A benchmark for one-stage and multi-stage HTTP intrusion detection systems

Fin Vermehr
fin.vermehr@mail.utoronto.ca
Spherical Defense Ltd
London, UK

Jack Hopkins
jack@sphericaldefense.com
Spherical Defense Ltd
London, UK

Akbir Khan
akbir@sphericaldefense.com
Spherical Defense Ltd
London, UK

Abstract

In recent years, the complexity of web-based application traffic has increased dramatically, introducing new vulnerabilities and motivating the development of new security solutions. Despite this, few resources exist to evaluate the performance of web security solutions on complex application traffic. In this work, we present a dataset for benchmarking application-layer intrusion detection systems. The labelled dataset contains 892,833 sanitized HTTP requests derived from the logs of a popular open source application - "Domination", and addresses several of the issues of previous web benchmarks: a lack of complex payloads, unrealistically balanced training sets, a lack of multi-stage attacks, and a lack of bot traffic. In addition to providing this dataset, we also introduce two tasks: namely, a one-stage attack detection task, and a multi-stage attack detection task. We present experimental results for several common application security approaches on these tasks. On the single-stage task, the linear SVM classifier trained on augmented TF-IDF embeddings had the highest performance, with an F1 score of 0.921. On the multi-stage task, the Random Forest classifier trained on the final hidden state of an TF-IDF + LSTM model had the highest performance, with an F1 score of 0.721.

To our knowledge, this is the first work to introduce a broad security benchmark developed from modern application HTTP traffic.

CCS Concepts: • **Security and privacy → Web application security**; *Artificial immune systems*; Web protocol security.

Keywords: datasets, neural networks, API security, intrusion detection

AISec '20, November 13–15, 2020, Orlando, Fl
© 2020 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/1122445.1122456

ACM Reference Format:
Fin Vermehr, Jack Hopkins, and Akbir Khan. 2020. A benchmark for one-stage and multi-stage HTTP intrusion detection systems. In *Proceedings of 13th ACM Workshop on Artificial Intelligence and Security (AISec '20).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/1122445.1122456

1 Introduction

Web applications are programs that run within a web server, accessed either by a human user using a web browser, or by another program interacting directly via an Application Programming Interface.

Communicating with web applications is usually handled with the HyperText Transfer Protocol (HTTP), a stateless application-level protocol for distributed, collaborative, hypertext information systems that predominates information exchange over the World Wide Web. HTTP requests can ask for information, or influence the state of the application they are sent to.

For this reason, web applications are at risk from attacks that are conveyed in specially crafted HTTP requests. These requests aim to elicit malicious or unwanted behaviour, and can cause data loss, data leaks, service downtime or degradation. When these attacks happen, detecting them early is imperative in order to limit damage, prosecute the attacker, deter future attackers and prevent future attacks from occurring [2]. This is known as incident response.

Application-layer attacks can be one-stage, or multi-stage. A one-stage web attack exploits vulnerabilities or conveys illegal information in a single request, usually at any point in the application life-cycle.

Conversely, a multi-stage attack unfolds over a sequence of requests to a server . In many cases, the application vulnerability is only exploited in the final request, with each antecedent request conditioning the application server into a vulnerable state. Each constituent request may be perfectly legitimate in the right context, but when brought together in the wrong order or with the wrong timing they may indicate an attack. E.g replay attacks, bot traffic and authentication bypasses.

Validating contextual behaviour in web-applications remains an unsolved problem, due to the complexity and the difficulty in explicitly defining assumptions and contexts in order to apply formal methods [5].

In this work, we contribute a benchmark for the evaluation of both HTTP misuse and anomaly detection systems, on a wide variety of one-stage and multi-stage attack classes. This dataset has been designed to benchmark machine learning and rule-based models on the task of API and web threat detection, and was derived from the complete HTTP request logs of a popular open source application with roughly 20,000 active users. There are several notable features of this application that make it interesting for use as a benchmark: a chat function, authentication functionality, authorisation functionality, complex JSON payloads directing the multi-turn game-play and inter-user dynamics.

This benchmark is subdivided into 2 tasks: a One-stage Task (1), and a Multi-stage Task (2).

The One-stage task consists of a training set and a test set. The training set consists of 852,369 normal requests, and 20,430 requests containing one-stage attacks. The test set consists of 6,500 normal requests and 6,500 requests containing attacks. The Multi-stage task consists of a training set of 11,175 normal sessions and 11,175 sessions conveying multi-stage attacks. The malicious training and test sets of this benchmark are additionally labelled with the type of attack, to enable bench-marking multi-class classifiers.

This benchmark has been introduced to address the following limitations of previous HTTP benchmarks :

Balanced training data. In production settings, the overwhelming majority of application traffic is normal. Previous benchmarks either only provide exclusively normal traffic for training unsupervised models [13], or provide a balanced classes when training supervised models[11]. A more realistic scenario when training unsupervised models is to include some attacks in the data to evaluate robustness. Conversely, when training supervised models, a data imbalance skewed towards normal traffic is more realistic. In our test sets, we ensure that the classes are balanced however.

Simple payloads. Previous web security benchmarks only include browser-driven web traffic, and omit common features of API-driven traffic common in production settings, such as complex JSON and XML payloads in the bodies of POST requests. These payloads are common in every non-trivial modern application, and so are included here.

One-stage attacks only. Multi-stage attacks on the application-layer have grown more frequent as web-application logic has grown more sophisticated. These attacks are underrepresented in the literature because they are difficult to explicitly encode as signatures or rules in misuse and anomaly detection systems.

2 Related Work

In this section, we will first describe existing datasets for benchmarking web intrusion detection approaches. We will then discuss the techniques used for intrusion detection on HTTP traffic, distinguishing between misuse detection systems and anomaly detection systems.

2.1 Benchmarks

In the last decade, at least twelve datasets have been released for the evaluation of network-level intrusion detection and prevention systems on web servers [37]. These include compromised URLs [1], VPN sessions [10], Botnets [4, 18], Slow Loris attacks [41], TOR [22] and Denial-of-Service [17, 38]; specific environments such as Android; and general purpose intrusion detection datasets [28, 30, 37, 42]. These benchmarks rely on captured network flow data, and does not provide visibility into their constituent HTTP requests, because the application payload is distributed across multiple TCP/IP packets that obfuscates their content. Furthermore, when encryption technologies such as SSL are used, the payloads of these packets are difficult to interpret.

Application-level intrusion detection systems require specific benchmarks, of which there are two examples, the CSIC2010 WAF Benchmark [13] and ECML-PKDD 2007[11].

CSIC2010 is a synthetic dataset of HTTP requests representing inbound traffic to an online store. This dataset is designed for training unsupervised systems, and includes a test set with examples of many common attacks. The payloads of each HTTP object in this dataset are simple, and omit much of the complexity commonly found in modern applications. For example, no nested data-types are included in any POST body in the dataset.

The ECML-PKDD 2007 challenge set is a benchmark published for detecting HTTP attacks, and the intervals in between. This benchmark includes labels for the types of attack, and has therefore been used to train supervised intrusion detection systems [11, 32].

2.2 Intrusion Detection Systems

Intrusion Detection Systems (IDSs) monitor a computer system or network for policy violations and malicious activity [8]. Such systems generally fall into two classes: misuse detection and anomaly detection.

Misuse detection solutions aim to detect instances of malicious attacks by comparing inbound activity against a model of possible threats. These models may be represented by an array of patterns (such as regular expressions) that are developed by human experts, such as the popular IDS tools Snort [6] and ModSecurity [36]. There has been some work to develop static models for the detection of specific types of attack such as XSS [44] and SQL [12, 35]. More recently, misuse detection systems have increasingly relied on supervised machine learning approaches, in which the threat model is learned implicitly through exposure to labeled attacks [25]. These threat identification models are often called signature models [27], and while they are effective at detecting known threats, they generalise poorly to unseen threats. An example of a misuse detection pipeline can be seen in Figure 2.

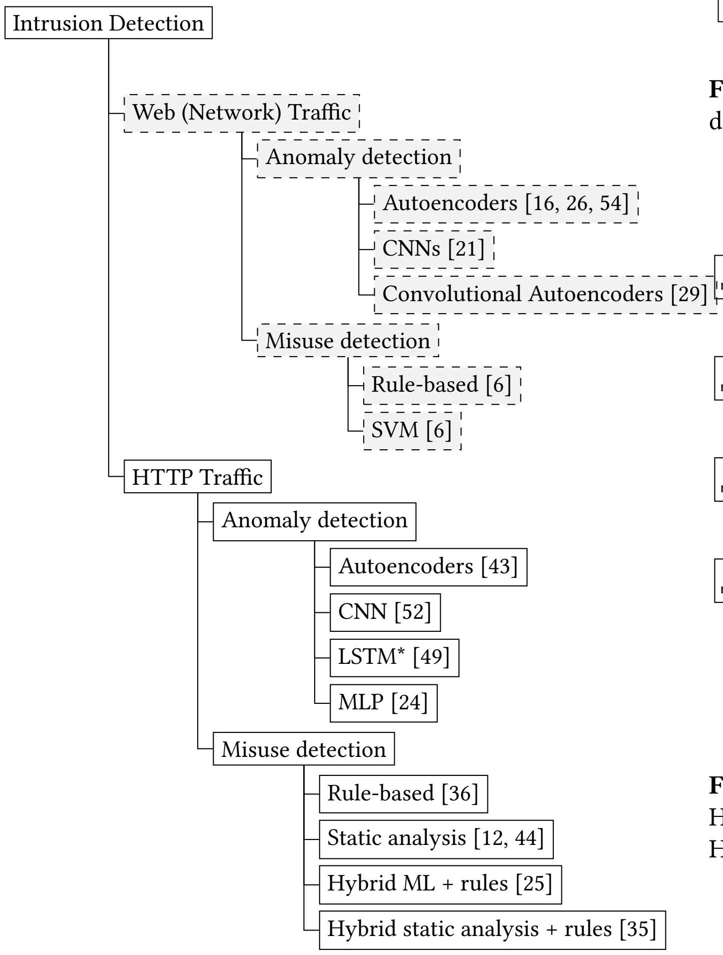

Figure 1. Taxonomy of web intrusion detection systems. Systems to which our benchmark *does not* apply are dashed.

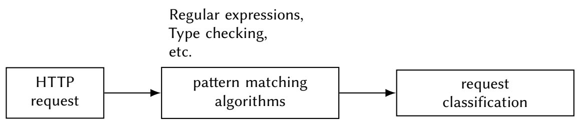

Figure 2. Pipeline for performing HTTP request misuse detection.

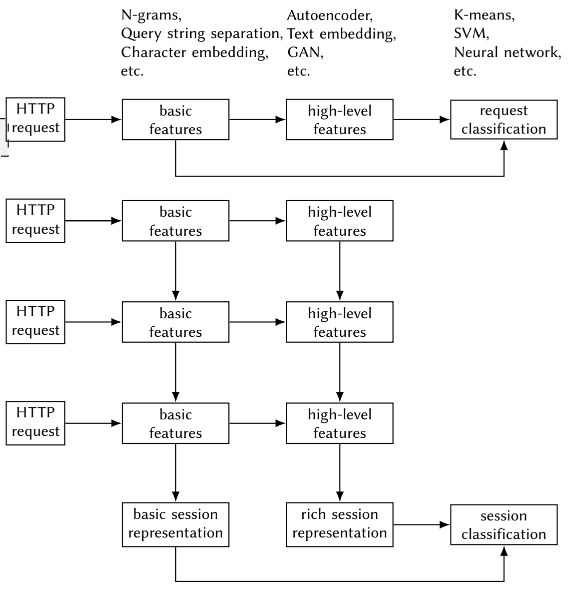

Figure 3. From top to bottom: (1) Pipeline for performing HTTP request anomaly detection. (2) Pipeline for performing HTTP session anomaly detection.

Anomaly detection solutions aim to detect instances of malicious web attacks by comparing inbound activity against a model of normal application requests. Historically, this has been performed using type validation, in which individual inputs are compared against an explicit defined model of their expected types [20]. These can be basic types like 'integer', or as complex as highly formatted strings like 'credit card number'.

Web-layer network traffic models have grown more sophisticated over time, employing machine learning approaches that offer improved performance compared to expert systems [19, 51]. More recently, deep learning techniques have been applied to extract high-level representations from network traffic for intrusion detection. An example of a machine learning anomaly detection pipeline can be seen in Figure **??**

Auto-encoders are one such technique, and create rich representations from input data by introducing an information bottleneck (which also mitigates overfitting). These architectures have achieved strong performance in network attack detection. [16] combines a sparse autoencoder with a Support Vector Machine (SVM) classifier. Other types of autoencoder have also been shown to work including contrastive, convolutional and denoising variants [29]. [26] developed an ensemble of autoencoders each trained on a correlated cluster of features extracted from the network traffic. [54] combined a gaussian-mixture model with an autoencoder for network anomaly detection. [43] detected web network attacks with a stacked autoencoder fed by N-gram representations with an Isolation Forest[23] one-class classifier. Convolutional Neural Networks (CNN)[21, 52] and Generative Adversarial Networks (GAN)[50] have been used for feature extraction from network traces.

In these examples, normal network behaviour is represented implicitly in the weights of the models. Deep learning models have also been used to explicitly learn ontologies describing normal network behaviour [53]. In addition to work on network traffic flows, there has also been some research on performing anomaly detection on HTTP objects directly.

Classical approaches extract N-grams from the HTTP objects, and pass them for downstream classification using hybrid SVMs [9] or Discrete Finite Automaton (DFA) induction [15]. Other work has focused on developing richer representations with techniques such as Byte-Pair Encoding (BPE) into Term Frequency-Inverse Document Frequency (TF-IDF) representations [51]; and Self-Organizing Maps [3].

More recently, attention-based LSTMs have been applied either to classify anomalies directly [49], or to first detect anomalous regions, and then classify them [24]. These newer approaches hold the SOTA accuracy on the CSIC2010 WAF benchmark.

Finally, there has also been some work in detecting certain application-level multi-stage attacks, such as Command and Control botnet sessions using SVMs on packet metadata [48], but such work has received less attention than network-level multi-stage attack detection [31, 39, 40]. An example of a machine learning multi-stage attack detection pipeline can be seen in Figure 4.

3 Dataset

In this section we will introduce the "Domination" web application, the dataset that we derived from it, and the two tasks that we created to benchmark HTTP intrusion detection systems. The dataset can be downloaded at the link below [1].

3.1 Application

The dataset presented here was extracted from the HTTP logs of a popular open-source [2] desktop and Android application with 20,000 users, receiving a mean of 500,000 HTTP requests per day. The application is a 'Risk'-like game [46], in which multiple players take turns placing armies in one of a number of territories, before sending armies to capture the territories of other players. Results are determined by a dice roll, with the advantage going to the defender. Each player authenticates either using an OAUTH [14] provider or through creating an account using the service. Following authentication, they enter a lobby where they can either select a public game to join, or create their own. There is a public chat function in which messages are publicly conveyed in individual HTTP requests. After joining a game, a number of moves are played by all participants. Eventually, the game is either won or abandoned, and the participants reenter the lobby or exit.

Instances of Personally Identifiable Information have been removed and replaced with public and synthetic examples that share a similar morphology. For example, private user identifiers were replaced with uniformly sampled public user identifiers from a popular online forum. Names were replaced by those sampled uniformly from a list of common English baby names. Email addresses were automatically generated, and UUID and authentication tokens are replaced.

[1]https://gofile.io/d/G9oYHl

[2]https://sourceforge.net/projects/domination/

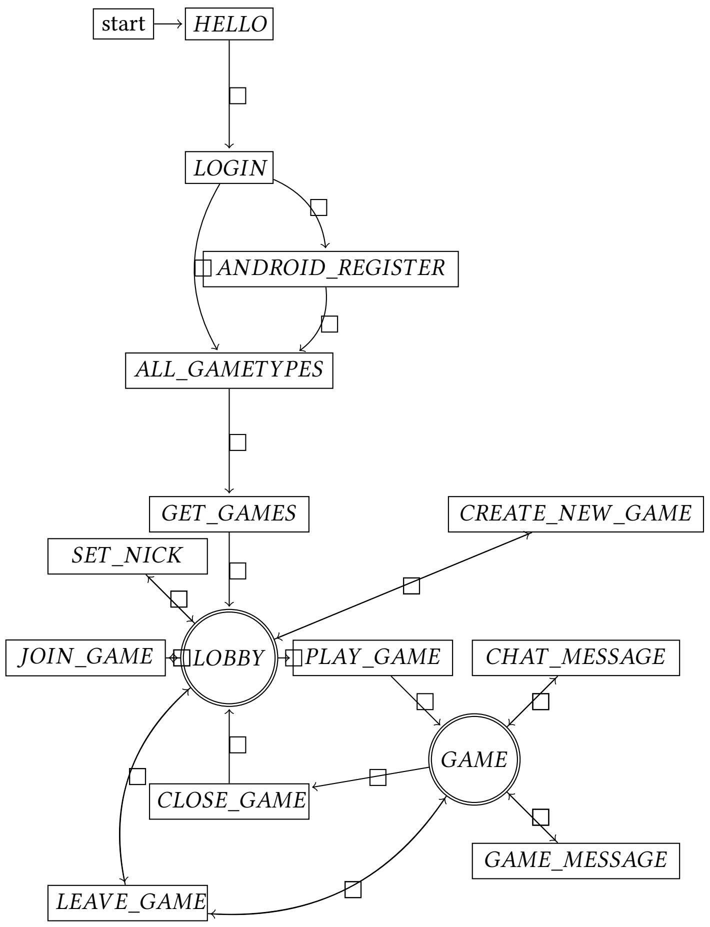

Figure 4. State diagram of the application

Public chat messages were left uncensored and unaltered.

User sessions were re-balanced to ensure that the length of games are kept short. Each session may contain multiple games, and each game has an upper limit of moves that it can contain. This upper limit is sampled from a normal distribution with $\mu = 25$ and $\sigma = 5$.

Each HTTP request was parsed into a JSON object before being added to the dataset (see Figure 5).

3.2 Single-stage Task

This task was designed to supersede earlier benchmarks such as CSIC2010 and ECML/PKDD in the task of evaluating HTTP intrusion detection systems. Recent deep learning architectures have demonstrated almost perfect accuracy on these benchmarks, necessitating a more challenging dataset. This task contains a wide range of single-stage attacks and illegal requests, and is significantly more complex than those of earlier contributions due to the inclusion of sophisticated HTTP payloads, such as a natural language chat field. The

Prefix	Endpoint	Description	Frequency
REQUEST	GAME_MESSAGE	Command to change game state.	268090
REQUEST	PLAY_GAME	Starts a new game.	15111
REQUEST	CLOSE_GAME	Closes the application.	13564
REQUEST	LOGIN	Authenticates user.	11527
REQUEST	HELLO	Establishes a chat session.	11496
REQUEST	ALL_GAMETYPES	Gets all game settings.	10840
REQUEST	GET_GAMES	Gets all public games.	10687
REQUEST	LEAVE_GAME	Leaves a game.	6214
REQUEST	JOIN_GAME	Joins an existing game.	5910
REQUEST	CREATE_NEW_GAME	Configure a new game.	3644
COMMAND	CHAT_MESSAGE	Send a public chat message.	2320
REQUEST	SET_NICK	Set nickname of user.	512
REQUEST	ANDROID_REGISTER	Register device to server.	367

data is labelled, making it suitable for evaluating supervised as well as unsupervised systems.

This task includes 4 files:

- Training normal (852,369 HTTP requests)
- Training anomalous (20,430 HTTP requests)
- Test normal (6,500 HTTP requests)
- Test anomalous (6,500 HTTP requests)

In both anomalous data-sets, each attack is annotated with its class. Included types of single-stage attacks are listed as follows:

3.2.1 Static attacks.
Requests for hidden or non-existent resources that are unintentionally left accessible by the web server.

1. Directory traversals (2190 train, 710 test)
2. OS directory index (23 train, 13 test)

3.2.2 Dynamic attacks.
Requests that aim to exploit vulnerabilities in the application implementation.

1. SQL injections (2830 train, 855 test)
2. XSS injections (2327 train, 869 test)
3. LDAP injections (142 train, 55 test)
4. NoSql injections (65 train, 23 test)
5. Command injection (4409 train, 1798 test)
6. Server-side includes (319 train, 114 test)
7. Local file includes (3831 train, 1265 test)
8. XML injections (2872 train, 91 test)

3.2.3 Unauthorized access attempts.
Requests that attempt to access disallowed resources in the application or web server (79 train, 49 test).

3.2.4 Policy violations.
Requests that contain content that violates the terms of service, including spam messages and phishing attempts (2130 train, 386 test).

3.2.5 Unintentional illegal requests.
Requests that contain illegal data types.

1. Malformed POST payloads (367 train, 127 test)
2. Unicode injection (396 train, 132 test)

Both test sets are balanced with respect to the API endpoint. In addition, the anomalous test set is also balanced with respect to the aforementioned attack classes.

```
{
  "method": "POST",
  "headers": {
    "Accept": "text/html, image/jpeg, *; q=.2, */*; q=.2",
    ...
    "timestamp": "684305"
  },
  "path": "/lobby/COMMAND_CHAT_MESSAGE",
  "query": null,
  "body": {
    "room_id": 1905922,
    "message": "hi. who is not ai?",
    "profile_picture": "source/data/profile_pictures/hecatonk.png"
  }
}
```

```
{
  "method": "POST",
  "headers": {
    "Accept": "text/html, image/jpeg, *; q=.2, */*; q=.2",
    ...
    "timestamp": "191142"
  },
  "path": "/lobby/REQUEST_GAME_MESSAGE",
  "query": "N \",NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL
  )%20waitfor%20delay%20'0:0:20'%20/* one",
  "body": {
    "game_id": 1908314,
    "message": "placearmies 2 1"
  }
}
```

Figure 5. From top to bottom: (1) Example of a normal HTTP request, taken from this dataset. (2) Example of a malicious HTTP request containing an SQL injection attack, taken from this dataset. Headers have been omitted.

3.3 Multi-stage Task

This task was designed as a challenge set for HTTP intrusion detection systems, and exclusively contains multi-stage attack and misuse examples. To our knowledge, this is the first dataset that describes the interaction between users and a web application in terms of HTTP requests.

This dataset contains examples of anomalous and malicious user HTTP sessions. These sessions were collated by sampling all requests that have the same user id, and concatenating them into arrays in order of their internal timestamp. An anomalous session may consist of individually normal requests in an anomalous or illegal combination. This task therefore requires a deeper understanding of the context a request occurs in. It requires the classification algorithm to identify discrepancies between subsequent requests, and the legal request transitions within the Domination application.

This task includes 4 files:

- Training normal (11175 HTTP sessions)
- Training anomalous (11175 number of HTTP sessions)
- Test normal (4840 HTTP sessions)
- Test anomalous (4840 number of HTTP sessions)

In both anomalous datasets, each attack instance is labelled. No instance of attack included in the training set is represented in the test set. Included types of multi-stage attacks are listed as follows:

3.3.1 Bots. Automated sessions designed to simulate user behaviour. A bot session consists of a set of a pre-defined, non-random paths that contains a chat message. This chat message will contain spam. Each request in the session will have a similar timestamp delta (475 train, 256 test).

3.3.2 Replays. Sessions containing actions which are maliciously repeated, such as repeating a move in an attempt to get an advantage over an opponent (2675 train, 1146 test).

3.3.3 Broken Authorization. Sessions containing attempts to take unauthorized actions.

1. Unauthenticated user taking a move (1337 train, 573 test)
2. Authenticated user switching session ID to take other moves (1338 train, 573 test)

3.3.4 Illegal Sessions. Sessions that contain requests that break intended state.

1. Switching game IDs (1338 train, 573 test)
2. Illegal state transitions, e.g leaving a game that was never joined. (1337 train, 573 test)

4 Evaluation

In this section, we will evaluate a number of approaches to HTTP misuse and anomaly detection, evaluating against existing datasets for comparison. For brevity, we present the details for re-implementation and access to our code [3].

We report both the F1 score and the Matthews correlation coefficient (MCC)[7]. The F1-score is the harmonic mean of the precision and recall of a classifier. However, given the imbalanced training sets, we also report the MCC, a more comprehensive metric which accounts not only for performance on positive samples but also negative samples.

[3]Link redacted in draft

	Train		Test	
	Normal	Anomalous	Normal	Anomalous
CSIC2010	36,000	0	36,000	25,000
EMC-PKDD2007	40,000	0	0	10,000
Single-Stage Task	852,369	20,430	6500	6500
Multi-Stage Task	11175	11175	4840	4840

Table 1. The dataset composition of both single-stage and multi-stage tasks compared to prior work.

4.1 Single-Stage Task

We evaluate a number of self-taught learning models [16], where a primary representation model learns an informative vector representation of a request, and a secondary classification model learns a decision boundary over the representation space to distinguish benign or malicious requests. Self-taught vector representations were learned from the following models:

TF-IDF Using the 1-gram, 2-gram and 3-gram tokenization of a request we create 3 sequences of terms which we are then used to calculate term frequency–inverse document frequency (tf-idf) for each document, resulting in 2271 total features. We then use feature selection to reduce each representation to a tractable 114 features. This was implemented using the Sci-kit library [34].

Bi-LSTM We train a bi-directional Long Short Term Memory (LSTM) network for next token prediction within a sequence, akin to an auto-regressive language model task. We uniformly sample 13,000 requests for our training dataset. Tokenization is learned via the byte-pair encoding algorithm, with a vocabulary of 2000 tokens (excluding special tokens) and a max sequence length of 500 [4]. Tokens are placed through a linear embedding layer with an output size of 50. This is then fed to both forward and backward LSTMs with a hidden size of 50. The output of the LSTM is passed through a linear layer with an output size of 2000 and a finally a softmax layer. The cross entropy loss is minimised. We train with an Adam optimiser with a learning rate of 10^3. We train this model for a single epoch with a batch size of 64. Finally, we construct a learned representation by concatenating the final hidden state of both forward and backward LSTMs.

Seq2Seq We train an Autoencoder network, with encoder and decoder as LSTM networks (hidden sizes 50 and 25 respectively). We take the hidden and cell states from the final step of the encoder and place these

[4]Implemented using the HuggingFace Transformer library [47]

into the decoder. For speed of training, teacher forced learning was also implemented. We use the aforementioned tokenization, batch size and optimiser from the BI-LSTM. After the encoder step, the final hidden state is given to the decoder network. Both neural networks were implemented with the Pytorch library [33].

We use a number of classical algorithms to learn decision boundaries over the representation space. We evaluate both unsupervised and supervised algorithms, including Support Vector Machines (SVMs), Random and Isolation Forests, Gradient Boosting and Naive Bayes. We implement all algorithms using the Sci-Kit Learn library and train until convergence.

In the unsupervised problem we evaluate one-class support vector machines (SVMs) using a Radial-Basis Function (RBF) kernel, and for Isolation and Random Forests we use 100 estimators. For Naive Bayes we apply a Gaussian prior.

4.2 Multi-stage Task

For the session level analysis task, we used the richest embeddings from the previous task, i.e TF-IDF. We performed two sets of experiments for session classification.

In the first experiment set, we simply sum the embedded requests together into a single vector for use as the feature vector of the session. Then using a two-class SVM, one-class SVM, Random Forest, Isolation Forest we classify the session as either normal or anomalous.

In the second experiment set, we train a unidirectional auto-regressive Long Short Term Memory (LSTM) network on the request embeddings, using the set of losses over a session as a feature vector. We apply Linear-SVM, RBF-SVM and Random Forest classifiers to benchmark the supervised problem, and an Isolation Forest and one-class SVM for the unsupervised problem.

4.3 Results

We present results within Table 2 and Table 3. For the single-stage task, we find that classifiers tend to perform better on CSIC2010 than our dataset. This suggests the quality of representations learnt on the Domination dataset may be insufficient for separation. To support this claim we provide visualizations of the training data for each representation learning model in Figure 6, demonstrating a clear overlap in attack and normal data. In general this reinforces the difficulties in creating informative representations for downstream tasks. These visualisations were created using principal component analysis (PCA) to reduce the training datasets to 2 dimensions.

The aforementioned problem of weak representations is exacerbated in the case of Neural Networks (Bi-LSTM and Seq2Seq). Referring back to Figure 6 we notice that while TF-IDF anomalous requests are in a single cluster within a homogeneous space of normal requests - we find the inverse is true for the Neural Networks. Neither case should make a difference for the supervised classifiers (there should be a symmetry given the balanced dataset), this highlights the marginal improvement the Neural Network representations have in the unsupervised case. Finally in the case of the supervised classifiers, we note that the area of overlap between normal to anomalous requests is far greater in the case of the neural networks, potentially alluding to the weak performance.

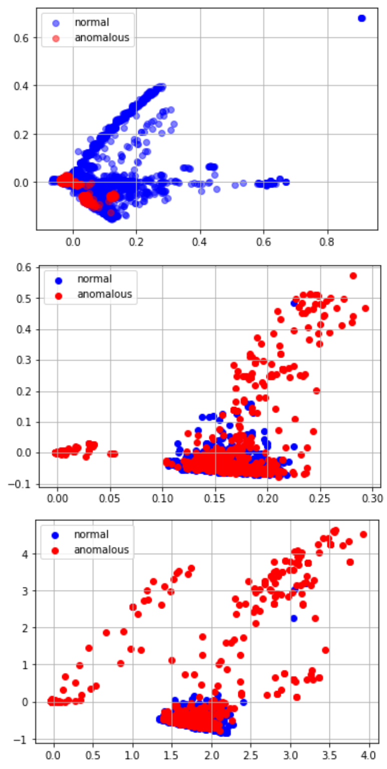

Figure 6. PCA plots of each representation learning algorithm. From top to bottom: (1) TF-IDF, (2) LSTM, (3) Seq2Seq

5 Conclusion

In this work, we presented a new dataset for benchmarking application-layer intrusion detection systems. This benchmark contains two tasks. The first task focuses on single-stage HTTP attacks, and is motivated by the near-perfect scores attained by recent HTTP anomaly detection systems on earlier benchmarks [5]. The second task is the first to include HTTP session data and multi-stage attacks, and was motivated by lack of resources for perfoming anomaly detection on multi-request user behaviour. We evaluated several types of machine learning architectures of both tasks. On the unsupervised Single-stage task, we report that the Bi-LSTM

[5]CSIC2010: 99.1% for Locate-Then-Detect. 99.7% for DeepHTTP.

	Unsupervised		Supervised				
F1 (MCC)	1-class SVM	Isolation forest	SVM (rbf)	SVM (linear)	Random forest	Gradient Boosting	Naive Bayes
Single-Stage Task							
TF-IDF BoW	Did not converge	0.660 (-0.086)	0.910 (0.835)	**0.921 (0.848)**	0.870 (0.767)	0.879 (0.780)	0.902 (0.799)
Bi- LSTM	0.593 (0.090)	**0.669 (0.195)**	0.027 (0.080)	0.040 (0.100)	0.159 (0.210)	0.289 (0.256)	0.155 (0.150)
Seq2Seq	0.146 (-0.082)	0.654 (0.112)	0.0131 (0.053)	0.000 (0.000)	0.092 (0.080)	0.152 (0.130)	0.334 (0.087)
CSIC2010							
TF-IDF BoW	Did not converge	0.735 (0.333)					
Bi- LSTM	0.599 (0.261)	0.624 (0.349)					
Seq2Seq	0.136 (-0.154)	**0.739 (0.219)**					

Table 2. Experimental results for the single-stage task, compared against the previous CSIC2010 benchmark.

	Unsupervised		Supervised		
F1 (MCC)	1-class SVM	Isolation forest	SVM (rbf)	SVM (linear)	Random forest
TF-IDF + BoW	0.503 (0.038)	0.664 (0.133)	0.490 (0.092)	0.374 (0.300)	**0.705 (0.500)**
TF-IDF + LSTM	0.689 (0.247)	0.681 (0.211)	0.606 (0.401)	0.678 (0.361)	**0.721 (0.526)**

Table 3. Experimental results for the multi-stage task.

with Isolation Forest had the highest F1 score of 0.669. On the supervised Single-stage task, we report that the TF-IDF Bag of Words with Linear SVM had the highest F1 score, at 0.921. On the unsupervised Multi-stage task, we report that the TF-IDF + LSTM with 1-class SVM had the highest F1 score, 0.689. On the supervised Multi-stage task, we report that the TF-IDF + LSTM with Random Forest had the highest F1 score, at 0.721.

We have ensured that the benchmarks herein contain a wide range of one-stage and multi-stage attack types. Despite this, it should be noted any security solution that performs well on these tasks should not be treated as a panacea. There are a number of security risks that are not represented in this benchmark, and should be addressed separately [45], including: security misconfigurations, improper assets management and insufficient logging.

For future work, we will perform additional experiments to determine the best approach for session-level anomaly detection.

Acknowledgments

To Yura, for her invaluable help in explaining the 'Domination' application and instructing us on how to sanitise it.

References

[1] 2016. ISCX-URL-2016. https://www.unb.ca/cic/datasets/url-2016.html

[2] Sara Althubiti, Xiaohong Yuan, and Albert Esterline. 2017. Analyzing HTTP requests for web intrusion detection. (2017).

[3] David Atienza, Álvaro Herrero, and Emilio Corchado. 2015. Neural analysis of http traffic for web attack detection. In *Computational Intelligence in Security for Information Systems Conference*. Springer, 201–212.

[4] Elaheh Biglar Beigi, Hossein Hadian Jazi, Natalia Stakhanova, and Ali A Ghorbani. 2014. Towards effective feature selection in machine learning-based botnet detection approaches. In *2014 IEEE Conference on Communications and Network Security*. IEEE, 247–255.

[5] Muffy Calder. 1998. What use are formal design and analysis methods to telecommunications services?. In *Feature Interactions in Telecommunications and Software Systems*, Vol. 5. IOS Press, 10–31.

[6] Brian Caswell and Jay Beale. 2004. *Snort 2.1 intrusion detection*. Elsevier.

[7] Davide Chicco and Giuseppe Jurman. 2020. The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC genomics* 21, 1 (2020), 6.

[8] Roberto Di Pietro and Luigi V Mancini. 2008. *Intrusion detection systems*. Vol. 38. Springer Science & Business Media.

[9] Ying Dong and Yuqing Zhang. 2017. Adaptively detecting malicious queries in web attacks. *arXiv preprint arXiv:1701.07774* (2017).

[10] Gerard Draper-Gil, Arash Habibi Lashkari, Mohammad Saiful Islam Mamun, and Ali A Ghorbani. 2016. Characterization of encrypted and vpn traffic using time-related. In *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*. 407–414.

[11] Matthieu Exbrayat. 2007. ECML/PKDD challenge: analyzing web traffic a boundaries signature approach. *ECML/PKDD'2007 DISCOVERY CHALLENGE* (2007), 53.

[12] Xiang Fu, Xin Lu, Boris Peltsverger, Shijun Chen, Kai Qian, and Lixin Tao. 2007. A static analysis framework for detecting SQL injection vulnerabilities. In *31st Annual International Computer Software and Applications Conference (COMPSAC 2007)*, Vol. 1. IEEE, 87–96.

[13] Carmen Torrano Giménez, Alejandro Pérez Villegas, and Gonzalo Álvarez Marañón. 2010. HTTP data set CSIC 2010. *Information Security Institute of CSIC (Spanish Research National Council)* (2010).

[14] Dick Hardt et al. 2012. *The OAuth 2.0 authorization framework*. Technical Report. RFC 6749, October.

[15] Kenneth Ingham III. 2007. Anomaly detection for HTTP intrusion detection: algorithm comparisons and the effect of generalization on accuracy. (2007).

[16] Ahmad Javaid, Quamar Niyaz, Weiqing Sun, and Mansoor Alam. 2016. A deep learning approach for network intrusion detection system. In *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*. 21–26.

[17] Hossein Hadian Jazi, Hugo Gonzalez, Natalia Stakhanova, and Ali A Ghorbani. 2017. Detecting HTTP-based application layer DoS attacks on web servers in the presence of sampling. *Computer Networks* 121 (2017), 25–36.

[18] Nickolaos Koroniotis, Nour Moustafa, Elena Sitnikova, and Benjamin Turnbull. 2019. Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset. *Future Generation Computer Systems* 100 (2019), 779–796.

[19] Rafał Kozik, Michał Choraś, and Witold Hołubowicz. 2016. Evolutionary-based packets classification for anomaly detection in web layer. *Security and Communication Networks* 9, 15 (2016), 2901–2910.

[20] Christopher Kruegel and Giovanni Vigna. 2003. Anomaly detection of web-based attacks. In *Proceedings of the 10th ACM conference on Computer and communications security*. 251–261.

[21] Donghwoon Kwon, Kathiravan Natarajan, Sang C Suh, Hyunjoo Kim, and Jinoh Kim. 2018. An empirical study on network anomaly detection using convolutional neural networks. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 1595–1598.

[22] Arash Habibi Lashkari, Gerard Draper-Gil, Mohammad Saiful Islam Mamun, and Ali A Ghorbani. 2017. Characterization of Tor Traffic using Time based Features.. In *ICISSP*. 253–262.

[23] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 413–422.

[24] Tianlong Liu, Yu Qi, Liang Shi, and Jianan Yan. 2019. Locate-then-detect: real-time web attack detection via attention-based deep neural networks. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. AAAI Press, 4725–4731.

[25] Abdelhamid Makiou, Youcef Begriche, and Ahmed Serhrouchni. 2014. Improving Web Application Firewalls to detect advanced SQL injection attacks. In *2014 10th International Conference on Information Assurance and Security*. IEEE, 35–40.

[26] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. 2018. Kitsune: an ensemble of autoencoders for online network intrusion detection. *arXiv preprint arXiv:1802.09089* (2018).

[27] Peter Morley. 2001. Processing virus collections. *VIRUS* 129 (2001), 129–134.

[28] Nour Moustafa and Jill Slay. 2015. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In *2015 military communications and information systems conference (MilCIS)*. IEEE, 1–6.

[29] Sheraz Naseer, Yasir Saleem, Shehzad Khalid, Muhammad Khawar Bashir, Jihun Han, Muhammad Munwar Iqbal, and Kijun Han. 2018. Enhanced network anomaly detection based on deep neural networks. *IEEE Access* 6 (2018), 48231–48246.

[30] Sang Ni, Quan Qian, and Rui Zhang. 2018. Malware identification using visualization images and deep learning. *Computers & Security* 77 (2018), 871–885.

[31] Dirk Ourston, Sara Matzner, William Stump, and Bryan Hopkins. 2003. Applications of hidden markov models to detecting multi-stage network attacks. In *36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the*. IEEE, 10–pp.

[32] Konstantinos Pachopoulos, Dialekti Valsamou, Dimitrios Mavroeidis, and Michalis Vazirgiannis. 2007. Feature extraction from web traffic data for the application of data mining algorithms in attack identification. In *Proceedings of the ECML/PKDD*. 65–70.

[33] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. (2017).

[34] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research* 12 (2011), 2825–2830.

[35] Cristian Pinzón, Juan F De Paz, Javier Bajo, Álvaro Herrero, and Emilio Corchado. 2010. AIIDA-Sql: an adaptive intelligent intrusion detector agent for detecting sql injection attacks. In *2010 10th International Conference on Hybrid Intelligent Systems*. IEEE, 73–78.

[36] Ivan Ristic. 2005. *Apache security*. O'Reilly Media.

[37] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. 2018. Toward generating a new intrusion detection dataset and intrusion traffic characterization.. In *ICISSP*. 108–116.

[38] Iman Sharafaldin, Arash Habibi Lashkari, Saqib Hakak, and Ali A Ghorbani. 2019. Developing Realistic Distributed Denial of Service (DDoS) Attack Dataset and Taxonomy. In *2019 International Carnahan Conference on Security Technology (ICCST)*. IEEE, 1–8.

[39] Tawfeeq Shawly, Ali Elghariani, Jason Kobes, and Arif Ghafoor. 2019. Architectures for Detecting Interleaved Multi-stage Network Attacks Using Hidden Markov Models. *IEEE Transactions on Dependable and Secure Computing* (2019).

[40] Jinmyeong Shin, Seok-Hwan Choi, Peng Liu, and Yoon-Ho Choi. 2019. Unsupervised multi-stage attack detection framework without details on single-stage attacks. *Future Generation Computer Systems* 100 (2019), 811–825.

[41] External Data Source. 2017. 2017 SUEE Dataset. https://doi.org/10.23721/100/1504322

[42] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. 2009. A detailed analysis of the KDD CUP 99 data set. In *2009 IEEE symposium on computational intelligence for security and defense applications*. IEEE, 1–6.

[43] Ali Moradi Vartouni, Saeed Sedighian Kashi, and Mohammad Teshnehlab. 2018. An anomaly detection method to detect web attacks using Stacked Auto-Encoder. In *2018 6th Iranian Joint Congress on Fuzzy and Intelligent Systems (CFIS)*. IEEE, 131–134.

[44] Gary Wassermann and Zhendong Su. 2008. Static detection of cross-site scripting vulnerabilities. In *2008 ACM/IEEE 30th International Conference on Software Engineering*. IEEE, 171–180.

[45] Dave Wichers and Jeff Williams. 2017. Owasp top-10 2017. *OWASP Foundation* (2017).

[46] Wikipedia. 2020. Risk (game) — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Risk%20(game)&oldid=962027383. [Online; accessed 17-June-2020].

[47] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771* (2019).

[48] Kazumasa Yamauchi, Yoshiaki Hori, and Kouichi Sakurai. 2013. Detecting HTTP-based botnet based on characteristic of the C & C session using by SVM. In *2013 Eighth Asia Joint Conference on Information Security*. IEEE, 63–68.

[49] Yuqi Yu, Hanbing Yan, Hongchao Guan, and Hao Zhou. 2018. DeepHTTP: Semantics-Structure Model with Attention for Anomalous HTTP Traffic Detection and Pattern Mining. *arXiv preprint arXiv:1810.12751* (2018).

[50] Houssam Zenati, Chuan Sheng Foo, Bruno Lecouat, Gaurav Manek, and Vijay Ramaseshan Chandrasekhar. 2018. Efficient gan-based anomaly detection. *arXiv preprint arXiv:1802.06222* (2018).

[51] Junlang Zhan, Xuan Liao, Yukun Bao, Lu Gan, Zhiwen Tan, Mengxue Zhang, Ruan He, and Jialiang Lu. 2019. An effective feature representation of web log data by leveraging byte pair encoding and TF-IDF. In *Proceedings of the ACM Turing Celebration Conference-China*. 1–6.

[52] Ming Zhang, Boyi Xu, Shuai Bai, Shuaibing Lu, and Zhechao Lin. 2017. A deep learning method to detect web attacks using a specially designed cnn. In *International Conference on Neural Information Processing*. Springer, 828–836.

[53] Huangjie Zheng, Yuchen Wang, Chen Han, Fangjie Le, Ruan He, and Jialiang Lu. 2018. Learning and applying ontology for machine learning in cyber attack detection. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE, 1309–1315.

[54] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho, and Haifeng Chen. 2018. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. (2018).