2021 11th IFIP International Conference on New Technologies, Mobility and Security (NTMS) | 978-1-6654-4399-9/20/$31.00 ©2021 IEEE | DOI: 10.1109/NTMS49979.2021.9432640

SDN Security through System Call Learning

Danai Chasaki
Department of Electrical and Computer Engineering
Villanova University
Email: danai.chasaki@villanova.edu

Christopher Mansour
Department of Computing and Information Science
Mercyhurst University
Email: cmansour@mercyhurst.edu

***Abstract*—Software Defined Networking (SDN) has changed the way of designing and managing networks. This technology emerged from the efforts to make the networks more programmable. Programmability allows the development of various third-party applications that can be used by network administrators to implement essential networking functionalities. However, programmability also introduces security threats. In this paper we address the issue of malicious hosts running malicious applications that bypass the standard SDN based detection mechanisms. The SDN security system we are proposing periodically monitors the system calls utilization of the different SDN applications installed, learns from past system behavior using machine learning classifiers, and thus accurately detects the existence of an unusual activity or a malicious application.**

I. Introduction

In the past few years, Software Defined Networking (SDN) has become an established novel networking paradigm that facilitates the development and management of computer networks. Software Defined Networking has two main characteristics: a) it separates the control plane from the data plane, b) SDN centralizes the control plane, hence a single software control program controls multiple data planes. The SDN control plane executes direct control over the state in the data plane elements using a well-defined Application Programming Interface (API), such as OpenFlow [1].

SDN originally started as an academic experiment, however, it became popular in the industry over the years. Many vendors of commercial switches are including support of the OpenFlow API in their equipment. Additionally, many large information-technology companies such as cloud providers, carriers, equipment vendors, and financial-services firms have started implementing SDNs. These companies include Google, Facebook, Yahoo, Microsoft, Verizon, and Deutsche Telekom fund Open Networking Foundation (ONF). For example, Google deployed an SDN to interconnect its data centers worldwide, which has helped the company improving its operational efficiency and to reduce costs significantly. Moreover, NSX, a VMware's network virtualization platform, delivers a fully functional SDN based network in software, provisioned independent of the underlying networking devices [2].

Although SDNs have aggressively evolved in the past few years, the attack surface of SDN has become a trending research topic in order to detect and mitigate possible security breaches against SDN [3]. Adversaries can now target virtual hosts connected to the network. Shin et. al. showed an example of a malicious attack where the attack code can be stored in the payload of the packet and thus will be executed by the virtual node while the packet's header is being inspected by the SDN controller [4].

Additionally, counterfeit virtual hosts may exist that inject faults or malicious data that can corrupt services and threaten the integrity and trustworthiness of the information being exchanged. A counterfeit malicious host can be the single point of entry to the network giving the attacker an easy access to the networking infrastructure. Malicious code can be carefully inserted between legitimate lines of code in order to be executed later allowing attackers to gain access to systems and disclose information.

In this paper we use a type of "side channel analysis" to detect any type of malicious activity or application that changes the processing behavior of the virtual host. We chose our side channel to be the operating system calls of the virtual host. We feed extracted system call features - from a mix of benign and malicious applications - into machine learning classifiers to train our security system to accurately detect suspicious activities that change the host's processing behavior and consequently the system call mix.

Many authors have addressed the analysis of system calls for detecting anomalies in traditional systems. Furthermore, using system call sequences as inputs on Machine Learning classifiers have been explored for traditional systems. However, the introduction of machine learning techniques to detect possible threats in an SDN is a novel concept explored in this paper.

This paper focuses on the detection of attacks from malicious hosts on a Software Defined Network using their system calls as inputs into Machine Learning classifiers. Furthermore, this work presents the following contributions:

- Setting up an SDN using tools such as Docker and Open vSwitch.
- Establishing different testing scenarios, ranging from normal operation, where only the basic OS is running, to a mild buffer overflow attack scenario and a heavy ping attack scenario.
- Creating a test-bench from different samples of system call traces.
- Applying different machine learning classifiers to determine if a system call trace belongs to an attack or to a normal scenario.

The remainder of the paper is organized as follows: Section II discusses some related work. In Section III, we describe the methodology used to set up the SDN, trace the system calls, and train the different machine learning classifiers. Evaluation and results of the proof of concept implementation are presented in Section IV. Section V summarizes and concludes the paper with some future work.

II. Related Work

A. Attacks against SDNs

With the emergence and growing popularity of the SDNs, several authors have shown the variety of possibilities that attackers can use when it comes to attack SDNs. Kreutz, et al. [2] state that security and dependability should be top priorities in an SDN, and that these issues should be promptly addressed and investigated. Attack vectors such as forger traffic flows in the data plane, forwarding devices and to wreak havoc with the network, and attacks on vulnerabilities in administrative stations can affect SDNs. The authors mention that some attack vectors that are specific to SDNs include attacks on control plane communication and logically centralized controllers. In addition, they explored the security issues that OpenFlow can encounter, such as spoofing, tampering, repudiation, information disclosure, denial of service, elevation of privileges, and assumption that all applications are benign.

Lee, et al. [3], tested three attack scenarios that leverage SDN applications in order to attack the SDN network. These scenarios were tested against three known SDN controllers, which are ONOS, OpenDaylight and Floodlight. Additionally, they discussed possible defense mechanisms, however, they did not propose any detection mechanism for these attack scenarios.

Ropke, et al [5], focused on a rootkit technique to enable attackers to sabotage network operating systems. Their work focused on demonstrating potential threats for a wide range of network operating systems, by presenting two prototypes: an SDN rootkit for the controller OpenDaylight; and a version with basic rootkit functions for the commercial and non-OpenDaylight-based HP controller. Their rootkit is capable of actively hiding itself, hide malicious network programming, and also provide remote access.

Katta, et al. [6], introduced a fault-tolerant SDN controller platform named Ravana. This platform could process control messages transactionally and exactly once, maintaining these guarantees in the face of a crash of the controller or switch. The authors claim that replicated state machines can be extended with lightweight switch-side mechanisms in order to guarantee correctness, and without taking the switches into an elaborate consensus protocol. Additionally, the Ravana prototype enables SDN applications to execute in a fault-tolerant fashion. Ravana has achieved high throughput with reasonable overhead, with a failover time under 100ms.

Chandrasekaran, et al [7] focused their research in the reliability and fault tolerance of the SDN. They claim that the relationship between the SDN-Apps and controllers, and the SDN-App and the network are the points where the main issues rely. The authors re-designed the controller architecture in order to make the controllers and network resilient to SDN-Apps failures by eliminating the before mentioned relations. This design is centered around a set of abstractions, which are used to improve the reliability of an SDN environment.

B. System calls and Machine Learning analysis applied to other technologies

In [8] an anomaly-based and host-based approach to detect mobile botnets is presented. The authors use machine learning algorithms in order to identify anomalous behaviors in statistical features extracted from system calls. They used a self-generated dataset with 13 families of mobile botnets and legitimate applications. They tested the performance of this approach in a close-to-reality scenario, achieving great results such as low false positive rates and high true detection rates. The authors used the Random Forest classifier, the Linear SVM classifier, and the RBF SVM classifier for this approach.

In [9] a machine learning based approach to detect malicious Android apps using discriminant system calls is presented. The system calls were extracted using machine learning and then an empirical estimation of the system calls, derived from diverse datasets employing human interaction and random input. They defined two feature selection approach, the Absolute Difference of Weighted System Calls (ADWSC) and the Ranked System Calls using Large Population Test (RSLPT) to validate the results on five datasets. The evaluated the effectiveness of the classifier against adversarial attacks, finding that the classifiers are vulnerable to data poisoning and label flipping attacks.

Canzanese, et al. [10] presented a study where they used system call analysis to detect malware that evade traditional defenses. Their system would monitor executing processes to identify compromised hosts in production environments. The authors claim that the experimental results compare the effectiveness of multiple feature extraction strategies and detectors based on their detection accuracy at low false positive rates. They presented a case study that indicates that the detection system performs well against a variety of malware samples, benign workloads, and host configurations.

Kolosnjaji, et al. [11] used deep learning to classify malware system call sequences by constructing a neural network using convolutional neural networks and recurrent network layers to find the best features for a classification task. This led to a hierarchical feature extraction architecture combining convolution of n-grams with full sequential modeling. The authors claim that the results show that this approach outperforms previously used methods for malware classification, achieving an average of 85.6% on precision and 89.4% on recall.

Rosenberg [12] implemented an intrusion detection system based on a configurable machine learning classifier. In addition, he used the system calls executed by the inspected code during its run-time in a sandbox as features and measured the effectiveness of the IDS when detecting malware. Finally, he used the IDS to compare between different machine learning classifiers to find the fittest. He used the following classifiers: Decision Tree, Random Forest, K-Nearest Neighbor, Gausian

Naive Bayes, Bernoulli Naive Bayes, Ada-Boots, Linear SVM, RBF SVM, and linear Discriminant Analysis.

Existing literature is primarily focused on the detection of security attacks through packet inspection. in [13] Chasaki et al. showed that constant system call monitoring is a somewhat effective technique to detect denial of service attacks, with low number of false negatives but high number of false positives. However, using that framework, mild types of attacks are impossible to detect and all SDN applications need to have complete benchmarking data for the attack detection to work. In this paper we take this approach one step further, focusing on the analysis of SDNs system calls using machine learning to detect various types of malicious activities.

III. Methodology

A. SDN Setup

In order to emulate an SDN virtual environment we created a framework consisting of an Open vSwitch (OVS) and a Floodlight controller. This framework was developed using Docker containers. Once the containers are created, they can connect using a docker network created by default, however, for our work, a software defined network is created with OVS.

For our setup we used six containers, one that acts as the web Server and five that make periodical requests to the Web Server. The job of the controller is to gather system call data for each application that is running on the virtual hosts (system call type and frequency), run machine learning classifiers and decide whether the application at hand is benign or not. One of the virtual host performs a buffer overflow on the web server, which manifests into a mild type of attack in terms of system call variation, while the five containers perform a distributed denial of service attack on the web server, which leads to a severe type of attack manifestation.

B. System Calls Monitoring

System calls dictate how a program or application requests a service from the kernel of the operating system (OS). This is the way SDN applications interact with the underlying OS. System calls are the only entry points into the kernel, and thus provide a comprehensive view of the network node's processing behavior. Each application on the network node has a specific number and order of system calls that it utilizes when performing an operation or task.

The Linux utility strace [14] is very useful for diagnostic and debugging purposes. It helps monitoring and tampering with interactions between processes and the Linux kernel, such as system calls, among others. Capturing and monitoring the system calls can be very useful for the detection of malicious applications on virtual hosts, specifically those that are exploited to perform code injection attacks. The malicious application will use system calls in a different way than what is expected. This variation may be reflected with new calls, i.e. new type of calls, that have never been used before, a different order on the calls, additional number of existing function calls.

Figure 1 represents our system's architecture. Hundreds of training data are collected in order to help the machine

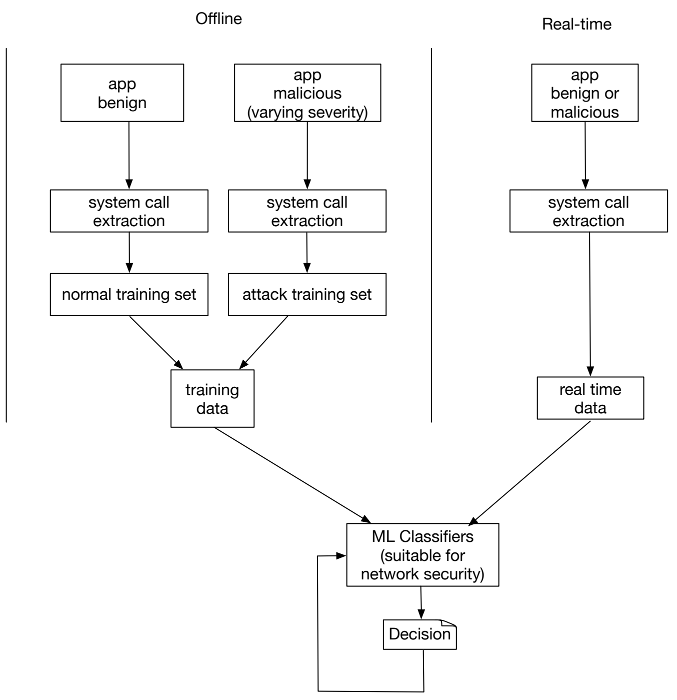

Fig. 1: System Architecture

learning classifiers decide whether the application running on the virtual host belongs to the benign class or the malicious class (binary classification).

First, there is an offline phase where benign networking applications running on our virtual hosts are analyzed. Their dominant system calls are extracted and used as normal training data sets for a handful of carefully selected machine learning classifiers. At the same time malicious applications that lead to mild, moderate and severe attacks are analyzed. Their system call mix is extracted as well, and is added to the benign training data.

We need to note here that traditional machine learning algorithms have the intrinsic assumption that the two classes - benign vs. attack - have roughly the same number of data points. In most cases, the majority of the network security data will be normal data. Only a small percentage of the total network traffic, files, system calls, etc., actually indicated malicious intent. In order to be consistent with real life we selected three training scenarios: a) 99% benign and 1% malicious traces b) 90% benign and 10% malicious traces, and c) 75% benign and 25% malicious traces. To solve the challenge posed by the class-imbalanced data sets instead of up-sampling (simply repeating the same minority class data points), we used the concept of Synthetic Minority Over Sampling Technique (SMOTE) proposed by Chawla et al [15].

In real-time, the controller will periodically poll the applications while running a particular task, will extract the type and frequency of the system calls involved, and with the help of the ML classifiers determine the existence of malicious activities.

TABLE I: Sample System Call Mix - Percentage Frequency

System Call Type	Benign App	Mild Attack	Severe Attack
fstat (%)	7.4	8	8.2
ioctl (%)	3.7	4	4.2
lseek (%)	3.7	4	4.2
read (%)	20.8	17.9	18.3
stat (%)	4.6	5	5
write (%)	7	7.1	7.9

C. Learning Process

For the system call learning process we first considered choosing the best performing machine learning classifier out of the ones that are popular in network security research. However, we came across the idea of multi model-training which was proposed by Zhou and Li [16] to include democratic-style learning with several models in semi-supervised learning mode. In this way, the different models help diversify their inherent inductive biases. At each iteration, if $N-1$ models agree on a label for a previously unlabeled data point, it is added to the Nth model as training data.The final prediction can be made with a majority mode scheme.

The models we selected are: Random Forest, Support Vector Machine (SVM) algorithms, the Nearest Neighbors Classifier (kNN) model and the Gaussian Naive Bayes model. The first two were were selected to provide two different view points for the multi-training analysis. These two algorithms have significantly different inductive biases, and have been proven to have good performance in past studies on network security data sets. kNN was also selected because it is a different type of classifier, a Link-based Object Classification (LOC) algorithm, and has been used in recent fraud investigation (most often class-imbalanced) techniques [17]. The Gaussian Naive Bayes model was also selected due to the amount of features our data has. There are multiple types of system calls and the GBN model assumes that the features are independent from each other, which holds true for most system call functions.

IV. Results and Evaluation

Table I illustrates three sample sets of system calls extracted by a) an SDN application acting normally, b) a malicious application that results in a mild attack, c) a malicious application that results in a severe attack. As we observe, the system calls do not vary drastically under normal or attack scenarios, which would make it impossible to detect an attack visually or by conventional monitoring. However, using our multi-model system call learning methodology we are able to detect attacks of various severity with high confidence.

In order to evaluate the efficacy of system call learning in SDN, we used the metrics most often used by data scientists [18]: Accuracy, Precision, Recall and F1 score.

The Accuracy can be considered the most intuitive performance metric, especially when the dataset is symmetric because it is defined as the ratio of correctly predicted observation to the total observations.

$$Accuracy = \frac{TruePositives + TrueNegatives}{TotalSampleSize} = \\ = \frac{TP+TN}{TP+FP+TN+FN} \tag{1}$$

The Precision can be defined as the ratio of correctly predicted positive observations to the total predicted positive observations. High precision is related to a low false positive rate.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} = \\ = \frac{TP}{TP+FP} \tag{2}$$

The Recall, also known as sensitivity, is considered the ratio of correctly predicted positive observations to all the observations in the actual class.

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} = \\ = \frac{TP}{TP+FN} \tag{3}$$

The F1 score is defined as the weighed average of Precision and Recall. This metric takes false positives and negatives into consideration.

$$F1Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{4}$$

When applied in the network security field, due to the class-imbalanced nature, the number of negative samples usually overwhelm the positive samples. This makes the Accuracy rather insensitive to the actual performance of the model. Because Precision focuses more on the predicted positives rather the negatives, it is affected less by the large number of true negative samples. It is a good metric to use when the cost of false positive detection is high. Recall, on the other hand, focuses on the actual positive samples. It is the metric of choice if the cost of false negative detections is very high.

F1 Score provides the balance between Precision and Recall by calculating the harmonic mean of Precision and Recall. It is very useful when it comes to an uneven dataset, which is our case. Therefore, the rest of the study will adopt F1 Score as the measure of study performance to take into account of the effect of both False Positive and False Negative predictions. However, in specific applications, model tuning should be adopted with the most appropriate metric.

Let's first define the three training scenarios we used to validate our methodology. To emulate realistic network attack type of traffic, we used the majority of our training data from the analysis of benign applications.

- The first scenario: $S1$ includes 99% training data from benign system operation, and 1% training data from malicious applications of varying severity.

- The second scenario: $S2$ includes 90% training data from benign system operation, and 10% training data from malicious applications of varying severity.
- The third scenario: $S3$ includes 75% training data from benign system operation, and 25% training data from malicious applications of varying severity.

Table II shows the F1 Score obtained for the different models and the three different scenarios. An F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0. As we can see, the Random Forest model got the highest F1 Score in Scenario 1, where we have very limited amount of malicious training data. This is the most common scenario in real network traffic. The k-Nearest Neighbors model got the best F1 Score on the second scenario. Finally, the Random Forest model got the highest F1 Score on the third scenario, where 25% of our training data belonged to malicious traces. The score reached almost perfect value of 0.951.

These results are very encouraging, and as other network security research concludes, the Random Forest model is the ideal candidate for malicious attack detection with low number of false positives and low number of false positives as well.

TABLE II: F1 Score for the different classification models under Scenarios 1, 2 and 3[1]

Model	Scenario 1	Scenario 2	Scenario 3
GNB (F1 score)	0.627	0.659	0.776
Linear (F1 score) SVM	0.909	0.889	0.938
RBF SVM (F1 Score)	0.923	0.898	0.938
kNN (F1 score)	0.91	0.928	0.938
RF (F1 score)	0.925	0.92	0.951

GBN does not perform well in our experiments; we attribute the low performance to the fact that the dominant system calls of our data as indicated by the $feature\_importances$ function happen to be reads and writes which are not completely independent types of system calls. In other types of applications GBN could have a better performance, if all the extracted features have no correlation.

V. Conclusion and Future Work

In this paper, we proposed the design and implementation of an SDN security framework that runs of the SDN controller and helps in detecting malicious activities in SDN networks through the use of a side channel. The proposed security framework periodically extracts system call data from the underlying virtual Operating System and uses the data as input to a multi-model machine learning scheme. Malicious applications when executing malicious code or even when they perform network denial of service attacks, will have different system calls utilization. We showed that by feeding our ML models real-time system call characteristics unusual activities can be detected accurately with low number of false positives and low number of false negatives.

[1]Scenario 1: 1% malicious training data, Scenario 2: 10% malicious training data, Scenario 3: 25% malicious training data

In the future, this design can be improved by learning other types of side channel information, like inter-arrival packet time, or cpu cycles. The decision of the ML models can always be fed back to the classifiers as input to close the loop and learn if the trace was labelled correctly. Overall, this is a very promising area of research that can be applied in other types of networked environments, like the IoT domain.

References

[1] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: an intellectual history of programmable networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87–98, 2014.

[2] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan 2015.

[3] S. Lee, C. Yoon, and S. Shin, "The smaller, the shrewder: A simple malicious application can kill an entire sdn environment," in *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks &#38; Network Function Virtualization*, ser. SDN-NFV Security '16. New York, NY, USA: ACM, 2016, pp. 23–28. [Online]. Available: http://doi.acm.org/10.1145/2876019.2876024

[4] S. Shin, L. Xu, S. Hong, and G. Gu, "Enhancing network security through software defined networking (sdn)," in *Computer Communication and Networks (ICCCN), 2016 25th International Conference on*. IEEE, 2016, pp. 1–9.

[5] C. Röpke and T. Holz, "Sdn rootkits: Subverting network operating systems of software-defined networks," in *Research in Attacks, Intrusions, and Defenses*, H. Bos, F. Monrose, and G. Blanc, Eds. Cham: Springer International Publishing, 2015, pp. 339–356.

[6] N. Katta, H. Zhang, M. Freedman, and J. Rexford, "Ravana: Controller fault-tolerance in software-defined networking," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, ser. SOSR '15. New York, NY, USA: ACM, 2015, pp. 4:1–4:12. [Online]. Available: http://doi.acm.org/10.1145/2774993.2774996

[7] B. Chandrasekaran and T. Benson, "Tolerating sdn application failures with legosdn," in *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, ser. HotNets-XIII. New York, NY, USA: ACM, 2014, pp. 22:1–22:7. [Online]. Available: http://doi.acm.org/10.1145/2670518.2673880

[8] V. G. da Costa, S. Barbon, R. S. Miani, J. J. Rodrigues, and B. B. Zarpelão, "Detecting mobile botnets through machine learning and system calls analysis," in *2017 IEEE International Conference on Communications (ICC)*. IEEE, 2017, pp. 1–6.

[9] P. Vinod, A. Zemmari, and M. Conti, "A machine learning based approach to detect malicious android apps using discriminant system calls," *Future Generation Computer Systems*, vol. 94, pp. 333–350, 2019.

[10] R. Canzanese, S. Mancoridis, and M. Kam, "System call-based detection of malicious processes," in *2015 IEEE International Conference on Software Quality, Reliability and Security*. IEEE, 2015, pp. 119–124.

[11] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," in *Australasian Joint Conference on Artificial Intelligence*. Springer, 2016, pp. 137–149.

[12] I. Rosenberg, "Advanced project in computer science: System-calls based dynamic analysis intrusion detection system with configurable machine-learning classifier," The Open University of Israel, Tech. Rep.

[13] C. Mansour and D. Chasaki, "Design of an SDN security mechanism to detect malicious activities," in *Tenth International Conference on Ubiquitous and Future Networks, ICUFN 2018, Prague, Czech Republic, July 3-6, 2018*, 2018, pp. 186–190. [Online]. Available: https://doi.org/10.1109/ICUFN.2018.8436794

[14] V. Chaykovsky. strace linux syscall tracer. https://strace.io/. Accessed: 4-30-2019.

[15] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *J. Artif. Int. Res.*, vol. 16, no. 1, pp. 321–357, Jun. 2002. [Online]. Available: http://dl.acm.org/citation.cfm?id=1622407.1622416

[16] Z.-H. Zhou and M. Li, "Tri-training: Exploiting unlabeled data using three classifiers," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 17, pp. 1529– 1541, 12 2005.

[17] T. Wood, “Link analysis and crime - an examination,” 2015. [Online]. Available: https://crimetechweekly.com/2015/12/17/link-analysis-and-fraud/

[18] G. Haixiang, L. Yijing, J. Shang, G. Mingyun, H. Yuanyue, and G. Bing, “Learning from class-imbalanced data,” *Expert Syst. Appl.*, vol. 73, no. C, pp. 220–239, May 2017. [Online]. Available: https://doi.org/10.1016/j.eswa.2016.12.035