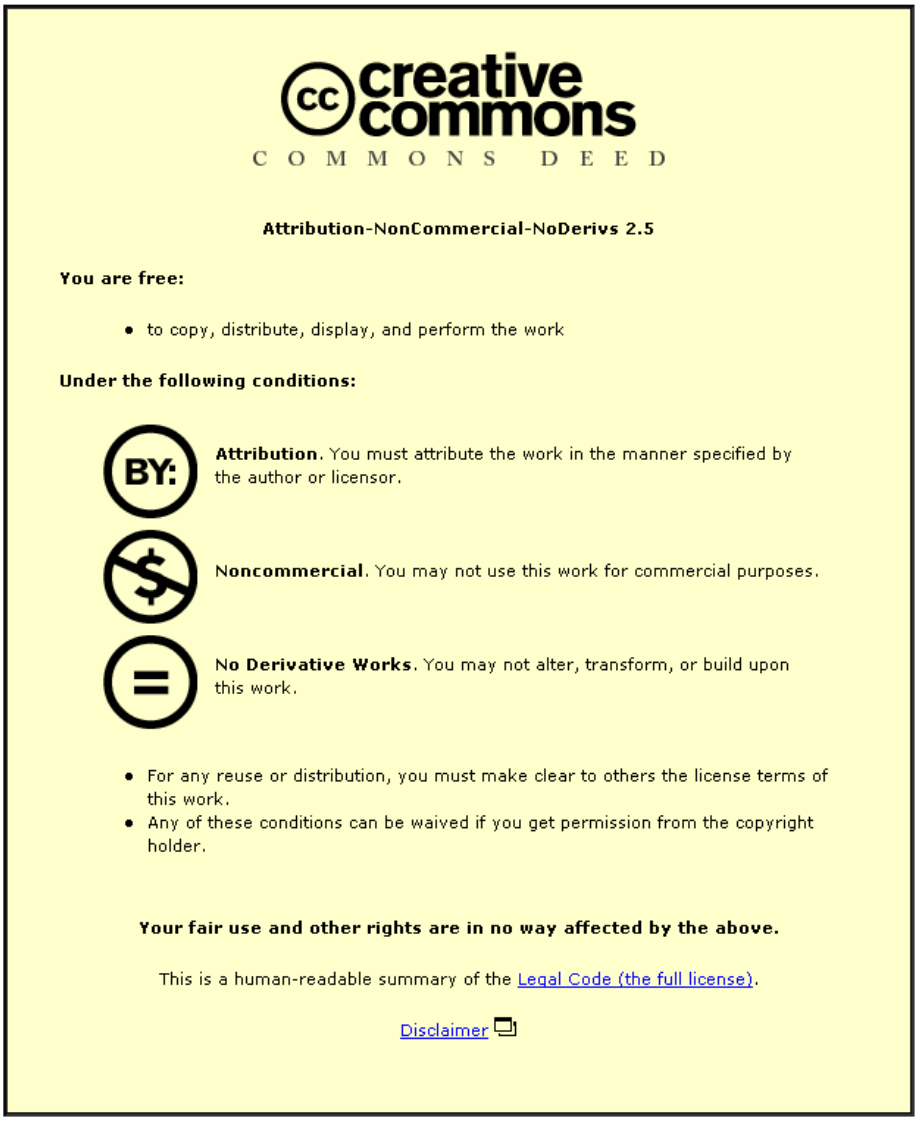

creative commons

COMMONS DEED

Attribution-NonCommercial-NoDerivs 2.5

You are free:

- to copy, distribute, display, and perform the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor.

Noncommercial. You may not use this work for commercial purposes.

No Derivative Works. You may not alter, transform, or build upon this work.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

This is a human-readable summary of the Legal Code (the full license).

Disclaimer

Augmented Attack Tree Modeling of SQL Injection Attacks

Jie Wang, Raphael C.-W. Phan, John N. Whitley and David J. Parish
High Speed Networks Research Group
Department of Electronic and Electrical Engineering,
Loughborough University, LE11 3TU, UK
Email: {J.Wang3, R.Phan, J.N.Whitley, D.J.Parish}@lboro.ac.uk

***Abstract*—The SQL injection attacks (SQLIAs) vulnerability is extremely widespread and poses a serious security threat to web applications with built-in access to databases. The SQLIA adversary intelligently exploits the SQL statement parsing operation by web servers via specially constructed SQL statements that subtly lead to non-explicit executions or modifications of corresponding database tables. In this paper, we present a formal and methodical way of modeling SQLIAs by way of augmented attack trees. This modeling explicitly captures the particular subtle incidents triggered by SQLIA adversaries and corresponding state transitions. To the best of our knowledge, this is the first known attack tree modelling of SQL injection attacks.**

Keywords-Augmented Attack Tree; SQL Injection Attack; Modelling

I. INTRODUCTION

Since web applications have become one of the most important communication channels between service providers and clients, more script kiddies and sophisticated hackers target victims either for fun, commercial reasons or personal gain. The increasing frequency and complexity of web based attacks has raised awareness of web application administrators of the need to effectively protect their web applications. The OWASP 2010 report places Injection Attacks, including SQLIAs, as the most likely and damaging [9]. SQLIAs are caused by attackers inserting a malicious SQL query into the web application to manipulate data, or even to gain access to the back-end database. The main reason contributing to the successful SQLIAs is due to bad web application design and implementation. In design and development phase, SQLIAs can be prevented by some adherence to guidelines including: validation of inputs before passing them to the database; the use of safe SQL statements for data access; and providing no error messages [4].

In assessing the risk of a web based system against SQLIAs and in facilitating detection of such intrusions, there is a need to properly model SQLIAs especially the subtly constructed SQL statements.

In this paper, we model SQLIAs with the Augmented Attack Tree (AAT) [6]–[8] technique and furthermore propose regular expressions as a solution to generically capture the subtle SQLIA-constructed statements. Attack trees describe attacks toward a system as a construction of atomic attacks modelled as states that an attack must go through to achieve success [2]. Such trees focus on the analysis of measurable goals that can ultimately be translated into specific tests against real-world implementations. While conventional attack trees have been widely used [1]–[3], [5], [10], they only concentrate on displaying the states of an attack. We argue that for SQLIAs it is crucial to explicitly model the transition process (edge of the attack tree) between states as well, and therefore conventional attack trees cannot provide sufficient information for analysis of SQLIAs. Our SQLIA modeling based on the Augmented Attack Tree allows to explicitly link regular expressions capturing generic signatures to different types of SQLIAs.

The rest of this paper is organized as follows. Section 2 provides background information about SQLIAs and describes the attack tree modeling techniques including the conventional attack tree and augmented attack tree. Section 3 shows our proposed SQLIA modeling. Section 4 reviews the related works. Finally in Section 5, conclusion and further work are presented.

II. PRELIMINARIES

Wherever Times is specified, Times Roman or Times New Roman may be used. If neither is available on your word processor, please use the font closest in appearance to Times. Avoid using bit-mapped fonts if possible. True-Type 1 or Open Type fonts are preferred. Please embed symbol fonts, as well, for math, etc.

A. Background on SQL Injection Attacks

SQLIAs are command-injection attacks where the attacker injects a malicious SQL query into back-end database through web application interface. The back-end database executes the system defined SQL statement with injected SQL query, and sends the corresponding execution results back to the attacker. The attacker could submit malicious SQL commands directly to the back-end database to extract confidential information or even obtain the root privilege of database.

SQLIAs are classified into seven types: Tautologies; Illegal/Logically Incorrect Queries; UNION Query; Piggy-Backed Queries; Stored Procedures; Inference and Alternate Encodings [13]. There are ten kinds of attack goals of SQLIAs: identify injectable parameters; identify database finger-prints; determine database schema; extract data; add or modify data; perform denial of service; evade detection; bypass authentication; execute remote commands; and escalate privilege [13]. Any of the attack goals can be achieved by any of the types, so there are a large combination of attacks.

978-1-4244-5265-1/10/$26.00 ©2010 IEEE

B. Attack Tree Modeling

1) **Conventional Attack Trees**. Attack trees were defined to model threats against computer network sys- tems [11]. According to varying attacks, attack trees provide a formal, methodical way to describe the security of system. The tree structure is utilized to represent attacks against a system, with the root node representing the ultimate attack goal and the branches representing ways to achieve the goal. Two connection types, OR and AND, are used to connect multiple child nodes with same parent node. OR means that the goal can be reached if any one of the subgoals is reached, whereas AND means that the goal can be reached only if all of subgoals are reached.

Construction of an attack tree depends on the expertise of the analyst. Any designing errors result in a flawed attack tree and lead to incorrect analysis. An attack tree is built from the attacker's point of view, therefore, the attack tree analyst must conceive as an attacker with infinite resources, knowledge and skill [2].

2) **Augmented Attack Tree.** Augmented Attack Tree (AAT) [6]–[8], [14] extended the conventional attack tree by associating with each branch a sequence of malicious operations that could have been used in the attack. The formalization of AAT is stated as follows [7], [8]:

Definition 1. *An augmented attack tree is a rooted labeled tree given by AAT= <V, E, ε, Label, $SIG_{u,v}$>, where*

- V is the set of nodes in the tree representing the different states of partial compromise or sub-goals that an attacker needs to move through in order to fully compromise a system. v ∈ V is a special node, distinguished from others, that forms the root of the tree. It represents the ultimate goal of the attacker, namely system compromise. The set V can be partitioned into two subsets, leaf nodes and internal nodes, such that
- leaf_nodes ∪ internal_nodes=V,
- leaf_nodes ∩ internal_nodes=Φ *;,*
- v ∈ internal nodes
- $E \subseteq V \times V$ constitutes the set of edges in the attack tree. An edge <u,v> ∈ E defines an atomic attack and represents the state transition from a child node v to a parent node u, u, v ∈ V. An atomic attack is a sequence of incidents. The edge <u, v> is said to be "emergent from" v and "incident to" u.
- ε is a set of tuples of the form <v, decomposition> such that
- v∈ internal nodes and
- decomposition ∈ [AND-decomposition, OR-decomposition]
- Label is the name of the exploit associated with each edge.
- SIGu,v is an attack signature which is defined as
- Definition 3 below.
- Definition 2. An incident-choice is a group of related incidents, the occurrence of any one of which can contribute towards the state transition in the attack tree.
-
- Definition 3. An attack signature SIGu,v is a sequence of incident-choices (incident-choice1, incident-choice2, . . ., incident-choicen) such that the sequence (incidenti,1, incidenti,2, . . . , incidentm,n) constitute an atomic attack.
- Modeling SQL Injection Attacks
- This section shows our modelling of SQLIAs with AAT and regular expressions. Of the seven well known SQLIA types [13], we model: Tautology Query; Logically Incorrect Query; UNION Query; Piggy-Backed Query; and Timing Inference Query. We ignored to model Stored Procedures, Alternate Encodings and Blind Injection Inference Query because of the following: Stored Procedures provides the functionality to consolidate and centralize logic that was originally implemented in applications. As such the content of a stored procedure is not distinct from other SQLIA at- tacks, so cannot be modeled separately. Alternate Encodings provide different coding practices and is used in conjunction with other SQL based attacks. The attacker injects a Blind Injection Inference Query mainly dependent on individual's intuition and experience without the assistant of database feedback information. The injection contents are the same as other SQLIA methods. Therefore, we focus on modeling with above mentioned five types. In this section, we first discuss the mechanism of modeling. Then, we model each type of SQLIAs in turn.
-
- Modeling Mechanism
- The modeling of SQLIA obeys the definition of AAT. For lack of a better name, we term our modeling result as Augmented SQL Injection Attack Tree (ASQLIAT).
- Node. A node in ASQLIAT represents the state. The child node represents the state during whole attack procedure. The root node represents SQLIA as the ultimate goal. Since the first step of computer network attack is usually to identify the attack object, we assume that the leaf node in all branches is the state that the attacker found out the targeted web application. There is a pair of brackets containing the symbol to distinguish each node. Those symbols indicate the parent node or child node in signature.

Edge. An edge in ASQLIAT represents the state transition from the child node to the parent node with a set of incidents.

Incident. An incident in ASQLIAT represents either the injection behavior or the web server execution behavior.

Label. A label in ASQLIAT represents the edge name.

Signature. An SQLIA signature is the related regular expression that captures the SQL statement constructed by

the SQLIA adversary to mount the particular type of SQL injection attack. For clarity of illustration, we model the signature with Perl's regular expression style [12] (although any regular expression syntax can be used w.l.o.g). Table 1 illustrates the elements of a modelled SQLIA signature and corresponding expression symbols. It is important to note that square brackets indicate the matching of any characters inside them, round brackets indicate the matching of the whole pattern inside them and plus mark indicates character matching more than 1 time. In addition, we utilize a pair of curly brackets to include a set of possible incidents or signatures in each edge.

TABLE I. . ELEMENTS AND EXPRESSIONS OF SIGNATURE

Element	Symbol
Alphanumeric	\w
Comment Mark	(\-\-)
Quotation Mark	['"]
Comparison Mark	([=<>]|[<>!]=)
Logical Keyword	(OR)
Type	(int|char|varchar)
Type Conversion Keyword	(CONVERT|CAST)
SQL Keyword	(SELECT|INSERT|UPDATE|DROP)
UNION Keyword	(UNION | UNION ALL)
Delimiter Mark	;
Delay Tim	\d+
White Space	\s
Bracket	\c
Case Insensitive	/i

Figure 1 illustrates the generated ASQLIAT. The leaf node, i.e. Found Web Application denotes the state where the adversary has found a particular attack target. Then, the attacker performs some actions to explore suitable places to inject command(s) into the database through web applications. Usually, the attacker injects the malicious command through either any input forms on the web application or via the URL header. Therefore, the transiting incidents along the edge can be taken from the set {Web Page Form Access, URL Header Access}; and upon successful execution of either incident, the adversary progresses to the next state: Found Injection Place. The expression below shows the incidents and their corresponding signatures. In signature, 2 indicates the second state as parent node and 1 indicates the first state as child node.

Incident ∈ {Web Page Form Access, URL Header Access}

$SIG_{2,1} \in \{SIG_{Web}, SIG_{URL}\}$

C. Tautology Query

The Tautology Query attack injects a piece of malicious code into one or more conditional statements so that they always evaluate to true and generate the result according to the evaluated true condition. Branch (1) in Figure 1 illustrates the generated Tautology Query in ASQLIAT. It is clearly show that it contains four states and three edges. The second edge which resides in between the state Found Injection Place and Web Server Execute Tautology Query is the Tautology Query Injection. For this kind of attack, the most significant feature of injection content is containing three key parts: OR; true condition; and comment mark. In the generated signature, it must include those three parts. In order to model and identify this attack is Tautology attack; the attacker injected code should meet the defined signature, which is shown as below,

Incident ∈ {Tautology Query Statement}

$SIG_{T3,\,T2} \in$ {/['"]\s+(OR)\s+\w+\s*([=<>]|[<>!]=)\s*\w+ \-\-/i}

When the state of Web Server Execute Tautology Query happened, the only way to transit into SQLIA state is through the edge Tautology Attack. There must be some incidents to indicate the achieving of SQLIA. The set of is {Bypass Authentication, Information Retrieval}. Once either of the incidents happen, the attack state achieve.

Incident ∈ {Bypass Authentication, Information Retrieval}

$SIG_{R,\,T3} \in \{SIG_{BA}, SIG_{IR}\}$

D. Logically Incorrect Query

Logically Incorrect Query attack is the attacker intent to obtain the error feedback message by injecting incorrect command into the database. The database structure and type information can be extracted according to the error message. Branch (2) in Figure 1 displays the generated Logically Incorrect Query branch in ASQLIAT. The labels of second edge and third edge are Logically Incorrect Query and Logically Incorrect Query Attack. The trick of this SQLIA is changing the data type of injected data and providing different data type against system defined data type. Database server returns error feedback message which can assist the attacker to further explore database server. Keywords to change different data type are CONVERT and CAST and either of them must be exist in the query. Therefore, the signature to detect this attack must contain those keywords. Once the attacker triggers the incident in second edge, the state transits into the state of Web Server Execute Logically Incorrect Query. The generated signature is stated as follows,

Incident ∈ {Logically Incorrect Query Statement}

$SIG_{LI3,LI2} \in$ {/(CONVERT|CAST)\s*\c+\s*(int|char| varchar)/i}

For the last edge which leading to SQLIA state, it's incident contains Return Error Message from Database and Information Retrieval.

Incident ∈ {Return Error Message from Database, Information Retrieval}

$SIG_{R,LI3} \in \{SIG_{EM}, SIG_{IR}\}$

E. UNION Query

The UNION Query attack is to inject UNION keyword following with another SELECT query statement. The result is database returns the dataset that is the union results of the original first query and the injected second query. The generated UNION Query branch is shown as Branch (3)

in Figure 1. The label of second edge is *UNION Query Injection* and the label of top edge is *UNION Query Attack*. The most important keywords in the injected code are UNION and SELECT. So, the generated signature is:

Incident ∈{UNION Query Statement}

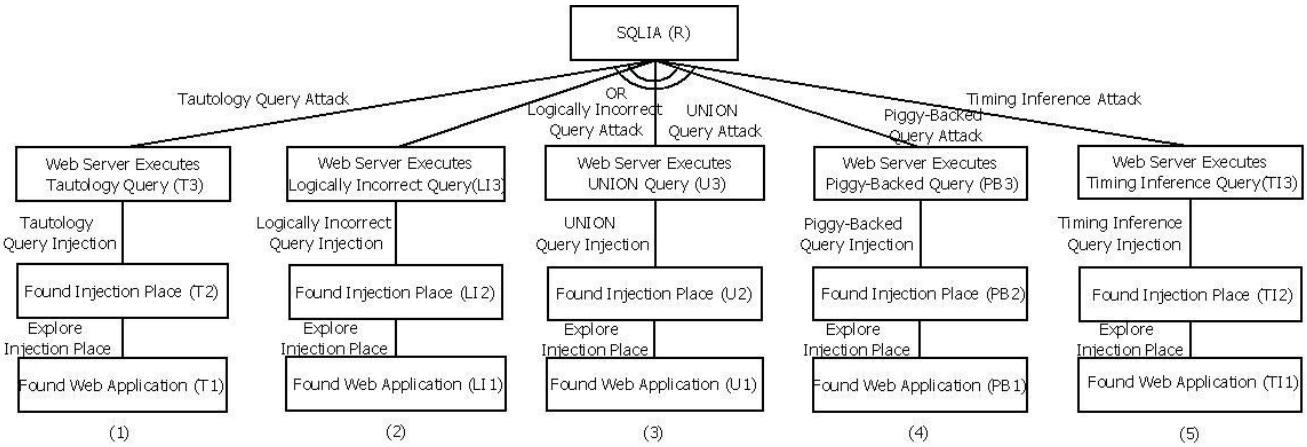

Figure 1. Augmented Attack Tree Modeling of SQL Injection Attacks

$SIG_{U3,U2}$ ∈{/["]\s+(UNION|UNION\s+ALL)\s+(SELECT) /i}

The label in top edge is UNION Query Attack. The possible incidents are Information Retrieval and Bypass Authentication.

Incident∈{Information Retrieval, Bypass Authentication}

$SIG_{R,U3}$ ∈{SIG_{IR} , SIG_{BA} }

F. Piggy-Backed Query

In Piggy-Backed Query attack, the query be extened by injecting additional queries after the original one. Consequently, the database receives multiple SQL queries and executes them in sequence. Branch (4) in Figure 1 illustrates the Piggy-Backed Query branch in ASQLIAT. The incident in the second edge, which label is Piggy-Backed Query Injection, is to inject multiple queries into the original one. Because Piggy-Backed Query combines several SQL query together, there must be some data manipulation keywords or data definition keywords. It's wise to model the signature with those keywords following with a delimiter which indicates the ending of previous query.

Incident∈{Piggy-Backed Query Statement}

$SIG_{PB3, PB2}$∈{/['"]\s*;\s*(SELECT|INSERT|UPDATE| DELETE |DROP)/i}

The incidents in the top edge, which label is Piggy-Backed Query Attack, can be Information Retrieval, Information Modification and Perform DoS.

Incident ∈ {Information Retrieval, Information Modifica- tion and Perform DoS}

$SIG_{R, PB3}$ ∈{SIG_{IR} , SIG_{IM} , SIG_{DoS} }

G. Timing Inference Query

The inference attack implemented according to the obtained result from a true or false evaluation about data values in the database. In this kind of attack, the target web server has been secured enough so that there is no enough or usable feedback error messages. In this situation, the attacker injects malicious command into the web server and then observes how the website changes.

For Timing Inference Query, the attacker injects query with both if/then evaluation statement and delay time. The attacker then obtains information from the database by monitoring the timing delay as the response of the database. If the timing delay take place, it means that the injected if/then evaluation statement been executed successfully by database server. Otherwise, the statement is wrong and need further modification. The last branch in Figure 1 displays the modeling of this kind of SQLIA. The label of edge between the state Found Injection Place and the state Web Server Execute Timing Inference Query, is Timing Inference Query Injection. It is important to contain the keyword WAITFOR and the length of waiting time in the generated signature. The generated signature is stated as follows,

Incident∈{Timing Inference Query Statement}

$SIG_{TI3, TI2}$ ∈{/(WAITFOR)\s+\d+/i}

Moreover, the label of top edge is Timing Inference Attack. The possible incidents are Information Retrieval, Information Modification and Identify Database Scheme.

Incident ∈ {Information Retrieval, Information Modification and Identify Database Scheme}

$SIG_{R, TI3}$ ∈{SIG_{IR} , SIG_{IM} , SIG_{DS} }

III. RELATED WORKS

Poolsapassit and Ray [6] proposed another Augmented Attack Tree (AAT). AAT [6] applied the conventional attack tree with extra attack probability labels. A label l is associated with a node S, given by the tuple <n, m> where m and n are positive integers greater than 0 with n≤m. m is termed the least effort to compromise subgoal S while n is termed the number of currently compromised subgoals under

S. The ratio n/m provides the measure of how far an attacker has progressed towards the ultimate goal in terms of the least effort along the most advanced attack path that s/he has been through. Wang et al. [14] improved augmented attack tree [6] with a notion of "minimal attack tree" and proposed a new trimming attack tree algorithm to reduce the redundant branches.

Byres et al. [1] described the first work of how the attack tree methodology be implemented to the SCADA protocol MODBUS/TCP with risk metrics. Their attack tree analysis was qualitative used to identify level of technical difficulty, severity of impact, probability of detection and the underlying critical vulnerabilities. Lin et al. [3] applied attack tree to model the threat of Cross Site Request Forgery (CSRF) attacks. Morais et al. [5] utilized attack tree to describe known attacks and derive injection test scenarios to evaluate the security properties of the protocol. Khand [2] presented the syntax and graphical representation of five kinds of new nodes as the extension of the conventional attack tree to model the security of systems. Saini et al. [10] constructed attack tree to study and evaluate the security of MyProxy system with all the possible threats, the cost of attack execution by attacker and the cost of damage from attack.

IV. CONCLUSION

In this paper, we have presented a method for modelling SQLIAs with the Augmented Attack Tree and regular expressions to capture subtle SQL statements formed by SQLIA adversaries. This approach is generic, thus it can be made to equally apply to other kinds of web based attacks.

REFERENCES

[1] E. J. Byres, M. Franz, and D. Miller. The Use of Attack Trees in Assessing Vulnerabilities in SCADA Systems. In IEEE International Infrastructure Survivability Workshop (IISW'04), Lisbon, Portugal, 2004.

[2] P. A. Khand. System Level Security Modeling Using Attack Trees. In 2nd International Conference on Computer, Control and Communication, 2009. IC4 2009, pages 1–6, February 2009.

[3] X. Lin, P. Zavarsky, R. Ruhl, and D. Lindskog. Threat Modeling for CSRF Attacks. In International Conference on Computational Science and Engineering, 2009. CSE '09, volume 3, pages 486–491, August 2009.

[4] S. Madan and S. Madan. Shielding Against SQL Injection Attacks Using ADMIRE Model. In First International Conference on Computational Intelligence, Communication Systems and Networks, 2009. CICSYN '09., pages 314–320, July 2009.

[5] A. Morais, E. Martins, A. Cavalli, and W. Jimenez. Security Protocol Testing Using Attack Trees. In International Conference on Computational Science and Engineering, 2009. CSE'09, volume 2, pages 690–697, August 2009.

[6] N. Poolsapassit and I. Ray. Using Attack Trees to Identify Malicious Attacks From Authorized Insiders. Lecture notes in computer science, 3679:231–246, 2005.

[7] N. Poolsapassit and I. Ray. Investigating Computer Attacks Using Attack Trees. IFIP International Federation for Information Processing, 242:331–343, 2007.

[8] N. Poolsapassit and I. Ray. A Systematic Approach for Investigating Computer Attacks Using Attack Trees. the 3rd IFIP TC-11 WG 11.9 Working Conference on Digital Forensics, January 2007.

[9] The Open Web Application Security Project. Owasp 10-2010. http://www.owasp.org/images/0/0f/OWASP T10 -2010 rc1.pdf.

[10] V. Saini, Q. Duan, and V. Paruchuri. Threat Modeling Using Attack Trees. Journal of Computing Sciences in Colleges, 23(4):124–131, 2008.

[11] B. Schneier. Attack Trees. Dr. Dobb's Journal, 24(12):21–29, 1999.

[12] R. L. Schwartz, T. Phoenix, and B. D. Foy. Learning perl. O'Reilly Media, Inc., 2005.

[13] J. Viegas, W. Halfond, and A. Orso. A Classification of SQL-Injection Attacks and Countermeasures. In International Symposium on Secure Software Engineering, 2006.

[14] H. Wang, S. Liu, and X. Zhang. An Improved Model of Attack Probability Prediction System. Wuhan University Journal of Natural Sciences, 11(6):1498–1502, 2006.