FEATURE

Applying AI in Application Security

Do you have something to say about this article?
Visit the *Journal* pages of the ISACA® website *(www.isaca.org/journal)*, find the article and click on the Comments link to share your thoughts.

http://bit.ly/2AWQFDL

Kiran Maraju, CEH, CISSP
Has 18 years of information security experience and is involved in multiple network, web application and mobile security vulnerability assessments. He also has experience with penetration testing, security code review, software development life cycle security, network security, database security and wireless security assessments. Maraju is working as a specialist leader for Deloitte. Prior to that, he worked as a scientist for the Indian Space Research Organization (ISRO).

The use of artificial intelligence (AI) in cyber security will help organizations enhance existing application security capabilities. Application security covers the security of web or thick client and mobile applications that pass through various phases of the security development life cycle, e.g., security design and security coding. Various AI areas such as machine learning and expert systems can be leveraged to improve application security to derive, predict or apply inferences to forecast security threats, identify security vulnerabilities and identify the security coding remediation guidance.

The following AI areas can be applied to application security:

- **Machine learning**—Decision-tree learning (DTL) during threat identification
- **Expert systems**—Forward chaining and backward chaining for security code review and code review guidance

Security auditors can make use of these techniques to automate the attack threat identification and code review process. The process involves developing various decision support inference rules for various application security vulnerabilities, applying the decision and inference rules to expert systems, and training the same systems using an algorithm with the various application security attack scenarios and attack paths. Security auditors can identify and infer the possibility of successful attacks by providing inputs to these machine-learning-based application security expert systems. **Figure 1** describes the relationship between DTL and expert systems in this process.

Machine Learning DTL—Application Security Attack Threat Identification

DTL is a form of inductive learning that uses a training set of examples to create a hypothesis that makes general conclusions. It also determines best attribute paths of attack (threats) if possible. During decision-tree development, a set of training examples is divided into smaller subsets and an associated decision tree is developed incrementally. A decision tree covering the training set is returned after completing the learning process. **Figure 2** describes threat decision trees.

Cross-site scripting (XSS) is a web application security vulnerability where input data submitted to the web application are not validated properly for malicious inputs. Attackers use this XSS vulnerability to hijack sessions and use cookies and for page defacement attacks.[1]

The following is a typical XSS scenario with training examples, including attack paths with different attributes available such as input data validation, output data validation and data type. The tainted data type values with possible data types are returned to the end user (reflective XSS), saved in the server (persistent XSS) or transferred to a different system as input. By applying entropy and gain values, the next-best-attribute decision tree with an entropy value selected as root node and with all subsequent branch nodes and root nodes is constructed as depicted in **figure 3**.

The decision-tree method is a type of supervised learning that involves attributes, training sets,

Figure 1—DTL, Forward Chaining and Backward Chaining

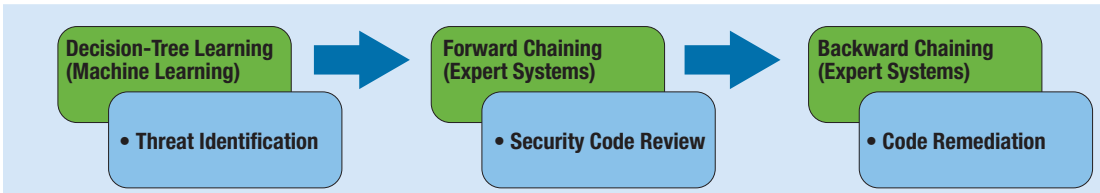
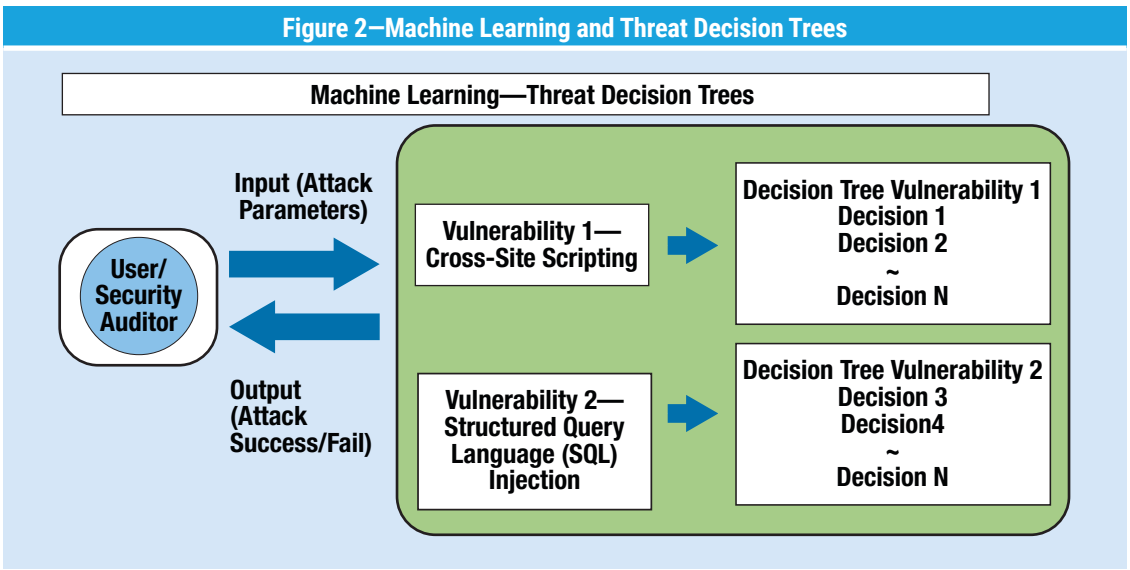


Figure 2—Machine Learning and Threat Decision Trees



and determining the best attribute according to entropy and gain with algorithms such as Iterative Dichotomiser 3 (ID3).[2]

The information gain (G), G(S,A) where A is an attribute and S is a sample of training examples

p_+ is the positive examples in S
p_- is the negative examples in S

- Entropy of S: Average optimal number of bits to encode information about certainty/uncertainty. Entropy (E) is the minimum number of bits needed to classify an arbitrary example as yes or no.

$$Entropy(S) = -p_+ \log_2 p_+ - p\text{-}\log_2 p_-$$

- Gain(S,A): Reduction in entropy after choosing attribute A

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values^{*}A!} \frac{|S_v|}{|S|} Entropy(S_v)$$

Therefore, the entropy of the training data, E(S), can be represented as E ([3+, 9-]) because out of the 12 training examples, three of them are attack success and nine of them are attack fails. **Figure 4** describes attack paths and their outcomes.

The previous attack paths are provided as training sets to the decision support system, and the input is passed through this decision-support system. This is depicted in **figure 3** as the XSS attack decision tree. When the input data validation is not performed, this will check for output data validation and, if output data validation is not performed, it will check for data type and provide the outcome as attack success (i.e., XSS attack is possible) or attack fail (i.e., XSS attack is not possible). For attack path P13, input data validation is performed with the decision-tree outcome attack fail. This means that XSS is not possible using the above training sets.

Figure 3 is the XSS vulnerability decision tree. Similar types of decision trees can be created for various application security vulnerabilities to correlate, identify and predict the security threats once these decision trees are constructed for various security vulnerabilities. These decision support system rules can be leveraged as inference rules for the application security expert systems.

Application Security Expert Systems

Expert systems are capable of interpreting the provided input, advising and deriving a solution. They also provide suggested alternatives to the problem and predict the results. **Figure 5** illustrates the components of an expert system—knowledge base and inference engine. The knowledge base comprises information that is factual and heuristic (guessable), including rules and facts. If-then-else rules are a part of the knowledge representation. Information acquired from security experts is fed to the expert systems in the form of if-then-else rules to develop security expert systems. Facts are the assertions.

Figure 3—XSS Attack Tree

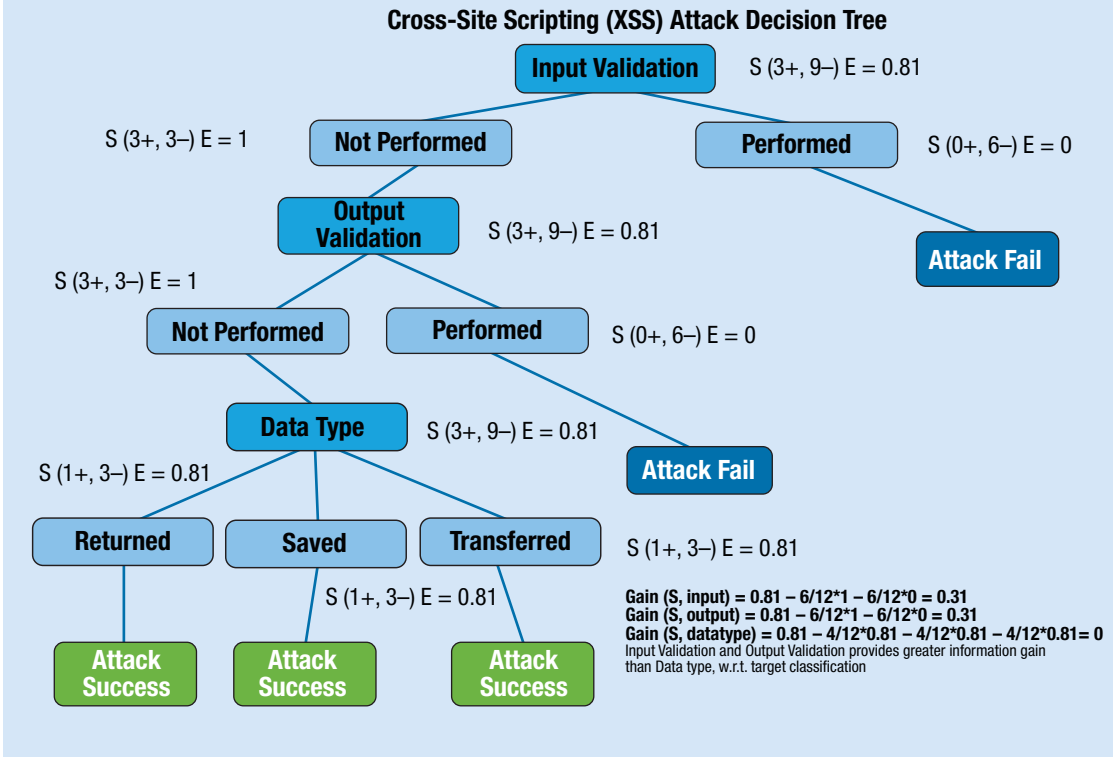
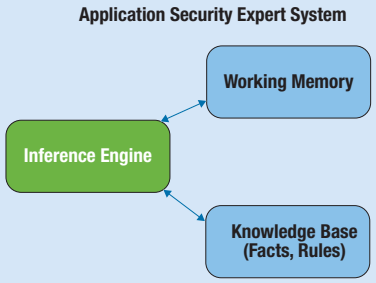


Figure 4—Attack Paths and Outcomes

Attack Path	Input Data Validation	Data Type	Output Validation	Result (XSS Injection Attack)
P1	Not Performed	Returned	Not Performed	Success
P2	Not Performed	Returned	Performed	Fail
P3	Not Performed	Saved	Not Performed	Success
P4	Not Performed	Saved	Performed	Fail
P5	Not Performed	Transferred	Not Performed	Success
P6	Not Performed	Transferred	Performed	Fail
P7	Performed	Returned	Not Performed	Fail
P8	Performed	Returned	Performed	Fail
P9	Performed	Saved	Not Performed	Fail
P10	Performed	Saved	Performed	Fail
P11	Performed	Transferred	Not Performed	Fail
P12	Performed	Transferred	Performed	Fail
P13	Performed	Returned	Not Performed	??

Figure 5—Expert System Components



Security Code Review Inference Engine—Expert Systems Forward Chaining

The security code review inference engine with forward chaining[3] can be used to predict values, i.e., deriving what can happen next. This will help the security code review engines to actually determine the type of attack. **Figure 6** details the SQL injection[4] vulnerability, with the forward chaining inference rules using if-then-else rules by matching the various conditions.

SQL injection is a web application security vulnerability where input data submitted to web applications are not validated properly for malicious inputs. Using SQL injection vulnerability attacks, attackers will inject malicious SQL commands to the back-end database and exfiltrate database details. **Figure 7** describes the security code review inference engine and forward chaining rules.

Forward Chaining Inference Rules for SQL Injection Vulnerability

The following are examples of the facts and rules used to demonstrate the forward chaining inference rules for SQL injection vulnerability.

Facts:

- **Fact 1:** X is a URL parameter.
- **Fact 2:** X contains a special character.
- **Fact 3:** X is not white-list input validation processed.
- **Fact 4:** X is passed through a database call.

Rules

- **Rule 1:** If X is a form parameter and X contains special characters, then X is a tainted input.
- **Rule 2:** If X is a URL parameter and X contains special characters, then X is a tainted input.
- **Rule 3:** If X is a cookie parameter and X contains special characters, then X is a tainted input.
- **Rule 4:** If X is a file import and X contains special characters, then X is a tainted input.
- **Rule 5:** If X is an HTTP header and X contains special characters, then X is a tainted input.
- **Rule 6:** If X is a tainted input and input is not white-list input validation processed, then X is an unvalidated input.

Figure 6—SQL Injection Vulnerability With Forward Chaining

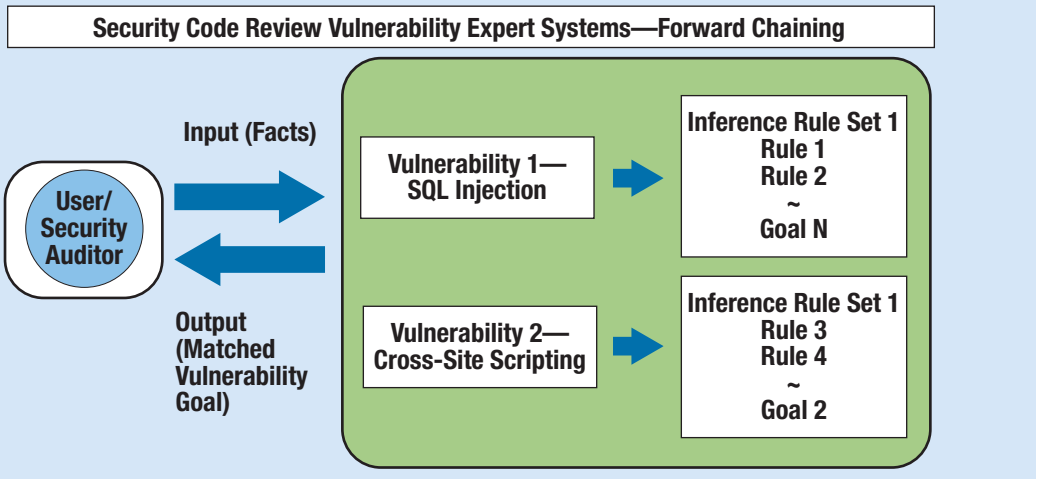
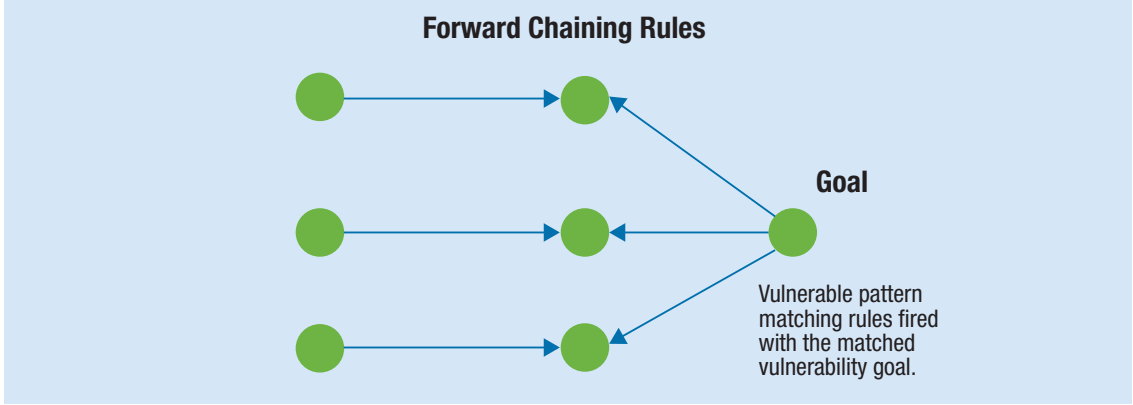


Figure 7—Security Code Review Inference Engine



- **Rule 7:** If X is a tainted input and X is not blacklist input validation processed, then X is an unvalidated input.
- **Rule 8:** If X is a tainted input and not an escaping input, then X is an unvalidated input.
- **Rule 9:** If X is an unvalidated input and X is not processed in prepared statements (with parameterized queries), then X is a potential SQL injection input.
- **Rule 10:** If X is an unvalidated input and X is passed through a database call, then X is a potential SQL injection input.

Figure 8 details the rules matching the current working memory, conflict set, rule fired and the next cycle after a rule has fired.

Application Security Vulnerability Remediation Guidance—Expert Systems Backward Chaining

The application security remediation guidance inference engine with backward chaining can be used for diagnosis of the values, i.e., deriving what happened. This application security remediation guidance expert system helps the developer to determine various possible sub-goals (solutions) to fix the vulnerabilities. **Figure 9** describes backward chaining expert systems that provide guidance for security vulnerabilities with inputs as goals and facts and outcomes as possible matched sub-goals.

Figure 10 details the application security remediation guidance rules for SQL injection remediation vulnerability where rules 14, 15 and 16 will be sub-goals. These sub-goals can be treated as possible solutions that a developer can implement to remediate the SQL attack, i.e., escape the input or implement blacklist/white-list input validation for special characters.

Backward Chaining Inference Rules for Identifying SQL Injection Vulnerability Recommendations

The goal is SQL injection, and the rules are:

- **Rule 11:** If X is a potential SQL injection input, then X is an unvalidated input.

Figure 8—Working Memory, Conflict Set and Rule-Fired Cycles

Cycle	Working Memory	Conflict Set	Rule Fired
0	URL parameter, special characters, tainted input	2	2
1	URL parameter, special characters, tainted input, input not whitelist input validation processed	2, 6	6
2	URL parameter, special characters, tainted input, input not whitelist input validation processed, X is passed through a database call	2, 6, 10	10
3	URL parameter, special characters, tainted input, input not whitelist input validation processed, X is passed through a database call, SQL injection	2, 6, 10	Halt

Figure 9—Application Security Vulnerability Remediation Guidance Expert Systems, Backward Chaining

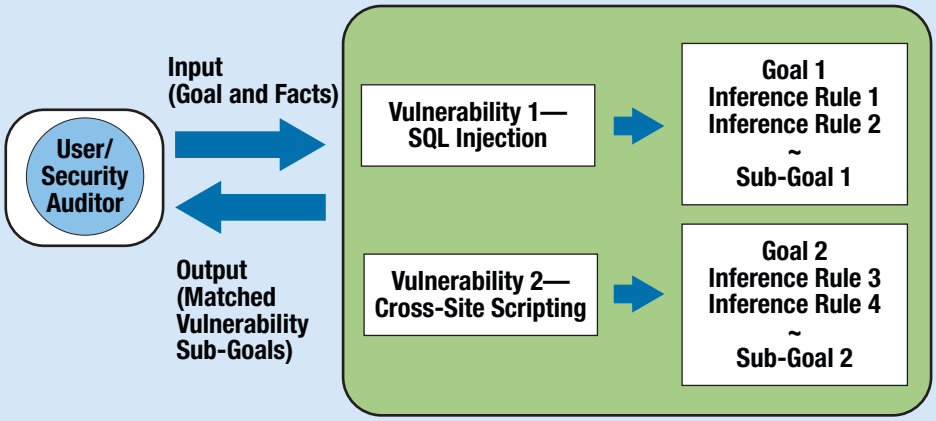
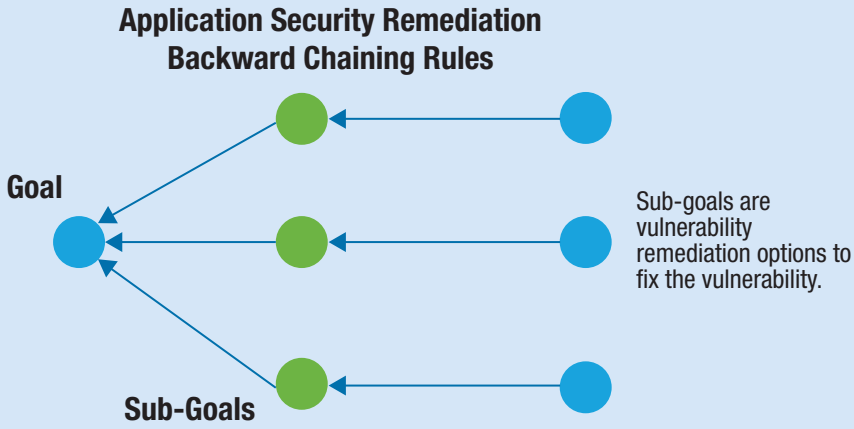


Figure 10—Application Security Remediation and Sub-Goal Development



- **Rule 12:** If X is a potential SQL injection input, then X is not processed in prepared statements (with parameterized queries).
- **Rule 13:** If X is a potential SQL injection input, then X is passed through a database call.
- **Rule 14:** If X is an unvalidated input, then X is a tainted input and not an escaping input (sub-goal).
- **Rule 15:** If X is an unvalidated input, then X is a tainted input and X is not blacklist input validation processed (sub-goal).
- **Rule 16:** If X is an unvalidated input, then X is a tainted input and X is not white-list input validation processed (sub-goal).
- **Rule 17:** If X is a tainted input, then X is from HTTP request header and X contains special characters.
- **Rule 18:** If X is a tainted input, then X is from file import and X contains special characters.
- **Rule 19:** If X is a tainted input, then X is from cookie parameter and X contains special characters.

Figure 11—SQL Injection Working Memory, Conflict Set and Rule-Fired Cycles			
Cycle	**Working Memory**	**Conflict Set**	**Rule Fired**
0	SQL injection input	11, 12, 13	11
1	SQL injection input, unvalidated input	11, 12, 13, 14, 15, 16	14
2	SQL injection input, unvalidated input, tainted input and not escaping input	11, 12, 13, 14, 15, 16	15
3	SQL injection input, unvalidated input, tainted input and not escaping input, not white-list input validation processed, not blacklist input validation processed	11, 12, 13, 14, 15, 16	16
4	SQL injection input, unvalidated input, tainted input and not escaping input, not white-list input validation processed, not blacklist input validation processed	11, 12, 13, 14, 15, 16	Halt

- **Rule 20:** If X is a tainted input, then X is from URL parameter and X contains special characters.
- **Rule 21:** If X is a tainted input, then X is from parameter and X contains special characters.

Figure 11 details the rules matching the current working memory, conflict set, rule fired and the next cycle after a rule has fired.

Working memory (WM) = ([unvalidated input] [tainted input] [not escaping input] [not white-list input validation processed] [not blacklist input validation processed]) comprises the various methods (sub-goals) that can be considered as possible solutions to remediate the SQL injection vulnerability.

Conclusion

Decision-tree machine learning and application security expert system techniques can be leveraged to automate decision-making to determine the next best attributes to use to identify the attack paths to classify/identify security threats, security vulnerabilities and code remediation guidance. This can be achieved by identifying and providing all possible attack scenarios to DTL and application security expert systems. Training sets are incrementally developed to create hypotheses to derive conclusions. The application security expert systems with forward and backward chaining can also be used to determine the security vulnerabilities, i.e., deriving consequences based on possible antecedents (matched rules), and can also be used for advising security vulnerability coding remediation solutions to fix the vulnerabilities.

Endnotes

1 Open Web Application Security Project, Cross-Site Scripting (XSS), *https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)*
2 Quinlan, J. R.; "Induction of Decision Trees," *Machine Learning 1*, p. 81–106, Kluwer Academic Publishers, USA, 1986, *www.hunch.net/~coms-4771/quinlan.pdf*
3 Al-Ajlan, A.; "The Comparison Between Forward and Backward Chaining," *International Journal of Machine Learning and Computing*, vol. 5, iss. 2, 2015, p. 106–113, *www.ijmlc.org/vol5/492-A14.pdf*
4 Open Web Application Security Project, SQL Injection, *https://www.owasp.org/index.php/SQL_Injection*