2007

A pragmatic method for integrated modeling of security attacks and countermeasures

Srdjan Pudar
Iowa State University

A pragmatic method for integrated modeling of security attacks and countermeasures

by

Srdjan Pudar

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Program of Study Committee:
Manimaran Govindarasu, Major Professor
Daji Qiao
Hridesh Rajan

Iowa State University

Ames, Iowa

2007

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

ABSTRACT

In recent years, research efforts in cyber security have steadily increased as a result of growing concerns for cyber attacks and also increasing trend in cyber attack incidents. One of the important areas of research that is gaining importance is modeling of attacks and countermeasures to quantify survivability and other security measures of interest. In this context, on one extreme, attack trees model has received attention due to its simplicity and ease of analysis, and on the other extreme, stochastic models have been advocated. While attack trees model does not capture complex dependencies among events and also is not amenable for modeling dynamic nature of the attacks and countermeasures, the fitness of stochastic models is yet to be established as there is not sufficient evidence to show that attack and defense behaviors follow some known distributions. With this motivation, a new attack modeling approach based on Petri nets, called PENET, is developed in this thesis whose goal is to significantly enhance the modeling power of attack trees. PENET introduces relevant concepts such as dynamic nature of attack, repairability of a system, and the existence of recurring attacks. Moreover, it attempts to find a balance between ease of use and representation power by providing set of constructs, parameters, performance metrics, and time domain analysis of attack progress. Time domain analysis produces valuable output such as time to reach the main goal and the path taken by the attacker. This output helps to evaluate system survivability and defense strategies. This approach is implemented as a software tool, called PENET Tool, which lets users draw model diagrams of a given system through intuitive user interface, perform time domain simulations and carry out security evaluations, and enable interactive ways to improve the survivability of the system.

CHAPTER 1. INTRODUCTION

1.1 Background

With rise of cyber attack activities in recent years, more emphasis was given to research the issues related to this phenomenon. One of such research efforts is modeling of cyber attacks and countermeasures. Several modeling approaches have emerged in research community and industry. In this context, most of existing approaches are based on two notable approaches, one being attack trees, and second group of stochastic models.

As one of most popular modeling approaches with high impact, attack trees defined by B. Scheneier [1] have been used to model attacker behavior with purpose of reaching some compromise objective, such as system failure or root access. This modeling tool has proved to be simple, easy to use and analyze results, but yet powerful in its modeling simplicity. Beside modeling attacker behavior, attack trees have found similar use of modeling system vulnerabilities and points of access. However, attack trees are limited to simple representations, because of limited construct set and static nature. Our effort uses Petri net constructs to augment and extend existing principles that are already proven useful in attack trees.

Attack tree structure heavily resembles to one of fault trees [11], and fault trees initially consisted of simple constructs. Similarly, fault trees were extended with additional elements such as dependency gate and priority AND gate that provide additional modeling capabilities beyond initial fault tree set. Due to exponential distribution of events modeled by fault trees, these additional gates can be converted to Markov chain representation, and analyzed using standard techniques for solving stochastic Markov chains. Unfortunately, this analysis approach cannot be used for purpose of extending attack trees because there is no evidence that probability of attack success follows exponential distribution. Thus, this work proposes alternative analysis approach that does not imply stochastic nature of the events.

The purpose of our proposed approach, called *Petri Net Attack Modeling* approach (PENET) is to provide intuitive modeling approach for modeling attacker behavior in vulnerable systems in security sense, based on concepts of attack trees and modeling abilities of Petri nets. Second purpose of PENET approach is to address issues present in attack trees and other attack models. For example, additional information for some events in attack tree model is often available but useless due to limitations in modeling representation, yet attack trees suffer of imprecision reflected by subjective assignments of values on its elements. This approach can be used from modeling vulnerabilities of a single entity, to a whole enterprise level.

Structure of this thesis is as follows. Related research work is examined in following section. Next, we provide quick overview of attack trees and Petri nets as both are basis of our approach. Chapter 3 contains formal definition of proposed PENET approach and analysis methods. Next two chapters are focused on *PENET Tool*, the software implementation of our approach. Chapter 5 contains case study used to illustrate and simulate our approach. Finally, we provide conclusion and future work.

The main contributions of this thesis are two-fold:

- Extending modeling capabilities of attack trees by using Petri net constructs in order to significantly improve the analytical capabilities of attack trees, specifically by:

  a) Addressing existing issues in attack trees such as limited representation power, imprecision, and lack of defined defense modeling.

  b) Introducing concepts of recurring attacks, defense modeling, and dynamic constructs.

  c) Introducing an analysis approach that follows attack execution in time domain.

  d) Providing means to evaluate system survivability and defense strategies.

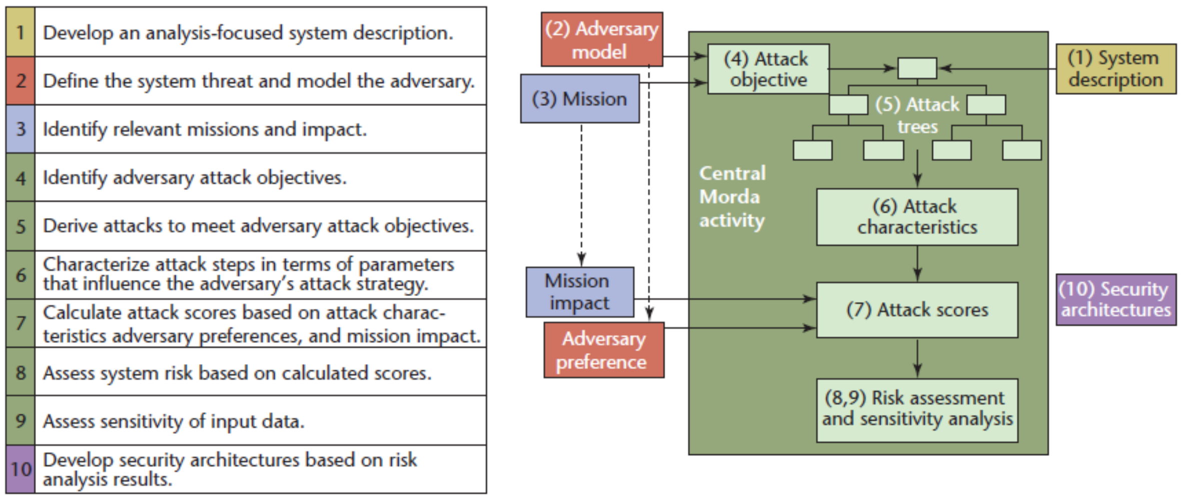
- Developing software tool that implements new approach and establishes its practical use.

1.2 Related Work

Since attack tree incursion, significant research effort has been made to find application for them, and to expand attack trees to new meaningful models. One of most prominent uses of attack trees is in Mission-Oriented Risk and Design Analysis (MORDA) [10] methodology developed by US National Security Agency. This methodology, shown on Figure 1.1, uses attack trees in its process steps. The 10-steps process starts with identification of items that are important in the process, namely system, adversary model, relevant missions, impact of successful attack, and attack objectives. Acquired information is then used to build attack trees that define attack space against which the system must defend. Last steps in this process involve analysis of attack tree model, assessments, and finally practical action. Assessment involves evaluation of risk based on attack tree analysis results, and sensitivity analysis of input data. Finally, defense strategies can be employed based on completed assessment.

Fung et al. [4] provides attack tree of MANET network with intrusion goal to compromise Confidentiality, Integrity, and Availability (CIA) of critical channels containing key information. Using service oriented architecture (SOA), authors analyze system and develop the attack tree model that represent such SOA system. This work adds notion of survivability to attack trees. Survivability analysis finds the system components susceptible to attacks an analyzes their ability to survive the attacks. Based on attack tree, authors deduct intrusion scenarios and provide quantitative values that determine which scenario has the lowest difficulty from attacker standpoint.

Bistarelli et al. [5] introduce defense aspect in attack trees and additional quantitative metrics such as return on investment (ROI) and return on attack (ROA). Defense trees are attack trees extension that incorporate counter-measures used to address intrusion attempts on leaves of attack trees. Higuero et al. [3] apply attack tree modeling to a case of digital content security. Authors illustrate attack scenarios and provide analysis based on various quantitative values that characterize attacker and attacker's threats. Attack patterns were introduced by Moore et al. [8] as a way to reuse generic segments or whole attack trees for applicable contexts. Authors show how to apply generalized attack tree scenarios such as buffer overflow to their particular case study. The re-usability factor is very important in application of attack tees.



1	Develop an analysis-focused system description.
2	Define the system threat and model the adversary.
3	Identify relevant missions and impact.
4	Identify adversary attack objectives.
5	Derive attacks to meet adversary attack objectives.
6	Characterize attack steps in terms of parameters that influence the adversary's attack strategy.
7	Calculate attack scores based on attack characteristics adversary preferences, and mission impact.
8	Assess system risk based on calculated scores.
9	Assess sensitivity of input data.
10	Develop security architectures based on risk analysis results.

Figure 1.1 Morda assessment [10] process steps

Attack trees are just one of many modeling approaches addressing security. Most of these approaches were derived from existing methodologies used to model dependability and reliability. Nicol et al. [12] provide overview of dependability and security modeling methods through a taxonomy and motivate for further research in this area. These modeling approaches can be divided to combinatorial, such as attack and fault trees, and stochastic, such as Markov chains and Petri nets. Authors conclude that there is merit for using stochastic techniques in attacker behavior modeling, but that significant additional work is required in order to create new model-based framework.

Further, Dalton et al. [2] provide conversion of attack trees to generalized stochastic Petri nets (GSPN) and perform steady state analysis of resulting GSPNs. Conversion methodology is based on Petri net equivalent models for AND and OR event gates commonly used in attack trees. The PIPE tool [13] was used to analyze derived Petri nets. McQueen et al. [19] introduce compromise graphs. The goal of their work is to quantify risk reduction estimation in a SCADA control system from cyber risk perspective. Quantitative risk model is defined by risk of undesired event that takes into account probability and consequence of undesired event. Compromise graph is directed graph with nodes as potential attack states and edges as successful compromises. Values on edges are expected time to compromise that are defined as functions. As a main measure of system security and risk, authors used time to compromise for base and enhanced systems. Also, authors recognize dominant attack path as a path with minimal time to compromise.

Other approaches use stochastic models to describe systems. These models are commonly converted to Markov chain, and analyzed using steady state transition matrix which contains desired system metrics. Madan et al. [7] use dynamic state diagram to describe behavior of intrusion tolerant system. This generic state diagram is a semi-Markov process model that is further solved using embedded discrete time Markov chain (DTMC). Advantage of this work is that presented generic model can be reused for specialized security attacks. The quantitative analysis of model produces two useful metrics, steady-state availability and *mean time to security failure* (MTTSF). Similarly, Sallhammar et al. in [6] develop stochastic model for security and dependability evaluation. Their approach uses game theory to model attacker behavior, as expected attacker behavior is driven by cost to perform attack and reward for successful attack. In this stochastic model intrusions are modeled as transitions between system states. Some system states are vulnerable to malicious faults, and game theory can be applied to these states.

In [14], McDermott introduces attack nets that are derived from attack trees and Petri nets. Attack net is a Petri net with states that represent point of security interests, and transitions that represent events. Unlike other approaches that follow attack tree approach, this model is intended for penetration testing. This model has capability to show how collection of seemingly unrelated flaws can have larger consequence for a whole system.

In conclusion of related work, we find that stochastic models promise strong modeling power that is lacking in attack trees. Unfortunately, due to fact that attacker behavior does not follow any of known distribution functions used in stochastic models, we cannot use stochastic models with defined probability distribution to develop precise attack model. However, other useful attributes of stochastic models can be used. Our approach, which detailed description follows in next sections, will try to find compromise between modeling power of stochastic models and simplicity of attack trees.

1.3 Foundations for Petri Net Attack Modeling Approach

1.3.1 Attack Trees

Attack trees are designed for the simplified evaluation of system security, often on the enterprise system level. Attack trees offer methodological means to analyze security of the system, such as its strengths and weaknesses with regards to intrusion attempts.

The principle of work of attack tree is simple and intuitive. Attack tree consists of three parts: root, leaves and sub-tree components. Root describes access or achieveability of primary goal, or compromise of desired system or a significant system module. Leaf nodes describe attack attempts as indivisible simple actions. Sub-trees are used for description of dependence of its children for achieving the upper goal. Sub-tree components can be AND and OR events. AND nodes describe the all dependence, meaning that all children are required to be compromised by attacker in order for that component to be compromised. Similarly, OR nodes describe that it is sufficient for one children to be compromised in order to compromise that sub-tree. Sample attack tree is shown on Figure 1.2.

Once attack tree model has been created, designer can associate values for each of its leaf nodes. The values can be of quantitative or qualitative nature. Choice whether to opt for quantitative or qualitative values depends on the ability to define reasonable values and desired precision level of the outcome. Most commonly used values are probability and monetary cost of attack, and they can be both qualitative and quantitative. The values can be determined by further researching attributes of particular attack that is represented by that leaf node. For example, assume that one leaf node describes attacker utilizing existing vulnerability on victim's internal DNS server, as shown on attack tree example in Figure 1.2. This event can be described by its statistical likeliness measured in percents. In this example, the value is found by examining the vulnerabilities of machine that allow server to be remotely taken over to better estimate the likeliness of this event. Similarly, we can associate cost for taking

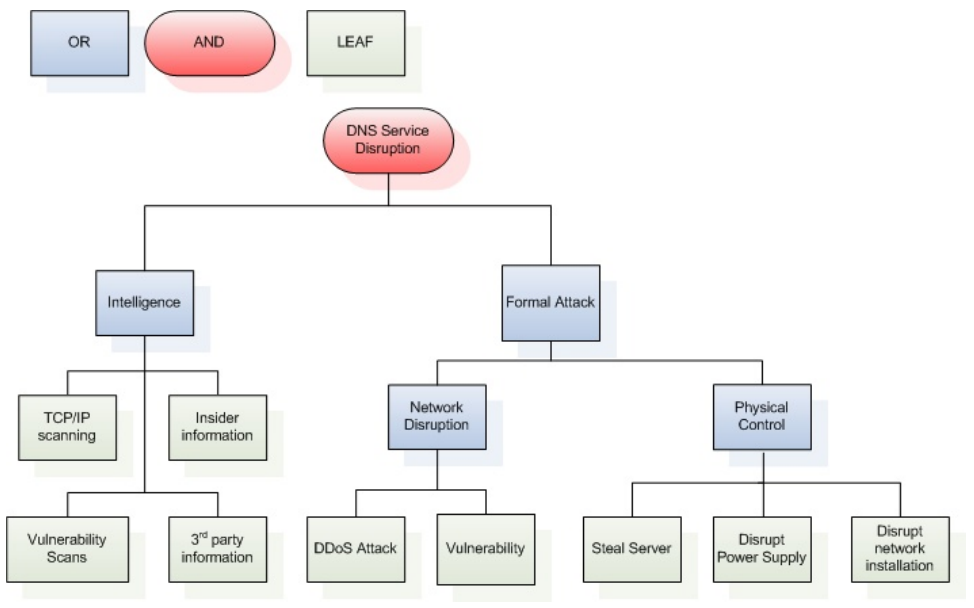


Figure 1.2 Attack tree representing simple DNS attack

advantage of such vulnerability.

Common attack tree analysis methods include finding selected value (often probability or cost) at root node, or performing attack scenarios analysis. Attack scenarios [4, 3], are sets of minimal combinations of enabled leaf nodes that lead to root access. For attack tree on Figure 1.2 there are 20 attack scenarios, each consisting of one leaf of left sub-tree and one leaf on right sub-tree. Attack scenarios can be used for finding various additional information such as the lowest cost attack and highest chance of success attack. Often attack trees contain custom additional parameters on leaves, and analysis can focus on them as well. An example of such parameter is required skill for performing an attack.

1.3.1.1 Weaknesses of Attack Trees

When modeling system using attack trees, often additional information about modeled system is available, but limited modeling capabilities of Attack trees cannot take this valuable information into account. This can result in a model that is less precise than optimal model of a different approach, that manages to fully utilize input information.

Assigning values to leaves is crucial for accurate analysis. However, this assignment is often crude and subjective due to fact that many unknown variables influence assignment of values. This leads to distorted outcome of the analysis. This subjectivity and inability to find precise values is one of biggest weaknesses of attack trees.

Because of simple modeling approach, attack trees can only present static model of system security and attacker behavior. In addition, limited constructs of attack trees fail to model dependability between events and existence of sequencing events. If system changes through a time, or new vulnerabilities appear, the original attack tree will be no longer valid. Thus, the static model is valid only for a limited time.

1.3.2 Deterministic Time Transitions Petri Nets

Petri nets are often used for various system modeling due to their ability to model various systems on a relatively high level, especially when compared to Markov chains. Petri nets are used in modeling and evaluation of system dependability [12], multiprocessor system performance evaluation [17], and modeling manufacturing systems and processes, but not limited to above listed. In this thesis, we focus primarily on a specific variant of Petri nets, called deterministic timed transitions Petri nets (DTTPN).

The definition of deterministic timed transitions Petri net is derived from basic definition of Petri net. DTTPN [9] is a 6-tuple $(P,\ T,\ I,\ O,\ M_0,\ \tau)$ where $(P,\ T,\ I,\ O,\ M_0)$ defines Petri net as 5-tuple of five sets: places P, transitions T, input functions I that define arcs from places to transitions, output functions O that define arcs from transitions to places, and initial markings M_0.

In a simplified notion, transitions t_i on DTTPN can fire after specified time delay τ_i. In a formal sense, τ is function that associates these time delays to each transition on DTTPN. Additionally, we will assume that some transitions are immediate, in other words time delay $\tau_i = 0$ for some transitions.

Figure 1.3 shows simple DTTPN. This net consists of four places (P_0 to P_3), two timed transitions (T_0 and T_1), and tokens in places defined by initial marking $M_0 = (1,\ 1,\ 0,\ 0)$.

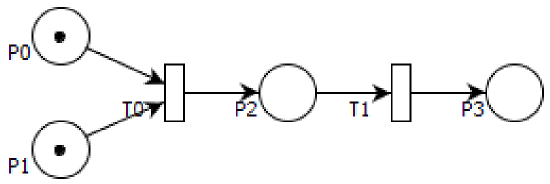


Figure 1.3 Simple deterministic time transitions Petri net (DTTPN)

After time τ_0 token is present in place P_1.

Often we will use notion of color to regulate firing requirements for a transition. In these cases, some places will generate differently colored tokens. Then transitions can require tokens of different colors in order to fire. Such nets are commonly called colored Petri nets (CPN) [9].

Other variant of Petri nets, named stochastic timed Petri nets [9] are commonly used for system modeling. Some variants of these nets such as stochastic Petri nets (SPNs) and generalized stochastic Petri nets (GSPNs) assume exponentially distributed firing time for transitions. For security modeling purposes, this assumption does not hold due to lack of evidence that attacker behavior can be modeled by exponentially distributed time delay. Instead of using such function for time delay on transitions, we used transitions with constant delay time. Thus, DTTPN were chosen instead of stochastic Petri nets for our modeling approach.

CHAPTER 2. PETRI NET ATTACK MODELING APPROACH (PENET)

2.1 Formal Definition

In previous section, we provided overview of existing deterministic timed transitions Petri nets (DTTPN) that are used as a basis for our modeling approach. Our approach can be interpreted as an modification of attack trees using Petri nets as modeling tool. The goal of PENET is to provide alternative to attack Tree modeling that addresses mentioned weaknesses found in attack trees.

Our approach uses loosely defined DTTPN as a modeling tool. PENET model is a DTTPN-alike net with defined set of constructs and specific times associated with each transition. Places P represent attackers places of interest, subgoals and main goal; similarly to concept of node employed in attack trees. Transitions T represent time delay needed for attacker to compromise next goal. Places with token represent sub-goals that attacker has managed to compromise. Main goal of attacker is represeted by root place, at which attacker has accomplished his goal.

Formally, PENET model is a 8-tuple (P, T, I, O, M_0, A, C, R) where:

- *(P, T, I, O, M_0)* carry same meaning as in DTTPN definion and are used to build constructs of PENET approach.
- Parameter set (A, C, R) that replaces time delay τ_i in DTTPN definition with time delays a_i, c_i, and r_i.

We introduce three time delays used in PENET transitions: periodic arrival, compromise, and repair times. Periodic arrival time a_i models cost that incurs to attacker for each attack attempt to some goal. Compromise time c_i models time delay needed to accomplish particular

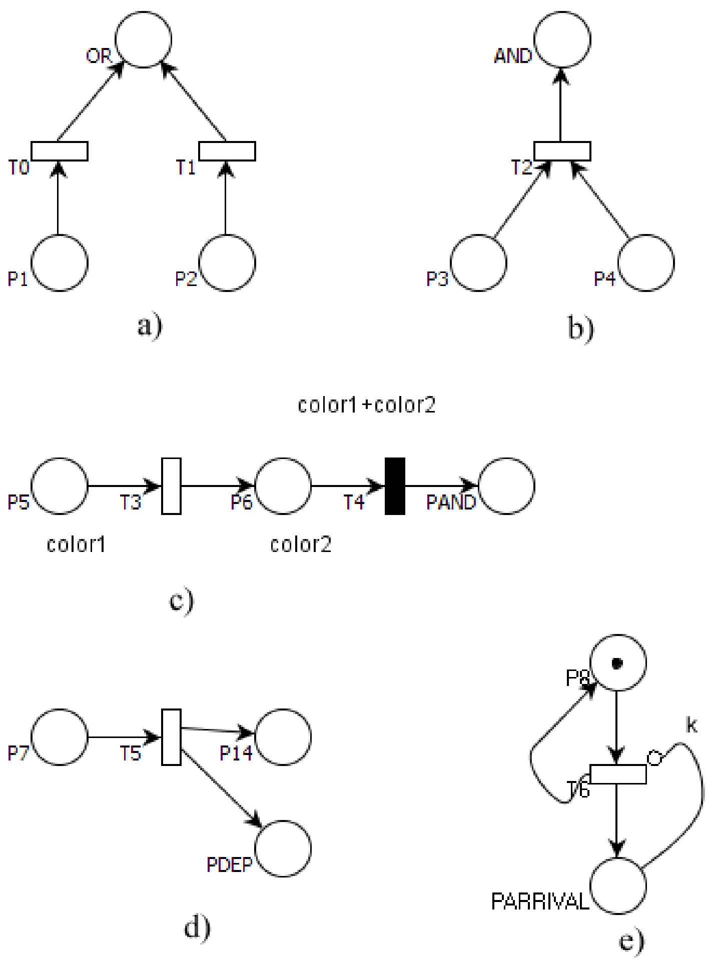


Figure 2.1 PENET constructs a) OR event b) AND event c) PAND event d) dependence event e) arrival construct

goal targeted by attack attempt. Repair time r_i models repair response when victim notices that some system module has been compromised. Thus, instead of single parameter, often probability used in attack trees, we provide finer modeling with three introduced parameters.

Constructs in PENET can be static and dynamic. Static constructs are AND and OR gates inherited from attack trees. As their name implies, they are used to model sub-system that can described by static structure. On other hand, some attacker behavior in a system cannot be described with such static modeling approach. As mentioned before, notions of sequential events and dependency between events cannot be modeled using attack trees. In these situations dynamic constructs of PENET approach find their use.

The common attack tree OR and AND gates are simply converted to PENET approach [2]. OR and AND gate models with any number of input children events are shown on Figure 2.1. The structure of these constructs is intuitive; for OR construct token will be present on top place if any transition fires. In case of AND construct, tokens must be present on every incoming place in order for transition to fire and bring token to top place.

PENET approach introduces dynamics in the model of the insecure system. In this sense, Petri nets are good choice as they provide various constructs for modeling dynamic behavior of a system. Thus, any construct represented by Petri net can be used in PENET approach. This provides additional degree of freedom to designer, as designer can develop new constructs that suit the particular application. In this work, focus will be given to two such constructs, priority AND (PAND) and dependency constructs. These gates are commonly found as extension of fault trees into Dynamic Fault trees [11], and they proved to be useful in modeling of even simple attack models.

Priority AND gate (Figure 2.1.c) models sequential nature of multiple events. This construct defines that events must occur in specified order, from left to right. Dependency gate (Figure 2.1.d) models dependence of an event upon prior occurrence of another event. The difference between these two is that dependence construct does not require strict sequencing between dependant and controlling event, only that controlling event has occurred. Arrival construct [18] from Figure 2.1.e models consecutive attack attempts that occur after attack has failed on some higher level, and attacker has to start from beginning. Factor k models maximum number of tokens that can be present at arrival place.

2.2 Defense Modeling

Although quantitative and qualitative metrics of attacker behavior provide values necessary for modeling, notion of defense modeling is mostly left out in existing research models. Unlike defense tree constructs in [5], most other attacker modeling approaches ignore repairability and defensive aspects of a vulnerable system. Most systems are repairable, both during critical failure state and intermediate failure states. This concept is widely used in system reliability modeling using repair rate μ. In our context of modeling attack behavior, this means that compromised modules could be repaired, if attacker does not move to higher level goal in a timely fashion. When this happens, attacker is forced to start over from some lower level place or from the start, leaf place. Repairability of a place is represented by introduced parameter r_i.

2.3 Converting Attack Trees to PENET Model

As mentioned earlier, some known information about system and attacker cannot be fully utilized by attack tree constructs. Attack trees can be converted to an equivalent PENET approach model. During this process it is desirable to include omitted information that will augment the model. PENET approach has ability to improve attack tree design using Petri net constructs and additional parameters that describe the known information. These constructs and parameters were introduced in PENET definition.

Algorithm 2.1 lists procedure for conversion. Conversion consists of several stages. In first stage, leaves are converted to initial places with arrival construct. In second stage, all OR gates are converted to their corresponding PENET model (Figure 3). In next stage, AND gates of attack tree are examined to determine whether they match priority AND or regular AND construct of PENET approach, and then converted to the appropriate one. Next, whole tree is examined for presence of dependencies that are not shown on attack tree. Finally, time delays are assigned to PENET model using existing attack tree quantitative or qualitative values and additional information that was obscured by attack tree design.

2.4 Coverability Tree Analysis

Coverability tree analysis is the first of two provided analysis methods for PENET approach. This method is similar to attack scenario analysis introduced in [4]. This analysis is useful for finding attacks of interests such as the most likely attack. The coverability analysis is common analysis method for all Petri nets [9]. First step in this analysis is to construct coverability

```
    **Input**: Attack tree $AT$
    **Output**: Corresponding PENET model
1   Assign leaves to places
2   Convert OR gates to equivalent model
3   Convert AND gates to PAND or AND model
4   Find dependencies and connect them to rest of net
5   Assign arrival $a_i$, compromise $c_i$, and repair $r_i$ times
```

Algorithm 2.1: *ATtoPENET(AT)*, Procedure for converting attack tree to a PENET model

tree.

The procedure for finding coverability tree from [9] is presented in Algorithm 2.2. Based on initial marking M_0, new markings are explored until there are no more markings left to be found. Repair places and transitions can be ignored for coverability tree analysis. Once coverability tree has been constructed, all paths that lead from initial marking M_0 to markings M_r that are associated with root of PENET tree are of interest for further analysis. These paths are labeled with AP_i.

Consider all markings paths AP_i from initial markings M_0 to root place in PENET net representing main attacker goal. For purpose of construction of full coverability tree it was assumed that all at leaf places have initial marking. Generally, not all initial markings are necessary to achieve root place, because not all transitions from leaves to root are enabled. Thus, set of all AP_i consists of two subsets, subset of mutually independent markings paths and subset of dependent markings paths that are extended with optional leaf places that are not necessary. For example, OR constructs require only one fired transition, so each OR construct generates multiple markings paths, and only ones with single input transition enabled are independent. Thus, list of independent marking paths AP_i*'* represent attack scenarios.

For i-th attack scenario AP_i*'*, with number of leaf nodes n_i, *Total Attack Scenario Cost* DC_i and *Total Attack Scenario Chance of Success* DP_i can be calculated:

$$DC_i = \sum_{j=0}^{n_i} AC_j \tag{2.1}$$

$$DP_i = \prod_{j=0}^{n_i} AP_j \tag{2.2}$$

```
Input: Initial marking $M_0$
Output: Set of all markings $M_i$
Label the initial marking $M_0$ as the root and tag it "new"
while "new" markings exist do
    Select a new marking $M$.
    if $M$ is identical to a marking on the path from the root to $M$ then
        then tag $M$ "old" and go to another new marking.
    end
    if no transitions are enabled at $M$, then
        tag $M$ "dead-end."
    end
    while there exist enabled transitions at $M$ do
        foreach enabled transition $t$ at $M$ do
            Obtain the marking $M'$ that results from firing $t$ at $M$.
            On the path from the root to $M$ if there exists a marking $M''$ such that
            $M'(p) \geq M''(p)$ for each place $p$ and $M' \neq M''$, i.e. $M''$ is coverable, then
            replace $M'(p)$ by $\omega$ (infinity) for each $p$ such that $M'(p) > M''(p)$.
            Introduce $M'$ as a node, draw an arc with label $t$ from $M$ to $M'$, and tag
            $M'$ "new."
        end
    end
end
```

Algorithm 2.2: *CoverabilityTree(M_0)*, Construction of coverability tree for Petri net from [9]

Note that number of leaf places n_i varies for each attack scenario i. Index j in equations (2.1) and (2.2) denotes leaf places that are part of particular attack scenario i. In equation (2.1) *arrival cost* AC_j represents monetary equivalent to periodic arrival time parameter a_j. The mapping between time delay a_j and cost AC_j is intentionally left out to allow for custom mapping that is specific to the modeled case. For example, every arrival time can be multiplied with some constant value k [\$/hour]. In equation (2.2) *chance of success* AP_j represents probability equivalent of compromise time parameter c_j. The mapping between probability and compromise time is again left out to the modeler. Example of one such mapping is presented in case study in chapter 5.

Finally, *Attack Scenario Index* DI_i combines left sides of equations (2.1) and (2.2):

$$DI_i = \frac{DC_i}{DP_i} \tag{2.3}$$

Attack scenario index is a single quantitative measure of cost and success probability of an attack scenario. The attack scenario with lowest *Attack Scenario Index* DI_i represents attack with best price to performance ratio from attacker perspective. Such attack scenario offers the attacker highest impact for the lowest investment. From victim perspective, such attack scenario can be considered as most likely one. Similar index is employed in [5] and called *Return on Attack* (ROA). The complete procedure for finding the most likely attack is listed in Algorithm 2.3.

```
Input: Initial marking $M_0$
Output: Most Likely Attack Scenario
1 Construct the coverability tree
2 Acquire all markings paths $AP_i$ that lead to root place
3 From $AP_i$ acquire independent markings paths $AP_i$' that represent attack scenarios
4 Calculate values $DC_i$, $DP_i$, $DI_i$ for each attack scenario
5 Analyze attack scenario with lowest $DI_i$ index value
```

Algorithm 2.3: Attack scenario analysis

2.5 PENET Time Domain Analysis

Second analysis method for PENET approach fully utilizes novelties of our model, such as delay times a_i, c_i, and r_i assigned across the model. The benefits of PENET approach can be seen from proper time domain analysis. In this analysis, all constructs and parameters will influence the outcome.

To make the analysis more comprehensible, in this work we will assume that attacker will take route of most likely attack, for which attacker has highest ratio between success probability and invested resources. This path was found using coverability tree analysis method. In general, PENET model with all leaves enabled can be analyzed, without any difference in the procedure. Algorithm 2.4 specifies procedure for performing time domain analysis. Assumed is that PENET model of system is available, along with parameters that quantitatively describe the model.

Algorithm simulates attacker progress in time domain taking into account PENET structure described by various constructs and parameters. First step in algorithm execution is initialization where leaf places that are part of attack scenario are being recognized and their time delay values assigned (steps 1, 2 in Algorithm 2.4).

To keep track of token arrivals at various places in various time points, priority queue PQ stores sorted list of events *evt* that contain information about occurrence time τ, place of arrival P, and required tokens R leading to transition being fired. Token table TT contains information of token arrival and departure at each place. Every token has its own unique ID. Priority queue and token table are created in steps 3 and 4.

Next step is to generate initial periodic events at every place i that is part of attack scenario based on arrival times a_i and maximum simulation time. These events are stored in priority queue PQ (step 5). Note that every event represents token arrival at some place.

After initialization has been completed, execution reaches main *while* loop (step 6). This loop contains two parts. In first part, steps 7 to 17, current event is being evaluated for validity, based on whether required tokens for current event are found. If all required tokens specified by requirements *req* are not found, then event is discarded, and algorithm continues with next event in priority queue. If requirements *req* are met, then token table TT is updated to reflect arrival of new token, and departure of tokens that caused firing (steps 14, 15, and 16 in Algorithm 2.4)

In second part, algorithm evaluates if current event can lead to new firings. When token arrives at some place, it could enable firing on multiple new transitions. However, as soon as one of such transition fires, token is no longer available for other transitions. Algorithm handles this situation by generating events for each transition that potentially can fire and setting event requirements *req* to include individual token ID. Later, when each event is due, algorithm checks whether all firing requirements are still met. The event that has earliest start time will consume the token, and render all other potential events invalid, which is desired outcome.

Algorithm ends when there are no events left, or when simulation time expires.

Beside analysis for single set of values of each parameter, time domain analysis can be performed for a range of values of specific parameters of interest, usually ones that victim or

```
Locate all leaf places that belong to attack scenario
Assign $a_i$ times on leaf places, $c_i$ times on transitions, and $r_i$ times where available
Create priority queue $PQ(evt)$ that stores events $evt(P, \tau, REQ)$ consisting of place
  and time of event, and event requirements
Create token table $TT(P, \tau_1, \tau_2, ID)$ that stores tokens per places and interval of
  their presence
Assign periodic token arrival time events $\tau_i$ for all enabled leaves $i$, with period
  equal to arrival time $a_i$ and queue events in $PQ$
while simulation end time is not reached and PQ is non-empty do
    Dequeue first event event in queue $PQ$
    if event is token arrival then
        Verify that all requirements $req$ for this event are met
        if not then
            ignore event and continue
        end
    end
    Declare token arrival time $\tau_{k1}$ at place $k$ of event
    Declare departure times $\tau_{x2}$ for all places $x$ in event $evt$ requirements $req$ that
      lead to this token arrival
    Update token table $TT$ with values $\tau_{k1}$ and $\tau_{x2}$
    Determine what transitions connected to place $k$ can fire
    foreach such firing do
        Create event $evt$ for new transition and queue it into $PQ$
    end
    Update time to first event in $PQ$
end
```

Algorithm 2.4: PENET time domain analysis for single attack scenario

attacker have some degree of control. The influencing parameters could result in different level of penetration, or in different time spent to reach the main goal. In a nutshell, the analysis process is the same. Regular analysis procedure from Algorithm 2.4 is performed for every value of parameters. This parametric simulation will be explored in remaining sections.

2.5.1 Token Propagation Logic

At each place i, single token arrives as a result of some transition being fired on time τ_{i1}, and leaves on τ_{i2}. Time τ_{i2} is determined by outgoing transitions with compromise time c_i that could lead to new compromises (different place), or to a repair place regulated by repair time r_i. Due to periodic nature of arrival times, multiple tokens could be present at any place.

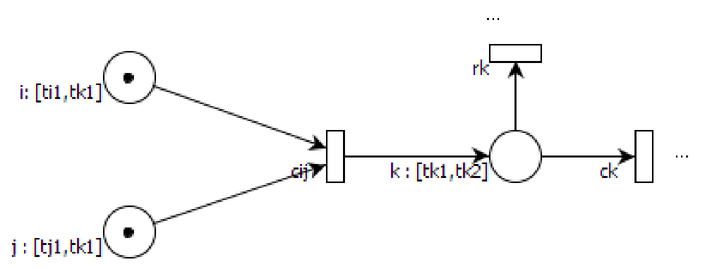


Figure 2.2 Token propagation in PENET model time domain analysis

Figure 2.2 illustrates the token propagation for a simple case of AND construct. Assumed is that places i and j have single token. In this particular case, token will be in place k for interval $[\tau_{k1}, \tau_{k2}]$ determined by:

$$\tau_{k1} = max(\tau_{i1}, \tau_{j1}) + c_{ij} \tag{2.4}$$

$$\tau_{k2} = \tau_{k1} + min(c_k, r_k) \tag{2.5}$$

In other words, single token will go to repair place or next compromise place depending whether new compromise takes more time than repair does. There will be two events generated for arrival of token at place k and its departure. If there are multiple tokens available, tokens will in general case distribute to both compromise and repair places.

General case equation for token progress cannot be specified because of existence of non-deterministic Petri net constructs. However, our PENET approach with current set of constructs does not include any non-deterministic constructs. Similar to AND event case, equation can be established for OR event when single tokens are present in both input places:

$$\tau_{k1} = min(\tau_{i1}, \tau_{j1}) + c_{ij} \tag{2.6}$$

$$\tau_{k2} = \tau_{k1} + min(c_k, r_k) \tag{2.7}$$

Remaining PENET constructs have trivial token progress.

2.6 Performance Metrics for Time Domain Analysis

Performance metrics provide additional quantitative information about PENET model and survivability of a system modeled by it. These metrics are suitable for comparing effectiveness of various attack or defense strategies for a given system. Usually these strategies manipulate value of some parameter that is controllable by attacker or by victim. Thus, efficacy of strategy change can be evaluated using these performance metrics:

- *Time to reach root place* provides time in hours from attack start to compromise of the root goal.
- *Number of intrusions accomplished at root place* specifies how many times attacker was able to compromise root goal within given time interval of attack.
- *Number of subgoals reached* specifies number of subgoals compromised out of total number of subgoals in PENET system. This goal is suitable for situations when root goal was not reached, but some of subgoals were reached. Smaller value reflects more effective defense. This parameter loses its meaning if it is not applied to PENET model that models the single attack scenario.

Values for each metric are obtained by performing time domain simulation of the PENET model, and analyzing contents of token table TT described in Algorithm 2.4.

2.7 Most Likely Attack Scenario Discovery Using Time Domain Analysis

Time domain analysis can be used to determine most likely attack in a dynamic sense. Recall that coverability tree analysis uses static, attack tree alike model to obtain most likely attack scenario. However, for different set of parameter values, victim can expect different most likely attack scenario. By performing time domain analysis, if token is present on root place, we can create attack scenario that has led to root. In this case, all leaf places that are necessary for compromise at root place belong to most likely attack scenario AP.

2.8 Evaluation of Survivability and Defense Strategies

Creating PENET model that precisely models the system is not end result of the PENET approach. Based on results of simulation, new steps can be taken towards enhancing the system survivability. In this sense, primary parameters that describe survivability, defense effectiveness, or some other desired defense descriptor are performance metrics that will be provided as a result of time domain simulation. Algorithm 2.5 lists procedure for improving survivability. Survivability can be improved by tweaking some parameter(s) of PENET model, or adding new defense constructs. Then, time domain simulation is re-ran for each tweak, until performance metrics are improved to the desired level.

This iterative process is compliant to process proposed in Morda methodology [10], presented on 1.1. Steps in Algorithm 2.5 are similar to 10 steps of Morda methodology. Both processes result in improved countermeasures based on analysis of attacker behavior and risk assessment.

```
1  Obtain relevant information of attacked system and components
2  Develop PENET model of system or attacker behavior
3  Assign the values to parameters in PENET model
4  Recognize parameters and other items of interest
5  Run time domain simulation for parameter set, either on selected attack scenario or
   on whole PENET model
6  Evaluate and validate cyber security and survivability using output from the
   simulation
7  if not satisfied with output then
8      Process changes that potentially improve system survivability or other desired
       metric
9      Go back to step 5;
10 else
11     Perform final steps
12 end
```

Algorithm 2.5: Iterative process for enhancing system survivability

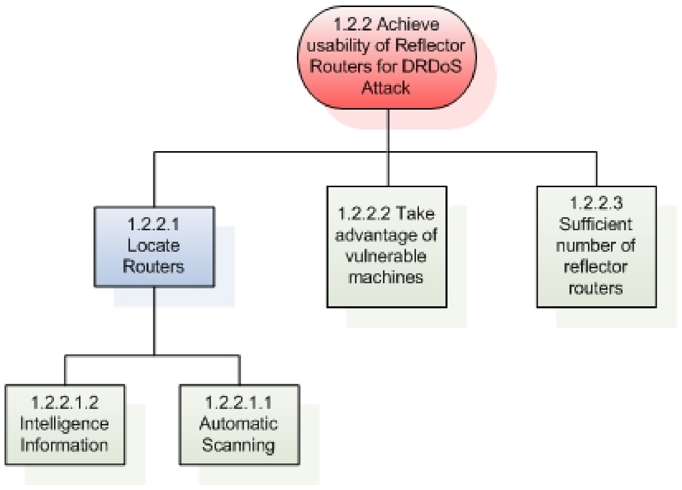

1.2.2 Achieve usability of Reflector Routers for DRDoS Attack
1.2.2.1 Locate Routers
1.2.2.2 Take advantage of vulnerable machines
1.2.2.3 Sufficient number of reflector routers
1.2.2.1.2 Intelligence Information
1.2.2.1.1 Automatic Scanning

Figure 2.3 Reflector routers attack pattern

2.9 Illustration of PENET Framework

To illustrate our approach, consider Attack subtree on Figure 2.3 that is an attack tree pattern [8] used in our case study discussed in the next section. This attack subtree can be easily converted to a PENET approach model, as is shown on Figure 2.4. For simplification reasons, arrival constructs (Figure 2.1.e) were not shown, but denoted with arrival time a_i at each leaf. Notice that AND root node (marking *1.2.2*) becomes Priority AND event when converted. From description of this attack it can be concluded that three leaf events have to occur in sequential order from left to right, but this information could not be captured in existing attack tree.

The PENET model of this pattern consists of four leaf places, one OR construct at place *1.2.2.1*, two PAND gates with compromise times c_{1222} and c_{1223}, and root place labeled with *1.2.2 goal*. Parameters include arrival times for each leaf event a_i, compromise times for each transition c_i, and repair times r_i for all leaf places.

Arrival constructs model periodic nature of attack attempts that reoccur after attack has been stopped because of repair efforts or timed-out attack. The formal representation adds unnecessary clutter that can be avoided by proper notations. Similarly, repair transitions end "in air", representing that token is being lost after successful repair. Sequential constructs on

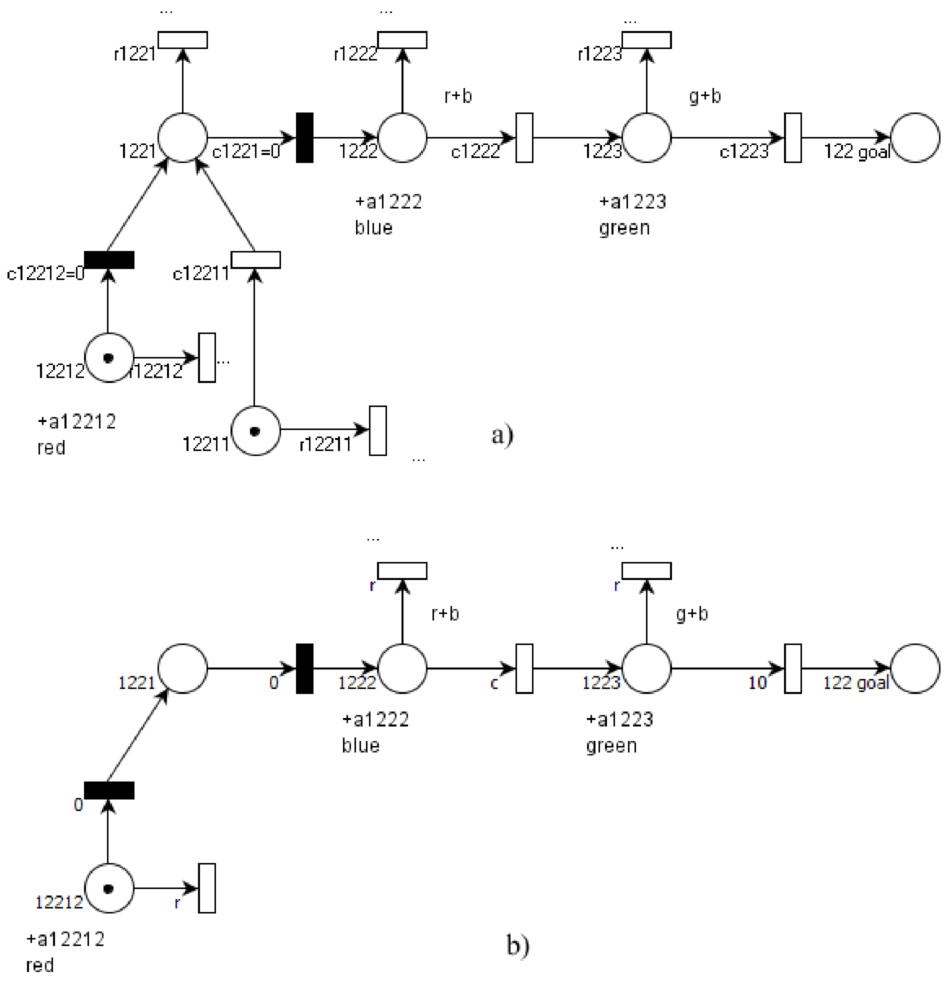


Figure 2.4 a) Reflector routers attack subtree converted to PENET model
b) Reduced model for time-domain analysis

Table 2.1 Parameter values for PENET model from Figure 2.4.b

Place / Value	12211	12212	1222	1223
Cost of arrival AC [\$]	100	100K	100	100K
Chance of success AP	0.1	1	0.001	0.1

transitions c_{1222} and c_{1223} require tokens from two different places in order to fire. To separate these tokens, introduced is notion of colored token, so for transition c_{1222} to fire both *red* and *blue* tokens need to be present on *1.2.2.2* place.

Figure 2.4.a shows initial PENET converted model. Because of OR event at *1.2.2.1* place, there are two attack scenarios. First scenario uses places *1.2.2.1.1*, *1.2.2.1*, *1.2.2.2*, *1.2.2.3* to reach *1.2.2* goal, and second scenario differs in first leaf that becomes place *1.2.2.1.2*. In order to come with most likely attack scenarios, values need to be fed into equations (2.1), (2.2) and (2.3). These values are assigned in table 2.1.

Thus, for first attack scenario total attack scenario cost and chance of success are

$$DC_1 = \sum_{j=0}^{2} AC_j = AC_{12211} + AC_{1222} + AC_{1223} = 100.2K\$$$

$$DP_1 = \prod_{j=0}^{2} AP_j = AP_{12211} + AP_{1222} + AP_{1223} = 10^{-5}$$

Finally, Attack Scenario Index for this attack scenario is

$$DI_1 = \frac{DC_1}{DP_1} = \frac{100.2K\$}{10^{-5}} = 10.02M\$$$

Similarly, for second attack scenario

$$DC_2 = \sum_{j=0}^{2} AC_j = AC_{12212} + AC_{1222} + AC_{1223} = 200.1K\$$$

$$DP_2 = \prod_{j=0}^{2} AP_j = AP_{12212} + AP_{1222} + AP_{1223} = 10^{-4}$$

$$DI_2 = \frac{DC_2}{DP_2} = \frac{200.1K\$}{10^{-4}} = 2.001M\$$$

From these values, it can be concluded that second attack scenario is more likely, due to having lower overall index.

Table 2.2 Simulation results for PENET model from Figure 2.4.b

Parameter a value [hours]	Goal accomplished	Time to accomplish top goal [hours]	Token count on top goal
2	122 goal	24	8
4	122 goal	28	4
6	122 goal	34	2
8	1222 sub-goal	N/A	0
10	1222 sub-goal	N/A	0

Next, time domain analysis can be performed on most likely attack scenario. Before that, the number of variables can be reduced by taking advantage of relations and properties of them. First, we recognize that

$$a_{1223} \geq a_{1222} \geq a_{12212}$$

due to fact that events described by these arrival times are in sequential order, so arrival time of later events cannot be shorter than of events that precede it. In a worst case from victim perspective, sequential attacks will arrive as soon as they can:

$$a_{1223} = a_{1222} = a_{12212} = a$$

Thus, all arrival times can be replaced with a single parameter a. Similarly, we can assume that repair time is constant within same sub-system, and modeled with single parameter r. Remaining parameter c denotes likelihood of attacker ability to take advantage of vulnerable machines (place *1.2.2.2*). This simplified model is shown on Figure 2.4.b.

Table 2.2 shows results of PENET time domain simulation for various values of parameter a. The simulation was performed using simulator specifically developed for this purpose. Simulator is based on time domain analysis procedure from Algorithm 2.4. Beside variable parameter a, other two parameters have assigned values: repair time $r = 8$ hours, and compromise time of $c = 10$ hours. Run time for generation of attacks was set to 40 hours.

From Table 2.2 it can be concluded that more frequent attacks saturate repair capabilites, decrease time to accomplish root goal, and increase number of root goal penetrations. Another interesting scenario worth mentioning is when compromise time is increased to c = 100 hours. This change could represent higher difficulty to penetrate more secure system. In this case, for parameter a value of a = 4 hours, the best attacker could do is reach *1222 sub-goal* at time τ = 4 hours.

CHAPTER 3. PENET TOOL OVERVIEW

3.1 Introduction

PENET Tool is a software implementation of introduced *Petri Net Attack Modeling* approach. PENET Tool was designed with a purpose to establish practical use of described attack modeling effort through a single and comprehensive software tool. This purpose can be translated to set of objectives translated to desired features: ease of use, ability to use graphical interface to draw and edit diagrams, and ability to perform time domain analysis and calculate performance metrics. Special attention was given to post-simulation features that are of special value to end-user. These features assist user in a vulnerability assessment, evaluation of defense strategies, and provide interactive tools to improve security of the system.

Secondary objective of PENET tool is to use established software engineering practices such as object-oriented design in order to ensure quality of software and sound design.

PENET Tool was completely written in C# .NET using Visual Studio 2005 as a development environment. It requires .NET 2.0 framework [21] to run. Because of these requirements, it is not suitable for operating systems other than Microsoft Windows.

Primary audience of this tool is individuals and organizations who want to use our approach in vulnerability evaluation of cyber attacks and developing defense strategies for their systems. Secondary audience is research community desiring to learn more about attacker behavior modeling and PENET approach.

3.2 Features

PENET Tool is a graphical user interface (GUI) application that provides one-stop utility for designing, testing, and evaluating attack models using PENET approach.

PENET Tool features diagram editor, file management tools, export and printing tools, integrated time domain simulator, and post-simulation tools. In most cases, these features are driven by user interaction.

Diagram editor provides means for drawing and editing elements and constructs of PENET attack model. Editor allows user to add new items, edit properties of exiting items, remove items, and move their position on the diagram. These features will be described in full detail in following sections.

File management operations support operations with model files, such as opening and saving files using Windows built-in interfaces, and converting XML files [22] to PENET representations.

Export and printing features support imaging and printing operations. PENET Tool is capable of printing diagrams and creating bitmap files from visual representation of a diagram.

Built-in simulator is based on implementation of PENET time domain analysis algorithm. It performs time domain simulations for a given PENET model.

Post-simulation analysis module performs presentation of simulation results in a fashion that is helpful for evaluation of system survivability and defense strategies. Defined performance metrics are utilized for this purpose.

PENET Tool is distributed using its installer for Microsoft Windows. Installer allows users to install PENET Tool as they would any other Windows application.

3.3 Using the PENET Tool

Figure 3.1 shows main user interface window of PENET. At user's disposal are drawing area, toolbar and menu bar, status bar and right- click menu.

3.3.1 File Operations

Basic file operations can be accomplished using *File* menu. Such operations are performed usually on a single PENET model.

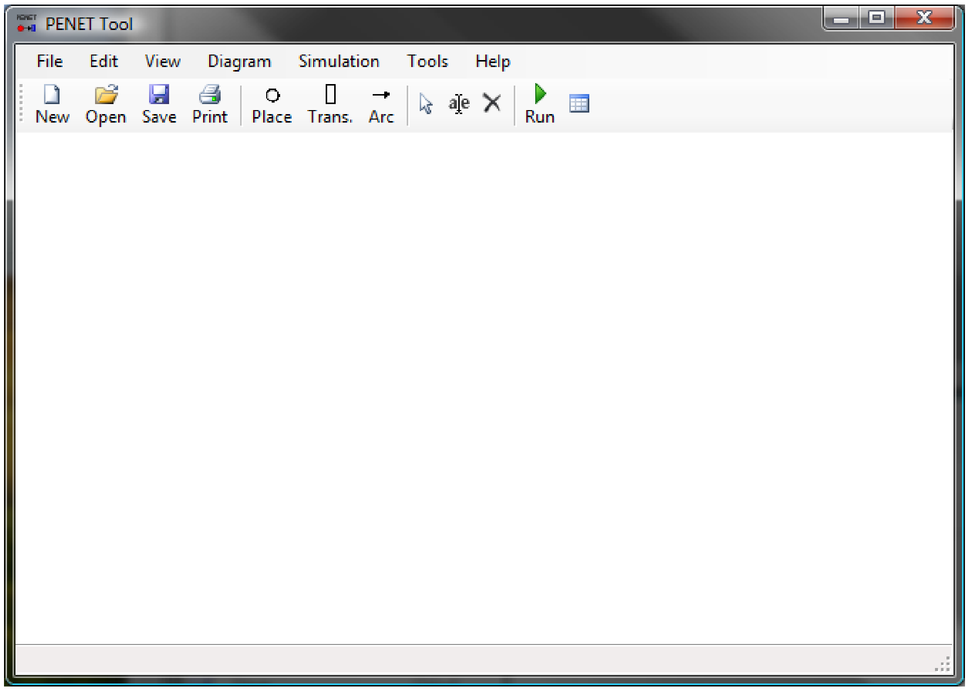


Figure 3.1 Main user interface window

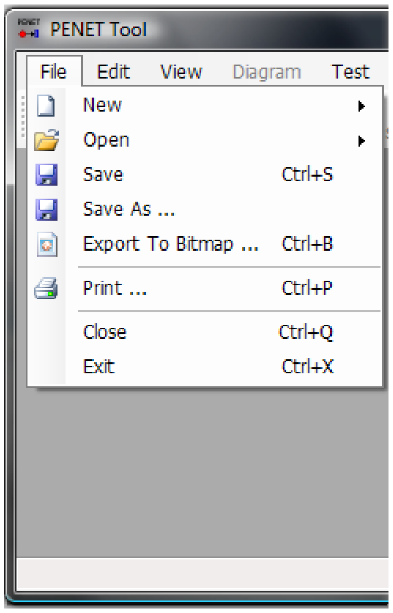


Figure 3.2 *File* menu of PENET Tool

Currently, users can create new diagrams that represent PENET models, and perform various essential operations on existing diagrams: opening, saving, exporting current diagram to an image file, and printing. Figure 3.2 shows *File* menu items, where all file-related operations are stored. Most of these options are available on tool's main toolbar as well.

3.3.2 Drawing and Editing Diagrams

Drawing and editing can be defined as a set of operations related to manipulations of diagrams that represent PENET model. In a nutshell, PENET model is a deterministic time transitions Petri net (DTTPN) [9] with specific parameters, but it follows almost all rules of timed Petri nets. Thus, diagram editor is basically a Petri net graphical editor with additional set features focused on supporting PENET models.

Built-in diagram editor is a crucial selling point of our tool. From drawing perspective, diagrams consist of places, transitions and links that connect them into a single model. Places are presented as circle-alike objects. Every place requires unique name, and might have optional token arrival information, in a case of a leaf place. Transitions enable firing - movement of tokens from one place to another. In PENET tool, as well as in traditional timed Petri nets, transitions are either immediate or timed. Immediate transitions are shown with black filled rectangle and represent zero firing delay time. In PENET model, transitions can be either compromise or repair transitions, based on whether they lead to further attack or repair state.

Place, *Trans.*, and *Arc* on toolbar and main application menu are provided for adding these basic elements. Drag-and-drop interface is implemented for adding all elements. User selects desired element on toolbar, and when mouse button is released on desired location on the drawing space, new item is dropped there. As already mentioned, arcs connect place and transition, and they can be added only between one place and one transition. After *Arc* is selected on toolbar, two mouse clicks within selected objects will draw link between them.

Figure 3.3 shows basic construct of PENET model, AND event, drawn using PENET Tool. There are three places (*P0*, *P1*, and *P2*), and one immediate transition (*T0*), as well as three arcs that connect immediate transition with three places.

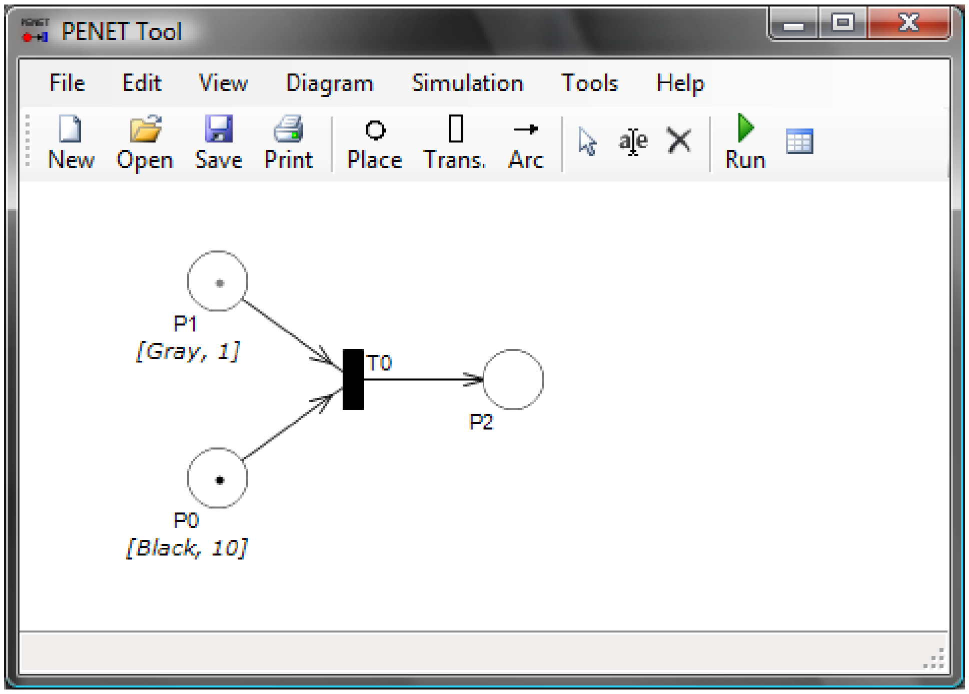


Figure 3.3 AND event shown in PENET Tool

As it can be seen on Figure 3.3, some places such as *P0* and *P1* contain additional information on the diagram. The label *[Black, 10]* means that this place regularly generates single black token every 10 time units.

There are multiple ways to edit existing items on a diagram. On Figure 3.4 selection sensitive right-click menu is presented. In this particular case, Place *P1* is selected, and right-click menu allows general actions and few actions specific to places only, such as *Set Token Arrival Data* and *Set As Root Goal*. Thus, every place can have two unique characteristics: token arrival data and whether it is a root goal or not.

When transition is selected, selection sensitive right-lick menu changes to operations reflecting transitions, like on Figure 3.5. Beside various *Set* actions, user can rotate transition for better fit on the diagram, and edit firing rules and results. *Edit Firing Rules and Results* action opens new windows as shown on Figure 3.6. These rules determine requirements for firing particular transition, and output of successful firing. Every transition must have defined requirements and rules for whole PENET model to be valid.

Diagram menu item that is part of main application menu bar is also selection sensitive,

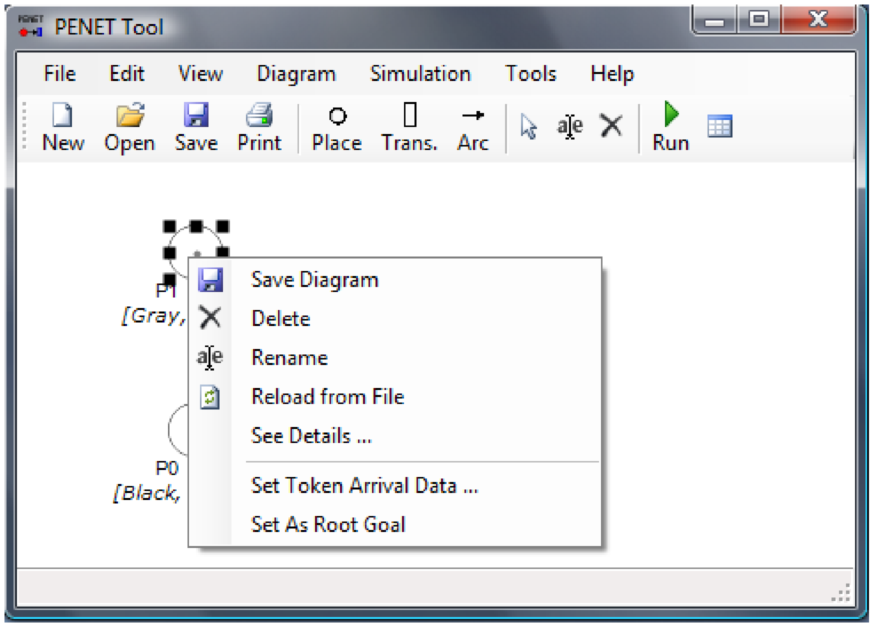


Figure 3.4 Right-click menu for selected place

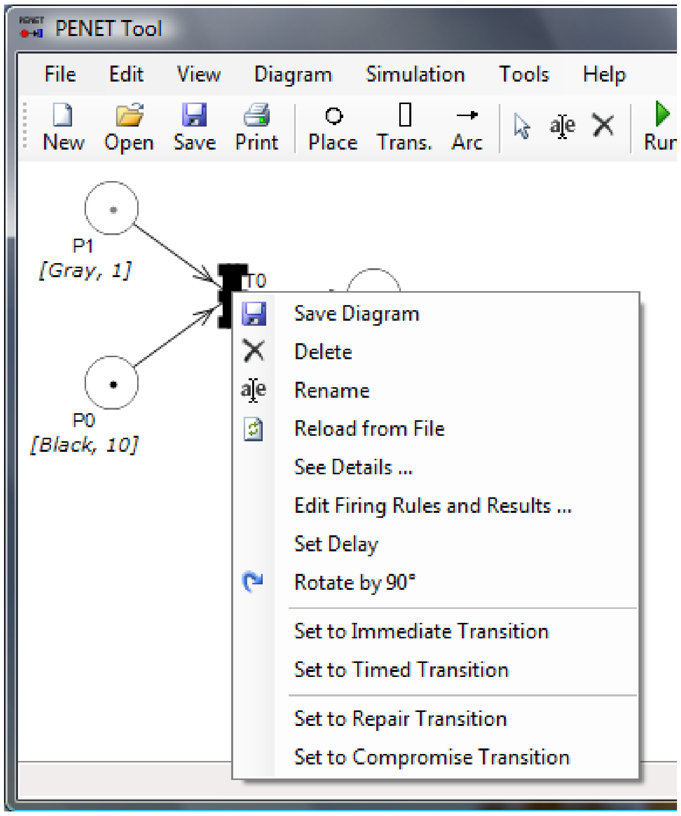
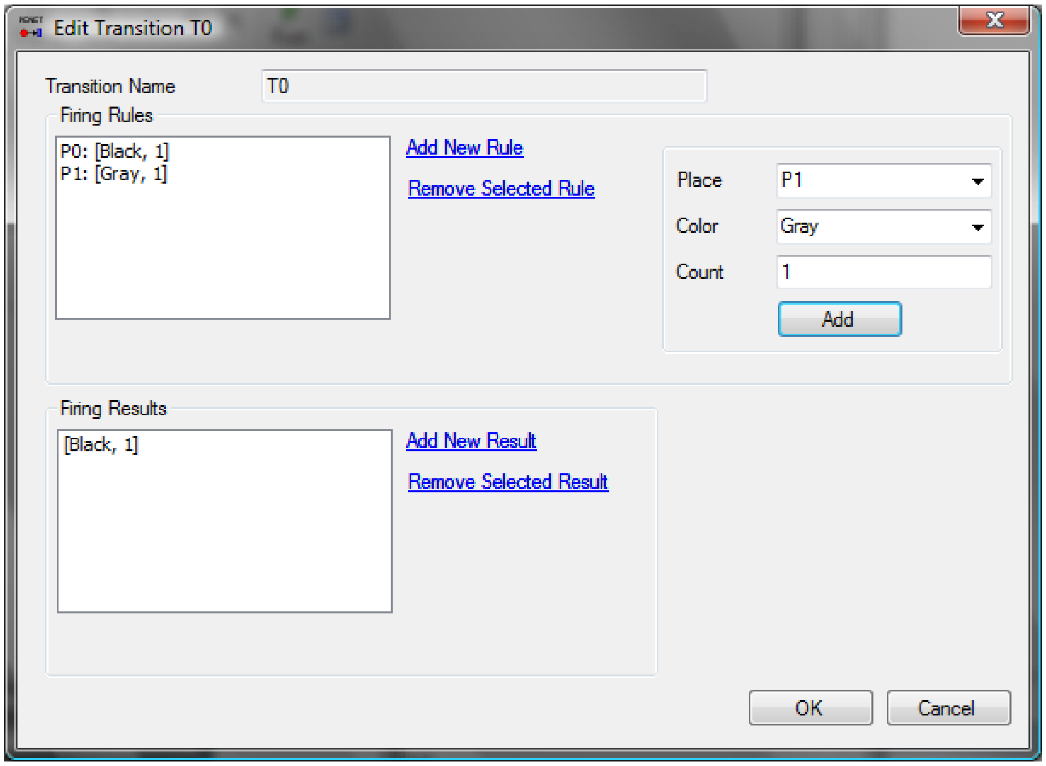


Figure 3.5 Right-click menu for selected transition


Edit Transition T0

Transition Name: T0

Firing Rules

P0: [Black, 1]
P1: [Gray, 1]

Place: P1
Color: Gray
Count: 1

Firing Results

[Black, 1]

Figure 3.6   Editing transition firing rules and results

and shadows presented actions available on right-click menu.

Beside crucial features such as adding and editing items, diagram editor contains additional features: moving, resizing, removing items, smart connect, and grid alignment. Moving feature provides ability to move any individual place or transition, or a collection of them. Similarly works remove feature. Resize allows any place or transition to be resized. Smart connect ensures that arcs connect places and transitions, but not two places or two transitions. Grid alignment aligns manually placed places and transitions to invisible grid points. *See Item Details* action provides read-only access to properties of a selected object.

3.3.3 Firing Rules and Results

Firing rules are set of rules that are required for transition to fire. Firing results are specification of output of a successful transition fire. Figure 3.6 shows firing rules and results for a transition *T0* on Figure 3.5. As it can be seen from figure, for *T0* to fire, both input places *P0* and *P1* have to contain at least one token: gray token in place *P1* and black in place *P0*. The output of such firing is one black token to all output places, in this case single place *P2*.

When places and transition are connected with arcs, PENET Tool tries to recognize firing rules that enable transition, and firing output of it. However, this is not always possible due to presence of tokens of various colors in incoming and outgoing places. Thus, it is necessary to review firing information using *Edit Firing Rules and Results* action in any non-trivial case that involves multiple colors.

3.4 Running Simulations and Result Analysis

Next logical step in PENET Tool use is running time domain simulations on created PENET model diagrams. PENET Tool incorporate time domain analysis algorithm that is used for running simulations and later results analysis. Simulations can be performed on valid PENET models.

3.4.1 About Valid PENET Models

Valid PENET model satisfies few requirements. Although simulations can be ran on invalid model, unexpected and unpredictable behavior may incur. First, there should not be floating elements, such as unconnected places or transitions. Second, no immediate transition shall have input and output at the same place. This would result in an infinite loop in the simulator. Next, there should not be any loops solely consisting of immediate transitions. Transition times should have non-negative values. And finally, firing rules on transitions should match places connected to them.

3.4.2 Running Simulations

There are two essentially same ways to run simulation. *Simulation* menu of main application menu bar contains *Run Simulation* and *Run Simulation from File* action items. For former, this will invoke simulation of PENET model currently opened in diagram editor. For later, user will be prompted to navigate to desired PENET diagram file.

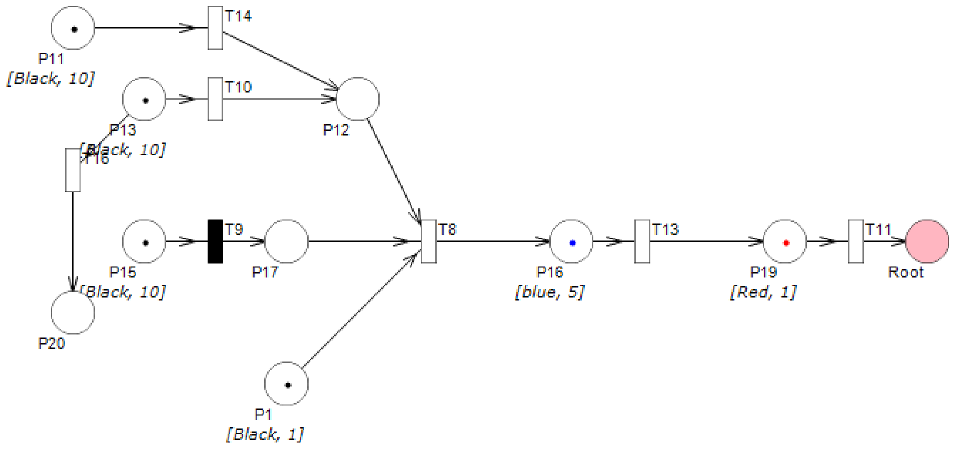


Figure 3.7 Sample PENET model diagram

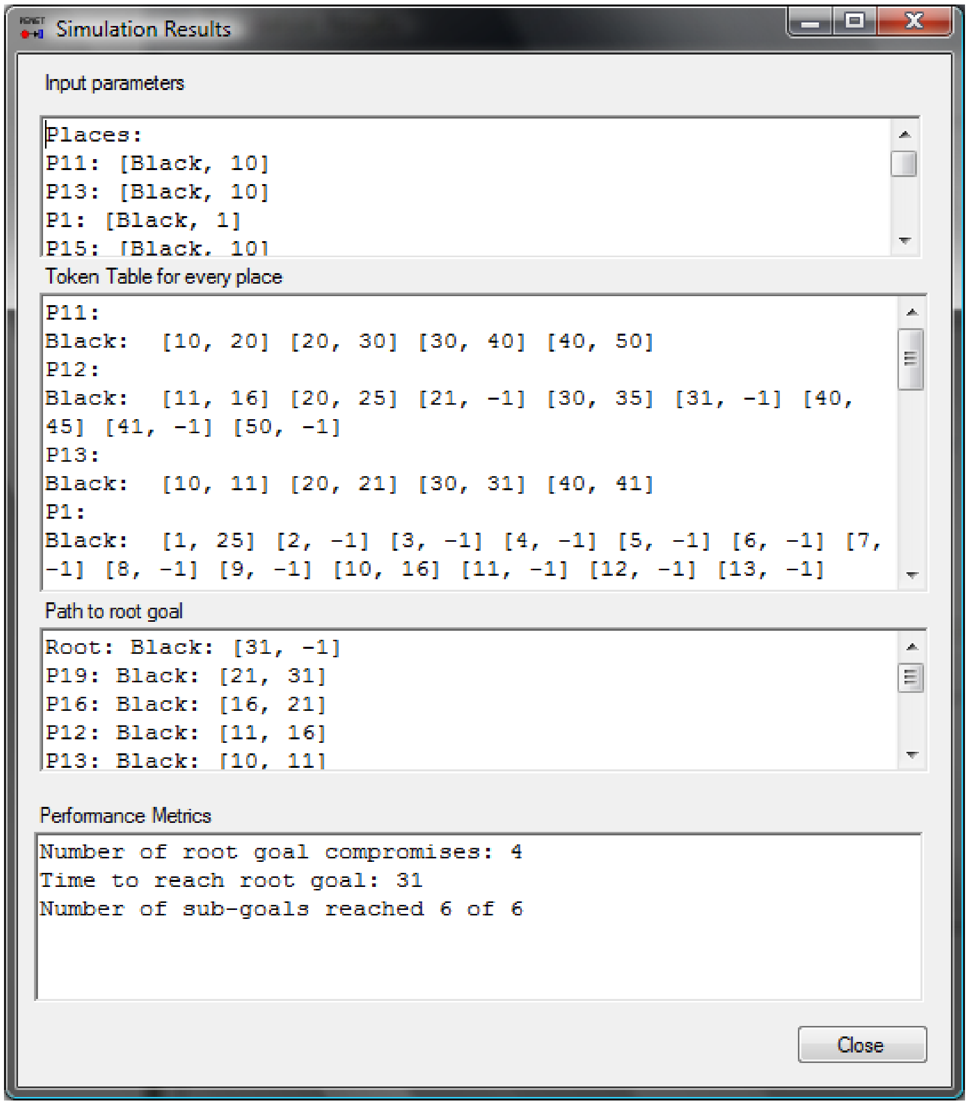
Default simulation run time is 40 hours (or time units!), which represents time in which new token arrivals at leaf nodes are being generated, and this value can changed using *Simulation* menu. However, actual simulation will last until there are events in queue left to execute, or until another threshold, *maximum simulation time* is reached.

3.4.3 Simulation Results and Analysis

To illustrate this subsection, we will use sample PENET model diagram like on Figure 3.7.

Figure 3.8 shows *Simulation Results* window that will be invoked once simulation of diagram from Figure 3.7 is completed. This screen consists of four sections. First section provides input parameters information such as arrival and compromise time delays for places and transitions. Second section shows contents of token table for each place in the PENET model. Token table (TT on Algorithm 2.4) shows each token that was present at a particular place. These entries on each place are separated by token color. Third section shows attack trace of a successful attack. Attack trace consists of tokens and places that were instrumental for attack to succeed. Last section contains performance metrics associated with simulation results. These performance metrics were defined earlier in PENET approach definition.

Beside simulation results window, trace is visually shown after simulation, like on the Figure 3.9. Places that were part of successful attack are filled with light blue color. Trace is



```
Places:
P11: [Black, 10]
P13: [Black, 10]
P1: [Black, 1]
P15: [Black, 10]
```

```
P11:
Black:  [10, 20] [20, 30] [30, 40] [40, 50]
P12:
Black:  [11, 16] [20, 25] [21, -1] [30, 35] [31, -1] [40,
45] [41, -1] [50, -1]
P13:
Black:  [10, 11] [20, 21] [30, 31] [40, 41]
P1:
Black:  [1, 25] [2, -1] [3, -1] [4, -1] [5, -1] [6, -1] [7,
-1] [8, -1] [9, -1] [10, 16] [11, -1] [12, -1] [13, -1]
```

```
Root: Black: [31, -1]
P19: Black: [21, 31]
P16: Black: [16, 21]
P12: Black: [11, 16]
P13: Black: [10, 11]
```

```
Number of root goal compromises: 4
Time to reach root goal: 31
Number of sub-goals reached 6 of 6
```

Figure 3.8 Simulation results for a sample model on Figure 3.7

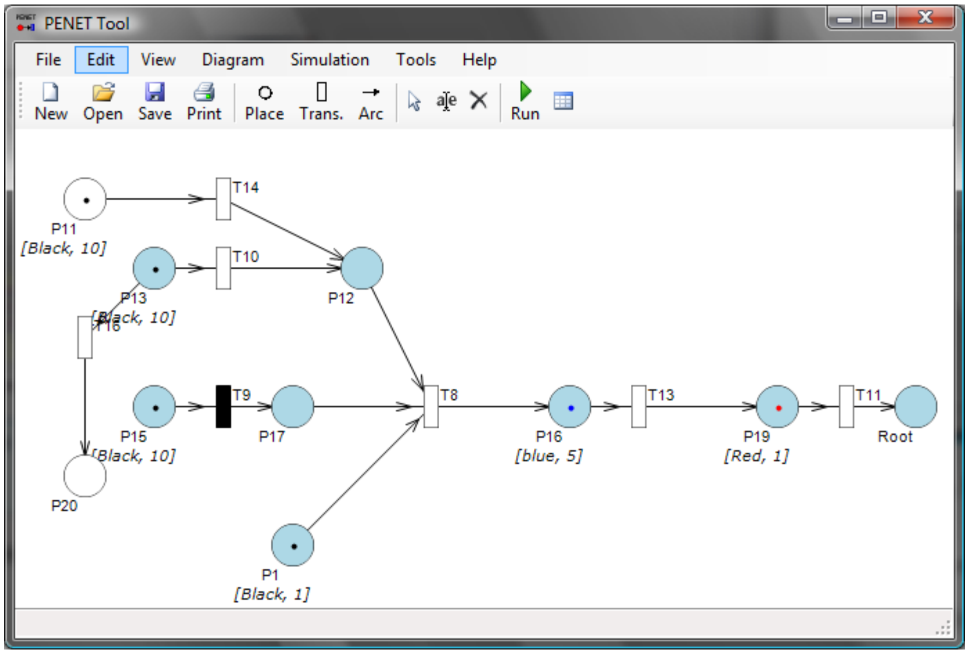


Figure 3.9 Successful attack trace with tracked places shown in blue

established by tracing all tokens that resulted in token being present at the root place. Since every particular token table entry contains ID field, it is possible to identify all tokens that were part of a successful attack. Trace starts with the token that is present at the root node, and recursively looks for tokens that lead to it. For example, token at root place was result of firing of *T11* by two tokens present at place *P19*.

3.4.4 Security Evaluation of Survivability and Defense Strategies

Although developing a representative model of system or attacker behavior might seem to be main goal of PENET Tool, ultimate goal of PENET Tool is to enhance the survivability of the modeled system.

Simulation Results window provides means for victim party to evaluate survivability, validate system security, and evaluate employed defense strategies. Performance metrics provide quantitative material necessary for such analysis. It is obvious that secure system has no compromises at root goal place and limited number of sub-goals compromised (the more sub-goals were compromised, the system is less secure).

The user can vary parameters of PENET model that are of interest and influence, add new elements that improve defense, and compare with existing results. Once simulation is re-ran, *Simulation Results* form will show new input and output (trace and performance metrics) for easy tracking how changes have affected the outcome. Parameters of interest have significant impact to performance metrics, and their value is influenced either by the attacker or the victim. Beside modifying parameters, defense strategy can be improved by adding additional valid repair places. As it will be shown in our TCP SYN case study (chapter 5), repair time significantly affects survivability of a vulnerable system.

Evaluation process can be performed systematically, and repeated if necessary until results are satisfactory. The evaluation and validation procedure is shown in Algorithm 2.5. From this algorithm it can be seen that process is essentially iterative.

CHAPTER 4. PENET TOOL DESIGN AND IMPLEMENTATION

4.1 Introduction

In this chapter focus will be given to internal design of PENET Tool. For easier understanding of design, the internal design will be described on the logical level, rather than on the code level. In a nutshell, PENET Tool is event-driven application where actions result of user interactions with applications user interface. In that sense, user operates on front-end consisting of various windows forms, and back-end of application performs various calculations and support for front-end.

To meet applications goals, various data structures are being used through the back-end. Most distinguished data structure in application describes the PENET model, the set of Petri net basic elements, constructs, parameters, and interactions. This structure is used by most PENET Tool modules, for example as an input for the simulator. Externally, PENET model data structure is converted to XML structure that manages to fully describe and capture the PENET model.

4.2 Block Diagram of PENET Tool Architecture

Figure 4.1 shows various modules of PENET Tool and their interaction with each other. Module interaction occurs in few different ways: the data could be shared between modules, one module uses another, or there could be two-way communication. The major functional modules are represented as larger and darker rectangles on Figure 4.1. By rule, minor modules appear only as a part of major module, or represent data structure that is used by a major module.

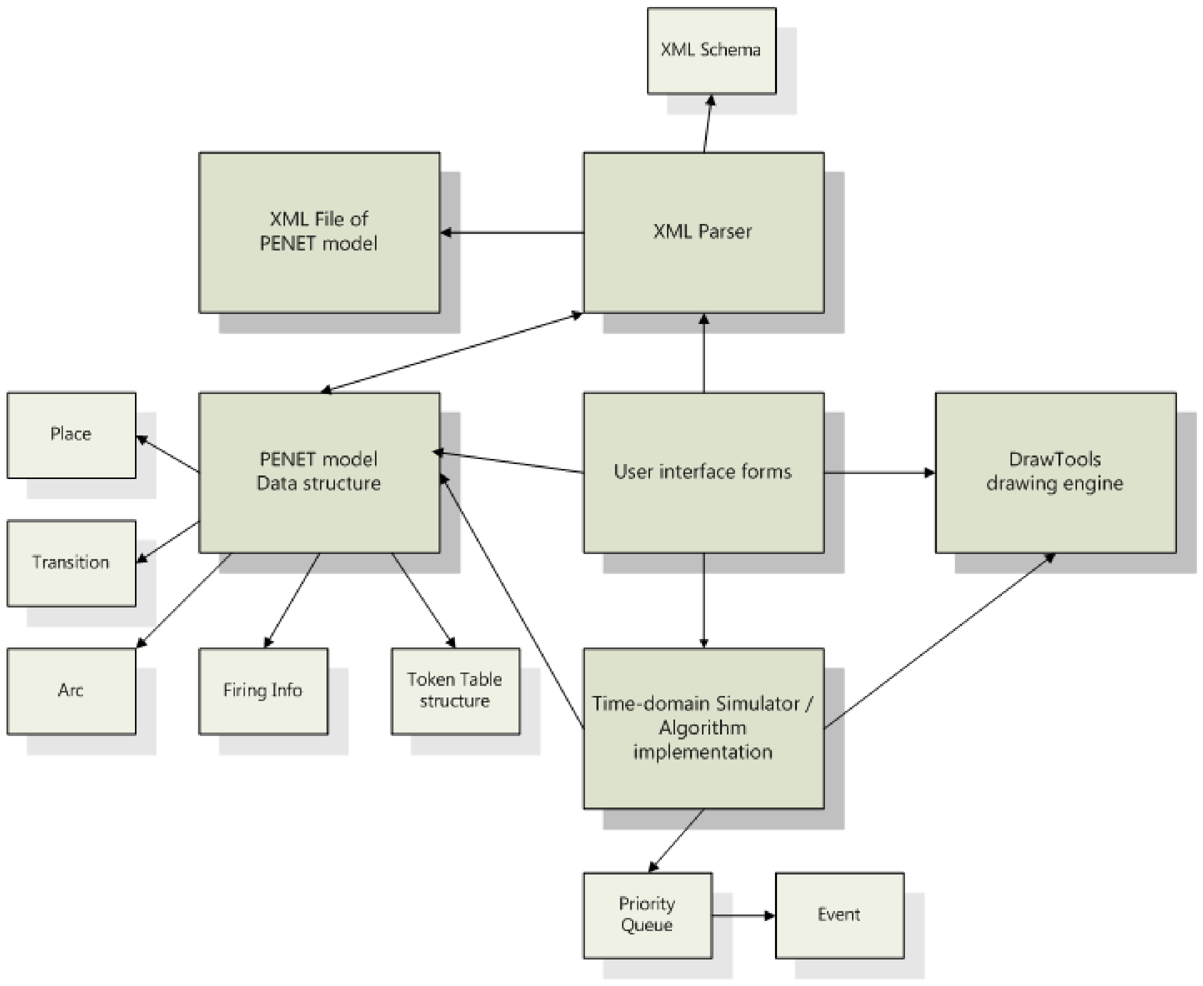


Figure 4.1 Block diagram of PENET Tool internal structure

4.2.1 XML modules

Primary purpose of XML modules is to provide permanent storage of created PENET models. Thus, PENET models created in graphical editor will be stored as XML files. XML format was chosen due to its ability to fully describe structure, relationship between elements, and parameters of any PENET model. *XML Parser* module is used for two-way conversion between XML file and data structures that are friendlier for further data manipulation. *XML Schema* [23] is template that fully defines PENET model. As already mentioned, XML format is able to fully capture the PENET model. This is accomplished by well-defined XML schema file. Thus, this schema file contains definition for places, transitions, arcs, and all their properties and relationships. XML schema was created using *XML Schema Definition Language* (XSD) and *XML Designer* [24]. Figure 4.2 shows XML schema in *XML Designer*. Further, XML schema file is used as a template for every creation or opening of the XML files that represent PENET model.

However, XML file is difficult to manipulate with in a native form. Application uses specific PENET model data structure for all internal purposes, aside of saving to permanent storage.

4.2.2 PENET Model Data Structure

The PENET model data structure (Figure 4.1) consists of definitions for each basic element (*Place*, *Transition*, *Arc*), and structures and relationships that are necessary to describe their interactions in the overall PENET model. *Firing Info* structure provides information about requirements for transmission firing, and information about result of transition firing. *Token Table* structure is used to keep track of individual tokens at each place. This is fundamental for ensuring that token progress is kept accurate.

As it can be seen from Figure 4.1, PENET model data structure is widely used in PENET Tool: it is input for performing simulations, *XML Parser* uses it to create the XML file, and drawing engine uses its information for proper display of PENET model.

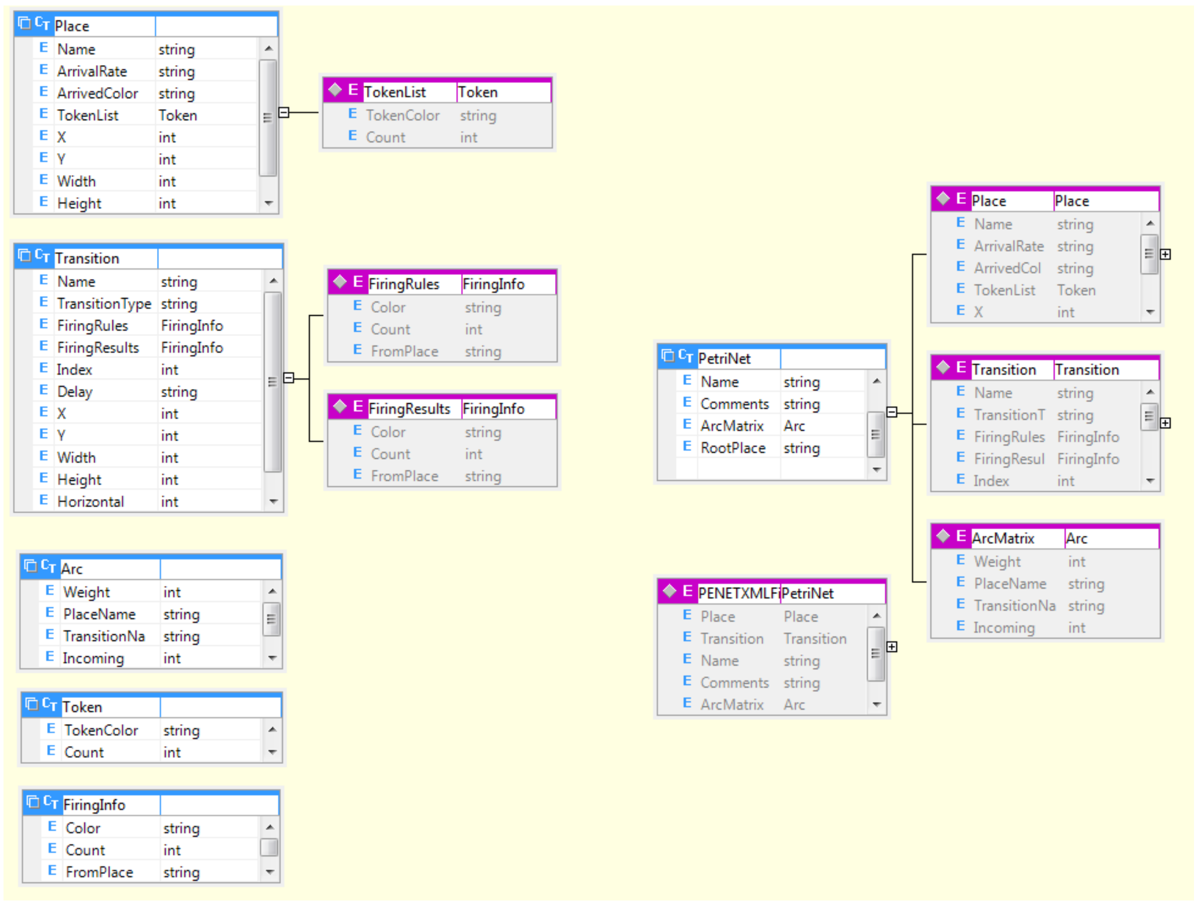


Figure 4.2 Block diagram of XML schema that describes PENET model

4.2.3 Drawing Engine

Drawing engine is a GDI+ based [25] control that provides drawing of PENET model on the application's main window. Drawing engine is one of most complex modules in the whole application, thanks to rich set of implemented features. The GDI+ libraries provide means for easy manipulation of graphics on the display devices. Drawing engine relies on information in the PENET model data structure for providing up-to-date display of the PENET model. Drawing engine has its own data structures for storing items that are drawn on the screen. This includes information such as coordinates of elements, their color and shape, labels, token colors, fill, etc. There is 1-to-1 mapping between full PENET model data and data structure in drawing engine used to draw elements. Interestingly, arcs are not stored in drawing engine but computed at run time using connected place and transition information which is sufficient to draw any arc. This approach was used because 2D coordinates of an arc strongly dependent of coordinates of place and transition connected by it. On the other hand, some actions performed on graphics engine propagate to PENET model. For example, removing an element on the diagram will invoke the removal of corresponding element in PENET model, and all arcs connected to it (if removed element was place or transition).

4.2.3.1 Polymorphism in Drawing Engine

As already mentioned, drawing engine recognizes three types of elements: places, transitions, and arcs. There are quite a few common actions and properties for each element. Each can be drawn, moved, removed, renamed, and such. In object-oriented design, there is concept of *polymorphism* that is strongly applicable to this situation. Applied to this case, it means that all objects drawn using drawing engine can be stored in a single container and that for majority of operations theres no need to examine type of the object. Additional benefit of using single container for all elements is that at run-time various elements are added to diagram strictly by user choice. Utilization of polymorphism has led to a simplified design.

4.3 Implementation of the Time Domain Analysis Algorithm

On block diagram of PENET Tool (Figure 4.1) it is evident that simulation module uses data structure of PENET model as input for running simulations. The simulator closely follows the PENET time domain analysis for single attack scenario in Algorithm 2.4. Thus, all structures described in Algorithm 2.4 have been implemented.

Simulator introduces new data structures used for simulation purposes: *Event*, *Priority Queue*, and *Token Table*. *Event* structure provides information about upcoming token arrival: place of arrival, fired transition, and what are firing requirements and results of firing. *Priority Queue* is used to sort events per their execution time. *Priority Queue* stores events in efficient way that guarantees that extracted first element from queue (dequeuing operation) will be the event with lowest start time. *Token Table* is structure that tracks progress of tokens. Each place of PENET model has its own token table. Each token table entry contains information about token arrival and leave times, color, and unique ID used for tracking purposes. Thus, every token arrival will invoke creation of new token table entry. Figure 3.8 shows contents of such table.

The simulator is event driven, in a sense that every transition firing and token generation are represented as single event. Every such event may lead to new events, for example token arrival at a place may enable firing of some connected transition. Thus, at every event incursion at some place algorithm evaluates prospective firings of all connected transitions. Simulation ends when there are no events left in queue, or if simulation time has expired.

Consider the situation when token arrival enables firing of multiple transitions. Based on PENET approach, the transition with lowest delay would be one that is fired, and remaining transitions would not fire. Transition with lowest delay has its own set of rules that might change from satisfied to not satisfied before scheduled firing occurs. In this situation, scheduled firing event will not be realized, but other remaining prospective transitions have to be rechecked. It is important to state that PENET Tool successfully handles such and even more complex cases. These cases are further discussed in challenges section.

4.3.1 Post-Simulation Analysis Implementation

Finally, last logical module of PENET Tool is post-simulation analysis. This module is automatically invoked after successful simulation. Purpose of this module was described in PENET Tool overview in previous chapter. Technically, it calculates performance metrics and the trace for completed simulation. Both are acquired by analyzing the token table after simulation has been completed. *Number of root goal compromises* is simply number of tokens registered at the root place. *Time to reach root goal* is simply arrival time of first entry in the token table. *Number of sub-goals reached* can be learned from last firing times of each transition that is marked as a compromise transition.

Trace path is also obtained using token table entries. Because every token entry has its own unique ID, it is possible to trace all firings and tokens that resulted in particular token at root place. For this purposes, every token table entry was augmented with firing requirements (that include unique token ID from input places) that have led to that entry.

4.4 Discussion of Challenges in Implementation of the PENET Tool

4.4.1 Complex Issues in Time Domain Simulator

In a event-based simulation, it might seem that events are linearly following each other in time domain, with events occurring as scheduled. Unfortunately, that is not a case. Consider a PENET model consisting of two AND events, shown on Figure 4.3. At time $\tau = 5$, requirements for firing transition *T0* will be met, and firing would occur at $\tau = 5 + 10 = 15$. However, at time $\tau = 8$, transition *T1* becomes enabled, and it could fire at $\tau = 8 + 4 = 12$, which is before firing of transition *T0*. Thus, in correct time-domain analysis implementation *T1* would fire, and *T0* would not fire at expected $\tau = 15$, because token at place *P1* would no longer be available.

Now consider even more complex situation, as shown on Figure 4.4. To explain point, we will assume that there is no periodic arrival of tokens. In this situation, based on arrival events, transition *T0* should fire at $\tau = 15$, followed by *T1* firing at $\tau = 12$, and *T2* firing at $\tau = 11$.

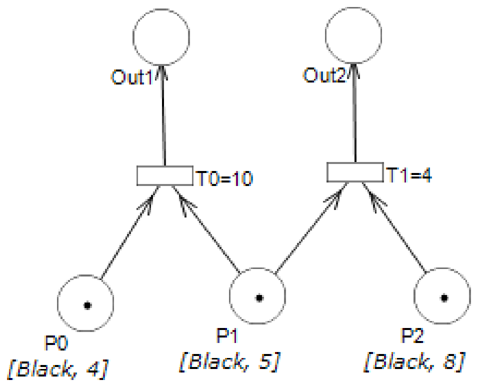


Figure 4.3 PENET model with an “anomaly“

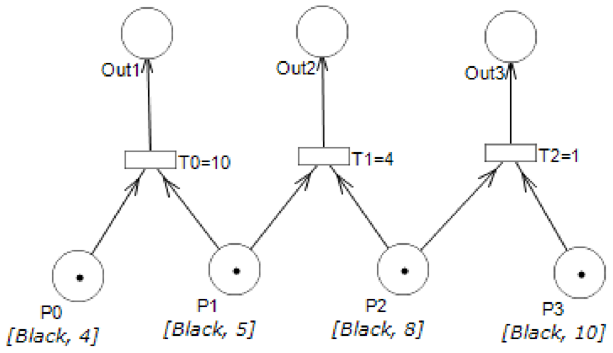


Figure 4.4 PENET model with multiple ”anomalies“

T2 firing at $\tau = 11$ will render *T1* firing at $\tau = 12$ invalid, but firing of *T0* at $\tau = 15$ should occur.

From these two cases we can draw conclusion that event-based simulation has anomalies, situations when events supersede other events and make them invalid and cancelled. Specifically, within transition delay time some other event can take one or more required tokens. In other words, between transition enabling time and actual firing time other events can occur that disturb order of events. These situations have to be properly handled by implemented time-domain simulator.

Practically, to ensure that simulator works as desired in presence of such anomalies, two steps must be taken:

- Simulator has to schedule all possible firings on token level, and

- Firing requirements have to be re-evaluated when transition firing is due.

To explain the first step, consider the PENET model on Figure 4.3 again. At time $\tau = 10$ there are two tokens at place *P0*, one at places *P1* and *P2*, and new token arrives at place *P1*. Thus, there are thee possible firings that need to be scheduled: two on transition *T0*, and one at transition *T1*. And every token present at *P0* has to be treated as a separate firing possibility. Therefore, set of events (in a general case) will be generated as a consequence of some single event. There could be multiple events from single enabled transition if there are multiple tokens in required places. Also, if event enables other transitions, these will get scheduled as well.

Second step is corollary of fact that only single firing will occur for whole set because set has one common requirement. Once one transition has used the common requirement for its firing, no other scheduled event from the set can fire. Thus, it is necessary to check at every firing whether requirements are still met, and to cancel events with invalid requirements.

4.4.2 Analysis of Algorithms Convergence and Complexity

It is necessary to show that time-domain algorithm converges towards solution if provided with valid input.

In this section, we will attempt to derive algorithm's ability to converge. In this sense, the important parameters are number of transitions n_t in PENET model, initial number of events generated from token arrivals at leaf places n_{e0}, maximal input or output degree of any event d. Note that every token arrival at some place is result of a single event. Also, acyclic PENET model is model where there are no transition firing combinations that would result in event at same place where they started.

From one event, in worst case $n_t \cdot n_{e0}^d$ events can be created. Each of these events has one common requirement, and that is the initial event that enabled possibility of new events to be generated. Once one if these events has been processed, remaining events will not meet their requirements. Thus, out of so many events created, only single transition will manage to be fired, and it will remove all remaining events. This translates to rudimentary rule that each event generates at most d new events.

If every event would generate up to d new events, algorithm would not converge. However, if PENET model is acyclic, every event propagates in direction towards root or repair place, and places are not being reused. This means that there will be at most $n_{e0} \cdot n_t \cdot d!$ events generated from initial n_{e0} events (that appear after token arrival events at leaf places have been processed). Thus, minimum priority queue will contain at most $n_{e0} \cdot (1 + n_t \cdot d!)$ elements. This puts bound on algorithms complexity and proves that algorithm will converge towards solution in case that theres no cycles in PENET model.

In a case of a cyclic PENET model with delay time in cycle larger than 0, simulation will run between time as if there were no cycles and maximum simulation time, depending of individual parameters of PENET model. If PENET model contains cycles with delay equal to 0, such model is invalid.

We have been shown that algorithm will converge unless there are zero-delay cycles, and that algorithms execution is efficient if there are no cycles. In case of existence of non zero-delay cycles in PENET model, in worst case algorithm will run until specified simulation time.

4.4.3 Additional Limitations of Time Domain Analysis Algorithm

Although design desire is to make PENET Tool 100 percent deterministic, in some cases there will be conflicting events occurring at exactly same time. In such case, there is no rule which event will occur, and which ones will be cancelled. Existing implementation does not consider this case, but mechanism could be added that recognizes such situations and notifies user that result was not deterministic.

Second issue occurs in situations of large token accumulation at some place. Because time domain algorithm evaluates every potential firing on individual token level, this will create large set of potential events for some connected transition. As already described, only single event will manage to fire, and requirements for remaining events in this set will not be met. The issue is that large number of events in the priority queue will slow down execution of the algorithm, regardless of fact that all but one event in newly created set will have unmet firing requirements. Data structures that store events in priority queue dynamically expand in run-time. Every expansion is performed by copying all events in the structure. Ratio of events generated to fired can be over 10^5 for simple case of three inputs AND gate and one place with 100 tokens present. This results in algorithm execution slowdown due to processing large number of invalid events. It is obvious that limit can be put in number of newly created events, because most of them get eliminated anyway due to unmet requirements. Different approach would be to create single event for whole set of mutually exclusive events, but even in this case large number of requirements would be part of *Event* data structure. Decision was made to find compromise between lower complexness and readability on one and performance improvement on the other side. The actual implementation has heuristic cut-off in which events generated at single firing are limited $n = 1000$. Experiments have shown that this cut-off does not affect result in any way. In other words, it is safe to assume that 1000 event combinations based on present tokens in input places are enough for single transition. Algorithm can be further developed to optimize these cases without theoretical loss of precision.

CHAPTER 5. CASE STUDY

In order to illustrate our Petri Net Attack Modeling approach, we will examine one of most common distributed denial of service (DDoS) attacks based on TCP SYN abuse in a case study. In this case study, we will describe the TCP SYN attack, and based on this description we will generate attack model by utilizing our novel attack modeling approach. Then, analysis methods defined in this thesis will be applied to generated PENET model. To obtain simulation results for time domain analysis, we have used earlier discussed PENET Tool that contains implementation of time domain analysis algorithm.

5.1 Model

The abuse, described in [15], works as follows (Figure 5.1). Initially, client (attacker in case of the abuse) sends request to establish connection by sending SYN message to the server. Server then responds with SYN-ACK message, and keeps half-open connection between itself and the attacker, waiting for final ACK response from the client. To keep track of all half-open

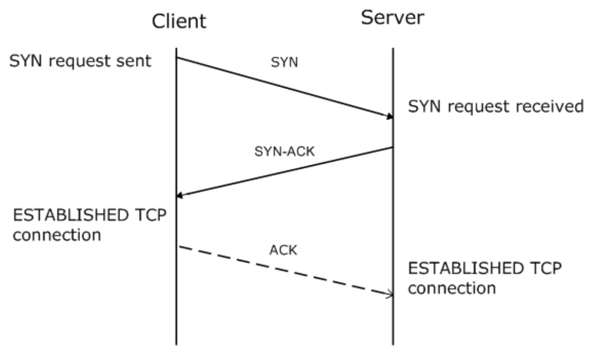


Figure 5.1 Establishing TCP connection using SYN message

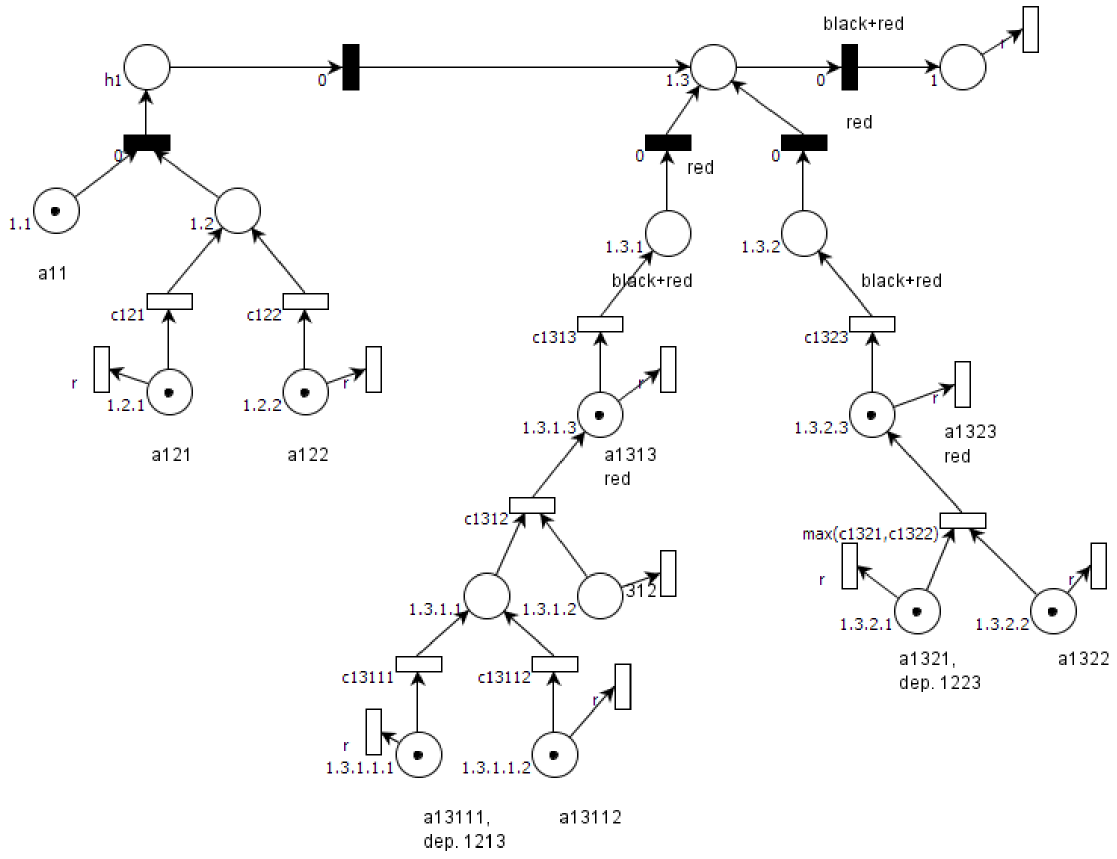


Figure 5.2 PENET model for TCP SYN DDoS Attack

connection, server stores in its memory list of all half-open connections. The attacker has spoofed its source address (TCP protocol does not provide checking for validity of source of the transmission), and server will never receive reply for its ACK message. As the server has limited number of available connections, those connections will eventually fill up, and server will be unable to service the legitimate users.

Figure 5.2 presents PENET model of described TCP SYN attack. Description of places is presented in Table 5.1. The goal of intrusion attempt is to disrupt or affect regular service utilizing TCP SYN DDoS attack. As it can be seen from Figure 5.2, model is structurally divided into two separate entities. Each entity has to be compromised by attacker in order for attack to succeed.

For simplicity reasons, and to illustrate attack patterns [8] often used in attack trees that are applicable to PENET model as well, we have replaced first entity with three patterns, shown on Figures 2.4, and 5.3.a and b. These patterns are not specific to TCP SYN abuse and can be used for various purposes. Figure 5.2 shows how the patterns were instantiated as places *1.1*, *1.2.1* and *1.2.2*.

Sub-goal *1.1*, represented by attack pattern on Figure 5.3.a depicts intrusion attempts that have goal of acquiring relevant information about victim of the attack: internal network configuration, operating system of target server, whether it is susceptible to known vulnerabilities, and such.

Sub-goal *1.2* is represented by attack scenarios on Figure 5.3.b (goal *1.2.1*) and Figure 2.4 (goal *1.2.2*). *Botnet Recruiting Attack Pattern* (Figure 5.3.b) models the ability of attacker to recruit zombie (also known as slave, agent, bot) machines that will amplify the attack or make it possible. The attacker first needs to obtain a list of machines that are vulnerable to be taken over (*1.2.1.1* sub-goal), and then exploiting known vulnerabilities or injecting them with viruses, Trojan horses or worms gain root access (*1.2.1.2* sub-goal). The number of machines in botnet is not arbitrary, but significantly large number of nodes in botnet is required for DDoS attack to have negative impact on victim's service. We model this requirement with *1.2.1.3 (Sufficient Number of Hosts in Botnet).*

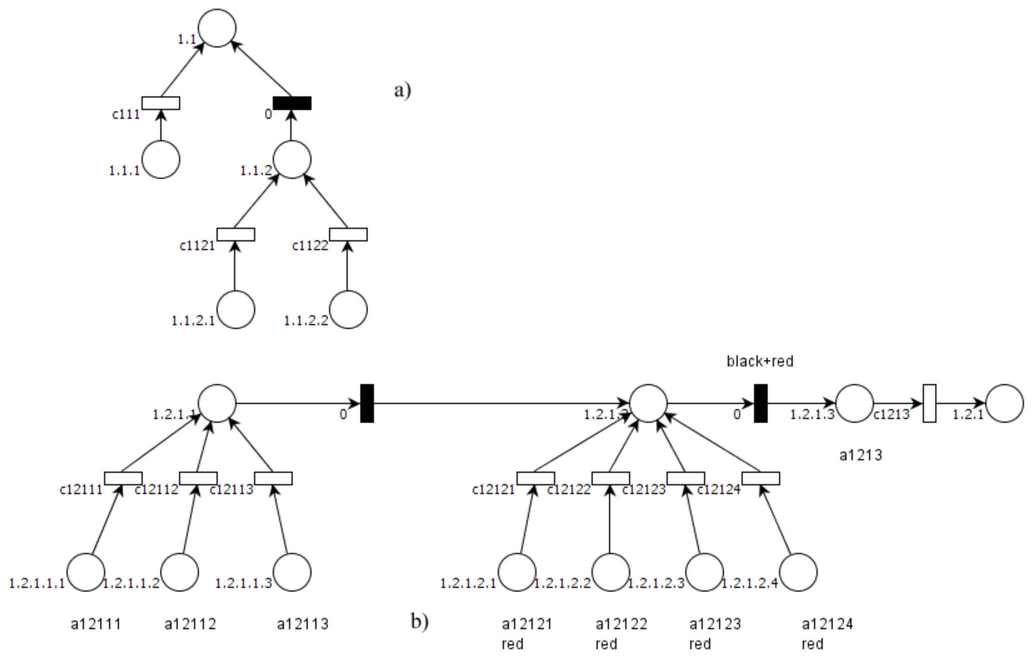


Figure 5.3 PENET Patterns a) Acquire relevant victim info pattern b) Compromise network and/or resources pattern

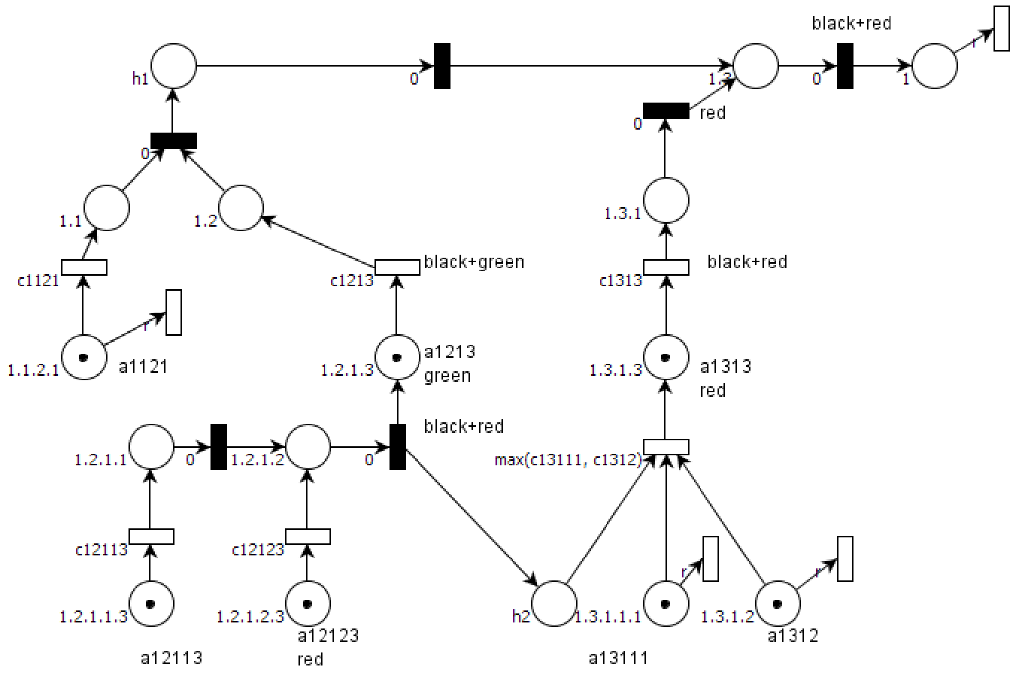


Figure 5.4 PENET model of most likely attack scenario

However, it is not necessary to use agent machines to perform TCP SYN DDoS attack; same effect can be created using *reflector routers* [15] that will amplify effect of false SYN requests by forwarding them to the victim. In this case, attacker has to follow similar path as in botnet recruitment. First, prospective routers need to be located, and then they need to be assigned to bot network. Finally, when satisfactory number of such routers is found, attacker has accomplished his subgoal of creating reflector router network. This is modeled by reflector routers attack pattern shown on the Figure 2.4.

Second entity on Figure 5.2 captures specifics of TCP SYN attack. Based on acquired information in other two sub-trees, attacker decides what kind of attack to perform. If he/she knows for that target is not vulnerable to TCP SYN attack, decision will be made to try brute-force DDoS attack that is not as refined as classic one, but instead relies on brute-force [16]. For TCP SYN attack to succeed, victim has to be vulnerable to TCP SYN abuse (*1.3.1.2* goal), and volume of abuse generated must reach critical level that accomplishes desired effect of attack. Volume of abuse is modeled with *Attack Amplifier Requirements* (*1.3.1.1* sub-goal).

Therefore, for attacker to achieve desired effect, the disruption in connection queue, he must compromise both parts of PENET model, where right part is specific to this attack, and left one instantiates attack patterns. The desired effect of attack is to compromise ability of victim to service legitimate users. As a impact of attack to victim, its network bandwidth, processing power, or response time will deteriorate; or in worst case, service will be unavailable.

5.2 Analysis of Presented Model

The first goal in analysis is to find most likely scenario, in other words to find leaf places that produce lowest *Attack Scenario Index DI*. Parameters are assigned as shown on table 5.1. These parameters are cost of arrival AC and chance of success AP. Based on this information, most likely attack scenario involves following leaf places: *1.1.2.1*, *1.2.1.1.3*, *1.2.1.2.3*, *1.2.1.3*, *1.3.1.1.1*, *1.3.1.2*, and *1.3.1.3*.

The *DC*, *DP* and *Attack Scenario Index DI* values for this attack scenario are:

Table 5.1 Description of Figure 5.2 with assigned parameters

Place index	Place name	Cost of arrival AC	Chance of success AP
1.1	Acquire relevant victim info		
1.1.1	Target host vulnerability scans	1K	0.1
1.1.2	Locate servers		
1.1.2.1	Automatic scanning	100	0.1
1.1.2.2	Intelligence information	100K	0.9
1.2	Compromise network and/or machine resources		
1.2.1	Recruit multiple machines in botnet		
1.2.1.1	Obtain list of zombie machines		
1.2.1.1.1	Random scanning	1K	0.001
1.2.1.1.2	Signpost scanning	5K	0.01
1.2.1.1.3	Hit list scanning	10K	0.1
1.2.1.2	Take advantage of vulnerable machines		
1.2.1.2.1	Virus infections	100	0.01
1.2.1.2.2	Worm infections	100	0.01
1.2.1.2.3	Operating system vulnerabilities	100	0.05
1.2.1.2.4	Trojan horse infections	10	0.02
1.2.1.3	Sufficient number of hosts in botnet	10K	0.1
1.2.2	Achieve usability of reflector routers	200.1K	0.0001
1.3	SYN abuse by attacker		
1.3.1	SYN attack requirements		
1.3.1.1	Attack amplifier requirements		
1.3.1.1.1	Sufficient number of hosts in botnet for TCP SYN	100	0.3
1.3.1.1.2	Sufficient number of reflector routers for TCP SYN	1000	0.01
1.3.1.2	Victim is prone to TCP abuse	1K	0.3
1.3.1.3	Master attacker decision: TCP SYN	0	1
1.3.2	Brute force attack requirements		
1.3.2.1	Sufficient number of hosts in botnet for TCP SYN	1K	0.1
1.3.2.2	Victim is prone to brute force abuse	10K	0.9
1.3.2.3	Master machine decision: brute-force DDoS	0	1

$$DC = \sum_{j=0}^{6} AC_j = 21.3K\$$$

$$DP = \prod_{j=0}^{6} AP_j = 4.5 \cdot 10^{-6}$$

$$DI = \frac{DC}{DP} = \frac{21.3K\$}{4.5 \cdot 10^{-6}} = 4.73G\$$$

In time domain analysis, we will utilize the PENET model of attack scenario shown on Figure 5.4. First step is to recognize parameters that will influence the outcome the most, that are also controlable (to higher or lesser degree) by attacker or by victim. Compromise time $c = c_{1121}$ at transition that fires the place *1.1.2.1* is significant factor because it is de-facto the entry point of the attack, and both attacker and victim can influence this delay time. Second parameter of choice is clearly repair time. Note that repair time is not present at all places; victim does not have repair capabilities on systems that are external to him, such as botnet network.

Colors (*red* and *green*) were used for proper modeling of priority AND construct. PENET parameters a_j and c_j were obtained from Table 5.1. Cost of arrival AC_j was transfered to periodic arrival time a_j by dividing each value with constant $k = 100$ that gives arrival time of $a = 1$ hour to leaves with lowest cost of arrival. Similarly, compromise times c_j were calculated as reciprocal values of chances of success AC_j. Also, it was assumed that attacker performed attacks on leaf places in duration of $\tau = 80$ hours.

Figures 5.5, 5.6, and 5.7 show defined performance metrics for varying repair time r and parameter c=c_{1121}.

5.3 Results

5.3.1 Time to root place

Figure 5.5 shows dependence of time to compromise root place of repair r and compromise c times. Initially, small values of compromise time c do not influence the outcome because other compromise times in PENET model play dominant role. As compromise time on place

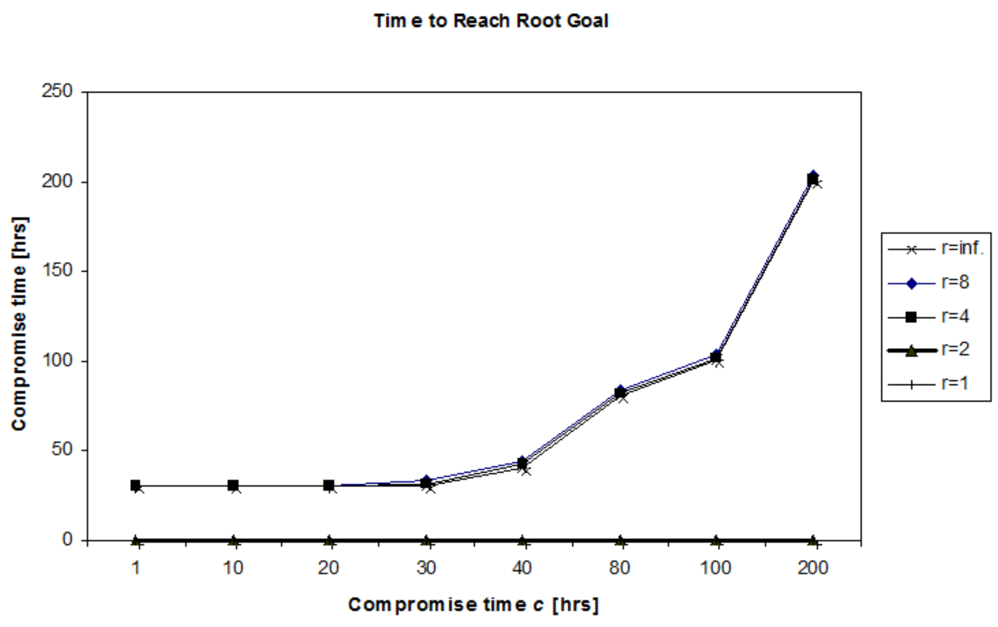


Figure 5.5 Time to root place compromise for TCP SYN model

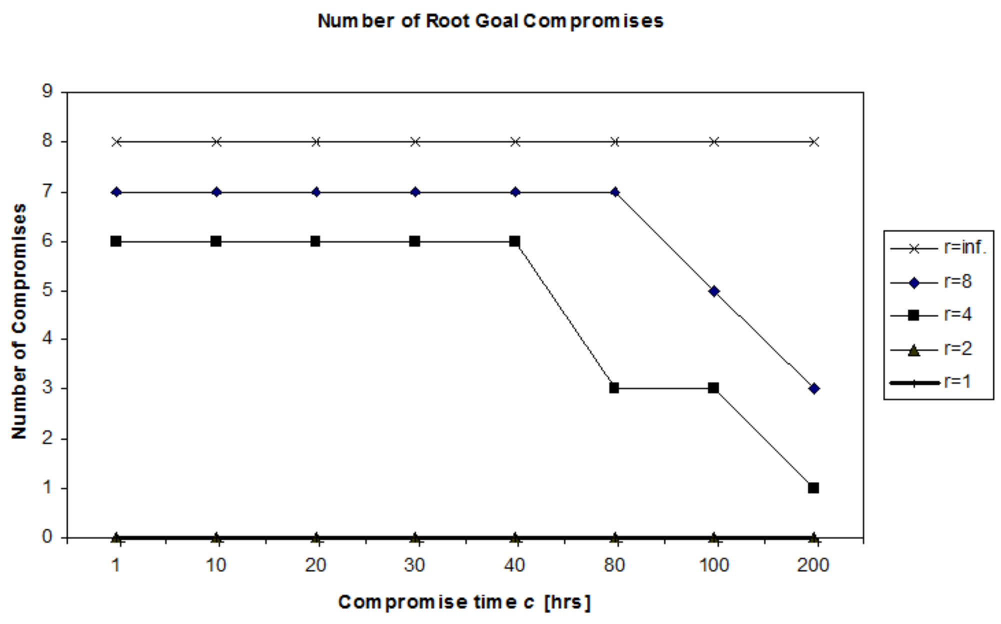


Figure 5.6 Number of root goal compromises for TCP SYN model

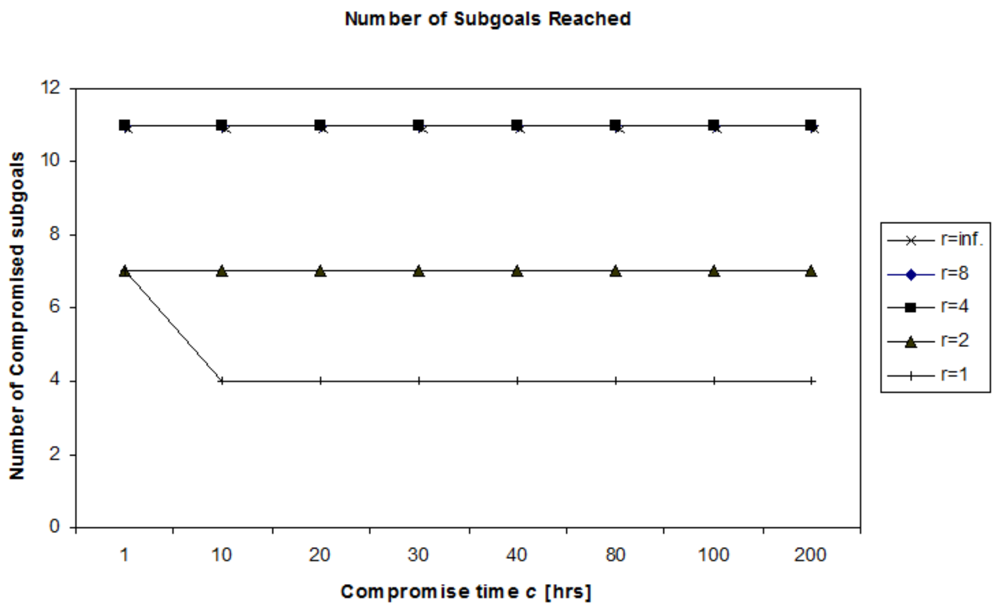


Figure 5.7 Number of subgoals compromised for TCP SYN model

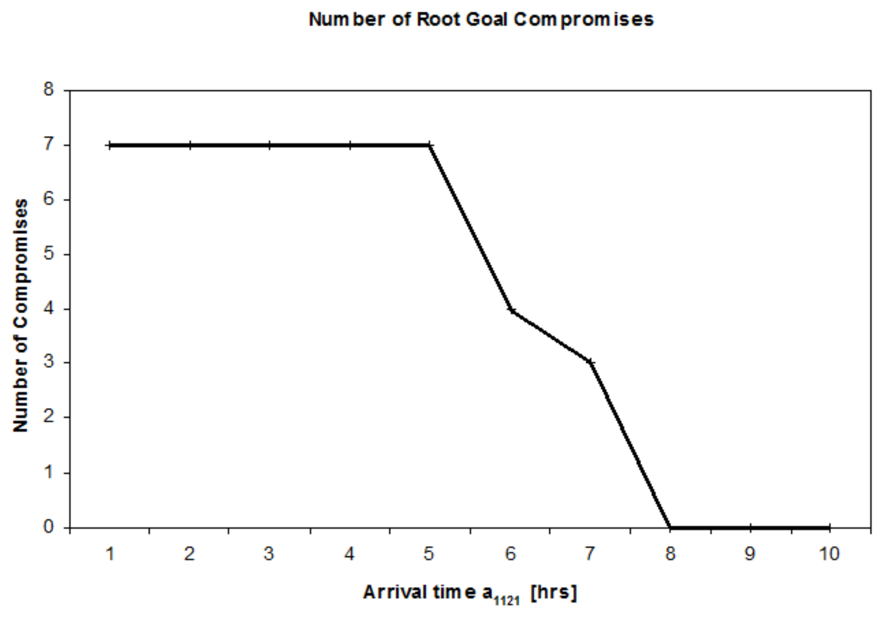


Figure 5.8 Number of root goal compromises for TCP SYN model for variable arrival time a_{1121}

1.1.2.1 increases beyond other compromise times in PENET model, it becomes dominant so the time to compromise root place starts to linearly increase with c. Also, graph shows that repair time only influence whether attack will succeed or not.

5.3.2 Number of root goal compromises

From Figure 5.6 we can conclude that compromise time c plays less important role compared to repair time r, when it comes to successful attack. Although number of compromises decreases as the compromise time increases, root place compromises terminate completely for small enough repair time (2 or less hours). As expected, when no repair capabilities exist (represented by infinite repair time), number of root goal compromises would not change regardless of compromise time.

5.3.3 Number of subgoals compromised

Figure 5.7 shows number of subgoals reached for various values of parameters c and r. The maximum number of subgoals reached is 11, which includes root goal. As we have seen from Figure 5.6, for small enough repair time attacker will not manage to reach root place. In these cases, attacker manages to compromise between 4 and 7 subgoals.

5.3.4 Effect of repair time

From all three diagrams, we can conclude that repairability and its promptness play crucial role in keeping the system safe from TCP SYN attack. The attacker has no means to penetrate system beyond few subgoals if repair is fast enough, in this particular case repair times r of 2 hours or less provide defense strategy that ensures survivability of a system, regardless of attacker's ability to decrease compromise time at the entry point.

5.3.5 Effect of arrival time

Figure 5.8 shows case when arrival time at place *1.1.2.1* was parameterized. Unlike other simulations, here compromise time c_{1121} was set to constant value of $c_{1121} = 10$, and arrival time a_{1121} varies from 1 to 10 hours. We kept repair time to $r = 8$ hours. This situation models attacker's strategy to increase intensity of attacks, in such way that attacks on place *1.1.2.1* occur more often. Simulation output nicely illustrates struggle between attacker and victim on the single resource: as soon as attacker gives enough time for repair (a_{1121} of 8 or more hours), victim is able to timely repair compromise before it reaches root goal.

CHAPTER 6. CONCLUSIONS AND FUTURE WORK

In this work, we have presented PENET, new attack modeling approach based on timed Petri nets. First objective of our modeling approach is to provide more precise quantitative parameterization and advanced modeling capabilities compared to attack trees, while keeping their intuitive features and simplistic approach. Second objective of PENET approach is to provide modeling tool that aids user in enhancing the system survivability.

The new modeling approach tries to find balance between complex stochastic models and static models such as attack trees. The weakness of many stochastic models is assumption that attack behavior follows exponential distribution. This assumption has no foundation in existing research. We have avoided such approach by using three simple timed delays to model attacker behavior. Attack trees' main weaknesses are subjective assignments of component's values, static model without notion of repairability, and limited modeling capabilities.

PENET model was defined using deterministic time transition Petri nets. Using quantitative parameters that model compromise, periodic arrival, and repair times we have provided means to create finer model of a vulnerable system. We have extended modeling capabilities by providing additional constructs, such as dependability and sequential constructs, while leaving open door for additional constructs that are possible due to modeling power of Petri nets. Additionally, we provided performance metrics to evaluate various aspects such as survivability of the system, total cost of attack scenario, and effectiveness of victim's repair efforts.

We have provided two analysis approaches for solving and simulating the created model. First analysis method produces attack scenarios and survivability indexes similarly to attack trees [4]. Second analysis method is a unique way to analyze survivability of a system in a time domain, employing novelty of our approach, such as new constructs and new delay times that quantitatively describe them. The output of this analysis tells us whether attacker will be able to reach root goal, at what time, what subgoals were accomplished, and how effective were repair efforts. We managed to provide dynamic sense what goes on during attack during its lifetime. The proposed approach follows Morda assessment process steps [10]. Finally, through case study, we have shown these concepts at work.

With concept of reccuring attack, we recognized new issue of *optimization of defense or attacker investment*. This problem can be formally stated as how to identify critical elements and utilize funds in recurring effort so the penetration is minimized or maximized. For other modeling techniques, where there is no notion of reoccurrence, this problem is simpler.

Finding *objective way to assign probability* of attack success remains significant issue for any attack modeling effort. PENET approach captures probability of attack success to three parameters: compromise, arrival, and repair rates. This approach is a step forward in resolving the issue of how to objectively, without subjectivity, assign probability of attack events. Some other authors [19] have decomposed this probability into set of attributes such as probabilities of being attacked, perimeter breach, being on attacker target list, etc. Related to this problem is problem of finding probability distribution of attacker behavior. Although Jonsson et al. [20] has provided significant effort, more research is necessary to obtain plausible results. Such results will open door for new modeling techniques that carry increased precision in comparison to existing ones.

Although cyber attacks are common, most modeling approaches do not *utilize real-world attack data*. Instead, researchers often use either simple cases to illustrate models, or they build generic models based on available information. Using real-world systems and attacks in developed models would be extremely beneficial to capture strengths and weaknesses of any attack modeling approach. Our approach would be more meaningful if it could be applied to an actual vulnerable system.

Another research effort that involves cyber security focuses on *interaction between physical and cyber worlds during attack*. In this area, researchers study consequences of cyber attacks on some critical infrastructures such as power plant system. Petri nets could prove useful in this scenario as a tool that is capable of modeling both physical and cyber worlds, and their interactions.

The proposed PENET attack modeling can be combined with attack trees to form a *hybrid model of attack trees and Petri nets*. Such model would appear as attack tree, with some constructs and events that cannot be modeled by attack trees represented with PENET model. The advantage of such model is it captures the sophistication of a PENET model and the simplicity of attack trees in an integrated manner.

BIBLIOGRAPHY

[1] Bruce Schneier. *Attack trees: Modeling security threats.* Dr. Dobb's journal, December 1999.

[2] Dalton, G.C., II; Mills, R.F.; Colombi, J.M.; Raines, R.A., *Analyzing Attack Trees using Generalized Stochastic Petri Nets,* 2006 IEEE Information Assurance Workshop , vol., no.pp. 116- 123, June 21-23

[3] Higuero, M.V.; Unzilla, J.J.; Jacob, E.; Saiz, P.; Aguado, M.; Luengo, D., *Application of 'attack trees' in security analysis of digital contents e-commerce protocols with copyright protection,* Security Technology, 2005. CCST '05. 39th Annual 2005 International Carnahan Conference on , vol., no.pp. 57- 60, 11-14 Oct. 2005

[4] Casey Fung; Yi-Liang Chen; Xinyu Wang; Lee, J.; Tarquini, R.; Anderson, M.; Linger, R., *Survivability analysis of distributed systems using attack tree methodology,* Military Communications Conference, 2005. MILCOM 2005. IEEE , vol., no.pp. 583- 589 Vol. 1, 17-20 Oct. 2005

[5] Bistarelli, S.; Fioravanti, F.; Peretti, P., *Defense trees for economic evaluation of security investments,* Availability, Reliability and Security, 2006. ARES 2006. The First International Conference on, vol., no.pp. 8 pp.-, 20-22 April 2006

[6] Sallhammar, K.; Helvik, B.E.; Knapskog, S.J., *Towards a stochastic model for integrated security and dependability evaluation,* Availability, Reliability and Security, 2006. ARES 2006. The First International Conference on, vol., no.pp. 8 pp.-, 20-22 April 2006

[7] Madan, B.B.; Gogeva-Popstojanova, K.; Vaidyanathan, K.; Trivedi, K.S., *Modeling and quantification of security attributes of software systems,* Dependable Systems and Networks, 2002. Proceedings. International Conference on, vol., no.pp. 505- 514, 2002

[8] Moore, A.P., R.J. Ellison, and R.C. Linger. *Attack modeling for information security and survivability.* Software Engineering Institute Technical Report CMU/SEI-2001.

[9] Wang, Jiacun. *Timed Petri Nets: Theory and Application.* Kluwer Acadamic Publishers, USA, 290 pages, October 1998.

[10] Shelby Evans, David Heinbuch, Elizabeth Kyule, John Piorkowski, James Wallner, *Risk-based Systems Security Engineering: Stopping Attacks with Intention,* IEEE Security and Privacy, vol. 02, no. 6, pp. 59-62, 2004.

[11] Dugan, J.B.; Bavuso, S.J.; Boyd, M.A., *Dynamic fault-tree models for fault-tolerant computer systems,* Reliability, IEEE Transactions on, vol.41, no.3pp.363-377, Sep 1992

[12] David M. Nicol, William H. Sanders, Kishor S. Trivedi, *Model-Based Evaluation: From Dependability to Security,* IEEE Transactions on Dependable and Secure Computing, vol. 01, no. 1, pp. 48-65, Jan-Mar, 2004

[13] J. Bloom, C. Clark, C. Clifford, A. Duncan, H. Khan, M. Papantoniou, T. Barnwell, M. Camacho, M. Cook, M. Gready, P. Kyme, and M. Tsouchlaris, *PIPE,* Imperial College DoC Group Project, 2004.

[14] McDermott, J. P. *Attack net penetration testing.* In Proceedings of the 2000 Workshop on New Security Paradigms (Ballycotton, County Cork, Ireland, September 18 - 21, 2000).

[15] Haining Wang; Danlu Zhang; Shin, K.G., *Change-point monitoring for the detection of DoS attacks,* Dependable and Secure Computing, IEEE Transactions on, vol.1, no.4pp. 193- 208, Oct.-Dec. 2004

[16] Mirkovic, J., Martin, J., and Reiher, P. *A Taxonomy of DDoS Attacks and DDoS Defense Mechanisms,* Los Angeles, CA, University of California Computer Science Department, 2001.

[17] M. Ajmone Marsan, G. Balbo, and G. Conte, *A class of generalized stochastic Petri nets for the performance analysis of multiprocessor systems,* ACM Trans. Computer Syst., vol. 2, no. 1, May 1984.

[18] Tridandapani, S.; Somani, A.K.; Sandadi, U.R., *Low overhead multiprocessor allocation strategies exploiting system spare capacity for fault detection and location,* Computers, IEEE Transactions on, vol.44, no.7pp.865-877, Jul 1995

[19] McQueen, M.A.; Boyer, W.F.; Flynn, M.A.; Beitel, G.A., *Quantitative Cyber Risk Reduction Estimation Methodology for a Small SCADA Control System,* System Sciences, 2006. HICSS '06. Proceedings of the 39th Annual Hawaii International Conference on, Vol.9, Iss., 04-07 Jan. 2006 Pages: 226- 226

[20] Jonsson, E.; Olovsson, T. *A quantitative model of the security intrusion process based on attacker behavior,* Software Engineering, IEEE Transactions on, Vol.23, Iss.4, Apr 1997 Pages:235-245

[21] Microsoft Corporation, *.NET Framework Development Center.* http://msdn2.microsoft.com/en-us/netframework/default.aspx. 2007

[22] World Wide Web Consortium, *Extensible Markup Language (XML).* http://www.w3.org/XML/. 2003

[23] World Wide Web Consortium, *XML Schema.* http://www.w3.org/XML/Schema. 2007

[24] Microsoft Corporation, *Introduction to XML Schemas (XML Designer.* http://msdn2.microsoft.com/en-us/library/efc70bx3(VS.80).aspx. 2007

[25] Microsoft Corporation, *About GDI+.* MSDN Documentation. 2005

ACKNOWLEDGMENTS

I would like to take this opportunity to express my deepest thanks to professor Dr. Manimaran Govindarasu for many years of his dedicated involvement, guidance and support in my research and professional development. Without his ideas, suggestions, and feedback this work would not see light of the day. Also, many thanks to professors Daji Qiao, Hridesh Rajan, and Check-Ching Liu that were involved in many ways in the reseach that produced this thesis.

Additionally, I would like to express warm thanks to my parents Spasoje and Velinka Pudar, and friend Mr. Pavle Ubiparipovic for their support and belief in me.

Finally, I would like to express thanks to fellow graduate students Kavitha Balasubramanian, Sudha Anil Gathala, Mohammad Fraiwan, and Chee-Wooi Ten for their valuable contributions and suggestions through my research endeavors.