is checked against the decision boundary itself. Thus, as shown in Eq. (2), the reward will be positive if the prediction is correct and the boundary was in a position to prevent *triaging*, otherwise the reward will be negative.

$$R_t(a, c_i) = \begin{cases} r, \left(\mathrm{P_i} \in (\mathrm{L_i}, \mathrm{U_i}) \; and \; y_{i,t} \neq y_t\right) \; or \; \left(P_i \notin (\mathrm{L_i}, \mathrm{U_i}) \; and \; y_{i,t} = y_t\right) \\ -r, \left(\mathrm{P_i} \in (\mathrm{L_i}, \mathrm{U_i}) \; and \; y_{i,t} = y_t\right) \; or \; \left(P_i \notin (\mathrm{L_i}, \mathrm{U_i}) \; and \; y_{i,t} \neq y_t\right) \end{cases} \tag{2}$$

3.4. MIDAS

Algorithm 2 describes the flow of execution. Specifically, the main steps are as follow:

1) Each classifier c_i initialises L_i and U_i, and available actions are set as increments of μ.
2) For each observation from the dataset, the classifier c_i makes a prediction, P_i.
3) Then, the decision boundary for the current prediction is selected by the chosen selection function using the predicted probability as a parameter.
4) The parameters of the selection function will determine the selection of an action either by *a)* exploitation, or *b)* exploration.
5) The prediction is evaluated against the new decision boundary position derived from a, and thus the decision of the classifier will either, *a)* remains as predicted, or, *b)* the observation is sent the next classifier c_{i+1} for further inference.
6) Finally, the action is evaluated against a set of criteria to determine the reward, which is then integrated into the value estimate to inform future selection.
7) The classification process continues while $i < n - 1$.

4. Evaluation

For the evaluation of MIDAS, a set of baseline detection models are established using commonly used supervised ML algorithms. Then, a series parameters are explored in terms of cost efficiency and attack detection performance and MIDAS is evaluated to assess the performance across different exploration algorithms and reward policies.

4.1. Datasets

The performance offered by MIDAS is evaluated using the following two datasets, namely *CCCS-CIC-AndMal-2020* (Rahali et al., 2020) and (Keyes et al., 2021) and *CIC-Bell-DNS-2021* (Mahdavifar et al., 2021). *CCCS-CIC-AndMal-2020* includes malware samples ranging from adware, backdoor, ransomware, and trojans. *CIC-Bell-DNS-2021* consists of lexical and open-source features derived from Domain Name System (DNS). These datasets were selected because of their realistic availability and importance across attack-based detection. In this paper, we refer to the datasets as AndMal, and Bell, respectively.

Since the classifiers in MIDAS follow a hierarchy or linearity based on cost, each dataset was split to form two feature sets to train the classifiers (*i.e.,* two classifiers per dataset). Importantly, the features were categorised as separate feature sets to represent the cost associated with collection and pre-processing. In particular, in *CCCS-CIC-AndMal-2020*, the API calls were used as the inexpensive set, (*i.e.,* layer 1). Then, the network, process, memory, battery, and logcat activity, were used as the costly feature set, (*i.e.,* layer 2). Similarly, in *CIC-Bell-DNS-2021,* the inexpensive set (*i.e.,* layer 1) uses the lexical features, as they are computationally inexpensive compared to features from third-party services, which were used as the costly set (*i.e.,* layer 2).

4.2. Experimental setup

Environment. The experiments were conducted with *Python 3.11* running on a HPE ProLiant DL380 Gen10 serving 64 cores (on Intel(R)

Algorithm 2
Decision Optimisation.

1: **Input:** $C = (c_1, c_2 \ldots, c_n)$, **X,** V **of dimensions** $k \times n$
2: **Initialization:** *nextLayer* ← *true*
3: **while** *nextLayer* **AND** $c_i \in C$ **do**
4: $P \leftarrow predict(c_i, X_i)$
5: $a, L_i a, U_i a \leftarrow SelectAction(P, V)$
6: **if** $P_i > U_{ia}$ **OR** $P_i < L_{ia}$ **then**
7: *nextLayer* ←, *false*
8: **end if**
9: R ← Reward(P, L_{ia}, U_{ia}, nextLayer)
10: $Vt{+}1 \underline{(a)} V_t + \alpha[R_t - V_t]$
11: **end while**

Xeon(R) Silver 4216 CPUs @ 2.10 GHz), 512GB memory and 12 TB of storage. In order to run experiments in parallel for the two different datasets we provisioned two virtual machines running each Ubuntu Desktop (version 22.0), which equally shared these resources, on a ESXi (Version 8) infrastructure. Moreover, the neural network models were trained on the Apple M2 2022 architecture with 16GB memory, using *TensorFlow2.0*, and *Scikit-learn* for random forest. Initially, the best *MAB* parameters were discovered using 50 trials (see Table 7 - Table 10), and then those selected based on performance were ran at 10,000 trials, where each trial processed a newly shuffled version of the dataset.

We opted for two detection algorithms across the experiments, namely a decision tree approach (Random Forest) and a neural network approach. Random forests are made up of a collection of decision trees and are known to be accurate (Tidjon et al., 2019). However, in cases of complex data, they will generalise poorly. Therefore, a feedforward neural network was implemented using a sequential model architecture to demonstrate a more generalisable approach. The model consists of an input layer that takes the n-dimensional shape of each given dataset. The first dense layer has 128 neurons with a ReLU activation function. This was followed by a dropout with a rate of 0.3 to prevent overfitting. The second dense layer has 64 neurons with a ReLU activation function. Finally, the output layer uses a sigmoid function to output a probability-based decision. The selected neural network parameter values for our experiments were determined using the grid-search technique (see Table 2). A *callback function* was used for early stopping. While the parameter values will differ depending on factors, such as dataset size, these worked well for our experiments.

To compare our proposed method to existing detection approaches, different baselines were used, namely: i) the full capacity to maximise detection of a model by training using all the available features in the dataset as a single model, and ii) a stacked approach is trained, where two separate classifiers using two different feature sets is trained (see Section IV.A). In each dataset we compare the performance with these baselines using both the neural network and random forest algorithms.

Detection Metrics. To evaluate detection performance the following metrics are used:

1) *Accuracy:* The accuracy of a detection model is the measure of overall performance differentiating benign from malicious observations. It is measured as follows: $ACC = \frac{TP+TN}{TP+TN+FP+FN}$, where TP = correctly identified as *malicious, FP* = incorrectly identified as *malicious, TN* = correctly identified as *benign*, and FN = incorrectly identified as *benign*.
2) *Precision:* The precision measures the proportion of correct, but positive predictions, as $\frac{TP}{TP+FP}$.
3) *Recall:* measures the proportion of actual positives that were identified correctly. Thus, the greater the measures the greater the performance, as $\frac{TP}{TP+FN}$.
4) *F1-Score:* The F1-score represents the recall and precision as a single metric as $\frac{2 \times Precision \times Recall}{Precision+Recall}$.

Table 2
Experimental parameters.

Neural Network	Parameters
Optimizer	Adam
Batch size	40
Epochs	100
Neurons	128, 64
Random Forest	
n_estimators	100

MAB Metrics. To evaluate the effectiveness of the decision boundary optimisation in an attack detection system we define and use the following metrics:

1) *Total Triaged:* We count the number of observations that undergo subsequent inference due to their initial prediction value and the decision boundary at time step *t*. In this work, our effort is to increase cost savings and thus, a suitable measure of cost reduction is to count how frequently the expensive model is used.
2) *Remaining false predictions:* An indicator of how well the decision boundary performs is to measure the number of remaining false predictions, *i.e.,* false positive and false negatives. Whilst a small number of remaining false predictions could indicate a greater performance, there is a strong correlation with the total number of triaged observations, as shown in Fig. 2 and Fig. 3. Thus, one should consider prioritising either variable. In this work, we want to minimise the number of triaged observations, and so we demonstrate the number of remaining false predictions can be a useful metric to decide system parameters.

4.3. Parameter selection

Since a decision boundary could be any number between 0 and 1, it is sensible to limit the number of available actions as increments (μ) of 0.03, so $0 < a < 0.5$ and $0.5 < a < 1$, respectively for L_i and U_i. This means there is enough available actions, but not too many.

MIDAS' effectiveness on low-cost attack detection is evaluated with 9 different reward policies, see Table 3. The reward policies have different reward scales due to their individual designs, and thus this limits the use of evaluation metrics like average reward and cumulative regret for directly comparing between policies. Initially, to test the resilience of the approaches and to discover the most optimal parameters for each policy, a series of parameter combinations across 50 iterations were tested, using the following metrics, a) total number of triaged observations, and b) remaining false predictions.

In this work we combine reward policies to test the hypothesis that the unique strength of each policy can be harnessed to achieve greater levels of performance. First, the *contextual index* is integrated with *distance* to comprehend the action environment, in particular the significance of the action relative to the cumulative probability density of model predictions (*i.e.*, reward policy f8, Table 3). Our work also considers integrating *imbalanced cost* with *distance (as* in reward policies *f1, f6, f7,* refer to Table 3.*)*.

For parameter selection, the decay constant γ and exploration term τ is evaluated to determine the best performing value. Softmax and UCB is implemented using the update rule described in Section III. The decay constant $\gamma \in \{0.1,\ 0.2,\ 0.4,\ 0.8,\ 1\}$ and the exploration term $\tau \in \{0.1,\ 0.2,\ 0.4,\ 0.8,\ 1\}$. To decide the most optimal value, one can select based on: *a)* higher overall detection rate, *b)* higher overall cost-saving, or *c)* a balanced approach. In this work, we prioritise policies with a higher-overall cost-saving. These preliminary results are summarised in the Appendix (*i.e.*, in Table 7, Table 8, Table 9, and Table 10).

Despite the UCB variants showing the most cost-efficient in terms of overall performance in Table 9 and Table 10, two of our proposed reward policies provide remarkable flexibility using Softmax. The

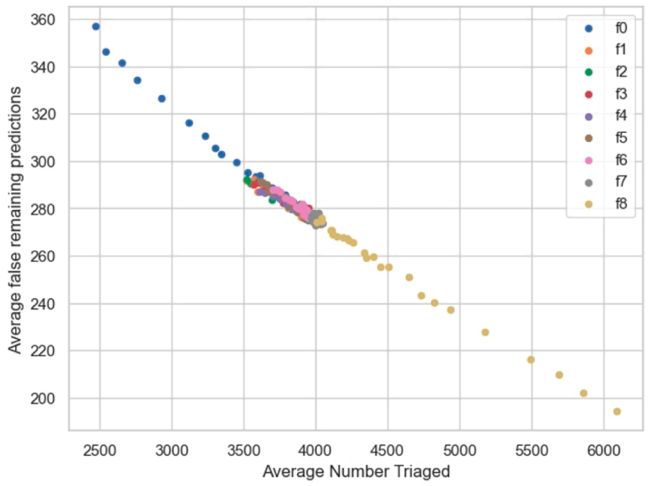

Fig. 2. Evaluating all combinations of the Softmax parameters, $\gamma \in \{0.1,\ 0.2,\ 0.4,\ 0.8,\ 1\}$, *and* $\tau \in \{0.1,\ 0.2,\ 0.4,\ 0.8,\ 1\}$ (Bell).

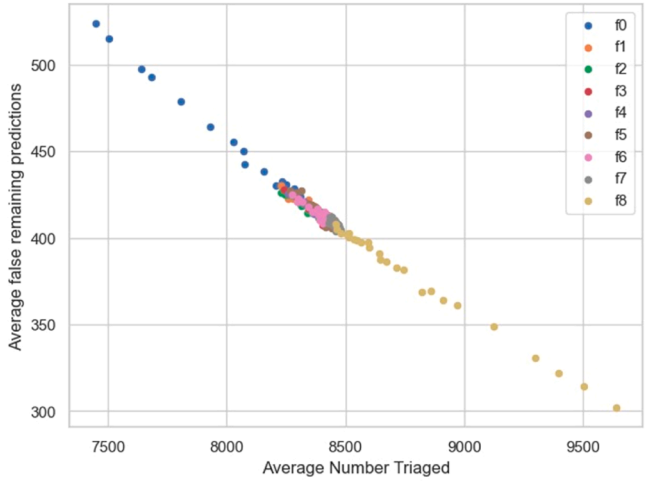

Fig. 3. Evaluating all combinations of the Softmax parameters, $\gamma \in \{0.1,\ 0.2,\ 0.4,\ 0.8,\ 1\}$, *and* $\tau \in \{0.1,\ 0.2,\ 0.4,\ 0.8,\ 1\}$ (AndMal).

Table 3
Reward policies used in our experiments.

Reward Policy	Description
f0	$log(k)$
f1	$\rho^{d_{P,a,t}}$
f2	0.9
f3	0.8
f4	0.7
f5	ρ
f6	$d^{\rho}_{P,a,t}$
f7	$\log\left(\rho^{d_{P,a,t}}\right)$
f8	$log(d_{P,a,t} \times k)$

correlation results are presented in Fig. 2 and Fig. 3. As shown, we can observe a strong correlation between remaining false predictions and triage rate across the range of parameters and reward policies. Specifically, *f0* and *f8* parameters can be selected across the correlation points, meaning that the system owner can adjust based on situational requirements, instead of solely relying on maximising cost-savings.

4.4. Cost-saving results

In this subsection, the cost-savings for each reward policy are explored. Here, the results of the top five performing reward policies are presented on the two datasets and two detection model types set out in Section IV.B. These results are summarised in Table 4.

The best performing reward policies show a significantly low triage rate. Across both datasets, the *f0* policy has the lowest triage rate, and consistently outperforms the other policies using different exploration functions. This indicates the reward policy is consistent and leads to cost-saving behaviour despite the type of dataset and model we used.

In this work the cost refers to the frequency of activation of the more expensive layer and thus cost-saving is calculated as $\frac{total\ observations - total\ triaged}{total\ observations}$. In *Bell* using random forest, the total cost with the use of MIDAS is 5.69 % (734.78/12,904), a cost-saving of 94.31 %. To compare, the neural network costs 13.16 % (1697.62/12,904) saving a total of 86.84 %, rivalled to an approach that utilises all resources. Similarly, MIDAS in the *AndMal* dataset with the neural network spends 23.91 % (3834.04/16,032), a cost-saving of 76.08 %, and random forest spends 11.99 % (1922.68/16,032), a cost-saving of 88 %.

Among the four selection functions under evaluation, the policies that use UCB, UCBe and Thomson outperform Softmax in terms of cost saving. The UCB variants competitively triage the most, where the dataset influenced UCB over UCBe. Softmax performs well in balancing cost saving with performance, suggesting Softmax is a good candidate for systems with different requirements (*e.g.,* increased detection performance), as seen in Fig. 2 and Fig. 3.

4.5. Classification risk

To demonstrate the risk associated with triaging, we measure the number of observations that undergo subsequent classification. The intuition here is that predictions from an initial classification (*i.e.,* layer 1) can be changed by subsequent predictions (*i.e.,* layer 2) enough to change the overall class, *i.e.,* a true negative could become a false positive.

Comparison results between reward policies and selection algorithms across the model type and datasets are presented in Fig. 4 and Fig. 5. It can be seen that *f0* consistently has the lowest risk across each dataset, model type and selection function. Here, we want to minimise intruding on the true prediction space, although the dataset can heavily influence this if there is overlapping probability distributions between true and false model predictions. Evidently, the triage rate correlates with the type of prediction going for subsequent classification. However,

Table 4
Top 5 reward policies for each dataset and model with regards to triage rate.

Dataset	Model	Reward policy	Selection func.	Triage rate
Bell	Neural Network	***f0***	**UCB**	**1697.62**
		f0	Thomson	2313.1
		f0	UCBE	2335.92
		f1	Thomson	3368.2
		f2	Thomson	3534.04
	Random Forest	***f0***	**UCBe**	**734.78**
		f0	UCB	833.49
		f0	Thomson	1361.10
		f1	Thomson	1669.62
		f2	Thomson	1716.40
AndMal	Neural Network	***f0***	**UCB**	**3834.12**
		f0	UCBE	4056.16
		f0	Thomson	5142.38
		f6	Thomson	5181.86
		f1	Thomson	6082.1
	Random Forest	***f0***	**UCBe**	**1922.68**
		f0	UCB	2001.16
		f0	Thomson	3069.08
		f6	Thomson	3412.7
		f1	Thomson	3473.54

a reward policy, such as *f0* that minimises the true prediction space is preferred, meaning the problem becomes predominantly down to detection performance by the initial model (*i.e.,* layer 1).

Usually, a model will have overlapping probability distributions, *i.e.,* coinciding false and true predictions. To maximise detection performance and marginally save on cost, one could consider *f8*, since it makes best attempts to capture as many negative observations as possible. A consequence of this design means an overlap of true and false prediction confidences can cause unnecessary classifications, thus incurring cost for those true predictions that do not need it (see Table 5).

4.6. Attack detection results

In this subsection, the detection performance of MIDAS is evaluated with the use of accuracy, precision, recall and F1-score. Our results, which are and ranked by F1-score, are summarised in Table 5.

In particular, the results suggest that the accuracy across the top 5 policies used by MIDAS in the Bell dataset does not fall below 86 %, and 95 % for AndMal. On the Bell dataset, *f2* using *Thomson* performed highly to the rest of the reward policies on detection performance. However, as shown in

Table 4, this policy was the least cost effective in the top 5 policies, an average of 52 % decline compared to *f0*. The *f0* policy consistently appears within the top 5 performant policies across all 4 experiments (*i.e.,* both datasets and both models), with the use of different selection algorithms (*i.e.*, Thomson, UCB and UCBE). The *f1* using *Thomson* is in second place in terms of accuracy, across 3/4 experiments. Overall, the experiments show that the detection performance does not have significant difference in the two datasets used for all policies (*e.g.,* ~1 % variation in accuracy). Finally, while Softmax selection did not perform as well as the top 5, it does provide some flexibility, as shown in Fig. 2 and Fig. 3.

Comparison with baseline architectures. We have compared the detection performance of MIDAS with the baseline architectures shown in Table 6. Our work is comparable with modern cloud-based phishing and/or malware detection engines, such as Microsoft Defender. Such a cloud-based detection engine, similar to our approach, includes a host-based, thus inexpensive, detection engine that undertakes processing and classification of the majority of files and/or URLs. It also uses a computational expensive, cloud-based, detection engine, which is activated only when the cheaper, host-based detection engine is not able to make a decision. As such, their different combinations, have been used as baseline architecture for the evaluation of MIDAS.

In particular, the *Large* model is trained on all the available features within the datasets, *i.e., Bell* and *AndMal*. The *Light* models, denoted L1 and L2, are models trained on a subset of the datasets' features, as described in Section IV.B. The Stacked model uses both L1 and L2, where the output is formed by averaging the predictive probabilities of both models in the stack. The cost of the stacked approach is comparable to the Large model. Finally, we compare the approaches with two MIDAS approaches based on *a)* the highest performance, and *b)* lowest triage rate, based on the aforementioned results (see Table 4 and Table 5).

As the results suggest in Table 6, the Large and Stacked model is the highest overall performing, as expected. Similarly, *L2* is expected to perform better than *L1* due to the use of more expensive features. There is also a minor degradation to detection performance for the MIDAS approaches where significant cost-savings are present. Both UCB variants and Thomson have either performed with the greatest performance, or the greatest cost-saving, and thus have made it into Table 5.

During the evaluation with Bell dataset, *f0* with the use of UCB and *f1* with the use of Thomson, using NN, we noticed that there is only a ~1 % degradation in performance to compensate for the 50.4 % cost savings by *f0* over *f1*. Compared to *Large* model, both *f0* and *f1* have a 1 % loss in accuracy, and a precision loss of 4 % and 3 %, respectively. Interestingly, the *L2* model outperforms the *Large* model on accuracy at 95 %, a 7 % gain over *Large*. In the Bell dataset where random forest was used in the

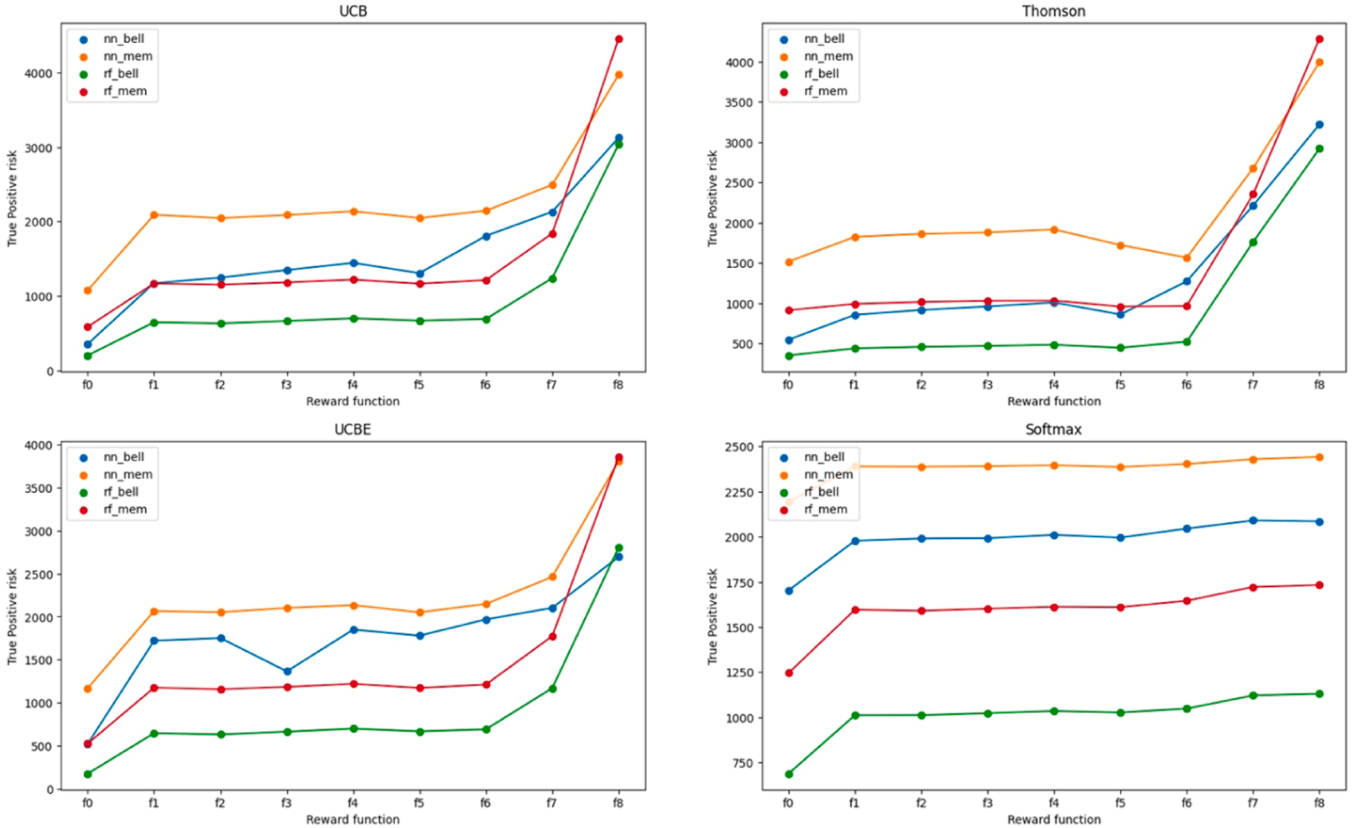

Fig. 4. The total number of true positives going to layer 2, where each plot represents the selection function.

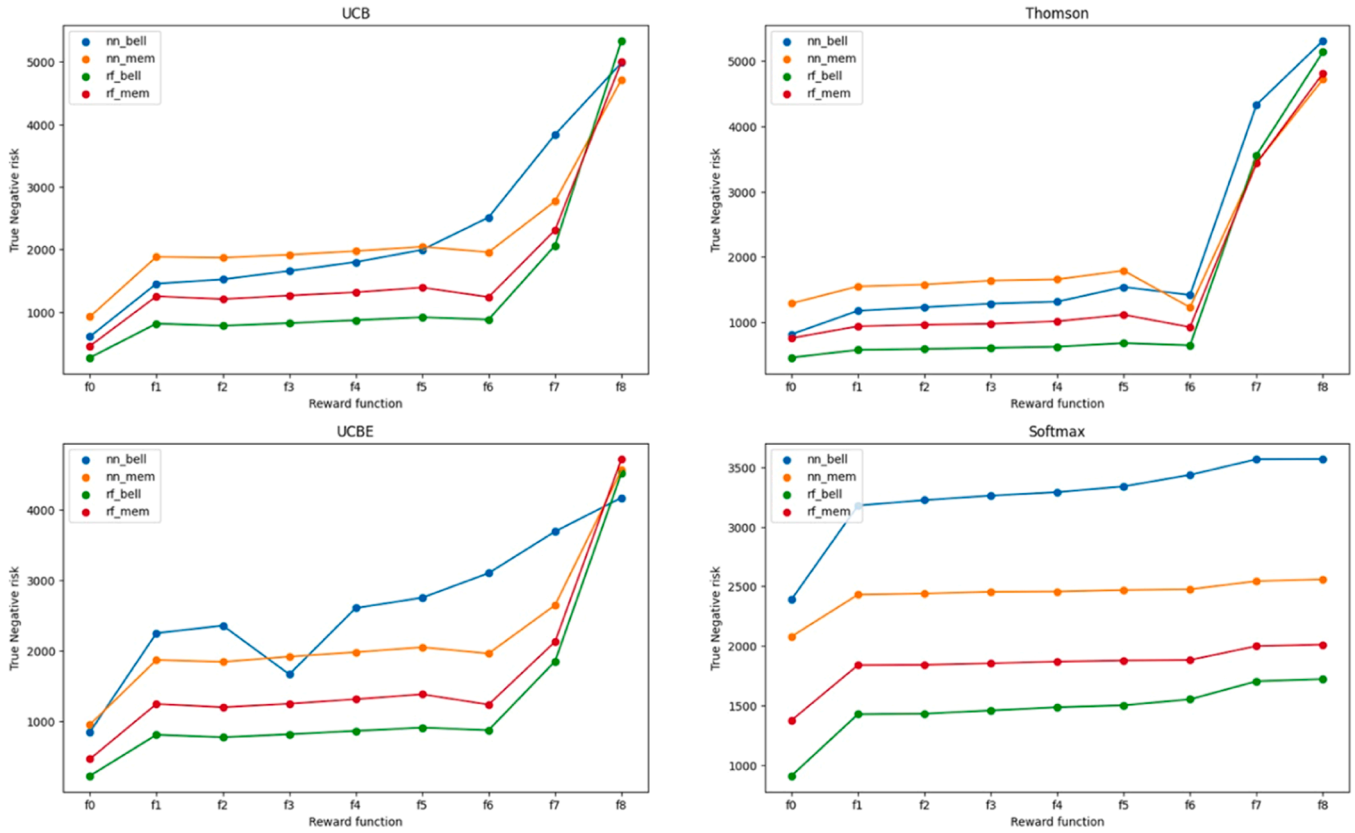

Fig. 5. The total number of true negatives going to layer 2, where each plot represents the selection function.

Table 5
Detection performance of each reward policy in order of F1-score (top 5).

Dataset	Model	Reward Policy	Selection func.	Accuracy	Precision	Recall	F1-score
Bell	Neural Network	***f1***	**Thomson**	**0.873**	**0.864**	**0.883**	**0.873**
		f2	Thomson	0.873	0.863	0.884	0.873
		f0	Thomson	0.871	0.86	0.88	0.87
		f0	UCBe	0.87	0.858	0.879	0.868
		f0	UCB	0.867	0.854	0.874	0.864
	Random Forest	***f2***	**Thomson**	**0.885**	**0.872**	**0.901**	**0.886**
		f1	Thomson	0.884	0.871	0.9	0.885
		f0	Thomson	0.883	0.87	0.899	0.884
		f0	UCBe	0.88	0.87	0.895	0.882
		f0	UCB	0.879	0.868	0.895	0.881
AndMal	Neural Network	***f0***	**Thomson**	**0.961**	**0.951**	**0.971**	**0.961**
		f6	Thomson	0.961	0.952	0.971	0.961
		f1	Thomson	0.961	0.951	0.97	0.961
		f0	UCB	0.959	0.948	0.971	0.96
		f0	UCBe	0.96	0.949	0.972	0.96
	Random Forest	***f0***	**UCBe**	**0.993**	**0.99**	**0.996**	**0.993**
		f0	UCB	0.993	0.99	0.996	0.993
		f0	Thomson	0.992	0.987	0.996	0.992
		f6	Thomson	0.992	0.987	0.996	0.991
		f1	Thomson	0.991	0.987	0.996	0.991

models, there is a 42.8 % greater cost saving by *f0* using UCBe. In comparison, *f2* with Thomson there is only a 1 % accuracy difference between the two at 88 % and 89 %. However, here *Large* outperforms all approaches with 96 % accuracy. As a result, the evaluation showed that using MIDAS in this instance, leads to a loss of ~8 % in detection performance to compensate for the cost-savings.

The evaluation with *AndMal* dataset show high detection performance for both models (i.e., RF and NN), which is likely due to the nature of the training on the dataset. When using NN, both MIDAS approaches, namely *f0* with Thomson and *f0* with UCB, had almost the same detection performance, where *f0* with UCB has a 74.5 % greater cost saving over *f0* with Thomson. Similarly, when random forest was used in *AndMal, f0* with UCBe was found to have both, the highest detection performance and cost savings, outperforming Stacked and comparable to *Large*.

5. Discussion

The results from the evaluation of our framework suggest that the parameters of the exploration algorithm greatly influenced the cost-to-performance dilemma. That is, a cost saving at the expense of a possible decrease in detection performance. In specific cases, Softmax

Table 6
Comparison of MIDAS with other detection architectures. Large model is trained on all the available features within the dataset. L1 and L2 are models trained on a subset of the datasets' features, as described in Section IV.B. The Stacked model uses both L1 and L2, with output being the average of their predictive probabilities.

Dataset	Model	Architecture	Accuracy	Precision	Recall	F1-score	Triage Rate
Bell-DNS	Neural Network	Large	0.88	0.89	0.85	0.87	N/A
		Stacked	0.88	0.90	0.83	0.86	
		Light (L1)	0.69	0.73	0.53	0.62	
		Light (L2)	0.95	0.88	0.86	0.87	
		MIDAS (f0, UCB)	0.87	0.85	0.87	0.86	1697.62
		MIDAS (f1, Thomson)	0.87	0.86	0.88	0.87	3368.2
	Random Forest	Large	0.96	0.95	0.96	0.95	N/A
		Stacked	0.94	0.95	0.92	0.94	
		Light (L1)	0.86	0.88	0.82	0.85	
		Light (L2)	0.96	0.95	0.96	0.95	
		MIDAS (f2, Thomson)	0.89	0.87	0.90	0.89	1716.40
		MIDAS (f0, UCBe)	0.88	0.87	0.90	0.88	734.78
AndMal	Neural Network	Large	0.97	0.97	0.97	0.97	N/A
		Stacked	0.99	0.99	0.99	0.99	
		Light (L1)	0.75	0.71	0.78	0.74	
		Light (L2)	0.99	0.99	0.99	0.99	
		MIDAS (f0, Thomson)	0.96	0.95	0.97	0.96	5142.38
		MIDAS (f0, UCB)	0.96	0.95	0.97	0.96	3834.12
	Random Forest	Large	0.99	0.98	0.99	0.99	N/A
		Stacked	0.98	0.98	0.98	0.98	
		Light (L1)	0.81	0.81	0.79	0.80	
		Light (L2)	0.99	0.99	0.99	0.99	
		MIDAS (f0, UCBe)	0.99	0.99	0.99	0.99	1922.68

was found to provide greatest flexibility across the parameter ranges, which would benefit those that have a varied risk appetite.

As expected during our experiments high detection performance was achieved by those approaches that triage the most, *i.e.,* have high cost. This is because the performance of multiple models combined is highest compared to a single model that is trained on a subset of features. However, as discussed in Section IV, our novel contextual index reward policy (*f0,* seeTable 3*)* allows MIDAS to provide cost-effective detection, where the detection performance is not far from the top detection architecture, described in Table 6. A system owner can opt to select reward functions that sacrifice computation cost for more detection and *vice versa*. Nonetheless, in a realistic scenario where the cost to operate multiple models is too high, in particular when humans are involved, one has to consider the cost implications. The analysis of our reward functions found that the contextual index policy (*f0)* outperformed the rest of the reward functions with regards to cost-saving across 3 out of 4 selection algorithms, across both classifier models and both datasets, thus showing consistency.

Our results indicate that the model's detection performance on its respective dataset contributes to the nature of the overall cost-saving. Not all detection models will classify with a clear separation between confident and non-confident predictions by its ground-truth. Therefore, there should be an expectation that overlap will exist. Also, not all datasets are rich enough to train a model so that all false predictions are low confidence and thus caught by our decision optimisation. For this reason, improving the detection performance should continue to be prioritised, so that our model can prioritise cost-saving in the 'right place', *i.e.,* within high-confidence prediction space.

Furthermore, as our experiments suggest, the strength of a detection model will influence the true prediction probabilities in terms of confidence scores. In our experiments, the two datasets and two classifier models provided sufficient variation in performance, and thus confidence ranges from the predictions. Ideally, a model should be trained so that incorrect predictions are more likely to be low in confidence compared to correct predictions.

MIDAS was evaluated using a detection architecture for malware/phishing having two layers, *i.e.,* an architecture that resembles that of commercial cloud-based detection products, such Microsoft Defender. In particular, Microsoft Defender uses, similar to MIDAS, an inexpensive detection engine that is on the customers premise, which is complimented by a more expensive cloud-service which is hosted by Microsoft. In Microsoft Defender architecture the expensive model is only activated when the local detection engine cannot make a decision. Nonetheless, MIDAS could be extended to use additional models, or model equivalents, such as a human analyst. However, one must consider an increase in detection models could introduce a challenge with ordering the models in sequence of cost and performance trade-offs, as set out in the design. We consider this outside the scope of this paper and leave it for future work.

Finaly, the sign of the reward signal (*i.e.,* positive or negative reward) depends on supervision, *i.e.,* a label, which could be ineffective if there are no new labels co-occurring with observations evolving. This can be seen as a concept drift problem and affects most model-based techniques. A deployed detection model may not have a constant set of labelled data and thus, the value estimation of an action will rely on a) intermittent labels, or b) probabilistic labels. It is assumed that a detection model will be regularly updated, either by model retraining, or using online models as new labelled data become available, *e.g.,* via human feedback.

6. CONCLUSION

In this paper, we propose a novel decision optimisation framework, MIDAS, for a multi-layered detection architecture, using a novel value estimate model, with aim to: a) correctly identify attacks, and b) minimise the expenditure on processing and inference. MIDAS provides the ability to adapt decision-making in multi-layered attack detection by using multi-armed bandits to adjust a dynamic decision boundary. MIDAS cost-effectiveness and detection performance was evaluated in a multi-layered detection architecture against two prevalent threats, phishing and malware. The first layer was inexpensive and undertook the classification of the majority of observations, whereas the second is more costly with regards to computation. This use case resembles the architecture of commercial detection solutions, such as Microsoft Defender, which use a computational expensive, cloud-based, detection engine only when the cheaper, local detection engine is not able to make a decision.

For the evaluation of MIDAS, we used CCCS-CIC-AndMal-2020 and CIC-Bell-DNS-2021 datasets, with models trained in an architecture using two layers. The evaluation showed that MIDAS is a cost-effective data-driven countermeasure against phishing and malware, without sacrificing detection performance. To further address the adaptability of

MIDAS, future work should explore the available actions with deeper analysis, in particular for multi-classification approach. Furthermore, the frequency of action selection should be investigated to avoid value estimate updates incurring unnecessary computational cost.

CRediT authorship contribution statement

Kieran Rendall: Writing – original draft, Resources, Methodology, Investigation, Formal analysis, Conceptualization. **Alexios Mylonas:** Writing – review & editing, Validation, Supervision, Resources, Methodology, Formal analysis, Conceptualization. **Stilianos Vidalis:** Writing – review & editing, Validation, Resources, Methodology, Formal analysis. **Dimitris Gritzalis:** Writing – review & editing, Validation, Methodology, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix

To demonstrate MIDAS across many different available parameters, specifically for Softmax and UCBe, we ran each policy across the parameter combinations to identify its cost saving performance. Tables 7–10 present the results for the best performing parameter/s.

Table 7
Parameter Selection (Softmax, Neural Network) - 50 Trials.

Softmax, Neural Network								
	Bell				Mal			
Reward Policy	γ	τ	Triaged	F. Remaining	γ	τ	Triaged	F. Remaining
f0	1	1	6499.82	2023.02	1	1	6499.82	2023.02
f1	1	1	8009.28	1633.06	1	1	8009.28	1633.06
f2	1	1	8076.68	1619.44	1	1	8076.68	1619.44
f3	1	1	8127.92	1604.56	1	1	8127.92	1604.56
f4	1	1	8205.30	1582.82	1	1	8205.30	1582.82
f5	1	1	8253.24	1571.06	1	1	8253.24	1571.06
f6	1	1	8457.90	1522.00	1	1	8457.90	1522.00
f7	0.1	0.2	8694.24	1469.26	0.1	0.2	8694.24	1469.26
f8	0.1	0.1	8700.64	1469.10	0.1	0.1	8700.64	1469.10

Table 8
Parameter Selection (Softmax, Random Forest) - 50 Trials.

Softmax, Random Forest								
	Bell				Mal			
Reward Policy	γ	τ	Triaged	F. Remaining	γ	τ	Triaged	F. Remaining
f0	1	1	2471.78	994.88	1	1	4462.76	1259.14
f2	1	1	3517.80	812.44	1	1	5556.86	973
f1	1	1	3518.34	812.44	1	1	5558.92	973.88
f3	1	1	3565.60	812.76	1	1	5598.08	963.46
f4	1	1	3613.02	801.44	1	1	5640.58	958.86
f5	1	1	3617.46	796.64	1	1	5632.62	957.58
f6	1	1	3705.58	797.00	1	1	5663.48	962.6
f7	0.1	0.2	3971.42	750.90	0.1	0.1	5923.32	897.4
f8	0.1	0.1	4008.30	742.88	0.1	0.1	5966.32	884.92

Table 9
Parameter Selection (UCBe, Neural Network) - 50 Trials.

UCBe, Neural Network						
	Bell			Mal		
Reward Policy	γ	Triaged	F. Remaining	γ	Triaged	F. Remaining
f0	1	1710.04	3363.68	0.2	3748.52	2361.28
f1	1	4256.46	2609.70	1	7000.26	1151.98
f2	1	4483.22	2550.38	1	6932.22	1173.04
f3	0.8	4849.22	2457.20	1	7052.08	1136.26
f4	1	5187.36	2358.32	1	7224.08	1081.16
F5	1	5306.38	2331.42	1	7192.84	1106.3
f6	1	6780.70	1937.70	0.8	7210.6	1099.1
f7	0.1	8888.50	1423.24	0.1	8592.04	734.48
f8	0.1	10,338.44	1121.68	0.1	12,346.96	258.04

Table 10
Parameter Selection (UCBe, Random Forest) - 50 Trials.

UCBe, RF						
	Bell			Mal		
Reward Policy	γ	Triaged	F. Remaining	γ	Triaged	F. Remaining
f0	0.2	770.94	1429.88	0.2	1921.74	2164
f1	1	2222.86	1033.16	1	4219.24	1303.56
f2	1	2305.84	1005.20	0.1	4110.92	1344.78
f3	1	2316.48	1015.14	1	4217.44	1322.04
f4	0.8	2383.62	1034.58	1	4358.88	1275.56
F5	1	2423.68	994.92	1	4381.02	1273.76
f6	1	2449.32	985.80	0.8	4171.94	1378.14
f7	0.1	4215.98	719.82	0.1	6178.2	833.62
f8	0.1	8961.10	328.60	0.1	11,618.7	129.02

Data availability

No data was used for the research described in the article.

References

Al-Sarem, M., Saeed, F., Al-Mekhlafi, Z.G., Mohammed, B.A., Al-Hadhrami, T., Alshammari, M.T., Alreshidi, A., Alshammari, T.S., 2021. An Optimized Stacking Ensemble Model for Phishing Websites Detection. Electronics. (Basel) 10 (11), 1285. https://doi.org/10.3390/electronics10111285.

Auer, P., Cesa-Bianchi, N., Fischer, P., 2002. Finite-time Analysis of the Multiarmed Bandit Problem. Mach. Learn. 47 (2/3), 235–256. https://doi.org/10.1023/A:1013689704352.

Bahnsen, A.C., Bohorquez, E.C., Villegas, S., Vargas, J., Gonzalez, F.A., 2017. Classifying phishing URLs using recurrent neural networks. In: 2017 APWG Symposium on Electronic Crime Research (ECrime), pp. 1–8. https://doi.org/10.1109/ECRIME.2017.7945048.

Bertrand Van Ouytsel, C.-H., Dam, K.H.T., Legay, A., 2024. Analysis of machine learning approaches to packing detection. Comput. Secur. 136, 103536. https://doi.org/10.1016/j.cose.2023.103536.

Bhat, P., Behal, S., Dutta, K., 2023. A system call-based android malware detection approach with homogeneous & heterogeneous ensemble machine learning. Comput. Secur. 130, 103277. https://doi.org/10.1016/j.cose.2023.103277.

Birman, Y., Hindi, S., Katz, G., Shabtai, A., 2022. Cost-effective ensemble models selection using deep reinforcement learning. Information Fusion 77, 133–148. https://doi.org/10.1016/j.inffus.2021.07.011.

Chen, T., Zeng, H., Lv, M., Zhu, T., 2024a. CTIMD: cyber threat intelligence enhanced malware detection using API call sequences with parameters. Comput. Secur. 136, 103518. https://doi.org/10.1016/j.cose.2023.103518.

Chen, X., Qiu, W., Chen, L., Ma, Y., Ma, J., 2024b. Fast and practical intrusion detection system based on federated learning for VANET. Comput. Secur. 142, 103881. https://doi.org/10.1016/j.cose.2024.103881.

Chohra, A., Shirani, P., Karbab, E.M.B., Debbabi, M., 2022. CHAMELEON: optimized feature selection using particle swarm optimization and ensemble methods for network anomaly detection. Computers and Security 117. https://doi.org/10.1016/j.cose.2022.102684.

CrowdStrike. (2023). *2023 Global Threat Report*.

Das, A., Baki, S., El Aassal, A., Verma, R., Dunbar, A., 2020. SoK: a Comprehensive Reexamination of Phishing Research from the Security Perspective. IEEE Communications Surveys and Tutorials 22 (1), 671–708. https://doi.org/10.1109/COMST.2019.2957750.

Dekel, L., Leybovich, I., Zilberman, P., Puzis, R., 2023. MABAT: a Multi-Armed Bandit Approach for Threat-Hunting. IEEE Transactions on Information Forensics and Security 18, 477–490. https://doi.org/10.1109/TIFS.2022.3215010.

Deng, Z., Chen, K., Meng, G., Zhang, X., Xu, K., Cheng, Y., 2022. Understanding Real-world Threats to Deep Learning Models in Android Apps. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, pp. 785–799. https://doi.org/10.1145/3548606.3559388.

Dodia, P., AlSabah, M., Alrawi, O., Wang, T., 2022. Exposing the Rat in the Tunnel: using Traffic Analysis for Tor-based Malware Detection. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, pp. 875–889. https://doi.org/10.1145/3548606.3560604.

Dong, F., Wang, L., Nie, X., Shao, F., Wang, H., Li, D., Luo, X., Xiao, X., 2023. DISTDET: a Cost-Effective Distributed Cyber Threat Detection System. In: 32nd USENIX Security Symposium (USENIX Security 23), pp. 6575–6592. https://www.usenix.org/conference/usenixsecurity23/presentation/dong-feng.

Doshi, J., Parmar, K., Sanghavi, R., Shekokar, N., 2023. A comprehensive dual-layer architecture for phishing and spam email detection. Comput. Secur. 133, 103378. https://doi.org/10.1016/j.cose.2023.103378.

Fang, W., He, J., Li, W., Lan, X., Chen, Y., Li, T., Huang, J., Zhang, L., 2023. Comprehensive Android Malware Detection Based on Federated Learning Architecture. IEEE Transactions on Information Forensics and Security 18, 3977–3990. https://doi.org/10.1109/TIFS.2023.3287395.

Fatima, A., Maurya, R., Dutta, M.K., Burget, R., Masek, J., 2019. Android Malware Detection Using Genetic Algorithm based Optimized Feature Selection and Machine Learning. In: 2019 42nd International Conference on Telecommunications and Signal Processing (TSP), pp. 220–223. https://doi.org/10.1109/TSP.2019.8769039.

Gao, X., Shan, C., Hu, C., Niu, Z., Liu, Z., 2019. An Adaptive Ensemble Machine Learning Model for Intrusion Detection. IEEe Access. 7, 82512–82521. https://doi.org/10.1109/ACCESS.2019.2923640.

Gritzalis, D.A., 2004. Embedding privacy in IT applications development. Information Management & Computer Security 12 (1), 8–26. https://doi.org/10.1108/09685220410518801.

Gupta, N., Jindal, V., Bedi, P., 2022. CSE-IDS: using cost-sensitive deep learning and ensemble algorithms to handle class imbalance in network-based intrusion detection systems. Comput. Secur. 112, 102499. https://doi.org/10.1016/j.cose.2021.102499.

Gupta, S., Singh, B., 2024. An intelligent multi-layer framework with SHAP integration for botnet detection and classification. Comput. Secur. 140, 103783. https://doi.org/10.1016/j.cose.2024.103783.

Heartfield, R., Loukas, G., Bezemskij, A., Panaousis, E., 2021. Self-Configurable Cyber-Physical Intrusion Detection for Smart Homes Using Reinforcement Learning. IEEE Transactions on Information Forensics and Security 16, 1720–1735. https://doi.org/10.1109/TIFS.2020.3042049.

Iliadis, J., Spinellis, D., Gritzalis, D., Preneel, B., Katsikas, S., 2000. Evaluating certificate status information mechanisms. In: Proceedings of the 7th ACM Conference on Computer and Communications Security, pp. 1–8. https://doi.org/10.1145/352600.352603.

Jacobs, A.S., Beltiukov, R., Willinger, W., Ferreira, R.A., Gupta, A., Granville, L.Z., 2022. AI/ML for Network Security: the Emperor has no Clothes. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, pp. 1537–1551. https://doi.org/10.1145/3548606.3560609.

Keyes, D.S., Li, B., Kaur, G., Lashkari, A.H., Gagnon, F., Massicotte, F., 2021. EntropLyzer: android Malware Classification and Characterization Using Entropy Analysis of Dynamic Characteristics. In: 2021 Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge (RDAAPS), pp. 1–12. https://doi.org/10.1109/RDAAPS48126.2021.9452002.

Kuleshov, V., Precup, D., 2000. Algorithms for multi-armed bandit problems. Journal of Machine Learning Research 1–48.

Lin, E., Chen, Q., Qi, X., 2020. Deep reinforcement learning for imbalanced classification. Applied Intelligence 50 (8), 2488–2502. https://doi.org/10.1007/s10489-020-01637-z.

Liu, S., Peng, G., Zeng, H., Fu, J., 2024. A survey on the evolution of fileless attacks and detection techniques. Comput. Secur. 137, 103653. https://doi.org/10.1016/j.cose.2023.103653.

Mahdavifar, S., Maleki, N., Lashkari, A.H., Broda, M., Razavi, A.H., 2021. Classifying Malicious Domains using DNS Traffic Analysis. In: 2021 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech), pp. 60–67. https://doi.org/10.1109/DASC-PICom-CBDCom-CyberSciTech52372.2021.00024.

Maniriho, P., Mahmood, A.N., Chowdhury, M.J.M., 2024. MeMalDet: a memory analysis-based malware detection framework using deep autoencoders and stacked ensemble under temporal evaluations. Comput. Secur. 142, 103864. https://doi.org/10.1016/j.cose.2024.103864.

Nisioti, A., Mylonas, A., Yoo, P.D., Katos, V., 2018. From Intrusion Detection to Attacker Attribution: a Comprehensive Survey of Unsupervised Methods. IEEE Communications Surveys & Tutorials 20 (4), 3369–3388. https://doi.org/10.1109/COMST.2018.2854724.

Otoum, S., Kantarci, B., Mouftah, H.T., 2020. A Novel Ensemble Method for Advanced Intrusion Detection in Wireless Sensor Networks. In: ICC 2020 - 2020 IEEE International Conference on Communications (ICC), pp. 1–6. https://doi.org/10.1109/ICC40277.2020.9149413.

Paleyes, A., Urma, R.-G., Lawrence, N.D., 2023. Challenges in Deploying Machine Learning: a Survey of Case Studies. ACM. Comput. Surv. 55 (6), 1–29. https://doi.org/10.1145/3533378.
Paya, A., Arroni, S., García-Díaz, V., Gómez, A., 2024. Apollon: a robust defense system against Adversarial Machine Learning attacks in Intrusion Detection Systems. Comput. Secur. 136, 103546. https://doi.org/10.1016/j.cose.2023.103546.
Rahali, A., Lashkari, A.H., Kaur, G., Taheri, L., GAGNON, F., Massicotte, F., 2020. DIDroid: android Malware Classification and Characterization Using Deep Image Learning. In: 2020 the 10th International Conference on Communication and Network Security, pp. 70–82. https://doi.org/10.1145/3442520.3442522.
Rao, R.S., Vaishnavi, T., Pais, A.R., 2019. PhishDump: a multi-model ensemble based technique for the detection of phishing sites in mobile devices. Pervasive Mob. Comput. 60, 101084. https://doi.org/10.1016/j.pmcj.2019.101084.
Rendall, K., Nisioti, A., Mylonas, A., 2020. Towards a multi-layered phishing detection. Sensors (Switzerland). https://doi.org/10.3390/s20164540.
Rookard, C., Khojandi, A., 2024. RRIoT: recurrent reinforcement learning for cyber threat detection on IoT devices. Comput. Secur. 140, 103786. https://doi.org/10.1016/j.cose.2024.103786.
Sagi, O., Rokach, L., 2018. Ensemble learning: a survey. WIREs Data Mining and Knowledge Discovery 8 (4). https://doi.org/10.1002/widm.1249.
Saha, S., Afroz, S., Rahman, A.H., 2024. MAlign: explainable static raw-byte based malware family classification using sequence alignment. Comput. Secur., 103714 https://doi.org/10.1016/j.cose.2024.103714.
Shen, F., Liu, Z., Perigo, L., 2023. Strategic Monitoring for Efficient Detection of Simultaneous APT Attacks with Limited Resources. International Journal of Advanced Computer Science and Applications 14 (3). https://doi.org/10.14569/IJACSA.2023.0140303.
Slivkins, A. (2019). *Introduction to Multi-Armed Bandits*.
Tidjon, L.N., Frappier, M., Mammar, A., 2019. Intrusion Detection Systems: a Cross-Domain Overview. IEEE Communications Surveys & Tutorials 21 (4), 3639–3681. https://doi.org/10.1109/COMST.2019.2922584.
Tseng, C.H., Chang, Y.-T., 2023. EBDM: ensemble binary detection models for multi-class wireless intrusion detection based on deep neural network. Comput. Secur. 133, 103419. https://doi.org/10.1016/j.cose.2023.103419.
van Geest, R.J., Cascavilla, G., Hulstijn, J., Zannone, N., 2024. The applicability of a hybrid framework for automated phishing detection. Comput. Secur., 103736 https://doi.org/10.1016/j.cose.2024.103736.
Wang, B., Yang, C., Ma, J., 2023. IAFDroid: demystifying Collusion Attacks in Android Ecosystem via Precise Inter-App Analysis. IEEE Transact. Inf. Forensics and Security 18, 2883–2898. https://doi.org/10.1109/TIFS.2023.3267666.
Wang, S., Chen, Z., Yan, Q., Ji, K., Peng, L., Yang, B., Conti, M., 2020. Deep and broad URL feature mining for android malware detection. Inf Sci (Ny) 513, 600–613. https://doi.org/10.1016/j.ins.2019.11.008.
Wu, Y., Li, M., Zeng, Q., Yang, T., Wang, J., Fang, Z., Cheng, L., 2023. DroidRL: feature selection for android malware detection with reinforcement learning. Comput. Secur. 128, 103126. https://doi.org/10.1016/j.cose.2023.103126.
Xin, D., Miao, H., Parameswaran, A., Polyzotis, N., 2021. Production machine learning pipelines: empirical Analysis and Optimization opportunities. In: Proceedings of the 2021 International Conference on Management of Data, pp. 2639–2652. https://doi.org/10.1145/3448016.3457566.
Xu, T., Goossen, G., Kerem Cevahir, H., Khodeir, S., Jin, Y., Kerem, H., Sara, C., Yingyezhe, K., Li, J.F., Sagar, S.S., David, P., Pearce, F.P., 2021. Deep entity classification: abusive account detection for online social networks. In: Proceedings of the 30th USENIX Security Symposium. https://www.usenix.org/conference/usenixsecurity21/presentation/xu-teng.
Yang, H., Wang, Y., Zhang, L., Cheng, X., Hu, Z., 2024. A novel Android malware detection method with API semantics extraction. Comput. Secur. 137, 103651. https://doi.org/10.1016/j.cose.2023.103651.
Yang, Z., Liu, X., Li, T., Wu, D., Wang, J., Zhao, Y., Han, H., 2022. A systematic literature review of methods and datasets for anomaly-based network intrusion detection. Comput. Secur. 116, 102675. https://doi.org/10.1016/j.cose.2022.102675.
Zhang, P., Sun, Z., Kyung, S., Behrens, H.W., Basque, Z.L., Cho, H., Oest, A., Wang, R., Bao, T., Shoshitaishvili, Y., Ahn, G.-J., Doupé, A., 2022. I'm SPARTACUS, No, I'm SPARTACUS: proactively protecting users from phishing by intentionally triggering cloaking behavior. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, pp. 3165–3179. https://doi.org/10.1145/3548606.3559334.
Zhong, M., Lin, M., Zhang, C., Xu, Z., 2024. A survey on graph neural networks for intrusion detection systems: methods, trends and challenges. Comput. Secur. 141, 103821. https://doi.org/10.1016/j.cose.2024.103821.

Mr Kieran Rendall is a PhD student in Computer Science at the University of Hertfordshire and a member of the Cybersecurity and Computing Systems research group. He holds an BSc in Cyber Security Management from Bournemouth University, UK. He has 5 years working in security and data science roles. His research interests include intrusion detection attack stage attribution.

Dr Alexios Mylonas is with University of Hertfordshire, where he leads the Cybersecurity and Computing Systems research group. He holds a PhD in Information and Communication Security and a BSc (Hons) in Computer Science from Athens University of Economics and Business, as well as an MSc in Information Security from Royal Holloway, University of London. His research interests focus on IoT security, incident response, web security and fraud detection. He has published >40 papers in esteemed scientific venues and his work is well cited (>2400 citations, h-index: 24).

Dr Stilianos Vidalis is the Deputy Head of the Dept. of Computer Science at the University of Hertfordshire and a member of the Cybersecurity and Computing Systems research group. He leads the development of an innovative technology that automates threat assessments. He acted as an instructor for the British Armed Forces Intelligence Personnel on penetration testing and digital forensics and as a cyberexpert for the Welsh Government and for the Wales University Officers' Training Corps. His research interests are in digital forensics and intrusion detection.

Dr Dimitris Gritzalis is a professor of cybersecurity with the Dept. of Informatics, Athens University of Economics & Business, where he also serves as Director for the M.*Sc*. Programme in Information Systems Security. He holds a B.*Sc*. (Mathematics, Univ. of Patras), a M.*Sc*. (Computer Science, City University of New York), and a Ph.D. (Information Systems Security, Univ. of the Aegean). He served as Associate Rector for Research, President for the Greek Computer Society and Associate Data Protection Commissioner of Greece. His research interests include risk assessment, security architectures, and malware. He is the Academic Editor of Computers & Security and the Scientific Editor of the International Journal of Critical Infrastructure Protection.