Vietnam J Comput Sci
DOI 10.1007/s40595-017-0095-3

REGULAR PAPER

Computational intelligence anti-malware framework for android OS

Konstantinos Demertzis[1] · **Lazaros Iliadis**[1]

Received: 2 May 2016 / Accepted: 15 February 2017

Abstract It is a fact that more and more users are adopting the online digital payment systems via mobile devices for everyday use. This attracts powerful gangs of cybercriminals, which use sophisticated and highly intelligent types of malware to broaden their attacks. Malicious software is designed to run quietly and to remain unsolved for a long time. It manages to take full control of the device and to communicate (via the Tor network) with its Command & Control servers of fast-flux botnets' networks to which it belongs. This is done to achieve the malicious objectives of the botmasters. This paper proposes the development of the computational intelligence anti-malware framework (CIantiMF) which is innovative, ultra-fast and has low requirements. It runs under the android operating system (OS) and its reasoning is based on advanced computational intelligence approaches. The selection of the android OS was based on its popularity and on the number of critical applications available for it. The CIantiMF uses two advanced technology extensions for the ART java virtual machine which is the default in the recent versions of android. The first is the smart anti-malware extension, which can recognize whether the java classes of an android application are benign or malicious using an optimized multi-layer perceptron. The optimization is done by the employment of the biogeography-based optimizer algorithm. The second is the Tor online traffic identification extension, which is capable of achieving malware localization, Tor traffic identification and botnets prohibition, with the use of the online sequential extreme learning machine algorithm.

Keywords Android malware · Firmware malware · Mobile banking malware · Rootkits · Ransomware · Online sequential extreme learning machine · Tor traffic analysis · Botnets

1 Introduction

1.1 Android security model

One of the most important features that distinguish the android OS, is the adoption of user identifiers (UIDs) which imparts sophisticated security capabilities, compared to the modes of traditional OS. In particular, the android applications run as separate processes with different UIDs and different permissions each. In this way, there is no application capable to read/write data or code to another, whereas if it is necessary to make data exchange with another application, it requires the assignment of specific permission. It should be noted that the android uses the mandatory access control (MAC) model in all of the processes, even in those that run with root/superuser privileges [1]. Specifically, this model is based on a security label system, which is attributed to both "subjects" (e.g. applications, users) and "objects", which are the categories that manage information, related to different needs. This means that they can be assigned to sections of individual information within a system. Specific types of security clearance (classification labels) are assigned to the applications and to their corresponding data. In this way, the android is based on security clearances, on the classification data labels and on the system's security policies,

✉ Konstantinos Demertzis
kdemertz@fmenr.duth.gr

Lazaros Iliadis
liliadis@fmenr.duth.gr

[1] Lab of Forest-Environmental Informatics and Computational Intelligence, Democritus University of Thrace, 193 Pandazidou st., 68200 N.Orestiada, Greece

Published online: 28 February 2017

to reach a decision on which subject is entitled to access an object.

The classification data labels are assigned to each type of object (file, directory, device, network). Based on the security policies, the system checks the security clearances of a "subject" (e.g. user, application) by comparing them to the classification data labels of an "object" (e.g. data, files) to which access is sought. Access is not approved, if security policies are not met. The MAC model is called mandatory [1], because the classification of "subjects" and "objects" is performed automatically by the system, without the intervention of users who theoretically cannot change these rating levels.

1.2 Android rootkits

An android rootkit [2] is a set of executable scripts and configuration files, allowing the continuous access to the root/superuser privileges. It should be mentioned that they actively hide their presence from the system administrators. This is done by their incorporation into basic android OS files or other legitimate applications.

Thus, they enable secret maintenance of the system's control by executing commands or by stealing important data (e.g. credit card numbers, passwords, banking applications) totally unnoticed. Typically, an attacker installs an android rootkit by exploiting known security loopholes (zero-day exploit, unpatched), to obtain passwords (e.g. phishing, clickjacking), or to perform direct attack on encryption (brute-force attack, hijack attack) or through close-in attack (social engineering).

The action of the rootkit, starts after the installation of the android OS, which is equivalent to simultaneous acquisition root/superuser privileges and after the installation of the necessary "Payloads".

Then the rootkit is activated and redirects the system calls to completely conceal its presence. For example, when a system function accesses a DLL library, it is misled by the rootkit, which activates its own code to overtake the control of its files. The kernel level rootkits [2] (which are the most dangerous) have the following capabilities:

(a) To change the privileges of a process (privilege escalation).
(b) They can create or open doors against a security gap or program
(c) They can create a coded or encrypted communication channel with C & C servers (HTTPS, Tor)
(d) They can "load" drivers or collect and record information from the system in which they operate through key loggers or password sniffers (telephone number, country, IMEI, model, android OS version, list of installed apps).
(e) It is possible to perform unstructured supplementary service data (USSD) request [1], even to neutralize the defenses of the system by replacing legal with false and malicious applications (e.g. Rogue security software, fake antivirus) [3].

The malware response programs, if not themselves fake, they perform out a scan in a modified system, where changes cannot be traced, since rootkits distort the files so that every signature-based or difference-based control fails.

Thus, the user cannot revoke the full administrator rights from the malicious software, even if he uninstalls all applications that turned his phone a pawn of unknown forces, capable of an imperceptible files interception. It should be mentioned that there are cases which even require the full cancellation of the operating system.

The firmware malware [3], a special category of android rootkits, is extremely difficult to detect because the traditional virus scanners will not detect firmware threats.

Android rootkits ransomware encrypt data and then they demand money to unlock the victim's files. If the money is not paid within the period specified by the criminals, they threaten to hold the decryption key, which is kept only on the hacker's C & C server.

Finally, the android rootkits are mainly mobile banking malware, which have been developed with the objective of financial fraud. They are conducting illegal financial transactions and they steal money.

The memory dumps analysis method is the most serious approach of treating these threats. It performs a forced dump of the operating system's virtual memory to identify an active rootkit. However, this technique is highly specific, it requires access to private source code, it is time consuming and it requires specialized personnel with the respective tools (digital forensic investigation tools). Moreover, it does not have the ability to detect every type of threat, as a hypervisor rootkit is able to monitor and to overturn the lower level of the system in an attempt to read the memory [3].

1.3 Tor-based botnets and Tor traffic analysis

The objective of Tor [4] is to conceal the user IDs and their activity in the network to prevent the monitoring and analysis of the traffic and to separate the detection from the routing using virtual circuits, or overlays, which change periodically.

It is the implementation of onion routing [5], in which multiple layers of encryption are employed, to ensure perfect forward secrecy between the nodes and the hidden services of Tor, while launching randomly the communication via Tor nodes (consensus) operated by volunteers worldwide. Although the Tor network is operating in the Transport layer of the OSI, the onion proxy software shows customers the

secure socket interface (SOCKS) which operates in the session layer.

Also, a continuous redirection of traffic requests between the relays (entry guards, middle relays and exit relays), takes place in this network. Both the sender and recipient addresses and the information are in the form of encrypted text, so that no one at any point along the communication channel can decrypt the information or identify both ends directly [5]. The most famous types of malware are seeking communication recovery and its maintenance with the C & C remote servers on a regular basis, so that botmasters can collect or transfer information and upgrades to the compromised devices (bots). This communication is usually performed using hardcoded address or default lists address (pool addresses) controlled by the creator of the.

The mode of communication of the latest, sophisticated malware generations, lies in the creation of an encrypted communication channel, based on the chaotic architecture of Tor, to alter the traces and to distort the elements that define an attack and eventually to increase the complexity of the botnets.

Although modern programming techniques enable the malware creators to use thousands, alternating and different subnet IP address, to communicate with their C2 servers, the trace of those IPs is relatively straightforward for the network engineers, or for the responsible security analysts. Once identified, they are included in a blacklist and eventually they are blocked as spam. On the other hand, the limitation of the Tor-based botnets is extremely difficult because the movement of the Tor network resembles that of the HTTPS protocol.

1.4 Tor vs HTTPS

The Tor network not only performs encryption, but it is also designed to simulate normal HTTPS protocol traffic, which makes the identification of its channels an extremely complex and specialized process, even for experienced engineers or network analyzers. Specifically, the Tor network can use the TCP port 443, which is used by the HTTPS, so that the supervision and interpretation of a session exclusively with the determination of the door cannot constitute a reliable method.

A successful method for detecting Tor traffic is the statistical analysis and the identification of the secure sockets layer protocol differences (SSL) [6]. The SSL protocol uses a combination of public and symmetric key encryption. Each SSL connection always starts with the exchange of messages by the server and the client until the secure connection is established (handshake). The handshake allows the server to prove its identity to the client using public-key encryption techniques and then allows the client and the server to cooperate in the creation of a symmetric key to be used to quickly encrypt and decrypt data exchanged between them. Optionally, the handshake also allows the client to prove its identity to the server [6]. Given that each Tor client creates self-signed SSL, using a random domain name that changes around every 30 min, a statistical analysis of the network traffic based on the specific SSL characteristics can identify the Tor sessions, in a network full of HTTPS traffic.

2 Innovation of the proposed method

Android rootkits are the most sophisticated and highly intelligent malware techniques that make detection of "contamination" and analysis of malicious code, a very complex task. It is a fact that they spread through chaotic Tor-based botnets in which communication is done using the anonymity Tor network, which makes it impossible to identify and locate the C & C servers. In addition, the network traffic for the Tor packet is designed to simulate the respective traffic of the HTTPS protocol, which causes serious Tor traffic identification weaknesses by the motion analysis systems. Finally, given the passive mode of traditional android mobile security systems, which are unable in most cases to identify these types of major threats, the development and use of alternative more radical and more substantial methods appear as a necessity. This work proposes the development and testing of a novel computational intelligence system named CiantiMF. The system requires the minimum consumption of resources and it significantly enhances the security mechanisms of the android OS [7].

Specifically, the architecture of the proposed system is based on the hybrid use of two advanced ART JVM (ANDROID) extensions, namely the SAME and the OTTIE. The SAME uses a neural network, optimized with the BBO algorithm and it is capable of recognizing whether the java classes of an android application are benign or malicious. The OTTIE employs the OSELIM algorithm to perform malware localization, Tor traffic identification and botnets prohibition.

The CiantiMF system is a biologically inspired artificial intelligence computer security technique [8–12]. Unlike other existing approaches which are based on individual passive safety techniques, the CiantiMF is an integrated active safety system. It provides intelligent surveillance mechanisms and classification of malware, it is able to defend itself and to protect from Rootkits malware, it detects and prevents encrypted Tor network activities and it can efficiently exploit the potential of the hardware, with minimal computational cost.

A major innovation of the CiantiMF approach is related to the architecture of the proposed hybrid computational intelligence system, which combines for the first time two very fast and highly effective biologically inspired machine learning algorithms towards the solution of a multidimensional and complex IT security problem. Another novelty is the addi-

tion of a hybrid machine learning system as an extension to the ART JVM, under the android OS. This addition pours intelligence at compiler level, something that significantly enhances the defense mechanisms of the system, as well as controlling the outset dependencies of an application.

Furthermore, a major innovative feature of this proposal is related to the identification and separation of the Tor network traffic from the traffic of the HTTPS protocol, which is presented for the first time in static or dynamic network traffic analysis systems.

3 Related work

Several publications discuss android-specific security mechanisms, involving overall security assessment of the platform [13], malware detection [14], application permission analysis [15], and kernel hardening [16]. Significant work has been done in applying machine learning (ML) methods, using features derived from both static [17–19] and dynamic [20] analysis to identify malicious android applications [21], to network traffic classification [22], malware traffic analysis [23] and botnets localization [24]. In parallel, several other authors [25–27] have also summarized scientific effort of detecting the botnets while proposing novel taxonomies of detection techniques, introducing different classes of botnet detection and presenting some of the most prominent methods within the defined classes. Also, traffic analysis attacks have been extensively studied over the past decade [28,29]. The authors have acknowledged the potential of machine learning-based approaches in providing efficient and effective detection, but they have not provided a deeper insight into specific methods, neither the comparison of the approaches by detection performances nor evaluation practice.

On the other hand, Cheng et al. [30] proposed the use of ELM methods to classify binary and multi-class network traffic for intrusion detection with high accuracy. Hsu et al. [31] proposed a real-time system for detecting botnets based on anomalous delays in HTTP/HTTPS requests from a given client with very promising results. Also, Haffner et. al. [32] employed AdaBoost, hidden Markov, Naive Bayesian and maximum entropy models to classify network traffic into different applications, with very high secure shell (SSH, is a cryptographic network protocol operating at layer 7 of the OSI model to allow remote login and other network services to operate securely over an unsecured network) detection rate and very low false-positive rate, but they employed only few bytes of the payload. Furthermore, Alshammari et al. [33] employed repeated incremental pruning, to produce error reduction (RIPPER) and AdaBoost algorithms for classifying SSH traffic from offline log files without using any payload, IP addresses or port numbers. Holz et al. [34] proposed a passive method to locate botnets and Apvrille et al. [35] propose a heuristic engine that statically pre-processes and prioritizes samples to accelerate the detection of new android malware in the wild. Crowdroid [36] made a first step towards the use of dynamic analysis results for android malware detection by performing k means clustering based on system call invocation counts. Afonso et al. [37] dynamically analyze android apps to use the number of invocations of API and system calls as coarse-grained features to train various classifiers. Their monitoring approach relies on modifying the app under analysis, which is easily detectable by malware. Dini et al. [38] proposed a multi-level anomaly detector for android malware (MADAM) system to monitors android at the kernel level and user level to detect real malware infections using machine learning techniques to distinguish between standard behaviors and malicious ones. The Droid Dolphin [39] approach relies on repackaging an application with monitoring code. Chakravarty et al. [40] assess the feasibility and effectiveness of practical traffic analysis attacks against the Tor network using NetFlow data and proposed an active traffic analysis method based on deliberately perturbing the characteristics of user traffic at the server side, and observing a similar perturbation at the client side through statistical correlation. Almubayed et al. [41] proposed a research has considered many ML algorithms to fingerprint Tor usage in the network. Chaabane et al. [42] provides a deep analysis of both the HTTP and BitTorrent protocols giving a complete overview of their usage, depict how users behave on top of Tor and also show that Tor usage is now diverted from the onion routing concept and that Tor exit nodes are frequently used as 1-hop SOCKS proxies, through a so-called tunneling technique. Finally, Chakravarty et al. proposed methods for performing traffic analysis using remote network bandwidth estimation tools, to identify the Tor relays and routers involved in Tor circuits [43,44].

4 Architecture of the CIantiMF

The architecture of the CIantiMF requires the creation of two parallel extensions which act additionally and complementary to the function of the ART JVM. This injects artificial intelligence at android compiler level, significantly enhancing its active security. Specifically, SAME [45] analyzes the java classes before they load and run a java application (class loader). Introduction of the files in the ART JVM passes necessarily through the said extension in which it is checked whether the classes are benign or malicious. If they are found malicious, a decision is made, either automatically, if the accuracy of classification exceeds a desired threshold, or after an intervention of the system's operator for the rejection and non-installation of the application. If the control class is found benign, then the installation process continues

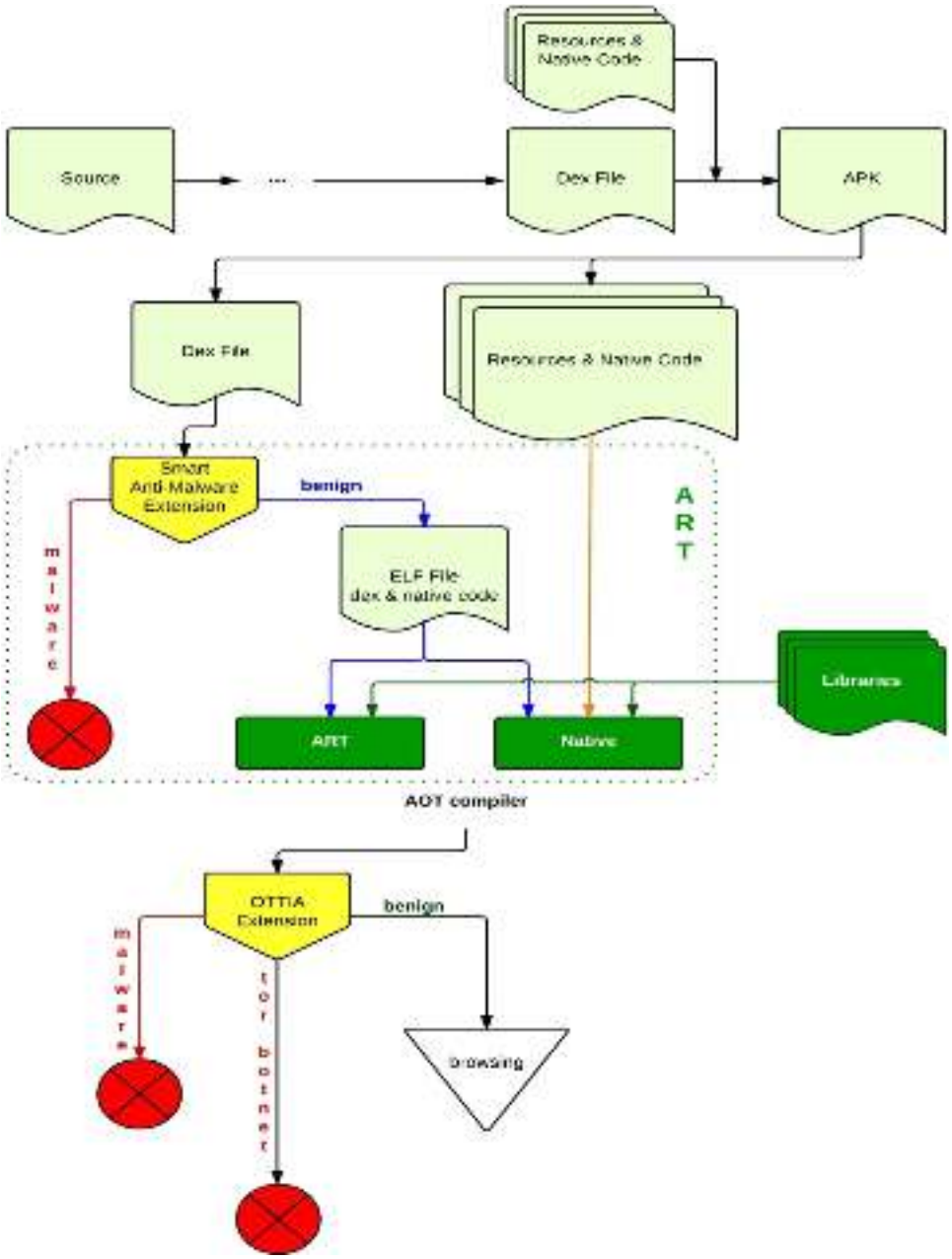

Fig. 1 The proposed architecture of the CIantiMF

normally without problems, while the user is informed that it is a safe application.

Then, when the application is executed, a control of the network traffic that is generated by the application is performed to determine whether it is related to malicious sources or not. A thorough analysis is also carried out to identify the potential encrypted traffic and accordingly if it is following the HTTPS protocol it is allowed, whereas if it is following the Tor protocol it is rejected by default as malicious.

The proposed architecture of the CIantiMF is presented in Fig. 1.

It should be emphasized that the above architectural shape operates based on the dynamic analysis of the android system's parameters, adapting the requirements of the running applications on the basis of stringent criteria and robust security policies.

This adaptation is the result of an automatic process, derived from computational intelligence technologies, thus overcoming the potential inability of users to take timely measures to protect themselves. Finally, it is important that these malware identification procedures require fewer steps than the processor to analyze an application, resulting in a better resources management and in less energy consumption.

4.1 Smart anti-malware extension (SAME)

In our previous work [45], we have proposed the SAME which introduces intelligence to the compiler and classifies malicious java classes in time to spot the android malwares. This is done by applying the java class file analysis (JCFA) approach and based on the effective BBO optimization algorithm, which is used to train a MLP.

Generally, the source code java files (.java) of a java application are compiled to byte code files (.class) which are platform independent and they can be executed by a JVM just like ART which is an ahead-of-time (AOT) compiler. The classes are organized in the .java files with each file containing at least one public class. The name of the file is identical to the name of the contained public class. The ART loads the classes required to execute the java program (class loader) and then it verifies the validity of the byte code files before execution (byte code verifier) [3]. The JCFA process includes also the analysis of the classes, methods and specific characteristics included in an application. The SAME, introduces advanced artificial intelligence (AI) methods, applied on specific parameters and data (obtained after the JCFA process) to perform binary classification of the classes comprising an application, in benign or malicious. More specifically the SAME system employs the biogeography-based optimizer to train a MLP which classifies the java classes of an application successfully in benign or malicious.

The architectural design of the SAME introduces an additional functional level inside the ARTJVM, which analyzes the java classes before their loading and before the execution of the java program (class loader). The introduction of the files in the ARTJVM, always passes from the above level, where the check for malicious classes is done. If malicious classes are detected, decisions are done depending on the accuracy of the classification. If the accuracy is high, then the decisions are done automatically, otherwise the actions are imposed by the user regarding the acceptance or rejection of the application installation. In the case that the classes are benign, the installation is performed normally and the user is notified that this is a secure application [45].

A basic innovation of the SAME is the inclusion of a machine learning approach as an extension of the ART JVM used by the android OS. This joint with the JCFA and the fact that the ART JVM resolves ahead-of-time all of the dependencies during the loading of classes, introduces Intelligence in compiler level. This fact enhances the defensive capabilities of the system significantly. It is important that the dependencies and the structural elements of an application are checked before its installation enabling the malware cases.

An another important innovative part of this research is related to the choice of the independent parameters, which was done after several exhaustive tests, to ensure the maximum performance and generalization of the algorithm and the consumption of the minimum resources.

Finally, it is worth mentioning that the BBO optimization algorithm (popular for engineering cases) is used for the first time to train an artificial neural network (ANN) for a real information security problem.

4.2 Online Tor traffic identification extension (OTTIE)

The TTIE is essentially a tool for analysis of web streaming traffic in fixed intervals, to extract timely conclusions in which some or all of the incoming data are not available for access from any permanent or temporary storage medium, but those arrive in a form of consecutive flows. For these data there is no control over the order in which they arrive, their size may vary and many of them offer no real information. Also the examination of individual IP packets or TCP segments can extract only a few conclusions and therefore the interdependence of the individual packets to each other, their analysis cannot be done with simple static methods, but it requires further modeling of traffic and the use of advanced analytical methods for the extraction of knowledge from complex data sets. This modeling in TTIE is achieved by the use of the computational intelligence online sequential extreme learning machine (OSELM) algorithm.

The extreme learning machine (ELM) as an emerging biologically inspired learning technique provides efficient unified solutions to "generalized" single-hidden layer feed forward networks (SLFNs) but the hidden layer (or called feature mapping) in ELM need not be tuned [46]. Such SLFNs include but are not limited to support vector machine, polynomial network, RBF networks, and the conventional feed forward neural networks. All the hidden node parameters are independent from the target functions or the training datasets and the output weights of ELMs may be determined in different ways (with or without iterations, with or without incremental implementations). ELM has several advantages, ease of use, faster learning speed, higher generalization performance, suitable for many nonlinear activation function and kernel functions.

According to the ELM theory [46], the ELM with Gaussian radial basis function kernel (GRBFK) $K(u, v) = \exp(-\gamma ||u - v||^2)$ is used in this approach. The hidden neurons are $k = 20$ that chosen with trial and error method. Subsequently, w_i are the assigned random input weights and b_i, $i = 1, \ldots, N$ are the biases. To calculate the hidden layer output matrix H, the Eq. (1) is used.

$$H = \begin{bmatrix} h(x_1) \\ \vdots \\ h(x_N) \end{bmatrix} = \begin{bmatrix} h_1(x_1) & \cdots & h_L(x_1) \\ \vdots & & \vdots \\ h_1(x_N) & \cdots & h_L(x_N) \end{bmatrix} \tag{1}$$

$h(x) = [h_1(x), ..., h_L(x)]$ is the output (row) vector of the hidden layer with respect to the input x. Also $h(x)$ actually maps the data from the d-dimensional input space to the L-dimensional hidden-layer feature space (ELM feature space) H and thus $h(x)$ is indeed a feature mapping. ELM is to minimize the training error as well as the norm of the output weights:

$$\text{Minimize} : ||H\beta - T||^2 \text{ and } ||\beta|| \tag{2}$$

where H is the hidden-layer output matrix of the equation (1), $||\beta||$ is used to minimize the norm of the output weights and actually to maximize the distance of the separating margins of the two different classes in the ELM feature space $2/||\beta||$.

To calculate the output weights β the function (3) is used:

$$\beta = \left(\frac{I}{C} + H^{\mathrm{T}}H\right)^{-1} H^{\mathrm{T}}T \tag{3}$$

where c is a positive constant is obtained and T resulting from the *function approximation of SLFNs with additive neurons*

$$T = \begin{bmatrix} t_1^{\mathrm{T}} \\ \vdots \\ t_N^{\mathrm{T}} \end{bmatrix} [46]$$

which is an arbitrary distinct sample with

$$t_i = [t_{i1}, t_{i2}, \ldots, t_{im}]^{\mathrm{T}} \in R^m [47]$$

The OSELM is an alternative technique for large-scale computing and machine learning approaches that used when data become available in a sequential order to determine a mapping from data set corresponding labels. The main difference between online learning and batch learning techniques is that in online learning the mapping is updated after the arrival of every new data point in a scale fashion, whereas batch techniques are used when one has access to the entire training data set at once. It is a versatile sequential learning algorithm because the training observations are sequentially (one-by-one or chunk-by-chunk with varying or fixed chunk length) presented to the learning algorithm. At any time, only the newly arrived single or chunk of observations (instead of the entire past data) are seen and learned. A single or a chunk of training observations is discarded as soon as the learning procedure for that particular (single or chunk of) observation(s) is completed. The learning algorithm has no prior knowledge as to how many training observations will be presented. Unlike other sequential learning algorithms which have many control parameters to be tuned, OSELM with RBFkernel only requires the number of hidden nodes to be specified [47,48].

The proposed method uses an OSELM that can learn data chunk-by-chunk with a fixed chunk size of 20×20, with RBF kernel classification approach to perform malware localization, Tor traffic identification and botnets prohibition in an energetic security mode that needs minimum computational resources and time [7]. The OSELM consists of two main phases namely: boosting phase (BPh) and sequential learning phase (SLPh). The BPh used to train the SLFNs using the primitive ELM method with some batch of training data in the initialization stage and these boosting training data will be discarded as soon as boosting phase is completed. The required batch of training data is very small, which can be equal to the number of hidden neurons [46–48].

The general classification process with OSELM classifier is described below:

Phase 1 (BPh) [47,48]

The process of BPh for a small initial training set $N = \{(x_i, t_i) | x_i \in R^n, t_i \in R^m, i = 1, \cdots, \tilde{N}\}$ is described as follows:

(a) Assign arbitrary input weight $w\beta^{(0)} = M_0 H_0^{\mathrm{T}} T_{0i}$ and bias b_i or center μ_i and impact width $\sigma_i, i = 1, \ldots \tilde{N}$, where $\tilde{N}$ number for hidden neuron or RBF kernel for a specific application.
(b) Calculate the initial hidden layer output matrix $H_0 = [h_1, \cdots, h_{\tilde{N}}]^{\mathrm{T}}$, where $h_i = [g(w_1 \cdot x_i + b_1), \ldots, g(w_{\tilde{N}} \cdot x_i + b_{\tilde{N}})]^{\mathrm{T}}, i = 1, \ldots, \tilde{N}$, where g activation function or RBF kernel.
(c) Estimate the initial output weight, where $M_0 = (H_0^{\mathrm{T}} H_0)^{-1}$ and $T_0 = [t_1, \ldots, t_{\tilde{N}}]^{\mathrm{T}}$.
(d) Set $k = 0$.

Phase 2 (SLPh) [47,48]

In the SLPh the OSELM will then learn the train data chunk-by-chunk with a fixed chunk size of 20×20 and all the training data will be discarded once the learning procedure on these data is completed. The essentials step of this phase for each further coming observation (x_i, t_1), where $x_i \in R^n, t_i \in R^m$ and $i = \tilde{N} + 1, \tilde{N} + 2, \tilde{N} + 3$, described as follows:

(a) Calculate the hidden layer output vector $h_{(k+1)} = [g(w_1 \cdot x_i + b_1), \ldots, g(w_{\tilde{N}} \cdot x_i + b_{\tilde{N}})]^{\mathrm{T}}$
(b) Calculate latest output weight $\beta^{(k+1)}$ by the algorithm $\hat{\beta} = (H^{\mathrm{T}}H)^{-1} H^{\mathrm{T}}T$ which is called the recursive least-squares (RLS) algorithm.
(c) Set $k = k + 1$

The proposed TTIE algorithm includes the following ruleset which is the core of its reasoning and described below.

Step 1 Perform malware localization by OSELM with malware localization dataset (MLD). If the malware analysis gives a positive result (Malware) the network traffic is blocked and the process is terminated. If the malware analy-

sis gives a negative result (Benign), no action is required and goes to step 2.
Step 2 Perform network traffic analysis by OSELM with network traffic classification dataset (NTCD). If the network traffic classification result is not a HTTPS, no action is required. If the network traffic classification result is a HTTPS, go to step 3.
Step 3 Performs Tor-traffic identification by OSELM with Tor-traffic identification dataset (TTID). If the botnet classification result gives a positive result (Botnet) the network traffic blocked and the process terminated. If the botnet classification result gives a negative result (HTTPS), no action is required.

The overall algorithmic approach of TTIE that is proposed herein is described clearly and in detail in the following Fig. 2.

5 Comparative testing

Performance evaluation was performed based on a thorough comparative analysis (20 trials for each test and we compute the average result) of the obtained prediction accuracy and generalization capability between the CIantiMF and the following six corresponding batch mode machine learning methods using a tenfold cross validation approach, namely: random forest (RF), k-nearest neighbors (k-NN), support vector machines (SVM), feed forward neural networks (FFNN), group methods of data handling (GMDH) and polynomial artificial neural network (PANN).

5.1 Datasets

The selection of the data was the result of an extensive research on the functionality of the protocol SSL, combined with a deep analysis of the independent variables, to obtain the ones that give the maximum precision under the strict condition of minimal computing resources consumption. This effort resulted in the creation of training sets, capable to properly train the employed learning algorithms.

Three datasets with high complexity were constructed and used for testing by the CIantiMF. The first MLD comprised 32 independent variables and 2 classes (benign or malware). This dataset containing 73,469 patterns (37,127 benign samples they were chosen from the Pcaps from National Cyber Watch Mid-Atlantic Collegiate Cyber Defense Competition and 36,342 malicious samples they were chosen from http://malware-traffic-analysis.net/) [49]. The idea of the network traffic analysis and the features extraction approach were based on the functional mode of the TCP protocol and moreover on the acknowledgement method of the reliable submission and receipt of the data. Also it relies on the error-free data transfer mechanisms between the network layer and the application layer, of the TCP header structure and the three-way handshake process [50].

The full list of the 32 features with the class is detailed in Table 1.

The second NTCD (Network Traffic Classification Dataset) comprised 22 independent variables and 12 network traffic classes (TELNET, FTP, HTTP, HTTPS, DNS, Lime, Local Forwarding, Remote Forwarding, SCP, SFTP, x11 and Shell). This dataset containing 137,050 patterns they were chosen from the Pcaps from Information Technology Operations Center (ITOC), US Military Academy [51].

The feature management and export (for Tables 2, 3) was based on the analysis of the network traffic and specifically on the methodology used in [52]. The full list of the 22 features with the corresponding classes is presented in the following Table 2.

Finally, the third TTID comprised 45 independent variables and 2 classes (Tor or HTTPS). This dataset containing 217,483 patterns they were chosen from the Pcaps from [53]. The full list of 45 features with their corresponding classes is presented in the following Table 3.

In the preprocessing process the duplicate records and records with missing values were removed. Also the datasets were determined and normalized to the interval $[-1, 1]$ to phase the problem of prevalence of features with wider range over the ones with a narrower range, without being more important [54].

Taking into account the limited capacity of resources and computing power of mobile devices and the limitations posed by their dependence on the battery, we have made a transformation of the TTID independent variables vector space, which is a very complex and extensive dataset. Principal components analysis (PCA) has been performed to obtain new linear combinations, capable to contain the largest possible part of the variance of the original information, without limiting the predictive capability and accuracy of the learning algorithm. However, due to the fact that the results have deteriorated enough (almost 12% less accuracy was obtained) the approach was abandoned.

Then, we have performed correlation analysis on various subsets of features which had the highest correlation with the obtained class, regardless of their interrelation. Also other subsets were used, which were highly correlated with the class and they appeared to have high cross-correlation (correlation attribute evaluation) [55].

Finally, we have tried to use subsets for which we have calculated the cost-sensitive classification, based on the cost matrix (cost sensitive subset evaluation). The method takes a cost matrix and a base evaluator. Cost matrix is a way to change the threshold value for a decision boundary. If the base evaluator can handle instance weights, then the training data are weighted according to the cost matrix, otherwise the training data are sampled according to the cost matrix. The

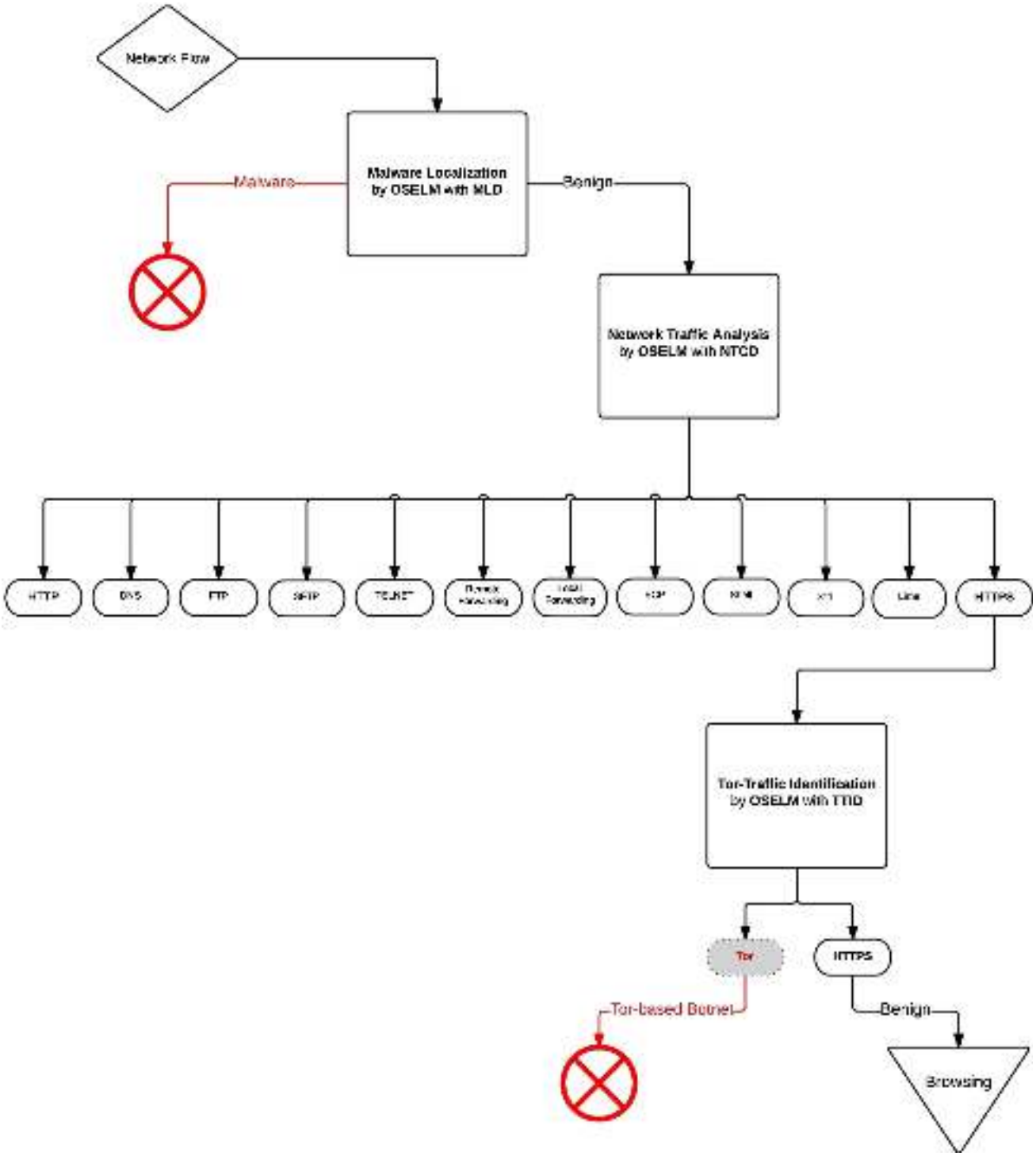

Fig. 2 The proposed architecture of the OTTIE

process of performing cost-sensitive subset evaluation is a very effective method because the error-based methods consider the classification errors as equally likely, which is not the case in all the real-time applications [55,56]. The subsets for which the value of each feature was calculated using the information gain with respect to the class (information gain attribute evaluation) [55].

Eventually, the subset chosen was based on the method of correlation-based feature subset selection (subsets of features that correlate highly with the class value and has low correla-

Table 1 MLD: extracted features from network traffic (31 independent and 2 depended)

ID	Feature name	ID	Feature name
1	seq_number	17	response_seq_number
2	ack_number	18	response_ack_number
3	src_port	19	response_src_port
4	dst_port	20	response_dst_port
5	fin	21	response_fin
6	syn	22	response_syn
7	rst	23	response_rst
8	psh	24	response_psh
9	ack	25	response_ack
10	window	26	response_window
11	check	27	response_check
12	ip_len	28	response_ip_len
13	ip_id	29	response_ip_id
14	ip_off	30	response_ip_off
15	ip_ttl	31	response_ip_ttl
16	ip_sum	32	Classes (benign or malicious)

Table 2 NTCD: extracted features from network traffic (22 independent and 12 depended)

ID	Feature name	ID	Feature name
1	min_fpktl	13	min_biat
2	mean_fpktl	14	mean_biat
3	max_fpktl	15	max_biat
4	std_fpktl	16	std_biat
5	min_bpktl	17	duration
6	mean_bpktl	18	proto
7	max_bpktl	19	total_fpackets
8	std_bpktl	20	total_fvolume
9	min_fiat	21	total_bpackets
10	mean_fiat	22	total_bvolume
11	max_fiat	23	Classes (TELNET, FTP, HTTP, HTTPS, DNS, Lime, Local Forwarding, Remote Forwarding, SCP, SFTP, x11 and Shell)
12	std_fiat		

tion with each other). From this dataset we have obtained the minimum error of the classifier in the training and test data, in relation to the value of each feature (attribute evaluation with particle swarm optimization) [55].

Finally, we have gained 33.5% reduction of the initial parameters, whereas the accuracy dropped only by 0.1% compared to the accuracy of the system that used all 45 features. The following Table 4 presents the 30 features included in the final dataset.

Table 3 TTID: extracted flow statistics from network traffic (45 independent and 2 depended)

ID	Feature name	ID	Feature name
1	srcip	24	max_biat
2	srcport	25	std_biat
3	dstip	26	duration
4	dstport	27	min_active
5	proto	28	mean_active
6	total_fpackets	29	max_active
7	total_fvolume	30	std_active
8	total_bpackets	31	min_idle
9	total_bvolume	32	mean_idle
10	min_fpktl	33	max_idle
11	mean_fpktl	34	std_idle
12	max_fpktl	35	sflow_fpackets
13	std_fpktl	36	sflow_fbytes
14	min_bpktl	37	sflow_bpackets
15	mean_bpktl	38	sflow_bbytes
16	max_bpktl	39	fpsh_cnt
17	std_bpktl	40	bpsh_cnt
18	min_fiat	41	furg_cnt
19	mean_fiat	42	burg_cnt
20	max_fiat	43	total_fhlen
21	std_fiat	44	total_bhlen
22	min_biat	45	dscp
23	mean_biat	46	Classes (Tor or HTTPS)

6 Results and comparative analysis

Given the complexity of the analysis and of the network traffic monitoring which is a realistic and very difficult task of computer security sector, the proposed system managed to perform with high accuracy.

Also it is really important that the created datasets appear to have particularly high complexity, as they emerged taking into account even the most unfavorable scenarios and almost all potential cases of network traffic that may occur. This is a factor that played an important role towards the generalization capacity of the proposed system. It is characteristic that during the classification process, in each case of different scenarios represented by the use of different datasets, the proposed system achieved accuracy rates of at least 94.2%. The value of this percentage can increase taking into account the training method of the OSELM algorithm which is gradually trained, using network traffic data bundles received in

Table 4 The final TTID feature vector after the feature selection process

ID	Feature name	Interpretation
1	srcip	The source IP address of the flow
2	srcport	The source port number of the flow
3	dstip	The destination IP address of the flow
4	dstport	The destination port number of the flow
5	total_fpackets	The total number of packets travelling in the forward direction
6	total_bpackets	The total number of packets travelling in the backward direction
7	min_fpktl	The minimum packet length (in bytes) from the forward direction
8	max_fpktl	The maximum packet length (in bytes) from the forward direction
9	min_bpktl	The minimum packet length (in bytes) from the backward direction
10	max_bpktl	The maximum packet length (in bytes) from the backward direction
11	min_fiat	The minimum interarrival time (in microseconds) between two packets
12	max_fiat	The maximum interarrival time (in microseconds) between two packets
13	min_biat	The minimum interarrival time (in microseconds) between two packets
14	max_biat	The maximum interarrival time (in microseconds) between two packets
15	duration	The time elapsed (in microseconds) from the first packet to the last packet
16	min_active	The minimum duration (in microseconds) of a sub-flow
17	max_active	The maximum duration (in microseconds) of a sub-flow
18	min_idle	The minimum time (in microseconds) the flow was idle
19	max_idle	The maximum time (in microseconds) the flow was idle
20	sflow_fpackets	The average number of forward travelling packets in the sub-flows
21	sflow_fbytes	The average number of bytes, travelling in the forward direction
22	sflow_bpackets	The average number of backward travelling packets in the sub-flows
23	sflow_bbytes	The average number of bytes, travelling in the backward direction
24	fpsh_cnt	The number of times the PSH flag was set for packets travelling in the forward direction
25	bpsh_cnt	The number of times the PSH flag was set for packets travelling in the backward direction
26	furg_cnt	The number of times the URG flag was set for packets travelling in the forward direction
27	burg_cnt	The number of times the URG flag was set for packets travelling in the backward direction
28	total_fhlen	The total header length (network and transport layer) of packets travelling in the forward direction
29	total_bhlen	The total header length (network and transport layer) of packets travelling in the backward direction
30	dscp	Differentiated services code point, a field in the IPv4 and IPv6 headers

real time. The analytical values of the classification efficiency of the algorithm and the comparison with the other six approaches are presented in Tables 5, 6 and 7. The validation metrics used are the following: accuracy (ACC), root mean square error (RMSE), precision, recall, F-score and receiver operating characteristic (ROC) (Table 8).

Classification accuracy is the number of correct predictions made divided by the total number of predictions made, multiplied by 100 to turn it into a percentage. Precision is the number of true positives divided by the number of true positives and false positives. Recall is the number of true positives divided by the number of true positives and the number of false negatives. The F-score is the 2 × ((precision × recall)/(precision+recall)). The true-positive rate (sensitivity) and false-positive rate (specificity) vary across the different threshold and the sensitivity is inversely related with specificity. Then, the plot of sensitivity versus 1-specificity is called receiver operating characteristic (ROC) curve and the ROC area is an effective measure of accuracy. The RMSE represents the sample standard deviation of the differences between predicted values and observed values.

The precision measure shows what percentage of positive predictions were correct, whereas recall measures what percentage of positive events were correctly predicted. The F-score can be interpreted as a weighted average of the precision and recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F-score is usually more useful than accuracy and it works best if false positives and false negatives have similar cost, in this case. Finally, the ROC curve is related in a direct and natural way to cost/benefit analysis of diagnostic decision making. The very high sensitivity

Table 5 Classification accuracy (ACC) and performance metrics for the MLDataset

Classifier	MLDataset						
	ACC (%)	RMSE	Precision (%)	Recall	F-score (%)	ROC	Evaluation
OSELM	98.2	0.3284	0.981	0.982	0.982	0.990	20×20
RF	97.3	0.3607	0.973	0.973	0.973	0.981	Tenfold CV
k-NN	92.7	0.5349	0.927	0.927	0.927	0.936	Tenfold CV
SVM	97.5	0.3542	0.975	0.975	0.975	0.975	Tenfold CV
FFNN	93.7	0.5172	0.939	0.940	0.940	0.962	Tenfold CV
GMDH	94.4	0.5017	0.944	0.944	0.945	0.955	Tenfold CV
PANN	90.9	0.5633	0.909	0.910	0.910	0.914	Tenfold CV

Table 6 Classification accuracy (ACC) and performance metrics for the NTCDataset

Classifier	NTCDataset						
	ACC (%)	RMSE	Precision (%)	Recall	F-score (%)	ROC	Evaluation
OSELM	99.6	0.2951	0.996	0.997	0.997	0.998	20×20
RF	98.4	0.3112	0.984	0.983	0.983	0.990	Tenfold CV
k-NN	95.8	0.4603	0.958	0.958	0.958	0.961	Tenfold CV
SVM	98.3	0.3217	0.983	0.983	0.983	0.987	Tenfold CV
FFNN	96.9	0.4489	0.970	0.969	0.970	0.972	Tenfold CV
GMDH	97.8	0.3983	0.978	0.978	0.978	0.978	Tenfold CV
PANN	96.6	0.4512	0.967	0.966	0.966	0.967	Tenfold CV

Table 7 Classification Accuracy (ACC) & Performance Metrics for the TTIDataset

Classifier	TTIDataset						
	ACC (%)	RMSE	Precision (%)	Recall	F-score (%)	ROC	Evaluation
OSELM	94.2	0.5018	0.942	0.942	0.942	0.950	20×20
RF	92.9	0.5206	0.923	0.929	0.923	0.931	Tenfold CV
k-NN	90.1	0.5437	0.901	0.901	0.901	0.910	Tenfold CV
SVM	93.5	0.5173	0.935	0.936	0.935	0.946	Tenfold CV
FFNN	92.4	0.5212	0.924	0.924	0.924	0.925	Tenfold CV
GMDH	88.8	0.5831	0.889	0.888	0.888	0.890	Tenfold CV
PANN	87.3	0.5937	0.873	0.873	0.873	0.874	Tenfold CV

rates, which represent the true malicious traffic identification cases (true positive rate), are typical and indicative of the quality of the process. This is also shown by the size of the ROC curves. This comparison generates very encouraging expectations for the recovery and identification of OSELM, as a robust model for such a complex real-time problem.

7 Discussion and conclusions

This research effort, presented a timely, innovative, small footprint and highly effective security system which relies on advanced methods of computational intelligence and it greatly enhances the android OS security mechanisms. The android OS was chosen as the application environment, due to its popularity that makes it one of the main attack targets of cyber criminals, especially for banking applications. The computational intelligence anti-malware framework (CIantiMF) uses two advanced extensions running under the ART JVM. The CIantiMF is able to recognize whether the java classes of an android application are benign or malicious. It performs network traffic analysis to identify the Tor-based Botnets.

The performance of the proposed system was tested on a novel dataset of high complexity, which emerged after extensive research of how the SSL protocol operates and after performing comparisons inspections and tests of independent variables which give the maximum precision, while requiring minimal computational resources.

The very high accuracy results obtained significantly enhance the overall methodology followed. The evaluation of the proposed architecture was based on a comparative analysis with six corresponding computational intelligence methods, with very promising output. Proposals for development and future enhancements of this system should be targeted to optimize the parameters of the ELM algorithm

Table 8 The initial parameters of compared algorithms

Algorithm	Parameter	Value
OSELM	Hidden neurons	25
	Activation function	RBF kernel
	Chunk size	20×20
RF	MaxDepth	Unlimited
	BagSize	100
	Number of iterations	100
k-NN	Number of neighbours	1
	Search algorithm	Linear search (euclidean distance)
SVM	Loss	0.1
	Eps	0.01
	Gamma	1/max_index
	Cost	1.0
	Kernel	RBF
FFNN	Hidden layers	12
	Max iterations	1000
	Learning rate	0.1
GMDH	Number of neurons in a layer	15
	Layers	4
	Selection pressure (in layers)	0.6
	Train ratio	0.85
PANN	Number of neurons in a layer	10
	Layers	3
	Train ratio	0.85

used, so as to achieve even more efficient, more accurate and faster classification.

Also it would be important for the proposed framework to be expanded with automatic extraction methods of network traffic characteristics, with semi- and unsupervised learning, so that it would fully automate the process of identifying malicious applications.

Finally, an additional element that could be studied towards the future expansion of this approach is to create an additional extension that can detect algorithmically generated malicious domains names, so that even more protection would be offered towards newest methods of malicious attacks.

References

1. https://source.android.com/security/index.html
2. Danisevskis, J.: Uncloaking rootkits on mobile devices with a hypervisor-based detector. Information Security and Cryptology-ICISC 2015, vol. 9558. Springer, Berlin (2016)
3. Rudd, E., et al.: A survey of stealth malware: attacks, mitigation measures, and steps toward autonomous open world solutions (2016). arXiv:1603.06028
4. Hayes, J.: Traffic confirmation attacks despite noise (2016). arXiv:1601.04893
5. Backes, M., et al.: Provably secure and practical onion routing. In: Computer Security Foundations Symposium (CSF), 2012 IEEE 25th. IEEE, New York (2012)
6. Bansal, D., Priya, S., Shipra, K.: Secure socket layer and its security analysis. Netw. Commun. Eng. **7**(6), 255–259 (2015)
7. Huang, Guang-Bin, Qin-Yu, Z., Chee-Kheong, S.: Extreme learning machine: theory and applications. Neurocomputing **70**(1–3), 489–501 (2006)
8. Demertzis, K., Iliadis, L.: A hybrid network anomaly and intrusion detection approach based on evolving spiking neural network classification (2014). Commun Comput Inf Sci **441**, 11–23 (2014). doi:10.1007/978-3-319-11710-2_2
9. Demertzis, K., Iliadis, L.: Evolving computational intelligence system for malware detection. Lect. Notes Bus. Inf. Process. **178**, 322–334 (2014)
10. Demertzis, K., Iliadis L.: Bio-inspired hybrid artificial intelligence framework for cyber security. In: Proceedings of 2nd CryptAAF (Cryptography and Its Applications in the Armed Forces), 2 April 2014, Athens, Greece. Computation, Cryptography, and Network Security. Computation, Cryptography, and Network Security. Springer International Publishing, Berlin, pp. 161–193. doi:10.1007/978-3-319-18275-9_7

11. Demertzis K., Iliadis L.: Bio-Inspired Hybrid Intelligent Method for Detecting Android Malware Proceedings of 9th International Conference on Knowledge, Information and Creativity Support Systems (KICSS 2014). ISBN: 978-9963-700-84-4 ("KICSS'2014 Proceedings")
12. Demertzis, K., Iliadis, L.: Evolving smart URL filter in a zone-based policy firewall for detecting algorithmically generated malicious domains. Statistical learning and data sciences. In: Series Lecture Notes in Computer Science. Third International Symposium, SLDS 2015, Egham, UK, April 20–23, 2015, Proceedings, vol. 9047, pp. 223–233. Springer International Publishing, Berlin. doi:10.1007/978-3-319-17091-6_17
13. Schmidt, A.D., Schmidt, H.G., Batyuk, L., Clausen, J.H., Camtepe, S.A., Albayrak, S., Yildizli, C.: Smartphone malware evolution revisited: android next target? In: Proceedings of the 4th IEEE International Conference on Malicious and Unwanted Software, pp. 1–7. IEEE, New York (2009)
14. Schmidt, A.D., Bye, R., Schmidt, H.G., Clausen, J., Kiraz, O., Yüksel, K., Camtepe, A., Albayrak, S.: Static analysis of executables for collaborative malware detection on android. In: IEEE International Congress on Communication (ICC) (2009)
15. Enck, W., Ongtang, M., McDaniel, P.: Understanding android security. IEEE Secur. Priv. **7**(1), 50–57 (2009)
16. Shabtai A., Fledel, Y., Elovici, Y.: Securing android powered mobile devices using selinux. IEEE Security and Privacy, vol. 99 (2009). **(PrePrints)**
17. Scandariato, R., Walden, J.: Predicting vulnerable classes in an android application (2012)
18. Shabtai, A., Fledel, Y., Elovici, Y.: Automated static code analysis for classifying android applications using machine learning. CIS. Conf. IEEE **2010**, 329–333 (2010)
19. Chin, E., Felt, A., Greenwood, K., Wagner, D.: Analyzing inter-application communication in android. In: 9th Conference on Mobile Systems, Applications, and Services. ACM, New York, pp. 239–252 (2011)
20. Burguera, I., Zurutuza, U., Nadjm-Tehrani, S.: Crowdroid: behavior-based malware detection system for android. In: 1st ACM Workshop on on SPSM. ACM, New York, pp. 15–26 (2011)
21. Glodek, W., Harang, R.R.: Permissions-based detection and analysis of mobile malware using random decision forests. In: IEEE Military Communications Conference (2013)
22. Zhang, J., et al.: An effective network traffic classification method with unknown flow detection. IEEE Trans. Netw. Serv. Manag. **10**(2), 133–147 (2013)
23. Gardiner, J., Shishir, N.: On the reliability of network measurement techniques used for malware traffic analysis. Secur. Protoc. **XXII**, 321–333 (2014)
24. Wang, H.T., et al.: Real-time fast-flux identification via localized spatial geolocation detection. In: Computer Software and Applications Conference (COMPSAC). IEEE, New York (2012)
25. Tu, T.D., Cheng, G., Liang, Y.X.: Detecting bot-infected machines based on analyzing the similar periodic DNS queries. In: 2015 International Conference on Communications, Management and Telecommunications (ComManTel). IEEE, New York (2015)
26. Sangroudi, A.A., Seyed, J.M.: Botnets detection for keeping the security of computer systems based on fuzzy clustering. Ind. J. Sci. Technol. **8**(28), 1 (2015)
27. Soltanaghaei, E., Kharrazi, M.: Detection of fast-flux botnets through DNS traffic analysis. Scientia Iranica. Trans D Comput Sci Eng Electr 22(6), 2389 (2015)
28. Wright, M.K., Adler, M., Levine, B.N., Shields, C.: An analysis of the degradation of anonymous protocols. In: Proceed. of the Network and Distributed Security Symposium (2002)
29. Shmatikov, V., Wang, M.H.: Timing analysis in low-latency mixnetworks: attacks and defenses. In: Proceedings of ESORICS (2006)
30. Cheng, C., Peng, T.W., Guang-Bin, H.: Extreme learning machines for intrusion detection: IJCNN. In: International Joint Conference (2012). doi:10.1109/IJCNN.2012.6252449
31. Hsu, C.H., Huang, C.Y., Chen, K.T.: Fast-flux bot detection in real time. In: 13th International Conference on Recent Advances in Intrusion Detection, ser. RAID'10 (2010)
32. Haffner, P., Sen, S., Spatscheck, O., Wang, D.: ACAS: auto-mated construction of application signatures. In: Proceedings of the ACM SIGCOMM, pp. 197–202 (2005)
33. Alshammari, R., Zincir-Heywood, N.A.: A flow based approach for SSH traffic detection. In: IEEE International Conference on Cybernetics, ISIC, pp. 296–301 (2007)
34. Holz, T., Gorecki, C., Rieck, K., Freiling, F.: Measuring and detecting fast-flux service networks. In: NDSS '08: Proceedings of the Network & Distributed System Security (2008)
35. Apvrille, A., Strazzere, T.: Reducing the window of opportunity forandroid malware: Gotta catch 'em all. J. Comput. Virol. **8**(1–2), 61–71 (2012)
36. Burguera, I., Zurutuza, U., Nadjm-Tehrani, S.: Crowdroid: behavior-based malware detection system for android. In: ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM) (2011)
37. Afonso, V.M., de Amorim, M.F., Gr'egio, A.R.A., Junquera, G.B., de Geus, P.L.: Identifying android malware using dynamically obtained features. J. Comput. Virol. Hack. Techniq. (2014)
38. Dini, G., Martinelli, F., Saracino, A., Sgandurra, D.: MADAM: a multi-level anomaly detector for android malware. In: Proceedings of 6^{th} MMM-ACNS, St. Petersburg, Russia (2012)
39. Wu, W.-C., Hung, S.-H.: DroidDolphin: a dynamic androidmalware detection framework using big data and machine learning. In: Conference on Research in Adaptive and Convergent Systems (RACS) (2014)
40. Chakravarty, S., Barbera, M.V., Portokalidis, G., Polychronakis, M., Keromytis, A.D.: On the effectiveness of traffic analysis against anonymity networks using flow records. In: Proceedings on 15th International Conference, PAM 2014, pp 247–257, Springer, Berlin (2014)
41. Almubayed, A., Hadi, A., Atoum, J.: A model for detecting tor encrypted traffic using supervised machine learning, I. J. Comput. Netw. Inf. Secur. **7**, 10–23 (2015)
42. Chaabane, A., Manils, P., Kaafar, M.A.: Digging into anonymous traffic: a deep analysis of the tor anonymizing network. In: 4th International Conference on Network and System Security (NSS), pp. 167–174 (2010)
43. Chakravarty, S., Stavrou, A., Keromytis, A.D.: Traffic analysis against low-latency anonymity networks using available bandwidth estimation. In: Proceedings of the 15th European Conference on Research in Computer Security, Ser. ESORICS'10, pp. 249–267. Springer, Berlin (2010)
44. Chakravarty, S., Stavrou, A., Keromytis, A.D.: Identifying proxy nodes in a tor anonymization circuit. In: Proceedings of the 2nd Workshop on Security and Privacy in Telecommunications and Information Systems (SePTIS), December 2008, pp. 633–639
45. Demertzis, K., Lazaros I.: SAME: An Intelligent Anti-Malware Extension for Android ART Virtual Machine, Computational Collective Intelligence, pp. 235–245. Springer, Berlin (2015)
46. Liang, N.-Y., Huang, G.-B., Saratchandran, P., Sundararajan, N.: A fast and accurate on-line sequential learning algorithm for feedforward networks. IEEE Trans. Neural Netw. **17**(6), 1411–1423 (2006)
47. Cambria, E., Guang-Bin, H.: Extreme learning machines. IEEE InTeLLIGenT SYSTemS 541-1672/13 (2013)
48. Huang G.-B., Liang N.-Y., Rong H.-J., Saratchandran P., Sundararajan N.: On-line sequential extreme learning machine, IASTED (2005)
49. http://malware-traffic-analysis.net/

50. Haining, W., Danlu, Z., Kang, G.S.: Detecting SYN flooding attacks, proceedings on INFOCOM 2002. Twenty-First Annu. Joint Conf. IEEE Comput. Commun. Soc. **3**, 1530–1539 (2002)
51. http://www.netresec.com/?page=PcapFiles
52. Arndt, D.J., Zincir-Heywood, A.N.: 2011 IEEE Symposium on A Comparison of Three Machine Learning Techniques for Encrypted Network Traffic Analysis, Computational Intelligence for Security and Defense Applications (CISDA), pp. 107–114
53. http://contagiodump.blogspot.gr/
54. Iliadis, L.: Intelligent Information Systems and Applications in Risk Estimation. ISBN: 978-960-6741-33-3 A. Stamoulis Publication, Thessaloniki (2008)
55. Bailey, M., Oberheide, J., Andersen, J., Mao, Z.M., Jahanian, F., Nazario, J.: Automated classification and analysis of internet malware. In: KrÃijgel, C., Lippmann, R., Clark, A. (eds.). RAID of Lecture Notes in Computer Science, vol. 4637, pp. 178–197. Springer, Berlin (2007)
56. Desai, A., Jadav, P.M.: An empirical evaluation of adaboost extensions for cost-sensitive classification. Int. J. Comput. Appl. **44**(13), 34–41 (2012)