Interactive Visual Decision Tree for Developing Detection Rules of Attacks on Web Applications

Tran Tri Dang, Tran Khanh Dang
Faculty of Computer Science and Engineering
Ho Chi Minh City University of Technology, VNU-HCM
Ho Chi Minh City, Viet Nam

Truong-Giang Nguyen Le
IT Security Division
EVN Finance Joint Stock Company
Ho Chi Minh City, Viet Nam

***Abstract*—Creating detection rules of attacks on web applications is not a trivial task, especially when the attacks are launched by experienced hackers. In such a situation, human expertise is essential to produce effective results. However, human users are easily overloaded by the huge input data, which is meant to be analyzed, learned from, and used to develop appropriate detection rules. To support human users in dealing with the information overload problem while developing detection rules of web application attacks, we propose a novel technique and tool called Interactive Visual Decision Tree (IVDT). IVDT is a variant of the popular decision tree learning technique introduced in research fields such as machine learning and data mining, with two additionally important features: visually supported data analysis and user-guided tree growing. Visually supported data analysis helps human users cope with high volume of training data while analyzing each node in the tree being built. On the other hand, user-guided tree growing allows human users to apply their own expertise and experience to create custom split condition for each tree node. A prototype implementation of IVDT is built and experimented to evaluate its effectiveness in terms of detection accuracy achieved by its users as well as ease of working with. The experiment results prove some advantages of IVDT over traditional decision tree learning method, but also point out its problems that should be handled in future improvements.**

***Keywords*—Interactive analytics; security visualization; visual decision tree; web application security**

I. Introduction

Decision tree learning is a popular technique for solving object classification problem, which is a common task in research fields such as machine learning and data mining. One of the reasons for this popularity is its ability to handle multi-dimensional data well. Another notable feature of decision tree learning is its human friendly presentation of learning result. Compared with other classification techniques like artificial neural network or support vector machine, whose results are rather black boxes to external users, decision tree learning produces hierarchical sequences of classification rules that are easy to understand and follow. It is even better when these sequences of rules are displayed in visual forms in which the most popular one is node-link diagram. In these diagrams, nodes correspond to collection of data objects, and links (i.e. edges) correspond to conditions to partition similar objects into subsets. Human users can classify a data object manually by starting at the root node and subsequently following links whose conditions the object satisfies, until reaching a leave node, with a specific class label, i.e. the classification result.

The presentation of decision trees and the way those trees are constructed are more or less can be done manually by a human user. Indeed, there are several unique benefits by integrating human users into decision tree building process. Firstly, it is the direct application of users' specific domain knowledge to construct trees. Automatic tree building algorithms tend to serve as general solutions to classification problems, so it is difficult or not natural to inject expert knowledge to guide the tree building process. Furthermore, algorithms implementers are usually data scientists, not domain experts, hence they may not see the problems being solved the way actual users see. In contrast, when real users are involved into tree building process, they can guide the construction using their own knowledge and experience. As an example, although "first name" and "password" are both, textual data, their meaning and natural pattern are different. Considering them as text, as general decision tree algorithms do, will lost their important differences. These losses will not happen when domain knowledge is applied appropriately when building trees.

Secondly, when human users actively participate in tree construction phase, they will understand more clearly and use the resulted tree more effectively in classification phase. By joining the tree construction process, human users get an overview picture of the dataset and the distribution of each attribute. From this knowledge, they see the reasons behind node splitting rules and priorities among data object's attributes. This gained insight, in turn, helps them work more effectively with the resulted tree in classifying data objects. They can even start over the tree construction phase when seeing the final result not satisfying. Starting over with experience gained from previous works can create a total different tree. Again, this effect is not trivial to implement in automatic tree construction.

The last, but not least, benefit is the flexibility in dealing with different data types and forming node splitting conditions human users can achieve. In automatic tree construction, only a limited data types, such as numeric or nominal, can be used directly. More complicated types need to go through a reprocessing phase before they can be used. Because the reprocessing and construction phases are independent, it is not possible to determine in advance if a reprocessing method is suitable for a particular data attribute. This makes the

reprocessing phase complex and time-consuming. On the other hand, in human-guided tree construction, reprocessing task is integrated into tree construction phase. As a result, using and changing reprocessing methods have instant feedbacks, thereby reducing time and complexity. Furthermore, unlike automatic algorithms which only produce simple predefined splitting rules, human users can enter any computable Boolean formulas for each tree node, making it quite flexible for classifying complex objects.

With the above benefits, it is natural to integrate human users into decision tree building technique to solve complex classification problems, one of which is the attack recognition of users' inputs on web applications. There are several reasons to integrate human security administrators into decision tree building process for the web application attack recognition problem. The first reason is computer attacks in general, and web application attacks in particular, are usually complicated. Because the web is one of the most popular platforms used today, its users have many differences in technical skills and background. This results in the determination whether a piece of user input is normal or not cannot be completely objective. For a particular instance of user input, one security administrator may see it as normal, yet another administrator may see it as abnormal. That is because besides the input data itself, it is necessary to consider its surrounding context, such as the user who creates it and the environment in which it resides, to make a fully informed decision. Without the direct involvement of a human administrator, it is not easy to integrate this context information into the classifier.

The second reason is related directly to the technique used to recognize web application attacks. As mentioned earlier, among machine learning methods, decision tree learning is one of the most human-friendly approaches, both for constructing classifiers as well as using learned results to classify data objects. As a result, integrating human users into decision tree learning is a natural choice to take advantages of the human reasoning capability and powerful computer data processing at the same time. This leads to the next question that we need to address: how to integrate human administrators into decision tree building effectively?

Because human users interact with computing systems via the user interface components, these are the places where most integration happens. The user interface should be designed to communicate effectively with its main users, in this case are the web application security administrators. Although there are many methods of communication, interactive visual interface is one of the most preferred methods. Unlike traditional text-based command line interface, interactive visual interface can provide more natural interaction and intuitive appearance to the users. Furthermore, when working with a high volume of input data, displaying it visually helps users overcome the information overload problem.

From the reasons mentioned above, in this paper we describe a method and tool called Interactive Visual Decision Tree (IVDT) whose purpose is to support web application security administrators in creating attack detection rules. IVDT is an interactive visual tool for building decision trees, specifically targeting at web application input classification. Given a collection of data input objects and their respective labels, security administrators can use IVDT to build visual decision trees which are used later to classify unknown inputs. But before proceeding further, it should be noted that although IVDT provide some unique advantages, it is not meant to replace automatic classification methods. Instead, we believe it should be used together with other techniques to utilize the strengths of all, especially in a complicated domain like security.

The rest of the paper is structured as follow: In Section II, we review the related works for this research; in Section III, the user interaction and data visualization of IVDT are described in detailed; the prototype implementation of IVDT is introduced in Section IV; Section V is used to describe our experiments and their results; and finally Section VI concludes the paper with lessons learned and a plan for future works.

II. Related Works

A. Decision Tree Learning

One of the most popular decision tree learning algorithms is the ID3 (Iterative Dichotomiser 3) proposed by Quinlan [1]. The input for this algorithm is a dataset containing data objects in many classes. Each data object in the dataset has the same number of data attributes and is in one particular class. At each step, ID3 selects a data attribute that has not been used and creates a formula on it to split the data objects into subsets. All data objects in a common subset have the same outcome when applying the split formula. The decision to select which unused attribute at a step to create a split formula is based on a value called information gain of that attribute.

The information gain (IG) is defined as

$$IG = H(parent) - \sum H(children) \quad (1)$$

In (1), *H(parent)* is the entropy of the parent node before splitting, and *ΣH(children)* is the weighted sum of the entropy of all child nodes that are the result of the split. Entropy of a node *N*, in turn, is defined as

$$H(N) = -\sum(p_i * logp_i) \quad (2)$$

In (2), p_i is the percentage of data objects in node *N* having i as the common class label. The goal of ID3, and other decision tree learning methods, in creating split conditions is to have the child nodes purer than the parent node. One of the disadvantages of ID3 is its inability to work on continuous domain data types without a preprocessing step.

Some of the limitations of ID3 are solved in C4.5, another decision tree learning method also proposed by Quinlan [2]. Among the improvements, C4.5 algorithm can handle continuous data attributes by generating a threshold for each of them, and use this value to split the parent node into two child nodes. Data objects having attribute values higher than the respected threshold are put into one child node, and the rest are put into the other child node. Another notable improvement of C4.5 compared with ID3 is that it can work with data objects missing values on some attributes. These missing values are simply not used in entropy and information gain calculations when deciding which attributes to use in split formulas.

CART (Classification and Regression Trees) is another popular technique for decision tree learning, proposed by Breiman et al. [3]. This technique is useful not only for classification, but also for regression. The output of CART is determined by the type of the dependent variable: if the dependent variable is categorical, the resulting tree is a classification tree; on the other hand, if the dependent variable is numeric, the resulting tree is a regression one. Like other decision tree learning methods, CART algorithm follows a greedy approach, i.e. at each step a split on a node is chosen as to maximizing the purity of the resulting child nodes. The purity metrics can be deviance, entropy, or gini index.

B. Interactive Decision Tree Learning

One of the earliest works on interactive decision tree learning was proposed by Ankerst et al. [4]. In that work, the authors used a multidimensional visualization method on the training data to support the users in selecting the split point optimally. The visualization technique is pixel-oriented, similar to the Circle Segments method [5], in which each attribute value is mapped to a particular color based on the class of the data object having that attribute value. Different attributes' values are positioned in separate areas. For data objects with D attributes, a circle with D segments, each segment for one attribute, is used to represent them. In each segment, the data values of the respective attribute are positioned from the center of the circle to the outer border line-by-line, each line is orthogonal to the segment halving line. The human users can select an unused attribute to create a split formula on it to grow the decision tree. They can also remove an attribute from the current decision tree, to backtrack to a previous state.

Another research on interactive decision tree learning is the PaintingClass by Teoh & Ma [6]. PaintingClass is a system for interactive construction, visualization and exploration of decision trees. Although the main objective of PaintingClass is to interactively construct decision trees, its other two objectives, visualization and exploration, are not less important. In fact, visualization and exploration features help PaintingClass's users have a better understanding of the underlying data, which in turn, make them be able to create small and accurate trees. Parallel coordinate visualization [7] is used in PaintingClass to display multi-dimensional objects in two-dimensional screen.

BaobabView is another work in this category [8]. Its user interface design has some differences compared with the previous works. The three most important visual areas of it are decision tree main view, attribute view, and confusion matrix view. The decision tree main view displays the decision tree being built in a node-link diagram. The nodes are containers of data objects, while the links display the flow of data objects from parent nodes to child nodes. Sizes of nodes and links correspond to the number of data objects they contain. Details about attributes' values of a selected node are shown in the attribute view. These values are sorted based on some impurity metrics such as information gain [1], gain ratio [2], and gini gain [3] to help users in seeing the overall data value distribution, and using that information to create appropriate split formula for the selected node. The confusion matrix view displays the correct and incorrect classified objects, to give users instant feedback for their choices.

C. Security Visualization

Security visualization is a multi-disciplinary research field studying the use of information visualization techniques to solve computer security problems. In security visualization systems, human user is an integral part. The reason behind this tight integration is to combine the powerfully graphical capability of computers with the efficiently visual analysis of human to solve complex security problems. This combination creates new advantages that are difficult to achieve when either human analysis or computer processing is used individually. One of the notable advantages is to help human users overcome the information overload issue when working with security data, thereby letting them make informed decisions.

In a research, Choi et al. used parallel coordinate method [7] to visualize network traffic for security administrators to detect large-scale internet attacks such as worms, DDoS, or network scanning activities [9]. Because each type of attacks has a particular visual pattern when visualized (the authors called them visual attack signatures), administrators can easily and quickly recognize them. After recognition, the administrators can further investigate the traffic data source in more detailed to see if the attacks are true or not. Although the final decision is made by human users, this visualization does enhance the decision making process by reducing the time and effort of administrators in analyzing high volume traffic data.

Security visualization is not only helpful for security experts, but also for average end users. End users need simple but effective software interface to accomplish their tasks. But if the tasks are security – related, the interface is rarely simple. One such task is sharing files between users. Because there are many rules governing the final permission of the shared files or folders, the interface to configure permission is rather complicated. To make the file sharing task simple and secure at the same time, Heitzmann et al. developed a visually secure interface for NTFS file system [10]. This interface uses Treemap [11] to display hierarchical folder structure with different colors for different permission levels. Because colors can be effortlessly differentiated by human, it is easy for a user to know if moving/copying files/folders to new locations violates their original permissions or not. Another task deserves looking at is web browsing. This is a popular task and involves many security decisions to be made by the users during a browsing session. As a result, browser software vendors invest much effort in designing visual interface components to communicate with their users about security information, such as sites protected by SSL/TLS, cookies, possible phishing sites, etc. A more comprehensive survey about browsers' security designs is given in [12].

One similar research of this work was proposed by Dang and Dang [13]. In that research, the authors described a security visualization technique to analyze user inputs on HTML web forms. Multi – levels zooming is used to provide administrators different levels of detailed views, depends on their needs. Unlike traditional text display method, with this visualization technique, the authors demonstrated that thousands of user input data objects can be displayed and analyzed at a time. Furthermore, built-in interactions support security administrators in selecting and viewing specific subset data in more detailed. By working directly with user input data,

human administrators can have an overall understanding of the structure of the web application they are trying to protect as well as its environment (input forms, types of user, complexities of attacks, etc.). This understanding is not easily to obtain when they only work on the surface with automatic tools like IDS, IPS. The difference between that research and our work is that ours go a step further by providing a tool not only for analyzing user inputs, but also for developing attack detection rules in the form of decision trees.

III. Visualization and Interaction Design

A. Problem and Solution Specification

In this section, we describe the format of user input data used by administrators to develop attack detection rules and details about the decision trees being built. Although web users can input data at any accessible location, these data objects must follow some predefined structures. These structures are determined in advance by web developers. As an example, the data objects to log in to an e-commerce website may contain fields such as "username" and "password", while the data objects to sign up an account may contain more fields like "first name", "last name", "email", "address", "telephone number", etc. However, the data objects for a specific action will have the same structure. For each action, because of the similarity of its data objects, a decision tree can be built on it to classify data objects into either normal or attack.

More specifically, each user input record is considered as a separate data object. In general, if an action requires a data record with N input fields, the corresponding data object will have N attributes. In reality, administrators can choose to exclude some fields if they think these fields are not necessary for recognizing attacks. Each data object used in the tree building phase has a class label, which is either normal or abnormal. In the verification phase, the result tree is used to compute the class label for new data objects. In the beginning, all data objects are in the same node, root node, of the decision tree. The administrators then create a Boolean rule, with one unused attribute, for the root node. Data objects satisfy that rule are copied to a child node of the root, while the others are copied to the other child. This process continues until all data objects in a node having the same class, or when the percentage/number of data objects in one class is small/big enough. When a new object is put into the tree, it will follow the created rules until reaching a leave node whose label is assigned to the new node.

B. Visualization and Interaction Design

When building attack recognition decision tree, administrators selects a node which contains a collection of data objects, i.e. user input data records, enters a Boolean split expression for the selected node to spit its contained data objects into two child nodes. This process continues until the whole tree is built. To support the administrators in analyzing data and determining appropriate split expressions, we provide two supporting tasks, node analysis and tree analysis. The main tasks and their relationship are depicted in Fig. 1.

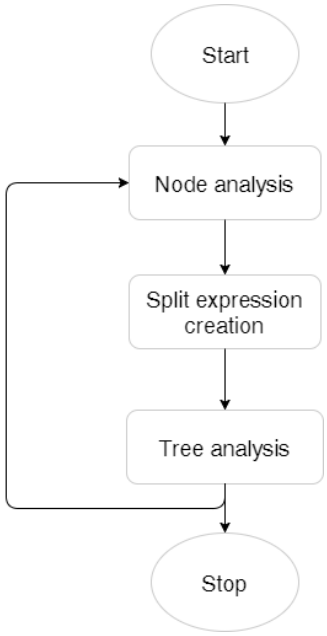

Fig. 1. The main tasks provided by ivdt for security administrators.

Node analysis: this task is used to inspect a selected node in details. The administrators can try different functions on any unused attributes to see how the data objects distribute. Usually, the pair of <*attribute*, *function*> that separates the data objects most should be chosen to create the split expression for the selected node. We provide a supported visual interface for this task, which is shown in Fig. 2. In Fig. 2, the pie chart displays the percentage of normal and abnormal data objects in the selected node. On the right of the pie chart are the distributions of data objects when custom functions are applied to unused attributes. The outputs of the custom functions are mapped to the X – coordinate, while the Y – coordinate is used to represent the number of data objects having the same X position, similar to the histogram presentation method. In Fig. 2, there are 2 distributions and it is easy to conclude that function 2 (segment (c) of Fig. 2) separates data objects better than function 1 (segment (b) of Fig. 2), and as a result, the administrators should use function 2 as the split expression.

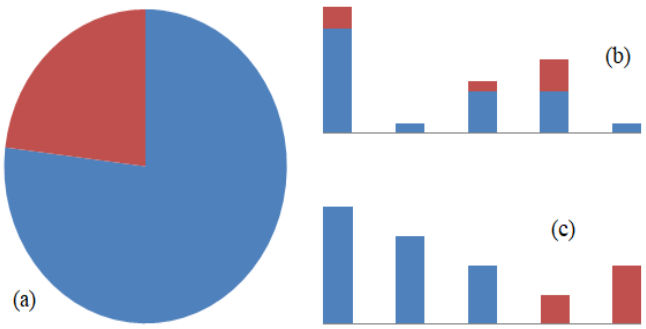

Fig. 2. Visualization of a node and custom functions: (a) Distribution of data objects over the whole node; (b) & (c) Distribution of data objects as histogram of custom functions on selected unused attributes.

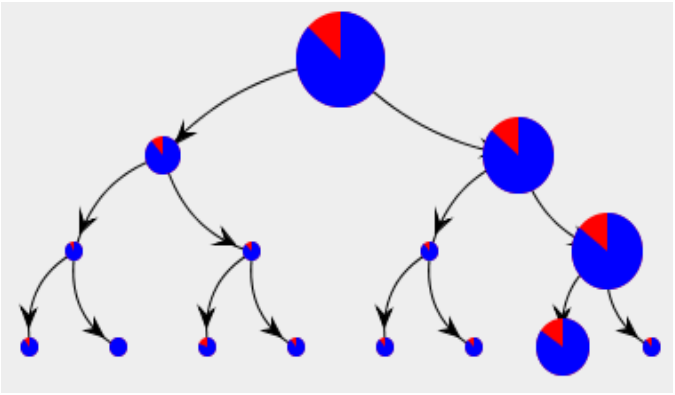

Fig. 3. Visualization of the attack detection decision tree.

Split expression creation: when the administrators finish analyzing a node, they can create a Boolean split expression for that node. The general form of the split expression is $S(a_i)$, in which S is any Boolean function and a_i is any unused attributes of the selected node. Because the split expression is a Boolean function, it maps the data objects into two groups. Data objects having the same output value are put into the same child node of the selected node. In this implementation, for no particular reason, the left child node is used to store data objects not satisfying the split expression and vice versa.

Tree analysis: after a split expression is created for a selected node, that node expands into two child nodes. This results in a new tree. The tree reflects the overall progress the administrators have made in developing attack detection rules. In contrast with node analysis, which provides a tool for local optimization, tree analysis focuses more on the global goal, i.e. it answers the question: how good is the attack detection decision tree being built? It does so by displaying the whole tree using pre-attentive visual attributes [14] to communicate crucial information with administrators quickly and naturally. More specifically, color is used for data objects' class label and node size is used for number of data objects in a node. These two graphical attributes together with tree's depth give administrators an approximate estimation of the goodness of the tree. Based on this subjective estimation, administrators can adjust their split expression strategy for new child nodes, or start over with a new tree. A sample visual decision tree is displayed in Fig. 3.

IV. Implementation

To experiment with the proposed technique, we've developed a prototype implementation of IVDT. The general architecture of the prototype is depicted in Fig. 4. The main components in the architecture are described below.

The User input Database: this is used to store the input records from external users of the target web application. Each input record contains a collection of <*name*, *value*> pairs. In each pair, *name* is a fixed value predefined in advance by the web application developer while *value* is an actual value entered by the web application user. All input records have the same structure, i.e. having the same number of <*name*, *value*> pair and the same *name* elements. If an input record has a different structure, there is a high chance that it is the direct result of form modification attack [15].

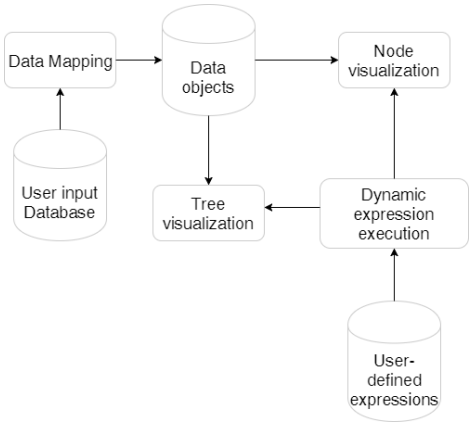

Fig. 4. Main components of the prototype implementation of IVDT.

The Data Mapping: the rules to retrieve data from the User input Database and map them into data objects suitable for processing are stored here. These rules specify the custom transformation for each input record field from its raw domain to another. For example, a textual input value of a field can be mapped into a numeric value for further processing. This component is also used to exclude fields that are not necessary in building decision tree. For example, fields like submit buttons always contain the same value for every input, so they do not provide any meaning in decision tree building and should be excluded at this step.

The Data objects: these objects are created from the user input records after applied the rules specified in the Data Mapping component. Besides a collection of <*name*, *value*> pair, each data object contains a class label, which is either normal or abnormal. The class label can be set by security administrators manually or by IDSs automatically.

The Dynamic expression execution: although the attributes contained in each data object can be of any type, e.g. Boolean, integer, real number, character string, etc. not all types are suitable for human analysis. In fact, we argue that choosing which data type/value to present an attribute is more or less a subjective decision. As an example, consider an attribute storing street addresses of buyers in an e-commerce web application, in our opinion, it will be more intuitive to analyze when these addresses are presented as distances to the target shop. But other people may prefer textual addresses to numeric distances and it is not unusual. To support different needs of different people, we develop the Dynamic expression execution component, which is responsible for converting attribute values from one domain to another at run time. The role of this component is somewhat similar to the role of Data Mapping component with one difference: the conversions here are done at run time. As a result, the transformation rules specified in Dynamic expression execution are not as complex as the ones specified in the Data Mapping component, but they can change data object attributes' values in real time. This real time support is important because it helps administrators in experimenting with different expressions to find the most suitable one to present an attribute. Once an expression is chosen, it is not difficult to turn it into a Boolean expression in order to create a split condition with that attribute. We use Java

Expression Library [16] to implement the Dynamic expression execution component.

The Node visualization: this component is responsible for displaying each selected node in detail according to the interface depicted previously in Fig. 2. As shown in Fig. 2, the selected node is visualized as a pie chart presenting the percentages of data objects with normal or abnormal class label. In addition to the pie chart, some bar charts presenting the distribution of the output values achieved by applying the Dynamic expression execution on an attribute are also displayed. The library JFreeChart [17] is used to implement the Node visualization.

The Tree visualization: this component is responsible for displaying the whole decision tree being built. The visualization method is depicted earlier in Fig. 3. As shown in Fig. 3, the relative size of each node corresponds to the number of data objects contained in that node. In turn, each node is visualized as a pie chart, similarly to the way the Node visualization does. Seeing the whole tree as it being built helps administrators in keeping track of their overall progress and also evaluating their current work result. We use JUNG (Java Universal Network/Graph Framework) [18] to organize tree elements, i.e. nodes, edges, layout, etc. as well as visualize it.

V. Experiments

There are two main objectives in our experiments. The first one is to measure the effectiveness of the proposed technique in recognizing attacks on web applications. Because attack detection rules are created by human users, in combination with the support of IVDT, the measured result does not only depend on our technique, but also depend on the skills and experience of the users. Despite this fact, the way users perform their analysis to create detection rules is also affected by the functionality of the provided tool, so the effectiveness we obtain is related to the proposed technique to some extent. In particular, the effectiveness is evaluated as the true positive rate and true negative rate of the resulted visual decision tree in classifying user inputs. The second objective of our experiment is to evaluate the ease or difficulty users have when using IVDT to create attack detection rules. For this objective, we follow a qualitative approach by interviewing the users directly.

A. Data Generation

Without loss of generality, the data being analyzed is supposed to be sent to a sign up action. This action is common on many types of web applications such as e-commerce, bulletin board, online social network, etc. On the input data, there are some fields which are easy to define which values are normal or not. Some examples are fields used to store email address, date of birth, and phone number. On the other hand, it is more difficult to define the patterns of normal values for fields like first name, last name, and password. Because both types of input fields are contained this example, we can see if there is a difference on the way users working on them.

The data objects in our experiments are generated automatically. They are labeled as either normal or abnormal. The abnormal data objects are malicious inputs on the sign up action. In particular, these malicious inputs are composed of SQL injection (SQLi) and Cross-site scripting (XSS) attacks. We choose SQLi and XSS because they are among the most popular attacks on web applications today [19], even though they existed long ago. The data generation processes are described below.

Normal data generation: We use the tool GenerateData [20] to generate normal data. It supports many different types of data such as human data (first name, last name, email, company, etc.), geo data (street address, city, region, latitude/longitude), credit card, text, numeric, etc. We also create a database table to store the artificial generated data. Each record of the database table corresponds to a data object and each table field corresponds to an attribute of the respective data object. There is an additional field used to store the label of the data object. For data objects generated by the GenerateData tool, their labels are assigned normal value.

Abnormal data generation: As described above, abnormal data are SQLi and XSS attacks, so they are generated by another tool. In the experiments, the attack values are created by the HackBar add-on for Mozilla Firefox browser [21]. The main purpose of this tool is to help web developers audit their code and look for security holes. The process we use to generate an abnormal data object is as follow: firstly, we use HackBar to generate an attack value; secondly, we select a random data object stored in the database table; and finally, we select a random field of the selected data object and replace its value with the generated attack value.

The total data objects generated is 1000, in which 800 are normal and 200 are abnormal. We further divide them into two sets: training set and testing set. Each set has the same number of normal and abnormal data objects (i.e. 400 normal and 100 abnormal objects in each set).

B. Experiment Settings

At this stage, our main objective is to get initial feedback about the proposed technique so that we can learn and improve them appropriately. Therefore, these experiments do not involve with many people. Instead, we invite three security practitioners, who are also members at our security lab, to try the prototype and give their opinion about it. These volunteer people all have adequate knowledge and skill about web application security. We record the action steps the security practitioners do during their assigned experiments, with their permission, to analyze further.

Before the security practitioners do their assigned tasks, we give them a brief tutorial about the main components of our prototype and the decision tree classification method. Because the security practitioners already know about SQLi and XSS attacks, these information are not covered in our tutorial. But we do stress the use of the Dynamic expression execution component as well as the visual display to analyze and develop SQLi and XSS attack detection rules. At the end of the tutorial, we tell the volunteers to write attack detection rules as precisely and compactly as possible.

C. Observed Action Steps

The common steps that the security practitioners do according to our observation are:

At first, when there is only the root node, they look through all attributes of the data objects. They do not apply any transformation on any field, but just sees the field's values in their originals. Maybe doing so can give them an overview of the distribution of values of each attribute.

After that, these people try some transformations on some attributes. We find that the most frequently used transformations they apply are length() and indexOf(), which return the length of an input string and the position of an input pattern in another input string respectively. Maybe these transformations are simple but effective enough to detect abnormal values, especially for human readable attributes like "first name" and "last name". These human readable attributes are also chosen before other attributes as split attributes.

Finally, for random attributes like password, it is more difficult for them to decide which value is normal and which one is not. As a result, the transformation rule they create on password field is rather complicated with many string functions combined together using Boolean functions such as AND and OR. Maybe foreseeing this complexity, split conditions for complex attributes are created near or at the leaves of the decision tree only.

D. Results and Discussion

After the security practitioners finish their assigned tasks, we ask them some questions to get their feedback on the usefulness or otherwise of the prototype. We also measure the detection performance of their result decision tree. Therefore, we divide our evaluation into two parts: quantitative result and qualitative result.

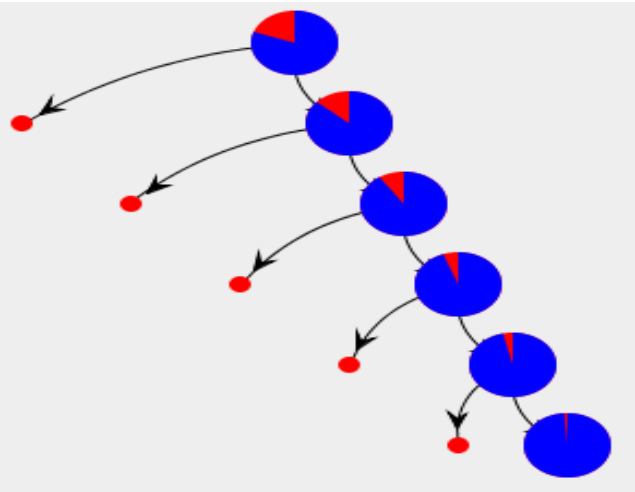

Fig. 5. All of the result trees have this similar form: they are skew toward the normal side.

Quantitative result: it takes around 15 minutes for the security practitioners to completely build attack detection rules in the form of a decision tree. Among the three people, one uses only 11 minutes to finish his task, while another uses 18 minutes. The action steps they do follow similarly to the ones described in the previous section. The height of all the resulted decision tree are 5 and they are rather unbalanced toward the normal class. This can be explained by the approach the security practitioners use to create the split condition for each node, i.e. these conditions focus on the recognition of attack patterns. So, as soon as a data object attribute's value matches such patterns, this data object is considered abnormal and the process can terminate immediately. On the other hand, when a data object attribute's value does not match any attack pattern, the remaining attributes should still be checked. When testing with our generated dataset described previously, the average true positive rate is 95% (92/97) and the average true negative rate is 100% (403/403). The visualization of one of the resulted trees is displayed in Fig. 5.

Qualitative result: we ask the security practitioners what they like or don't like about IVDT. Their responses are similar to what we expected. To summarize, all of them like the way we visualize each node (in Node analysis task) and the whole tree (in Tree analysis task) to support the tree building process. Two of them think the Dynamic expression execution feature is useful because it helps them in trying out different tests at run time, thereby being able to apply their relevant skills and knowledge directly. The volunteers also provide some useful feedbacks to improve IVDT. Two of them suggest that we should complement this tool with a feature to build trees automatically so that they can compare their results with the automatic result version, and/or they can use the automatic result as the base from which to make more refinements manually. One other important suggestion is that our tool should integrate with available web application IDSs or firewalls to reuse their large existing attack detection rules.

VI. Conclusions and Future Works

In this paper, we have proposed and developed a visual interaction technique to support security administrators in building detection rules of attacks on web applications. The technique is based on decision tree learning and is enhanced further by adding data visualization and user interaction supports into the tree building process. Unlike traditional decision tree learning, our technique is user – driven. In other words, human users manually create classification rules, with necessary supports from the IVDT tool. Our proposed technique possesses some advantages over traditional decision tree learning. Some of the most important advantages include:

Firstly, knowledge of users is applied directly. Because users directly analyze data objects and enter split conditions for each node, they can utilize their existing domain knowledge into the rules creation process. This can result in better classification trees.

Secondly, users gain not only result, but also insight about the target web application and its environment. When analyzing data to create split conditions, users do not only create decision trees but also have an overview picture of the data objects, attributes, and values' distribution. This insight is difficult to achieve when using an automatic tool.

Finally, classification rules can be made quite flexibly. Because classification rules are entered by human users, they can be quite flexible. For example, the data attributes used in split conditions can be of any type, not just numeric or nominal types; and custom functions used on data objects' attributes can produce many types of outcomes.

However, the proposed technique still has several issues that need to be addressed in the future.

Firstly, due to some external constraints, we can invite only three security researchers in our lab to join the experiments. Because of the small number of participants, the experiment result may not be representative for the general security administrators. In future experiments, we plan to invite more people with different backgrounds to have more feedbacks. In addition to security experts, we also invite students majored in computer science to join the experiments. The purpose of having computer science students in future experiments is to consider the appropriateness of using this tool for security teaching.

Secondly, because the number of data objects in the experiments is only average, we cannot evaluate the appropriateness of our using IVDT with very big data. For example, when there are many attributes in the training data objects, it is not trivial to select the first attribute to create the split condition. Similarly, when the decision tree being built is very big, displaying it on the computer screen at once in a meaningful way is also a difficult problem to which we will pay more attention in future works.

ACKNOWLEDGMENT

This research is funded by Vietnam National University HoChiMinh City (VNU-HCM) under grant number C2018-20-12.

REFERENCES

[1] J. R. Quinlan, “Induction of Decision Trees,” Mach. Learn., vol. 1, no. 1, pp. 81–106, 1986.

[2] J. R. Quinlan, “C4.5: Programs for Machine Learning,” Morgan Kaufmann San Mateo Calif., vol. 1, no. 3, p. 302, 1992.

[3] R. A. Breiman, Leo and Friedman, Jerome and Stone, Charles J and Olshen, “Classification and regression trees,” 1984.

[4] M. Ankerst, C. Elsen, M. Ester, and H.-P. Kriegel, “Visual classification: an interactive approach to decision tree construction,” in Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD ’99, 1999, pp. 392–396.

[5] M. Ankerst, D. Keim, and H. Kriegel, “‘Circle Segments’: A Technique for Visually Exploring Large Multidimensional Data Sets,” in Proc. IEEE Visualization ’96, Hot Topic Session, 1996, pp. 5–8.

[6] S. T. Teoh and K.-L. Ma, “PaintingClass: interactive construction, visualization and exploration of decision trees,” in Star, 2003, pp. 667–672.

[7] A. Inselberg, “The plane with parallel coordinates,” Vis. Comput., vol. 1, no. 4, pp. 69–91, 1985.

[8] S. van den Elzen and J. J. van Wijk, “BaobabView: Interactive construction and analysis of decision trees,” in IEEE S. Vis. Anal., 2011, pp. 151–160.

[9] H. Choi, H. Lee, and H. Kim, “Fast detection and visualization of network attacks on parallel coordinates,” Comput. Secur., 2009.

[10] A. Heitzmann, B. Palazzi, C. Papamanthou, and R. Tamassia, “Effective visualization of file system access-control,” in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2008, vol. 5210 LNCS, pp. 18–25.

[11] B. Johnson and B. Shneiderman, “Tree-Maps: S Space Filling Approach to the Visualization of Hierarchical Information Structures,” 1991, pp. 284–291.

[12] T. K. Dang and T. T. Dang, “A survey on security visualization techniques for web information systems,” Int. J. Web Inf. Syst., vol. 9, no. 1, 2013.

[13] T. T. Dang and T. K. Dang, “Visualization of web form submissions for security analysis,” Int. J. Web Inf. Syst., vol. 9, no. 2, pp. 165–180, 2013.

[14] A. Treisman, “Preattentive processing in vision,” Comput. Vision, Graph. Image Process., vol. 31, no. 2, pp. 156–177, 1985.

[15] D. Scott and R. Sharp, “Abstracting application-level web security,” in Proceedings of the eleventh international conference on World Wide Web - WWW ’02, 2002, p. 396.

[16] K. Metlov, “Java Expressions Library.” [Online]. Available: http://www.gnu.org/software/jel. [Accessed: 02-Oct-2017].

[17] D. Gilbert, “JFreeChart.” [Online]. Available: http://www.jfree.org/jfreechart/. [Accessed: 02-Oct-2017].

[18] J. Team, “Java Universal Network/Graph Framework.” [Online]. Available: http://jung.sourceforge.net/. [Accessed: 02-Oct-2017].

[19] OWASP, “OWASP Top 10 - 2017.” [Online]. Available: https://www.owasp.org/index.php/Top_10-2017_Top_10. [Accessed: 10-Jan-2018].

[20] GenerateData, “Generate Data.”. [Online]. Available: http://generatedata.com/. [Accessed: 10-Jan-2018].

[21] J. Adriaans and P. Laguna, “HackBar.” [Online]. Available: https://addons.mozilla.org/en-US/firefox/addon/hackbar/. [Accessed: 10-Jan-2018.