**Notice of Violation of IEEE Publication Principles**

**"A Petri-net Model of Network Security Testing"**
by Jun-long Yan, Ming-xin He, Tie-yuan Li
in the 2011 IEEE International Conference on Computer Science and Automation Engineering (CSAE), 2011, pp. 188 – 192

After careful and considered review of the content and authorship of this paper by a duly constituted expert committee, this paper has been found to be in violation of IEEE's Publication Principles.

This paper contains significant portions of original text from the paper cited below. The original text was copied with insufficient attribution (including appropriate references to the original author(s) and/or paper title) and without permission.

Due to the nature of this violation, reasonable effort should be made to remove all past references to this paper, and future references should be made to the following article:

**"Attack Net Penetration Testing"**
by J.P. McDermott
in the Proceedings of the 2000 Workshop on New Security Paradigms (NSPW 2000), 2001, pp. 15 – 21

# A Petri-net Model of Network Security Testing[1]

Jun-long YAN Ming-xin HE Tie-yuan LI
1. GuangZhou Information Technology Research Institute, Jinan University, GZ, CHINA
2. GuangZhou ShengTong Quality Testing of Construction co., ltd, GZ, CHINA
e-mail: sinber@yahoo.cn e-mail: mx.he@yeah.net e-mail: lty@stjc.com.cn

*ABSTRACT*—**Given the increasing dependence of our societies on networked information systems, the overall security of these systems should be measured and improved. Existing security metrics have generally focused on measuring individual vulnerabilities without considering their combined effects. In this paper, we proposed a Petri-net based model. It retains key advantages of the flaw hypothesis and attack trees approaches while providing some new benefits. This novel model provides a theoretical foundation and a practical framework for continuously measuring network security in a dynamic environment.**

*Keywords*: ***Petri-net; Network Security; Testing***

## I. INTRODUCTION

Our society has become increasingly dependant on the reliability and proper functioning of a vast number of interconnected information systems. To improve the security of these systems, it is necessary to measure the amount of security provided by different system[1]. Existing security metrics[2] have generally focused on measuring individual vulnerabilities without considering their combined effects.

A large part of security testing is art rather than science. The effectiveness of security testing depends on the skill and experience of the testers. Security testers need firm grounding in the first principles of information security but they also need an almost encylopedic knowledge of product or system trivial that have little apparent relationship to principles. Security testing also requires a special kind of insight that cannot be systematized.

In spite of this, there are widely used process models for security testing. Security testers that follow these models are more effective in their use of resources. Security testing process models are structured around some paradigm that organizes the discovery of potential attacks on the live system. In this paper we describe a new process model for security testing that uses the Petri-net[3][4] as its paradigm. Surprisingly, this approach provides increased structure to flaw generation activities, without restricting the free range of inquiry. This technique is particularly useful for organizing security testing by means of distributed or cooperative attacks. It also has the nice properties of easily depicting both refinement of specific attacks and attack alternatives in a manner similar to attack trees.

## II. PRELIMINARIES

### *A. Security Testing*

Security testing is a fundamental area of information system security engineering. It usually follows one of two approaches: flaw hypothesis[5]or attack trees[6][7]. We describe the basic flaw hypothesis approach here as a sequence of six activities:(1)Define security testing goals. (2) Perform background study. (3) Generate hypothetical flaws. (4) Confirm hypothesis. (5) Generalize discovered flaws. (6) Eliminate discovered flaws.

The flaw hypothesis approach[8] defines "a flaw as a demonstrated undocumented capability, which can be exploited to violate some aspect of the security policy". Security testing is first planned by setting the scope, establishing ground rules and objectives, and defining the purpose of the testing. Next, a background study is performed using all available resources. This background study may include system design documentation, source code, user documentation, and results of unit and integration testing. After the background study is complete, the security test team generates hypothetical flaws using brainstorming sessions.

**978-1-4244-8728-8/11/$26.00 ©2011 IEEE**

[1] Funded by: 1.Guangdong Emergency Technology Research Center of Risk Evaluation and Prewarning on Public Network Security Project (2010A032000002). 2.Network Information Security Testing, Monitoring and Protecting Service Project(06121040B0270124/3)

The generated list of flaws is analyzed, filtered, and ordered according to priority. The hypothetical flaws are then confirmed or refuted by test or source code analysis. The confirmed flaws are then analyzed for patterns, to see if the mistake that lead to the flaw might have been repeated elsewhere in the system. Finally, recommended fixes for the flaw are developed. Many of the concepts used in other forms of security testing originated with the flaw hypothesis approach, including assessment of the hypothetical attacker's motives, behavior, and goals; assigning priorities in order to perform the most critical tests first; and the importance of personal ethics for security testers.

The attack tree approach [7] was developed at Sparta. The attack tree approach is intended for security testing where there is less background information about the system to be tested. The basic idea is a combination of the work breakdown structure from project management and the familiar tree representation of a logical proposition. There are several ways to combine these ideas into an attack tree; we will describe a typical attack tree approach that starts with the target or goal as the root of the tree. Our example attack tree is taken from B.Schneier [7] and shows an attack on a physical safe. The root and most of the nodes in the tree are disjunctive nodes. The attack is accomplished if any of the actions described by its children are accomplished. Disjunctive nodes represent alternative attacks.

For the example shown in Fig. 1, the action Get Combo From Owner may be accomplished either by threats, blackmail, eavesdropping, or bribery. The lowest interior node, Eavesdrop, is a conjunctive node and requires all of its children to be accomplished before it is considered accomplished. In the example attack on the safe, the penetrators must not only listen for the combination, they must also trick the owner into repeating it out loud while the eavesdropping is taking place. Conjunctive nodes represent decomposition or refinement of a specific attack. The nodes of attack trees can be assigned various attributes, such as cost or likelihood, in order to analyze a given penetration testing situation and assign priorities to certain attacks. The power of the tree structure for showing decomposition is particularly helpful in organizing security tests of undocumented products or of operational systems. Attack trees are a top-down approach to security testing.

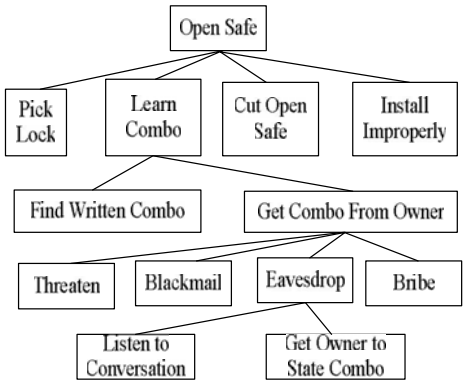


Figure 1: Representative Attack Tree

*B. Petri net[3][4]*

A Petri net graphis a 3-tuple $(S,T,W)$, where: S is a finite set of places; T is a finite set of transitions; S and T are disjoint, i.e. no object can be both a place and a transition. $W:(S\times T)\bigcup(T\times S)\to \mathrm{N}$ is a multiset of arcs, i.e. it defines arcs and assigns to each arc a non-negative integer arc multiplicity; note that no arc may connect two places or two transitions. The flow relation is the set of arcs: $F=\{(x,y)\,|\,W(x,y)>0\}$. In many textbooks, arcs can only have multiplicity 1, and they often define Petri nets using F instead of W.

A Petri net graph is a bipartite multidigraph $(S\bigcup T,F)$ with node partitions S and T. The preset of a transition t is the set of its input places: ${}^{\bullet}t=\{s\in S\,|\,W(s,t)>0\}$; its postset is the set of its output places: $t^{\bullet}=\{s\in S\,|\,W(t,s)>0\}$.

A marking of a Petri net (graph) is a multiset of its places, i.e. , a mapping $M:S\to \mathrm{N}$. We say the marking assigns to each place a number of tokens. A Petri net (called marked Petri net by some, see above) is a 4-tuple $(S,T,W,M_0)$, where (S,T,W) is a Petri net graph; M0 is the initial marking, a marking of the Petri net graph.

## III.TESTING MODEL

*A. Attack Net*

Our proposed new approach is to organize security testing according to an attack net. An attack net is a (disjunctivel) Petri net with a set P={ p0, p l, p2 . . . . pn } of places representing interesting (e. g. control or knowledge of) states or modes of the security relevant entities of the system of interest. The attack net also has a set T={t0, tl, t2, . . . , tin} of transitions that represent input events, commands, or data that cause one or more security relevant entities to change state. Places are connected to transitions and transitions are connected to places by directed arcs. The attack net has a set of tokens. Tokens move from place to place along the directed arcs to indicate the progress of the attack. If a token is at a place, then the attacker has gained control of the corresponding entity, in the state represented by the place. If place pi precedes place pj in the attack net, then the attack must achieve the control or knowledge represented by place pi before the control or knowledge represented by place pj is possible.

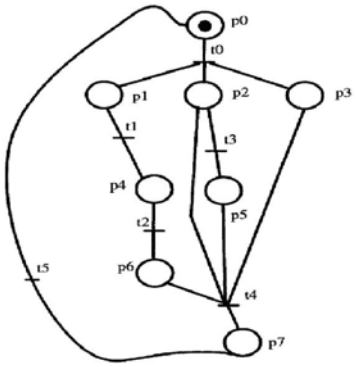


Figure 2 Attack Tree

An example will clarify this. Fig. 2 shows an attack tree representation of the so-called Mitnick attack[9], a combination of SYN flooding, TCP session hijacking, and Unix. Rhosts trust relationship spoofing. Place p0 represents the starting state of the exploit: the attacker with root access to some host on a TCP/IP network. Transition to represents an initial reconnaissance of the target system by the attacker, using finger, showmount, rcpinfo, ping, etc. A successful reconnaissance replicates the token into all three places pl, p2, and p3, concurrently. Place pl represents identification of a routable but unused address on the target network. Place p2 represents the identification of a trusted host and place p3 represents the identification of a trusting. Transition tl represents the construction of a SYN flood packet with a false source address, by the attacker. Transition t3 represents investigation of the trusted host's TCP sequence numbers by the attacker. Place p5 represents the attacker's ability to predict the trusted host's TCP sequence numbers. Transition t2 represents the initiation of a SYN flood attack on the trusted host and place p6 represents the system state when the trusted host's queues are flooded.

Now the attack net looks like Fig. 3, below. Transition t4 is the spoofing of a TCP session with the trusting host; place p7 represents the system state where the trusting host has established a trusted TCP session with the attacker. We include, just for nice, a transition t5 that represents the modification of the trusting host's. Rhosts file, which will give the attacker root access on the trusting host, thus allowing repetition.

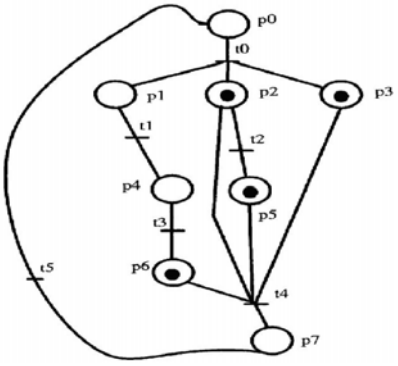


Figure 3 Trusted Host SYN Flooded

Attack nets do not need to have cycles describing recursive attacks. In most cases, they will not, since security testing seeks to improve the development or operation of systems. Establishment of a successful initial violation of some policy or security model of a system is sufficient to achieve that goal. The key features of attack nets are: (1)modeling concurrency and attack progress with tokens. (2)modeling intermediate and final objectives as places. (3)modeling commands or inputs as transitions. Attack nets are not intended to model the actual behavior of a system or component during an attack, in the sense of attack signaturesused by intrusion detection systems or the abuse cases used by abuse case based assurance arguments. They are used to organize the development of plausible attack

scenarios. Attack nets are a notation for discovering and discussing scenarios under development.

*B. Petri-Net Model and Case Study*

Although it might seem more reasonable to organize attack net security testing along lines similar to attack tree testing, in fact it is better to use an approach similar to flaw hypothesis testing. We will see why shortly, but for now we propose the following high-level organization for an petri-net based security testing project:(1)Define goals. (2)Background study. (3)Attack net generation. (4)Hypothesis verification. (5)Flaw generalization. (6)Flaw elimination.

All of the steps we use are the same as the flaw hypothesis approach, except we use petri-net based attack nets to construct hypothetical flaws or attacks. In our limited experience, the best way to construct an petri net is to start with a flaw database or background study and construct attack subnets. Most security flaws can be described (or hypothesized) as a single command or action that takes a product or system into an undesirable state. We model these as (the smallest possible nonempty) disjoint attack nets: two places connected by a single transition, as show by Fig. 4 below.

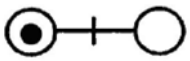

Figure 4:Intial Subtree Used to Model Individual Flaws

As an example of what we mean, we will show how a specific flaw can be translated into one of these initial single transition attack nets. Our example flaw concerns linuxconf, a graphical user interface for administration of the Linux operating system. In versions of linuxconf that were in use at the time this paper was written, there is a flaw that allowed ordinary users to retain the privilege of shutting down or restarting the system, after the linuxconf tool had configured their permissions to deny this. This was an implementation flaw that allowed an unsafe condition to arise. The administrator had established a policy that user x may not restart the system, but user x still had that privilege. This did not result in compromise of an account with root privileges, or even unauthorized disclosure. However, it may be used in conjunction with other flaws to compromise an account with root privileges. We depict this as the following single-transition attack net.

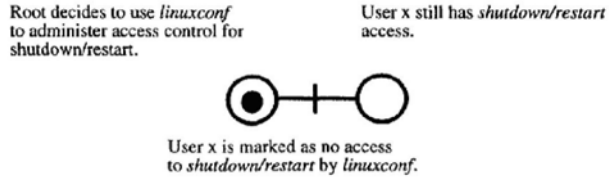


Figure 5: Linuxconf Flaw

Once we have a sufficient collection of single-transition attack nets, we then examine relationships between the interesting places by attempting to arrange transitions between them. In this way we can build up an attack tree that indicates the possibility of more serious flaws. The notation is sufficiently flexible to denote a wide range of hypothetical or confirmed flaws during brainstorming sessions. The notation also serves as a record of complex dependencies or causal relationships that may be suggested during the formulation of a hypothesis. We mentioned earlier that we would use attack nets for penetration testing in circumstances similar to those most suited to the flaw hypothesis approach, rather than the attack tree approach. We suggest this because attack nets are so useful for investigating combinations of flaws. This type of activity is more in keeping with the hypothesis generation process. Attack nets can be a powerful bottom up approach to security testing. While we could also use them top-down, it is not clear what the benefit would be. It is true that the diagrams can be more descriptive than attack trees. Consider the leaf nodes Listen to Conversation and Get Owner to State Combo of our example attack tree, shown in Figure 1. The conjunctive node indicates that these two actions must be accomplished together in order to complete the action Eavesdrop, but there is no indication of sequence or dependency. The assumed semantics of the actions makes it clear in this case, but an attack net could depict the situation as shown in Fig. 6.

Since the original example did not have this much detail, we are supposing some conditions. However, it is clear that the attack net model can provide much more information than the attack tree. We can see a necessary sequence: that the action of getting the owner to state the

combination must take place after we have achieved a situation where his conversation is being recorded. We could also be more specific about distinctions between actions and resulting states. Furthermore, an insight may arise from consideration of the place (state) where the owner's conversation is being recorded. We may discover other significant attacks from consideration of a security state where the safe owner's conversation is being recorded.

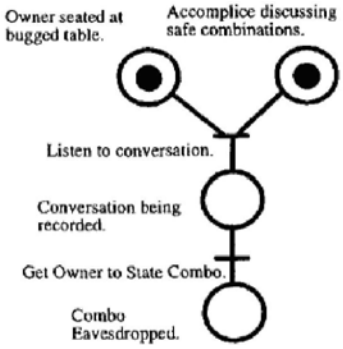


Figure 6: Attack Subtree For Eavedrop Action

## IV. CONCLUSIONS

The petri-net-based attack net approach to security testing is a departure from both the flaw hypothesis model and the attack tree model but it retains the essential benefits of both. Any security testing process is unavoidably dependent upon the flaw hypothesis process model. Any valid security testing process model will retain many of its features and so does the attack net model. Nevertheless, the petri-net-based attack net security testing process brings more discipline to the brainstorming activity without restricting the free range of ideas in any way. The petri-net-based attack nets also provide the alternatives and refinement of the attack tree approach. It is not clear whether attack nets are better than attack trees for top down testing of poorly documented operational systems. It provide a graphical means of showing how a collection of flaws may be combined to achieve a significant system security. This is important since an petri-net-based attack net can make full use of hypothetical flaws. It can model more sophisticated attacks that may combine several flaws, none of which is a threat by itself. The ability to use discovered transitions (i.e. security relevant commands) to connect subnets allows security testing teams to communicate easily about the cumulative effects of several minor flaws. The separation of security test commands or events from the attack states or objectives also increases the descriptive power of this approach. The basic notion of an initial security relevant state, the hostile test input, and the resulting security state is captured by the minimal Petri net representation.

In addition to specifying composition or refinement, petri-net-based attack nets can also model choices. The use of disjunctive transitions allows the movement of some tokens while other places are empty, thus modeling vulnerabilities that could be exploited in several ways or alternative attacks on a single goal. Attack nets can readily depict the precedence and progress relationships in concurrent and distributed attacks. This could be more significant as attacks become more sophisticated by making use of concurrency and cooperation.

Finally, there is an enormous body of literature regarding Petri-nets. It seems reasonable to expect further improvements can be made by applying some of these results.

## REFERENCES

[1] A. Jaquith, Security Metrics Replacing Fear, Uncertainty, and Doubt. AddisonWesley, 2007.

[2] L. Wang, T. Islam, T. Long, A. Singhal, and S. Jajodia, An attack graph-based probabilistic security metric. In Proceedings of The 22nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security (DBSEC'08), Vol.5094, pp. 283-296, DOI: 10.1007/978-3-540-70567-3_22, 2008.

[3] T. Murata, Petri Nets: Properties, Analysis and Applications. In Proceedings of the IEEE, vol. 77, no. 4, pp. 541-580, April 1989.

[4] C. A. Petri, W. Reisig, Petri net. Scholarpedia, vol. 3(4), pp. 64-77, 2008.

[5] WEISSMAN, C. System Security Analysis/Certification Methodology and Results. SP-3728, System Development Corporation, Santa Monica, CA, October 1973.

[6] SALTER, C., SAYDJARI, O., SCHNEIER, B. and WALLNER, J. Toward A Secure System Engineering Methodology. In Proceedings of New Security Paradigms Workshop, Charlottesville, Virginia, September 1998.

[7] SCHNEIER, B. Attack Trees. Dr. Dobbs Journal, December 1999.

[8] WEISSMAN, C. Security Testing. In Handbook for the Computer Security Certification of Trusted Systems. Naval Research Laboratory Technical Memorandum 5540:082a, January 1995.

[9] M. Abedin, S. Nessa, E. Al-Shaer and L. Khan, Vulnerability analysis for evaluating quality of protection of security policies. In 2nd ACM CCS Workshop on Quality of Protection, Alexandria, Virginia, October 2006.