COMPUTERS & SECURITY 82 (2019) 261–269

Available online at www.sciencedirect.com

ScienceDirect

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

Computers & Security

An ensemble learning approach for XSS attack detection with domain knowledge and threat intelligence

*Yun Zhou**, *Peichao Wang*

Science and Technology on Information Systems Engineering Laboratory, National University of Defense Technology, Deya Road 109, Kaifu District, Changsha, China

ARTICLE INFO

Article history:
Received 25 June 2018
Revised 26 October 2018
Accepted 31 December 2018
Available online 11 January 2019

ABSTRACT

Cross-site scripting (XSS) attack is one of the most dangerous attacks for web security. Traditional XSS detection methods mainly focus on the vulnerability itself, relying on static analysis and dynamic analysis, which appear weak in defending the flood of various kinds of payloads. In this paper, the XSS attack detection method is proposed based on an ensemble learning approach which utilizes a set of Bayesian networks, and each Bayesian network is built with both domain knowledge and threat intelligence. Besides, an analysis method is proposed to further explain the results, which sorts nodes in the Bayesian network according to their influences on the output node. The results are explainable to the end users. To validate the proposed method, experiments are performed on a real-world dataset about the XSS attack. The results show the priority of the proposed method, especially when the number of attacks increases. Moreover, the node sorting results could help the security team to cope with the attack in time.

1. Introduction

In the last two decades, modern society's dependence on web applications has increased tremendously, while cyber-attacks have been increasing significantly worldwide at the same time (Nikolaos et al., 2017). As one of the most serious web attacks that is an important part of the cyber-attack, XSS attack has caused enormous damage to economics and individual privacy (José et al., 2014).

XSS is the abbreviation of Cross-Site Scripting, while its name is used to distinguish from CSS (Cascade Style Sheets) and it is a common vulnerability in web applications; it allows attackers to inject malicious codes into web pages, and victims view the pages or click evil links will trigger the malicious scripts. XSS has been listed in OWASP (Open Web Application Security Project) as the top 10 application security risks for many times.[1] The combination of XSS and other web attacks will threaten the security of the web users, causing the steal of confidential information and user sessions.

Normally, XSS vulnerabilities are classified into three types: reflected XSS, stored XSS and DOM-based XSS.[2] Reflected XSS is the most common type of XSS: in the exploit of it, malicious scripts are usually written in the URL and an attacker will lure the victim to click it. Stored XSS is less common, but its damage is usually greater than the other two kinds. In a stored XSS attack, the attacker injects code into input data and the data is stored in the database; when creating a webpage that references the certain data, the vulnerability is exploited and

[1] https://www.owasp.org/index.php/Top_10-2017_Top_10
[2] https://www.owasp.org/index.php/Types_of_Cross-Site_Scripting

* Corresponding author.
E-mail address: zhouyun@nudt.edu.cn (Y. Zhou).
https://doi.org/10.1016/j.cose.2018.12.016

everyone who views the webpage is attacked. DOM-based XSS could be regarded as a special type of reflected XSS; the malicious script might alter the Document Object Model (DOM) and do harm to the web user.

To defend XSS, two popular ways - static analysis and dynamic analysis (Guo et al., 2015) - are applied for detecting XSS vulnerabilities in time. Static analysis analyzes the source codes without running the application, which requires enormous security domain knowledge. Dynamic analysis analyzes the information obtained during program execution to detect vulnerabilities. This method relies on the completeness of attack vectors and may easily result in a high rate of Type II error. Besides these two methods, data-driven analysis is a popular and recently developed method for cyber security analysis, which analyzes XSS payloads rather than application vulnerabilities.

Recently, machine learning techniques are widely used to effectively detect XSS attack (Dai et al., 2017). However, many works relying on black-box models that cannot explain the result explicitly. For example, SVM (Kevin, 2012) aimed to find the best hyperplane with the maximal geometric margin, which do not provide further explanations of probabilistic relationships between model features.

Moreover, few models consider the available domain threat intelligence, which is the evidence-based knowledge representing threats that can provide support for decisions (Wiemm and Helmi, 2018). For example, malicious IPs could be used to improve the attack detection performances. Threat intelligence is a valuable asset for enterprises (Eric et al., 2014) which consists of information from multiple sources and usually contains descriptions of new types of attacks.

In this paper, we propose a new XSS attack detection method based on an ensemble learning approach (Cortes et al., 2014), whose individual classifiers are Bayesian networks (BNs) (Zhou et al., 2014) learnt with domain knowledge and threat intelligence. Ensemble methods are general techniques in machine learning for combining several individual learners to create a more accurate classifier, which could increase generalization ability of the model.

In this paper, the XSS attack detection problem is modelled as a binary classification problem and the basic learner used in the ensemble model is the BN model. The final classification result is calculated via the majority voting method. Moreover, to build the individual BN, we firstly extract features from the domain knowledge —the created XSS attack ontology, and then use the state-of-the-art BN structure learning algorithm to learn the BN structure and parameters of the model. The collected threat intelligence is also integrated here to improve the model learning accuracy. Besides, we propose a model explanation method by evaluating nodes' importance in the constructed BNs to help find the important factors of detecting the attack. The experiments based on a real-world XSS dataset illustrate the effectiveness of the proposed method.

2. Related works

Detection of web attack is always a hot topic in the security field. Traditional detection methods mainly rely on IDS (Intrusion Detection System), which is designed to identify either hostile activities or policy violation activities within a network (Muhammad et al., 2017). However, facing attacks from the application layer such as XSS, traditional methods are usually ineffective or unavailable. Confronting XSS, popular methods mainly aim at detecting vulnerabilities in applications rather than discovering potential attacks from malicious users, and are divided into two types: static analysis method and dynamic analysis method (Philipp et al., 2007).

Static analysis is a manual analysis method that cyber security experts mine possible vulnerabilities directly from the source codes, like lexical analysis and data flow analysis. People have developed many analysis tools and algorithms in this field. Nenad et al. (2006) raised an open source tool for the detection of XSS in PHP by means of data flow analysis. Lwin and Hee (2012) proposed a two-phase XSS static analysis method where one phase was aimed to locate potential XSS vulnerabilities and the other was designed to escape them. Agosta et al. (2012) combined static taint analysis with symbolic code in the detection of security vulnerabilities and made a tool to perform their methodology. However, the weakness of static analysis is clear: it consumes a large amount of manual efforts and the source codes are usually not accessible.

Dynamic analysis is another manual analysis method that focuses on testing penetration tests in a certain application like a real attacker. It tries different payloads in the applications' possible injection points and does further analysis based on responses. The completeness of attack vectors plays a critical role in this approach, and there are also many works in this field. Wang et al. (2017) raised a dynamic attack vector generation method based on the Hidden Markov Model (HMM). The attack vector is acquired from the HMM and is evaluated by its similarities with normal requests. Fabien et al. (2014) proposed an XSS fuzzer called KameleonFuzz which has the ability to generate malicious inputs to evaluate. And this black-box scanner is used to detect XSS vulnerabilities in web applications and the attack vectors are generated in the learning of taint flow in the application. Moataz and Fakhreldin (2016) proposed a genetic algorithm based XSS attack payload generator which used a fitness function composed of many blocks designed on their own. The experiment on web applications developed by PHP and MySQL shows the effectiveness of the method in the generation of payloads.

Nowadays, traditional methods like static analysis and dynamic analysis are not effective due to the increasing forms of XSS payloads. To address this challenge, machine learning algorithms have been introduced in the detection of XSS and achieve good results. Krishnaveni and Sathiyakumari (2013) selected four kinds of features of a web page, and used Naive Bayes, Decision Tree and Multi-Layer Perceptron to classify normal pages and malicious pages. Rathore et al. (2017) proposed an XSS detection method aiming at the discovery of XSS in SNS, which abstracts certain features of SNS to improve the detection performance. However, most current works are block-box models and lack of the explanations of the results. BN is a good white-box model in machine learning field, and its result is explainable for end users. It can combine different sources of knowledge, and can be built with small and incomplete datasets (Sabarathinam et al., 2017), thus the BN is an increasingly popular model in cyber security commu-

nity (Chockalingam et al., 2017; Wu et al., 2013; Axelrad et al., 2013).

The most challenging part of utilizing BN is the model construction, which is usually expensive and needs to combine both expert knowledge and empirical data as the input. BN learning includes structure and parameter learning. Bayesian parameter learning is getting variable's Conditional Probability Tables (CPTs) based on a known network structure. Structure learning of BN is the primary mission of BN learning; in this case, the structures and parameters are both unknown and will be getting through existing data. There are three kinds of structure learning algorithms: constraint based learning algorithm, scoring and searching learning algorithm and hybrid learning algorithm. As discussed in previous works (Zhou et al., 2016), pure data-driven BN structure learning requires a large amount of high quality data. Thus, domain knowledge i.e. security ontology and rules are needed to support the BN construction (Stefan, 2012).

Recently, another kind of domain knowledge - open source threat intelligence is introduced as the complement of the model construction (Alper et al., 2011). Threat intelligence is the sum of security clues obtained from multiple channels to protect the core assets of the system. CVSS (Common Vulnerability Scoring System) score is a popular type of open source threat intelligence, which is a standard for specifying vulnerability attributes. It has many sub-scores measuring a vulnerability. Muñozgonzález et al. (2017) combined them in different ways to formulate their own evaluation metric in risk evaluation. Peng et al. (2010) transformed metric values into the probability values in BN's CPTs. However, except the vulnerability database, there still are many threat intelligence sources such as online malicious IP databases and malicious domain name databases that are not used. Thus, in this paper, we propose a new model to incorporate these threat intelligence resources to help detect the XSS attack.

3. Model formulation

3.1. *Illustration of model's implementation process*

Fig. 1 illustrates the framework of the proposed method for the detection of XSS. Firstly, the ontology is built for the XSS attack process, and knowledge is abstracted from it to create features that could represent the characteristic of the XSS attack. Then these features are set as nodes in each individual BN and values are extracted from raw data as the source of Bayesian learning. The learning algorithm applied in this paper is scoring and searching learning algorithm. Because each BN is an individual learner in the ensemble model, a voting method is applied here to ensemble the individual model to create the ensemble learner.

After the creation of the ensemble learner, the threat intelligence is figured out to improve the performance of the proposed model and collect them. The collected intelligence is utilized to create complement rules to face the concealing of the XSS attack. In practice, when the new data is input into the model, it will be detected by ensemble learner and complement rules. And the node importance sorting based on learnt ensemble BN learner is performed to figure out these

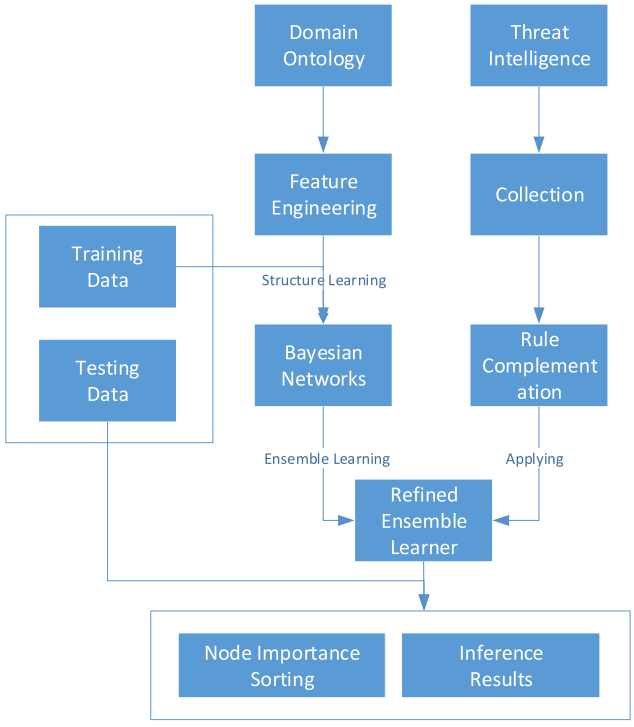

Fig. 1 – The framework of the proposed method.

nodes' influences on the final detection result. BN is a white-box model which means its result is explainable for users, and complement rules originating from threat intelligence could figure out the hidden attack in time; the combination of BN and rules from threat intelligence could help security staff cope with incidents in time.

3.2. *Model creation based on domain knowledge*

Ontology is an important expression of domain knowledge. The classical concept of ontology in the computer field is provided by Gruber (1993), where an ontology is a description of the formally existed concepts and provides a knowledge overview for a certain domain (Gruber, 1995). In this paper, the ontology model acts as a knowledge base and can help to capture the core concepts covering what an XSS could cause and its expression in its payload. Constructing an ontology by certain modeling language directly is a time-consuming work; in practice, an ontology editor with GUI is helpful for us to concentrate our efforts on the constructing of ontology itself rather than the modeling language. In this paper, Protégé is chosen as the modelling tool. Protégé[3] is a free and open-source framework to help researchers construct their own ontology. It is developed by Stanford Center for Biomedical Informatics Research through Java, and OWL (Web Ontology Language) is used as its modelling language. The high-level ontology of XSS attack used in this paper is shown in Fig. 2.

Specifically, an XSS payload may different from normal requests or inputs in four parts: an abnormal input length, containing sensitive words, sensitive characters and a redirection

[3] https://protege.stanford.edu/

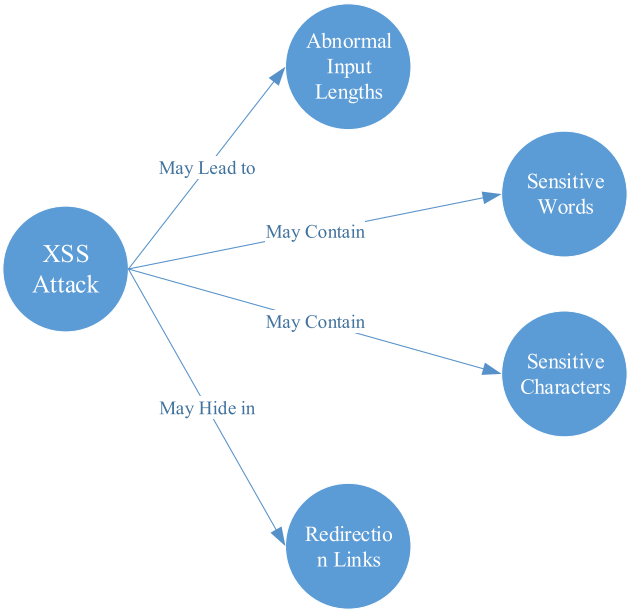

Fig. 2 – High-level ontology of XSS attack.

link. An XSS payload is usually longer than the normal one, because malicious codes are inside. Thus, the input length is extracted as one feature. Sensitive words and sensitive characters are significant factors to identify an XSS. There are many words and characters which may appear in one payload, such as *alert, eval, javascript*, < , and so on. These words and characters combine together to create one malicious payload, and their appearances can be used as features in the machine learning model. Besides, an XSS payload may utilize the redirection link to mask its original form, hiding the malicious codes in the redirection pages. In order to redirect the current page to another page, some protocols may appear in one payload, such as *http* and *https*. The appearance times of these protocols are also counted as features and the redirection address is extracted for further analysis.

In the individual BN model, abstracted features are regarded as nodes, and values are learnt from the raw data. A BN is a directed acyclic graphical model where the nodes represent random variables, and the directed edges represent the dependencies between random variables (Zhou et al., 2016). Let $\boldsymbol{X} = \{X_1, \ldots, X_n\}$ be a set of (continuous or discrete) random variables where each X_i may take finite values. A BN is a pair (S, **P**) where S stands for network structure that encodes a set of conditional independence assertions about variables in **X**, and **P** is a set of local probability distribution associated with each variable. $\boldsymbol{P} = \{P_i\}$ and $P_i = P(X_i|\pi(X_i))$, where $\pi(X_i)$ denotes the parents of node X_i in **X**. The **X**'s joint probability distribution which satisfies the local Markov property with respect to S can be written as:

$$P(\boldsymbol{X}) = \prod_i P(X_i|\pi(X_i)) \tag{1}$$

For a BN with discrete variables, the probability distribution is described by a CPT that contains the probability of each value of the variable given its parents' possible values (Zhou et al., 2015). After getting features and values, Bayesian structure learning is used to form the model for further detection. In this paper, we use the popular scoring and searching learning algorithm, where score functions are used to evaluate the fitness between data and structure while searching algorithms are used to seek the structures with the highest scores in candidate sets. To be specific, BDeu (Heckerman et al., 1994) is chosen as the scoring function and Tabu search (Glover, 1990) is chosen as the searching algorithm.

Tabu search is a popular searching strategy for optimal solutions in the optimization problem. In order to avoid local optimization, it applies a tabu list in its searching process to deny the repeated calculation of the algorithm, which makes it more likely to get a globally optimal solution. It is widely used in lots of fields and has many variant forms. BDeu (Bayesian Dirichlet equivalent uniform) score aims at maximizing the posterior probability of the DAG given data, while assuming a uniform prior over possible DAGs (Mauro et al., 2014). It is a widely used score function deriving from BDe and BDu (Heckerman et al., 1994).

In this paper, ensemble learning is applied to improve the accuracy of the proposed method, which utilises multiple single learners to achieve the learning aim. Specifically, bagging and majority voting methods are utilized. Bagging is a commonly used ensemble learning approach which is based on the bootstrap sampling of the original training dataset **D**. And the majority voting method is a popular way to combine the predicting results to get a final classification result. The ensemble BN learning process in this paper is shown Algorithm 1.

In practice, for a training dataset **D**, *T* times of bootstrap sampling is performed; each time one sub training dataset $\boldsymbol{D}^t$ is sampled, thus *T* sub training datasets are obtained. For each $\boldsymbol{D}^t$, the Tabu search algorithm and BDeu score are applied in the structure learning to get the BN^t. Then, *T* individual BNs are built. For the final prediction result, majority voting method is utilized. For one record R_k, if more than half of all individual BNs predict the target variable Ta_k as the same label 1 or 0, the ensemble classifier C_{BN} will output this label as the final result:

$$Ta_k = \begin{cases} 1 & if\ count(Ta_k^t = 1) > count(Ta_k^t = 0) \\ 0 & if\ count(Ta_k^t = 0) > count(Ta_k^t = 1) \end{cases} \tag{2}$$

In this Eq. (1) denotes the malicious record and 0 denotes the normal record; *count*(*f*) is the occurrences of the condition *f* happens in all *T* BNs.

3.3. *Model improvement based on cyber threat intelligence*

In practice, model just relying on data-driven machine learning and ignoring the threat intelligence might give the chance for attackers to achieve their goal. Cyber threat intelligence (CTI) is an important complement to cyber security. The UK's Centre for the Protection of National Infrastructure (CPNI) lists four types[4] of threat intelligence: tactical, technical, operational and strategic. In practice, most threat intelligence

[4] https://en.wikipedia.org/wiki/Cyber_threat_intelligence

Algorithm 1 – Ensemble BN learning algorithm.

Input: training dataset **D**, individual classifier number T
Output: the ensemble classifier C_{BN}
Begin
For $1, 2, \ldots, T$ do:
Use bootstrap sampling to sample data from **D** to get sub training dataset $\mathbf{D}^t$;
Use Bayesian structure learning algorithm to acquire individual BN^t through sub dataset;
End for
Use the majority voting method to combine all individual classifiers as the ensemble classifier C_{BN}.
End

is held by profit-making organizations. Luckily, there are four main types of open-source threat intelligence for research purposes: malicious IP addresses, malicious domain names, vulnerability databases and open-source information databases (such as blogs and forums). In the detection of XSS, the open-source threat intelligence which can be utilized automatically are malicious IP addresses and malicious domain names.

Malicious IP addresses may be the most accessible threat intelligence for research. These IP addresses are used in the blacklist of the web browser and the browser could block the request from these addresses. There are many websites providing the malicious IP addresses for the public, while the base maintained by FireHOL[5] maybe the best choice for research. This website contains malicious IP addresses from varied sources and updates its contents frequently.

Malicious domain names are helpful in the defending of botnets, phishing pages and malwares. When a user tries to visit a webpage whose address is listed in the malicious domain name list, the visit should be prohibited for the sake of user's security. Websites such as DNS-BH project[6] and PhishTank[7] make their resources accessible to the public, and these domain names can be added into the blacklist of the browser.

Open-source threat intelligence forms the complement rules of the model to improve its performance. Let $\mathbf{C} = \{C_1, \ldots, C_m\}$ be the complement rules, $\mathbf{R} = \{R_1, \ldots, R_M\}$ be the records. $\mathbf{Ta} = \{Ta_1, \ldots, Ta_M\}$ denotes the prediction results through the ensemble learner, $\mathbf{Ta'} = \{Ta'_1, \ldots, Ta'_M\}$ denotes the prediction results through both ensemble learner and complement rules. The value of Ta'_k could be calculated through equation the below:

$$Ta'_k = \begin{cases} 1 & if \max\{I_C(R_k), Ta_k\} = 1 \\ 0 & if \max\{I_C(R_k), Ta_k\} = 0 \end{cases} \quad (3)$$

In the equation, 0 denotes the normal record while 1 denotes the malicious record; $I_C(R_k)$ is an indicator function that indicates whether the extracted value meet the complement rules or not. For instance, if there is malicious redirection in R_k, which match the rules in **C**, the $I_C(R_k)$ equals to 1. A detailed example is discussed in Section 4.2.

If $I_C(R_k)$ equals 1, the final prediction type of the record is malicious; if $I_C(R_k)$ equals 0 and $Ta_k = 1$, which means the ensemble learner predict the current record as the malicious, and the final result is malicious; if $I_C(R_k)$ equals 0 and $Ta_k = 0$, which means the current record contains no malicious redirections and is a normal record through ensemble learner's prediction, the result indicates the current record is normal. When the detection result is provided, diagnosis inference in one individual BN could be further used to infer the key factors of the attack.

3.4. *Node importance sorting*

After building the refined ensemble BN learner, the inference is performed in each individual BN, which is the process of computing the updated distribution of variables of interest, given that other variables are set to certain values. The computational complexity of performing exact inference in a BN is known to be NP-hard (Gregory, 1990). However, in the late 1980s a major breakthrough was achieved with the development of exacting inference algorithms that computed efficiently for a large class of real-world BNs. The most commonly used of these exacting inference algorithms for discrete BNs is the junction tree algorithm (Lauritzen and Spiegelhalter, 1988), which propagates the evidences along the moralized and triangulated graph.

To reduce the inference computational complexity in this paper, we only consider nodes within the label node's Markov blanket in each individual BN, whose equation is shown below:

$$Mb(X) = \pi(X) \cup ch(X) \cup \left(\cup_{Y \in ch(X)} \pi(Y)\right) \quad (4)$$

In this equation, node X is one node in the BN, and $\pi(X)$ denotes X's parent nodes, and $ch(X)$ denotes X's child nodes, and $\cup_{Y \in ch(X)} \pi(Y)$ denotes parent nodes of X's child nodes. Once the node's Markov blanket is given, this node is conditionally independent from the nodes outside the blanket, thus further analysis is performed just for nodes in the blankets with standard junction tree algorithm. Specifically, the interested node is queried after observing the nodes in the Markov blanket. The inference result is a posterior probabilistic distribution of the label node.

The posterior distribution of the label node varied in different evidence. In inference, we are interested in how other nodes affect the target node. Thus, posterior distributions of the target node under different states of an observing node are calculated. In this paper, the target node denotes the node that could indicate the type of current record, and in practice it is usually the label node in a BN.

[5] https://github.com/firehol/blocklist-ipsets
[6] http://www.malwaredomains.com/
[7] https://www.phishtank.com

Algorithm 2 – The node importance sorting algorithm of the ensemble model.

Input: nodes' possible states in each BN
Output: node importance sorting list
Begin
For each individual $BN^t(t = 1, 2, \ldots, T)$:
For X_i^t in $\mathbf{X^t}$:
For each state in X_i^t's r_i^t possible states:
Calculate posterior probability distributions based on the state through the junction tree algorithm;
Calculate all *Dis* of Ta^t's posterior probability distributions under different X_i^t's observations.
Select the largest *Dis* in above distances as $Dis_{max}^{t,i}$;
Sort $Dis_{max}^{t,i}$ of different X_i^t in the descending order;
For the sorting list in BN^t:
Assign each node's weight $w_{t,i}$;
For each node:
Calculate its final weight $W_i = \sum_{t=1}^{T} w_{t,i}$;
Sort the nodes in an ascending order based on the node's W_i.
End

Considering the target node Ta^t in BN^t and its feature node $\mathbf{X^t} = \{X_1^t, \ldots, X_i^t, \ldots, X_N^t\}$, node X_i^t has r_i^t possible states to be observed. Nodes' importance is sorted based on the Algorithm 2, which is an improved version of the previous work (Wang et al., 2018):

In Algorithm 2, every time we change the evidence of one feature node X_i^t and fix the other nodes. Each X_i^t has r_i^t possible evidences, thus Ta^t has r_i^t posterior probability distributions about the detection results.

To quantify the X_i^t's importance/influence on Ta^t, the absolute distances of Ta^t's posterior probabilities are calculated, then there are $C_{r_i^t}^2 = \frac{r_i^t(r_i^t-1)}{2}$ distances.

For a pair of posterior probabilities $P(Ta^t = malicious|X_i^t = e)$ and $P(Ta^t = malicious|X_i^t = e')$ (P_1 and P_2 for short), their distances can be obtained through the equation below:

$$Dis(P_1, P_2) = |P_1 - P_2| \tag{5}$$

The maximum *Dis* in $C_{r_i^t}^2 = \frac{r_i^t(r_i^t-1)}{2}$ distances of BN^t denotes X_i^t's influence on the target variable Ta^t, named as $Dis_{max}^{t,i}$ in this paper. After getting $Dis_{max}^{t,i}$ of all feature nodes in each individual BNs, $Dis_{max}^{t,1}, Dis_{max}^{t,2}, \ldots, Dis_{max}^{t,N}$ are sorted in a descending order as the sorting list in BN^t, which means the feature node who has a greater influence on Ta^t has a smaller index in the sorting list.

Because we have several BNs in one ensemble model, here we utilize weighted voting method to get the final sorting list, in which the weight of each node is its sorting index in individual BN. Suppose there are T individual BNs; after preliminary calculation, T sorting lists are obtained. In each list, the node's index is assigned as its weight, thus each node gets T weights. And $w_{t,i}(t = 1, 2, \ldots, T)$ denotes node X_i^t's weight in the t-th sorting list.

Then the node's final weight can be obtained by summing its weights in all lists:

$$W_i = \sum_{t=1}^{T} w_{t,i} \tag{6}$$

The resulting final weight of each node $W_1, W_2, \ldots, W_N$ is sorted in the ascending order, which means node X_i with a smaller value of W_i has a greater influence on the final result of the target variable Ta. This sorting result could help network administrators in further investigation.

4. Experiments and discussions

4.1. Experimental setting

Our experimental environment is built with Python 3.6 and SPSS 24.0. XSS payloads[8] are collected from GitHub as the training dataset. There are 151,658 records, consisting of 16,151 XSS payloads and 135,507 normal records. Although they are different in trigger process, storage position and output position, their payloads have common characteristics, such as consisting of the same sensitive words and characters.

The testing dataset in our experiment is gathered from multiple GitHub sources and security forums, which consists of 6503 normal records and 3497 XSS payloads.

Then, the features of an XSS attack are created according to the ontology discussed in Section 3.2. After the refinement of ontology's concepts, 30 features are obtained. these features are numbered from No. 0 to No. 29, and node No. 30 is the type of record (XSS attack or normal one). The details of 30 features used in this paper are shown in Table 1.

4.2. Model learning with our method

Since features are obtained, this paper sets these features as nodes in BN. For the reason that an attacker may change the encoding method or case of the input, the extracted parameters are standardized into lower cases and the URL encoding, HTML encoding, JavaScript encoding, ASCII encoding, Unicode encoding and twice URL encoding are transformed into their original codes. Then values are extracted from processed

[8] https://github.com/duoergun0729/1book/tree/master/data

Table 1 – Features used in the experiments.

Index	Meaning	Index	Meaning	Index	Meaning
0	Length of input	10	Number of *href*	20	Number of *iframe*
1	Number of *alert*	11	Number of *javascript*	21	Number of *onclick*
2	Number of *script*	12	Number of *window*	22	Number of single quote mark (')
3	Number of *onerror*	13	Number of *fromcharcode*	23	Number of double quotation marks (")
4	Number of *confirm*	14	Number of *document*	24	Number of left angle brackets (<)
5	Number of *img*	15	Number of *onmouseover*	25	Number of right angle brackets (>)
6	Number of *onload*	16	Number of *cookie*	26	Number of back-slant (\)
7	Number of *eval*	17	Number of *domain*	27	Number of comma (,)
8	Number of *prompt*	18	Number of *onfocus*	28	Number of plus (+)
9	Number of *src*	19	Number of *expression*	29	Number of *http://*, *https://* and *file://*

Table 2 – Examples of normal requests and malicious redirection requests.

Normal	Malicious
<A HREF="http://146.112.52.126/"">Click<A>	<A HREF="http://34.0146.0 × 7.142/">Click</A>
<SCRIPT SRC=https://www.163.com/></SCRIPT>	<SCRIPT SRC=http://gsiworld.neostrada.pl/xss.js></SCRIPT>
<iframe src=https://www.taobao.com/>	<iframe src=http://sinopengelleriasma.com/index.html>

dataset to fit the proposed features. To decrease the computational complexity of learning and improve the model performance, entropy-based discretization method is applied to discrete values in datasets here. This kind of method is a supervised discretization method; in our experiment, SPSS 24.0 is used to complete this step.

After pre-processing, the algorithm is realized in Python package PyAgrum[9] to perform the BN structure learning. PyAgrum is a Python wrapper for the C ++ aGrUM, and includes many Bayesian algorithms which cover state-of-the-art structural learning, parameter learning and inference algorihtms. Here the Tabu search algorithm is applied in the package to perform the learning for each BN. The default scoring function is set as the BDeu score. In our experiment, the number of individual learners is 5. Besides data-driven learning, if there exists redirection in the records of the original dataset, we can extract these IPs or domain names and compare them with collected threat intelligence. The threat intelligence utilized here comes from two aspects: malicious IP addresses and malicious domain names. This information is collected from three resources: FireHOL, DNS-BH project and PhishTank.

Our threat intelligence mainly aims at the detection of redirection in the XSS payload. There are two ways to mark a resource in the network: IP or domain name. Domain name is a good way to locate one resource in the network, which is easy for the public to remember. An IP is another way to locate the resource, which is normally written in dotted decimal notation. Each part of the IP address could be written in different decimal forms, including binary notations, octal notations, decimal notations and sexadecimal notations; different forms could be used in an IP address simultaneously. Besides, one IP could be presented in a form of a large integer. Using different decimal forms in one IP is a good way to hide its malicious destination. For example, an IP address 10.0.3.193, it could be presented as 167,773,121, using the calculating process $10 \times 256^3 + 0 \times 256^2 + 3 \times 256 + 193 = 167,773,121$. To change the long integer into normal dotted decimal notation, it is rewritten into 32-bit binary notation, divided into 4 parts where each part containing an 8-bit figure, and the binary notation is restored into normal dotted decimal notation. Some examples of normal and transformed requests are listed in Table 2.

The complement rule set **C** in the experiment is the combination of varied malicious IP address and domain names. If record R_k has malicious redirections listed in **C,** the current record must be malicious. For instance, there is one record <A HREF ="http://12.16.123.147/"> Click < /A > , if the redirection address 12.16.123.147 is listed as malicious IP in the rule set **C**, the record is labelled as malicious.

4.3. *Performance and result explanation with node importance sorting*

To compare the proposed method to other popular classifiers, popular classifiers are chosen to do the classification. These classifiers include SVM, Naïve Bayes, Logistics Regression, Decision Tree and Random Forest. In the experiment, Python package scikit-learn[10] is applied to utilize the classifiers.

In the experiment, malicious domain names and different forms of malicious IP from threat intelligence are used to replace the redirection links in a payload or a normal request URL. Of course, these creations are malicious. With the help of the open-source threat intelligence, the proposed method could detect these new payloads. In the experiment, we randomly choose 0% (Original), 5%, 10%, 15%, 20%, 25%, 30%, 35%, 40% and 45% records in the testing dataset and replace them with the malicious creations separately. Then the performances of different classifiers are tested and the experiments are repeated for 10 times. The result is shown in Table 3.

As can be seen from the result, the proposed method's performance is not as good as others in original situations, but its performance improves with the malicious creations being

[9] http://agrum.gitlab.io/

[10] http://scikit-learn.org/stable/

Table 3 – Average classification accuracies of different methods under different percentages of randomly sampled malicious records in the dataset.

Malicious records percentage	Our method	SVM	Naïve Bayes	Logistics regression	Decision tree	Random forest
Original	96.96%	97.76%	**99.23%**	97.89%	97.23%	97.76%
5%	**97.59%**	95.43%	96.30%	95.55%	94.24%	95.45%
10%	**98.06%**	93.04%	93.37%	93.13%	91.16%	93.02%
15%	**97.89%**	90.64%	90.36%	90.68%	88.06%	90.58%
20%	**97.64%**	88.19%	87.41%	88.24%	84.90%	88.14%
25%	**97.78%**	85.76%	84.44%	85.82%	81.85%	85.76%
30%	**97.63%**	83.39%	81.52%	83.36%	78.64%	83.30%
35%	**97.88%**	80.98%	78.53%	80.90%	75.54%	80.86%
40%	**98.22%**	78.45%	75.54%	78.54%	72.46%	78.43%
45%	**98.54%**	76.16%	72.62%	76.10%	69.28%	76.02%

added into testing dataset, which shows the effectiveness in the detection of potential attack hidden in redirection links. Specifically, the proposed method outperforms the other classifiers when the ratio of generated malicious records is 5%. Moreover, the proposed method achieves 98.54% accuracy at the worst setting, which the malicious records occupy the largest ratio of all testing settings.

The reason why the proposed method outperforms others is clear that the transformed malicious addresses could avoid disguises such as sensitive words and characters, and the transformed IP addresses could make the payload shorter than normal payload. With the help of incorporated threat intelligence, we could detect and cope with these payloads in time.

To further explain the results of the proposed method, node importance inference is performed on the ensemble BN model to see how the features in Table 1 affect the label node (the output node). Thus, the difference of posterior distributions of the label node is calculated under different states of an observing node with the Eqs. (5) and (6).

As discussed in Section 3.4, large difference corresponds to large importance and smaller sorting number in the list. In this experiment, the final node importance sorting result calculated by the proposed Algorithm 2 is listed as below:

0 > 1 > 24 > 27 > 23 > 22 > 8 > 26 > 10 > 3 > 9
>2 > 5 > 15 > 29 > 25 > 12 > 11 > 18 > 6 > 7 > 4
>20 > 13 > 21 > 19 > 16 > 14 > 17

The first three nodes of the result are adopted for further analysis. As we can see, node 0 ("length of input") ranks the highest in the sorting list, which agrees with the practice that the length of the input of malicious record is usually much longer than normal. For example, many XSS attacks start with the scanning using automatic testing tools such as AWVS (Acunetix Web Vulnerability Scanner) and Netsparker, where payloads generated by these tools are usually strange and long.

The second important node is the node 1 (number of "alert"). Here, the sensitive character "alert" is a common form used in real XSS tests. *<script>alert(/XSS/)</script>* is the most common and typical XSS testing payload for an attacker.

The third important node is the node 24, which is the number of left angle brackets (<). In practice, the number of left angle brackets is widely used in the payload to start useful scripts, such as *<img src = x onerror = alert(/XSS/) >*. Besides, the reuse of brackets may avoid the elimination of sensitive characters by XSS filters to some extent.

In conclusion, it can be seen that the sorting result provides a meaningful ranking about important factors in real XSS attacks. In practice, when a real attack is detected, the security staff could find the possible causes according to the sorting list, which could save time in further investigation.

5. Conclusions

In this paper, we proposed an XSS attack detection method based on an ensemble learning approach learnt with domain knowledge and threat intelligence.

We acquired knowledge from ontology to abstract features of the XSS, and used the state-of-the-art structure learning algorithm to get the structure of the individual BN. To increase the generalization of our method, we used the bagging and majority voting method to create one ensemble model with multiple learnt BNs. To further explain the detection result, we proposed a node importance sorting method under the multiple BNs setting.

In our experiments, we collected open-source threat intelligence and used them in a real word XSS attack detection dataset. To simulate real attack scenarios, we inserted different percentages of sampled malicious records in the dataset. The results showed our method won the other methods in most cases, and achieved 98.54% accuracy even at the worst case. Moreover, the calculated node importance sorting results also make sense, and could be used in further (human) investigation.

In future work, we will test this method in many more datasets and practical scenarios. Moreover, we will integrate the proposed method and its outputs in a real web security risk assessment system. The system will both consider outside web attacks such as XSS and inside weakness such as vulnerabilities, and provide an overall evaluation score based on these two aspects.

Acknowledgments

YZ and PW contributed equally to this paper. This work was supported by National Natural Science Foundation of China

(grant no. 61703416) and Natural Science Foundation of Hunan Province, China (grant no. 2018JJ3614).

REFERENCES

Agosta G, Barenghi A, Parata A, et al. Automated security analysis of dynamic web applications through symbolic code execution. In: Ninth International Conference on Information Technology: New Generations; 2012. p. 189–94 April.

Alper C, Mike T, Dan D, et al. Behavioral analysis of botnets for threat intelligence. Info Syst e-Business Manage 2011;10(4):491–519.

Axelrad ET, Sticha PJ, Brdiczka O, et al. A bayesian network model for predicting insider threats. In: 2013 IEEE Symposium on Security and Privacy Workshops; 2013. p. 82–9 May.

Chockalingam S, Pieters W, Teixeira A, et al. Bayesian network models in cyber security: a systematic review. In: Nordic Conference on Secure IT Systems; 2017. p. 105–22 November.

Cortes C, Kuznetsov V, Mohri M. Ensemble methods for structured prediction. In: International Conference on Machine Learning; 2014. p. 1134–42.

Dai H, Li J, Lu X, et al. Machine learning algorithm for intelligent detection of webshell. Chin J Netw Info Secur 2017;3:51–7.

Eric WB, Michael DG, Panos K, et al. Taxonomy model for cyber threat intelligence information exchange technologies. In: Proceedings of the 2014 ACM Workshop on Information Sharing & Collaborative Security; 2014. p. 51–60 November.

Fabien D, Sanjay R, Jean-Luc R, et al. KameleonFuzz: evolutionary fuzzing for black-box xss detection. In: Proceedings of the 4th ACM Conference on Data and Application Security and Privacy; 2014. p. 37–48 March.

Glover F. Tabu search: a tutorial. Interfaces 1990;20(4):74–94.

Gregory FC. The computational complexity of probabilistic inference using Bayesian belief networks. Artif Intell 1990;42(2):393–405.

Gruber TR. A translation approach to portable ontology specification. Knowl Acquis 1993;5(2):199–220.

Gruber TR. Toward principles for the design of ontologies used for knowledge sharing. Int J Hum Comput Stud 1995;43(5–6):907–28.

Guo X, Jin S, Zhang Y. XSS vulnerability detection using optimized attack vector repertory. In: 2015 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery; 2015. p. 29–36 September.

Heckerman D, Geiger D, Chickering DM. Learning Bayesian networks: the combination of knowledge and statistical data. Mach Learn 1994;20(3):197–243.

José F, Nuno S, Marco V, et al. Analysis of field data on web security vulnerabilities. IEEE Trans Dependable Secure Comput 2014;11(2):89–100.

Kevin PM. Machine Learning: A Probabilistic Perspective. MIT Press; 2012.

Krishnaveni S, Sathiyakumari K. Multiclass classification of XSS web page attack using machine learning techniques. Int J Comput Appl 2013;74(12):36–40.

Lauritzen SL, Spiegelhalter DJ. Local computations with probabilities on graphical structures and their application to expert systems. J R Stat Soc 1988;50(2):157–224.

Lwin KS, Hee BKT. Automated removal of cross site scripting vulnerabilities in web applications. Info Softw Technol 2012;54(5):467–78.

Mauro S, Cassio PC, Marco Z. Min-BDeu and max-BDeu scores for learning Bayesian networks. In: Probabilistic Graphical Models 2014; 2014. p. 426–41.

Moataz AA, Fakhreldin A. Multiple-path testing for cross site scripting using genetic algorithms. J Syst Architect 2016;64:50–62.

Muñozgonzález L, Sgandurra D, Paudice A, et al. Efficient attack graph analysis through approximate inference. ACM Trans Privacy Security 2017;20(3):10.

Muhammad HK, Carsten M, Tim W, et al. A logitboost-based algorithm for detecting known and unknown web attacks. IEEE Access 2017;5:26190–200.

Nenad J, Christopher K, Engin K. Pixy: A static analysis tool for detecting web application vulnerabilities. In: 2006 IEEE Symposium on Security and Privacy; 2006. p. 258–63 May.

Nikolaos P, Elias P, Michalis P, et al. Recommender systems meeting security: from product recommendation to cyber-attack prediction. In: Communications in Computer and Information Science; 2017. p. 508–19 August.

Peng X, Li J, Xinming O, et al. Using bayesian networks for cyber security analysis. In: 2010 IEEE/IFIP International Conference on Dependable Systems & Networks; 2010. p. 211–20 June.

Philipp V, Florian N, Nenad J, et al. Cross-site scripting prevention with 'dynamic data tainting and static analysis. Proceedings of the Network and Distributed System Security Symposium, 2007.

Rathore S, Sharma PK, Park JH, et al. XSSClassifier: an efficient XSS attack detection approach based on machine learning classifier on SNSs. J Info Process Syst 2017;13(4):1014–28.

Sabarathinam C, Wolter P, Andre T, et al. Bayesian network models in cyber security: a systematic review. In: 20th Nordic Conference on Secure IT Systems; 2017. p. 105–22 November.

Stefan F. An ontology-based approach for constructing Bayesian networks. Data Knowl Eng 2012;73:73–88.

Wang D, Gu M, Zhao W. Cross-site script vulnerability penetration testing technology. J Harbin Eng Univ 2017;38(11):1769–74.

Wang P, Zhou Y, Zhu C, Tang L, Zhang W. In: Cyber security inference based on a two-level Bayesian network framework. Miyazaki, Japan: Seagaia Convention Center; 2018. p. 7–10 October.

Wiemm T, Helmi R. A survey on technical threat intelligence in the age of sophisticated cyber attacks. Comput Secur 2018;72:212–33.

Wu J, Yin L, Guo Y. Cyber attacks prediction model based on Bayesian network. In: IEEE International Conference on Parallel and Distributed Systems; 2013. p. 730–1 December.

Zhou Y, Fenton N, Neil M. Bayesian network approach to multinomial parameter learning using data and expert judgments. Int J Approx Reason 2014;55(5):1252–68.

Zhou Y, Fenton N, Zhu C. An empirical study of bayesian network parameter learning with monotonic influence constraints. Decis Support Syst 2016;87(C):69–79.

Zhou Y, Fenton N, Hospedales TM, et al. Probabilistic graphical models parameter learning with transferred prior and constraints. In: The Conference on Uncertainty in Artificial Intelligence; 2015. p. 972–81 July.

Yun Zhou received his Ph.D. degree in computer science from the Queen Mary, University of London in 2015. He is currently an Assistant Professor in Science and Technology on Information Systems Engineering Laboratory at the National University of Defense Technology, Changsha, China. His research focuses on Bayesian methods for prediction, risk management and decision making. He applies these techniques to a wide range of real-world problems, especially in web security applications. He has published several papers in reputed journals and conferences in this area, including INFOCOM, IJCAI, IJAR, UAI, and PGM.

Peichao Wang is a Master student in Science and Technology on Information Systems Engineering Laboratory at the National University of Defense Technology, Changsha, China. His research interests include web security analysis and machine learning.