Attack Net Penetration Testing

J.P. McDermott

Department of Computer Science

James Madison University

Harrisonburg, Virginia 22807

mcdermot@cs.jmu.edu

Abstract

The modeling of penetration testing as a Petri net is surprisingly useful. It retains key advantages of the flaw hypothesis and attack tree approaches while providing some new benefits.

1 Introduction

Penetration testing is a critical step in the development of any secure product or system. While many current businesses define penetration testing as the application of automated network vulnerability scanners to an operational site, true penetration testing is much more than that. Penetration testing stresses not only the operation, but the implementation and design of a product or system.

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

In spite of this, there are widely used process models for penetration testing. Penetration testers that follow these models are more effective in their use of resources. Penetration 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 penetration testing that uses the Petri net 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 penetration 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.

2 Penetration Testing

Penetration testing is a fundamental area of information system security engineering. The earliest published open reference to penetration testing is a paper by R.R. Linde [2]. Penetration testing usually follows one of two approaches: flaw hypothesis [6] or attack tree [4]. The flaw hypothesis approach began life as a proprietary testing process of SDC. It is now in general use and remains the best current approach to penetration testing of new products at the end of development. We describe the basic flaw hypothesis approach here as a sequence of six activities:

1. Define penetration 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." Penetration 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 penetration test team generates hypothetical flaws using brainstorming sessions, the Delphi technique, or similar approaches. 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.

The complete flaw hypothesis approach is much richer than the high-level summary we provide here. Many of the concepts

used in other forms of penetration 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 penetration testers.

The attack tree approach was developed at Sparta. The attack tree approach is intended for penetration 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 [5] and shows an attack on a physical safe.

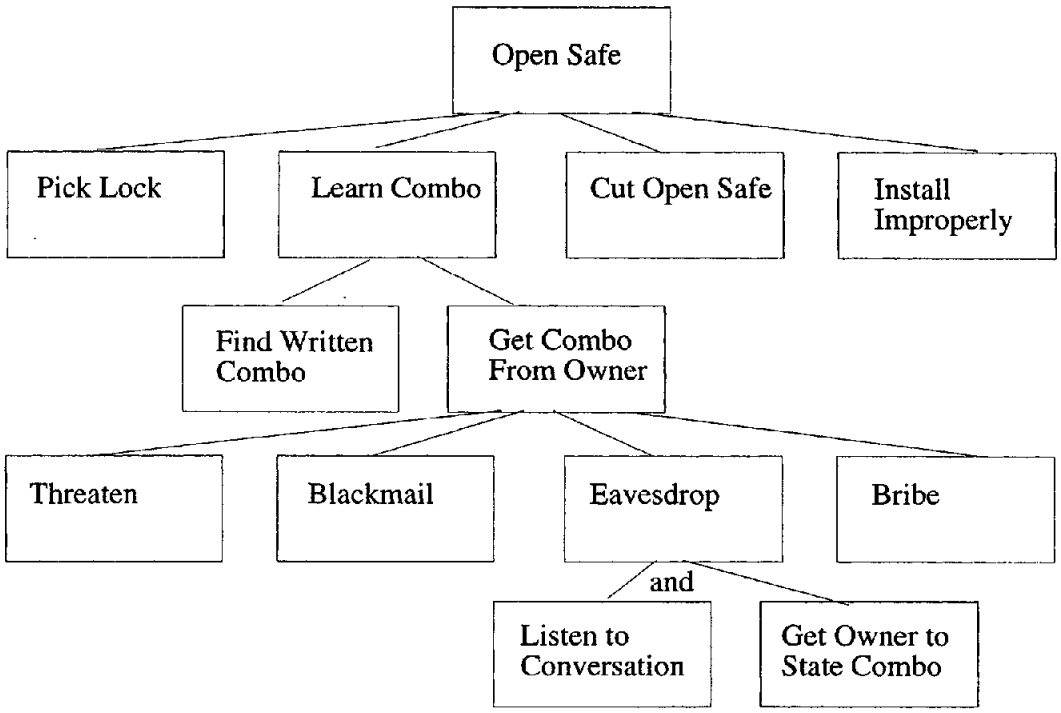

Figure 1: Representative Attack Tree

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 Figure 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 penetration tests of undocumented products or of operational systems. Attack trees are a top-down approach to penetration testing.

3 Attack Nets

Our proposed new approach is to organize penetration testing according to an attack net. An attack net is a (disjunctive1) Petri net with a set P={p0, p1, 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, t1, t2, ..., tm} of transitions that

[1] A disjunctive Petri net allows transitions to fire when at least one of its incoming places has a token, at nodes that are designated as disjunctive nodes. Otherwise, all incoming places must have a token to fire the transition.

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.

An example will clarify this. Figure 2 shows an attack tree representation of the so-called Mitnick attack, 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 t0 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 p1, p2, and p3, concurrently. Place p1 represents identification of a routable but unused address on the target network. (Strictly speaking, the attacker does not have complete control, but the lack of other controls gives the same practical effect.) Place p2 represents the identification of a trusted (spoofable) host and place p3 represents the identification of a trusting (target host). Transition t1 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.

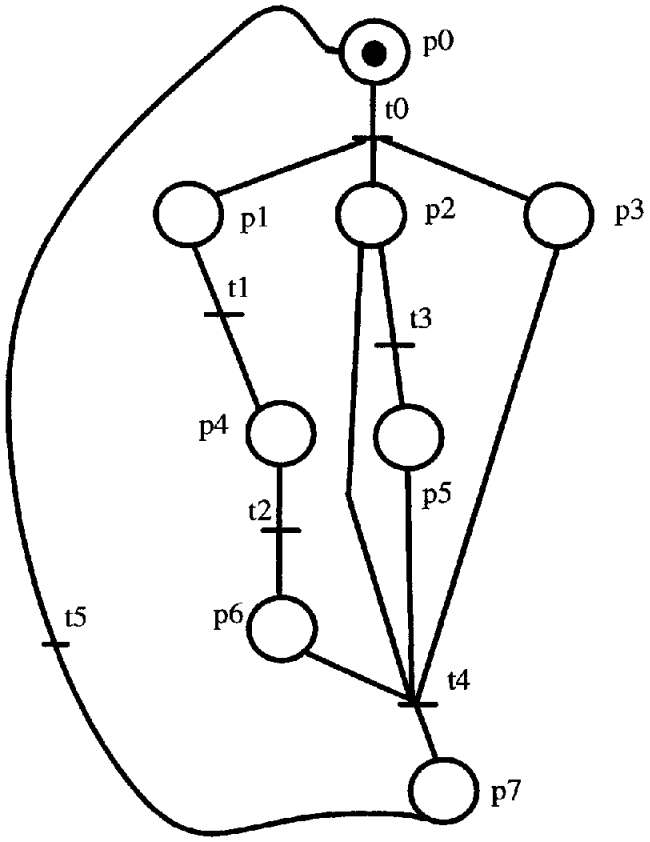

Figure 2: Attack Tree Representation of the Mitnick Attack

Now the attack net looks like Figure 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 (bogus) 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.

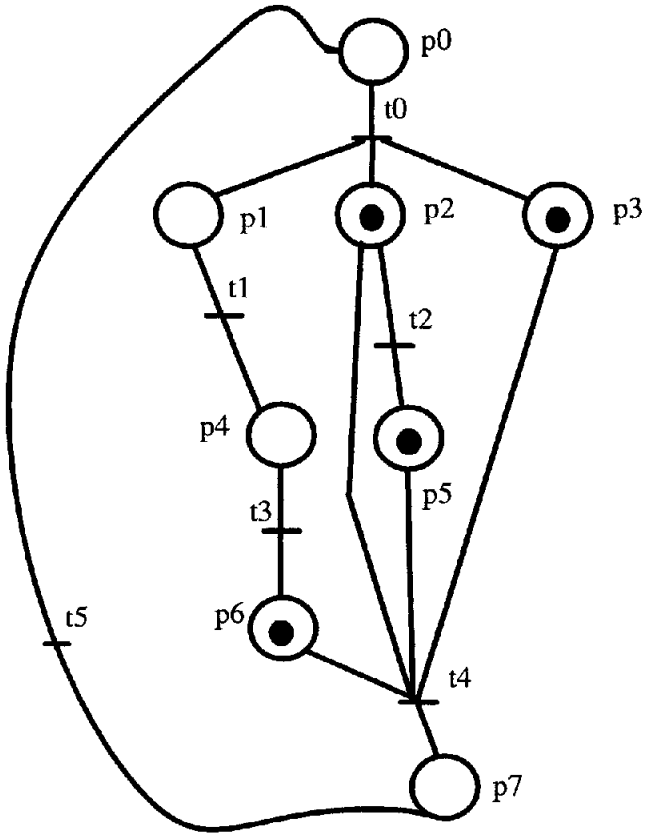

Figure 3: Trusted Host SYN Flooded

Attack nets do not need to have cycles describing recursive attacks. In most cases, they will not, since penetration 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. For an example of an acyclic attack net, Figure 4 below models a generic distributed denial of service attack on a single host. The first place, holding the single token, represents the attacker's initial control of the attacking host. The first level of transitions are high-level abstractions of the actions taken to gain control of multiple accomplice hosts. The multiple places in the middle represent the attacker's control of the accomplice hosts. The second level of transitions from these places represent the generation of spurious service requests by the accomplice hosts. The last place represents the condition where the victim host is flooded and cannot respond.

The key features of attack nets are

- modeling concurrency and attack progress with tokens
- modeling intermediate and final objectives as places
- modeling commands or inputs as transitions

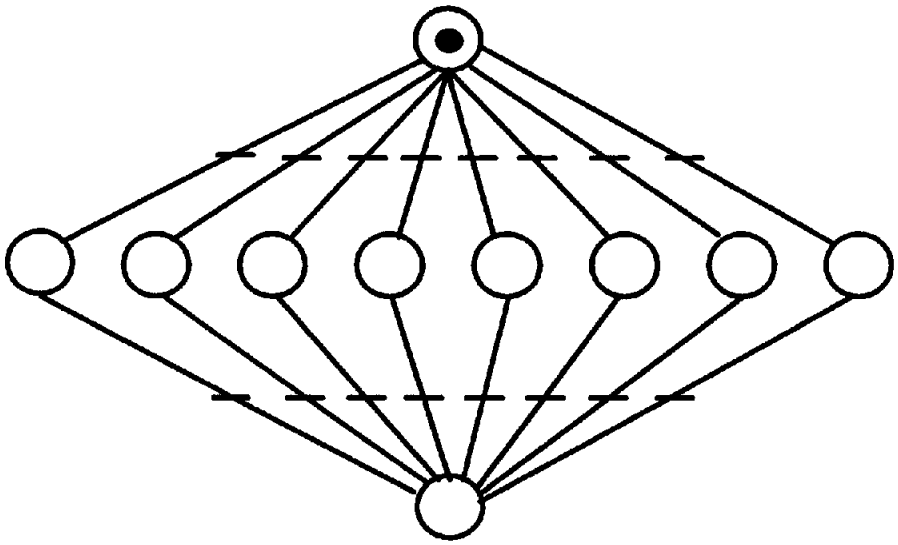

Figure 4: Attack Tree Representation of Generic Distributed Denial of Service Attack

Attack nets are not intended to model the actual behavior of a system or component during an attack, in the sense of attack signatures [1] used by intrusion detection systems or the abuse cases [3] 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.

We can also used labeled tokens to indicate which attacker is responsible for moving a particular component of the system of interest into a particular state. This might be most useful when modeling so-called "stealth" attacks that are designed to escape intrusion detection by dividing the attack responsibility among several attackers, or when we need to model mistakes on the part of the defenders.

4 Attack Net Penetration Testing

Although it might seem more reasonable to organize attack net penetration 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 attack-net-based penetration testing project:

Define goals

Background study

Attack net generation

Hypothesis verification

Flaw generalization

Flaw elimination

All of the steps we use are the same as the flaw hypothesis approach, except we use attack nets to construct hypothetical flaws or attacks.

In our limited experience, the best way to construct an attack 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 Figure 5 below.

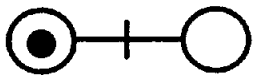

Figure 5: Initial Subtree Used to Model Individual Flaws

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.

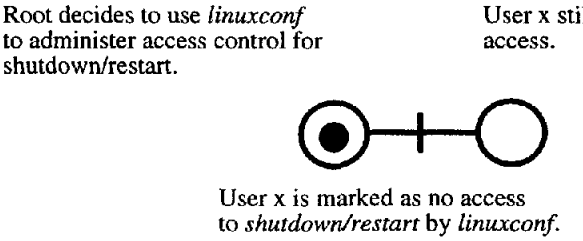

Figure 6: Linuxconf Flaw

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 penetration 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 Figure 7.

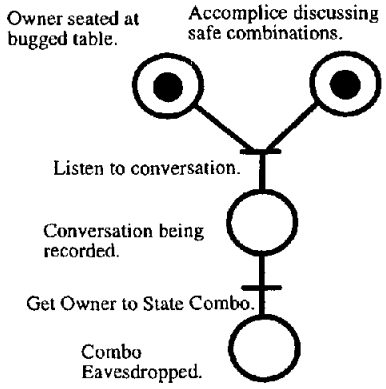

Figure 7: Attack Subtree For Eavesdrop Action

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.

5 Conclusions

The attack net approach to penetration testing is a departure from both the flaw hypothesis model and the attack tree model but it retains the essential benefits of both. Any penetration testing process is unavoidably dependent upon the flaw hypothesis process model. Any valid penetration testing process model will retain many of its features and so does the attack net model. Nevertheless, the attack net penetration testing process brings more discipline to the brainstorming activity without restricting the free range of ideas in any way. 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.

Attack nets provide a graphical means of showing how a collection of flaws may be combined to achieve a significant system penetration. This is important since an attack net can make full use of hypothetical flaws. Attack nets 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 penetration teams to communicate easily about the cumulative effects of several minor flaws.

The separation of penetration 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, 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. ESCAMILLA, T. Intrusion Detection: Network Security Beyond the Firewall. Wiley, 1998.
2. LINDE, R. *Operating System Penetration.* In Proceedings of the National Computer Conference, Vol 44. AFIPS Press, Montvale, NJ, 1975.
3. McDERMOTT, J. and FOX, C. *Using Abuse Case Models for Security Requirements Analysis.* In Proceedings of 15th Annual Computer Security Applications Conference, Phoenix, Arizona, December 1999.
4. 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.
5. SCHNEIER, B. *Attack Trees.* Dr. Dobbs Journal, December 1999.
6. WEISSMAN, C. *System Security Analysis/Certification Methodology and Results.* SP-3728, System Development Corporation, Santa Monica, CA, October 1973.
7. WEISSMAN, C. *Penetration Testing.* In Information Security Essays. Abrams M.D., Jajodia, S., Podell, H. eds. IEEE Computer Society Press, 1994.
8. WEISSMAN, C. *Penetration Testing.* In Handbook for the Computer Security Certification of Trusted Systems. Naval Research Laboratory Technical Memorandum 5540:082a, 24 January 1995.

New Security Paradigm Workshop 9/00 Ballycotton, Co. Cork, Ireland