Using Coloured Petri Nets in Penetration Testing

Ole Martin Dahl

Abstract

Network penetration testing is a well-known approach used for security testing. Penetrating testing can be a laborious task which relies much on human knowledge and expertise, with various techniques employed, and an extensive amount of tools used in the process. A methodical approach to penetration testing is therefore recommended. The flaw hypothesis methodology, used in this thesis, represent one of the most used models for penetration testing and have great similarities in other penetration testing methodologies and standards used today.

Petri nets represent a graph based mathematically sound modelling technique for concurrent systems, and provide a graphically intuitive approach for modelling, simulation and execution. A coloured Petri net is a high level Petri net that provides a significant increase in the expressiveness and compactness of Petri net models.

The flaw hypothesis methodology used together with coloured Petri net attack models is presented in the thesis. The use of coloured Petri nets is described and analysed through case studies elucidating several properties of Petri net variants and their suitability to modelling attacks in penetration testing. Advantages of modelling attacks with coloured Petri nets have been explored and are described. Coloured Petri nets have been found to have many usefull mechanisms for modelling, analysing, and automatically executing penetration attempts, e.g. through their ability to model states, transitions, concurrency, and timing.

The overall topic of the thesis is the technical aspects of penetration testing, what it is, and a methodical approach to it.

Sammendrag

Nettverkspenetrasjonstesting er en velkjent metode brukt innen sikkerhetstesting. Penetrasjonstesting kan være en omfattende oppgave som krever mye kunnskap og ekspertise, med mange forskjellige teknikker, og et vidt antall verktøy i bruk under prosessen. En metodisk tilnærming til penetrasjonstesting er derfor anbefalt. "The flaw hypothesis methodology", som denne oppgaven baserer seg på, representerer en av de mest brukte modellene for penetrasjonstesting. Det finnes likheter til denne modellen i de fleste metoder og standarder innen penetrasjonstesting brukt i dag.

Petri net er et graf basert modeleringsspråk, opprinnelig tiltenkt for modellering av systemer med samtidighetsproblematikk. Grafene gir en intuitiv måte å modellere, simulere og eksekvere modeller. "Coloured Petri net" er en type høynivå Petri net som gir flere muligheter til detaljmodellering og sammensetting av Petri net.

The flaw hypothesis methodology er brukt sammen med forskjellige typer coloured Petri net for modellering av systemangrep i penetrasjonstestingsprosessen. Metoden er beskrevet og analysert gjennom tre testscenarioer, som belyser egenskapene ved Petri net modellering av angrep i penetrasjonstesting. Fordeler ved Petri net angrepsmodelleringen har blitt utforsket og beskrevet. Mange av egenskapene til coloured Petri nets har vist seg å fungere bra til angrepsmodellering under penetrasjonstesting, for eksempel egenskapene grafene har til å modelere tilstander, transisjoner, samtidighet, og tidsbergning.

Preface

This thesis started as an awoken interest in the topic of penetration testing in 2004. The motivation was to learn more about the vast area of penetration testing and its usage in network security. In late 2004, the actual work started. Early, I realised that penetration testing was large and that much detailed research has been done in the area. During the literature review an article by John McDermott, about Petri net attack modelling, caught my interest and with eminent guidance from Dr Stephen D. Wolthusen, coloured Petri nets became my way of further refining and using Petri nets in penetration testing.

The thesis work has been very exciting and given me many challenges. I feel that the thesis turned out well after about six months of work, with moments of slight frustration. My enitre graduate education in computer security, including this final master thesis, has been a valuable personal experience for me and has extended my knowledge horizon.

This thesis work would not be possible without the help and support from many individuals. I am especially grateful to Dr. Stephen D. Wolthusen who has provided guidance, prominent expertise, valuable support and encouragement during the thesis work. I would also like to thank my supervisor Dr. Erik Hjelmås for valuable guidance, shared research experience, and encouragement while doing this thesis work and pursuing my graduate education in information security.

My thanks also go to the other individuals at Gjøvik University College that have contributed, for their help with my thesis work. Special thanks go to my fellow students and mates, Ole Kasper Olsen, Fredrik Skarderud, Torkjel Søndrol and Anders Wiehe for their motivation, opinions, and ideas. I also want to thank my family and friends for the patience and understanding they have shown for my work.

Last but not least, my thanks go to my girlfriend Sonja for her love, support, and understanding during the whole period that went into pursuing my graduate studies.

Ole Martin Dahl, 30th June 2005

Contents

List of Figures

1   Introduction

Penetration testing is a fundamental area of information system security. Two of the earliest published open references to penetration testing are a penetration testing methodology presented by Linde [42] in 1975, which is one of the seminal papers behind the flaw hypothesis methodology, and a vulnerability analysis report of the Multics[1] operating system from 1974 by Karger and Schell [33]. Multics is notable for its early emphasis on computer security by design, it was designed as a secure system from the ground up. In spite of this, early versions of Multics were broken into, not once, but repeatedly. The vulnerability analysis found major flaws that had not been found during the thorough system design, and major lessons were learned. To this day information systems are tested with similar methods and many of the flaws found then are the same today [34].

1.1   Topics Covered by This Thesis

Penetration testing is a security practice during which some trusted party attempts to detect and exploit weaknesses in an information system. Penetration testing traditionally tries to break down or break into a system using different hacking techniques, and thereby uncovering weaknesses the system.

Penetration testing has evolved into many different areas of security, from application testing [2, 69] in different system development phases to network security testing [39] of in-production systems. This thesis will focus on network penetration testing, although many of the ideas presented will be adaptable to other types of penetration testing.

The thesis uses the flaw hypothesis methodology [42, 76, 77, 78] as a framework for penetration testing and introduce an attack model using coloured Petri nets [31, 30, 32]. The flaw hypothesis methodology is a systematic and thorough approach to penetration testing. Coloured Petri nets are a formal and graphically appealing model language for design, specification, simulation and verification of systems.

The thesis focus on the technical aspects of penetration testing; what it is all about, a methodical approach to it, and a proposition to use coloured Petri nets to model penetration attempts.

Keywords: Technology, information security, network security, penetration testing/ethical hacking[2], the flaw hypothesis methodology, attack modelling, coloured Petri nets.

1.2   Justification, Motivation and Benefits

Keeping information systems in top condition to withstand attacks from adversaries have become vital to most organisations. In a perfect world, it would not be necessary to

---

[1]Modern operating systems, in particular the UNIX system, are in part descended from Multics
[2]An ethical hacker [64, 79] is often used synonymous to a penetration tester

conduct penetration testing when an information system is securely configured and set up from scratch. However, it is most often necessary to confirm security mechanisms, and to test whether the systems really are secure from the adversary's point of view. Under design, implementation, maintenance and operation of today's complex information systems, human and system errors affecting information security will occur. Even an immune system, if that can exist, does not provide absolute protection in the face of constantly evolving adversaries [54]. Using penetration testing as a method to confirm that the defensive measures are complete and working is important, both under development and in production. The goal is to locate vulnerabilities that make it possible for malicious adversaries to exploit the system, preferably finding them before an adversary do.

Finding vulnerabilities and their exploitation can be complex. Several attacks require concurrent execution, multiple cooperating agents, combinations of system flaws and timing by attackers to be successful. To become successfully many different methods and a lot of different tools can be used to help with the process. When more complex attack scenarios are hypothesised, error-prone manual testing techniques may be too resource consuming for the penetration testers.

Research results on how to use penetration testing recourses effectively, including when different attack scenarios appear in large-scale network environments such as multi-tier systems, e.g. electronic commerce systems, are the main motivation for this thesis work with the introduction of coloured Petri net attack modelling.

The stakeholders are security staff, organisations that hire security testing services and everyone in the growing security testing business.

1.3 Problem Description

There are a lot of different techniques, methods and tools for doing penetration testing. The flaw hypothesis methodology is a commonly used methodology for doing penetration testing. Most methods today build on such a model or methodology. However the methodology itself only provides limited high level guidance for the derivation of penetration attempts. Refining the flaw hypothesis methodology with the use of coloured Petri net attack models should be possible. By doing this we can exert the coloured Petri nets ability of graphically modelling concurrency and timing, and the possibility of executing and analysing hypothesised attack scenarios. This can make coloured Petri nets a helpful tool in precise modelling of penetration attempts, especially when complex attack scenarios are hypothesised.

1.4 Research Questions

In order to find out whether penetration testing can benefit from attack modelling with coloured Petri nets in a methodical approach like the flaw hypothesis methodology, the following questions has been explored:

1. Do penetration testing need the possibility of modelling custom attack scenarios, or do the combination of a high-level methodology and modern testing tools solve the need for custom penetration attempts satisfactory?

2. How can we use coloured Petri nets to model penetration testing attacks?

3. How can we use coloured Petri net attack modelling inside the flaw hypothesis methodology?

4. What can we gain from doing penetration testing with the use of coloured Petri nets for attack modelling?

1.5 Summary of Claimed Contributions

The contributions will mainly be:

- A methodical penetration testing approach with basis in the flaw hypothesis methodology with a mechanism for the modelling, partial analysis, and automatic execution of attacks, based on different types of coloured Petri nets.

- A study of coloured Petri nets, including the variants; timed coloured Petri nets and interval timed coloured Petri nets, for modelling penetration testing attacks.

- Three case studies that elucidate the properties of the methodology and different Petri net variants and their suitability to model different attacks.

1.6 Method

The method used is mainly a literature study and the development of a methodical approach to penetration testing. The methodical approach presented has evolved from previous work in the areas of penetration testing, attack modelling, and Petri net theory. The approach is applied to three different hypothetical test scenarios to verify theory, and to find limitations and benefits. Note that the scenarios do not represent a qualitative experiment, there have not been done any live penetration tests in the amount required to get enough statistical data for this. Conclusions and results are both based in the literature study and the test scenarios.

The study started with a wide and open area to research. Both the area of penetration testing and coloured Petri net theory are vast, therefore most of the time spent on the thesis has been spend reading relevant research and literature in the area. The search was gradually narrowed down as the author gained better understanding of the subject matter. This method approach has lead to wide understanding of penetration testing and its possibilities.

During the thesis work a paper [12] has been submitted for publication, co-written with Dr. Stephen D. Wolthusen, that cover some of the topics in the thesis (available in appendix B).

1.7 Terminology

The terms "attack" and "penetration" are used interchangeably throughout the thesis, this also applies to "penetration tester" and "attacker". When we refere to an attack or penetration done by an malicious adversary the term maliciuos is used. An attack or a penetration exploit a security flaw or vulnerability in a system, which again may lead to a system "breach", "compromise", or "intrusion" if sucessfull. The distinction between breach, intrusion, and compromise is neither strict nor crucially important for the dicussions in this thesis.

1.8 Outline of Chapters

Chapter 2 provide an introduction to different Petri nets and their background theory, which is used later in the thesis. In Chapter 3 the topic of penetration testing and its usage is explained. This Chapter include the theory of the flaw hypothesis methodology.

Chapter 4 present the most relevant related work to the thesis research.

Chapter 5 present the proposed methodical approach to penetration testing with coloured Petri nets for attack modelling. The testing stages of the flaw hypothesis methodology in combination with the coloured Petri net attack models are explained. Lastly the main benefits of the approach are summarised. Then, in Chapter 6, three penetration case studies are presented, where it is shown how the approach can be used in different penetration testing scenarios. Each scenario end with a small discussion about conclusions and observations found in the scenario.

Chapter 7 summarise the conclusions of the study and Chapter 8 suggests some future work that could be done.

Appendix A explain the different declarations and net inscriptions used in the designed coloured Petri nets througout the thesis. Appendix B include a paper [12] submitted for publication during the thesis work.

2   Petri Net Theory

This Chapter will present the necessary background theory of coloured Petri nets, which later in the thesis will be used for attack modelling in penetration testing.

2.1   Basic Graph Theory

We begin with the formal definition of a graph [59]:

> A graph $G = (V, E, \phi)$ consists of a nonempty set V called the set of *nodes* of the graph, a set E called the set of *edges* of the graph, and a mapping ϕ from a set of edges E to a set of pairs of elements of V.

An edge in a graph can be *directed*, that is a pair of nodes is ordered (annotated with an arrow). If all edges in a graph are directed, then the graph itself is a *directed graph* or *digraph*. A multigraph is a graph with multiple edges, i.e. edges that have the same end nodes. *Bipartite graph* means that the graph has two types of nodes. A property of a bipartite graph is that an edge can only connect two nodes that belong to different types.

2.2   Petri Nets

A Petri net [55, 59] also called a place/transition net [31], is a generalised graph introduced by Carl Adam Petri in 1962 [56]. The motivation behind Petri nets was the need to address problems of concurrency in systems.

A Petri net is a bipartite dircted multigraph, where the two types of nodes are places and transitions. In graphical representations, places are shown as circles or ellipses, and transitions as bars or rectangles. Figure 1 show the notations.
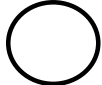
Place	◯ ⬭
Transition	── ▭
Token	⬤

Figure 1: Basics of Petri nets

The edges of a Petri net are called *arcs* and are always directed. Arcs are subdivided into *input* arcs, which connect places with transitions and *output* arcs, which start at a transition and end at a place. Places are also often referred to as input and output places when a specific transition is discussed.

A marking (state) of a Petri net is shown by tokens. Pictorially, a number of black dots (tokens) are placed in a place to mark states. Figure 2 shows a simple Petri net with two places connected by a single transition and with three tokens in one of the places. Types of Petri nets can be distinguished by the level of information associated with individual tokens which can range from simple Boolean information to structured tokens.
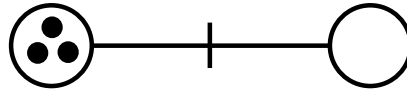
Figure 2: Petri net example

In order to simulate the dynamic behaviour of a system, a state or marking in a Petri net is changed according to transition rules. Firstly the transition is said to be enabled if each input place of a transition is marked by at least one token. Secondly an enabled transition may or may not fire depending on whether or not the event, decided by the transition, actually took place. Lastly a firing of an enabled transition removes a number of tokens from each input place of the enabled transition and adds a number of tokens to each output place of the transition.

The formal definition of a Petri net graph [55] is as follows:

A *Petri net graph* G is a bipartite directed multigraph, $G = (V, A)$, where $V = v_1, v_2, v_3, ..., v_n$ is a set of vertices and $A = a_1, a_2, a_3, ..., a_n$ is a multiset/bag[1] of directed arcs, $a_i = (v_j, v_k)$, with $v_j, v_k \in V$. The set V can be partitioned into two disjoint sets P and T such that $V = P \cup T$, $P \cap T = \varnothing$, and for each directed arc, $a_i \in A$, if $a_i = (v_j, v_k)$, then either $v_j \in P$ and $v_k \in T$ or $v_j \in T$ and $v_k \in P$.

Usage of Petri nets are widespread and diverse [41, 52, 55, 59], e.g. in network communication technology and in other systems requiring concurrent processes. Murata's paper [52] give a comprehensive survey of Petri nets. The web page "Petri nets world"[2] located at the University of Hamburg, Germany, provide a variety of online services for the international Petri Nets community and has a collection of research on the topic.

For a more complete formal explanation of Petri nets see [55, 59]

2.3 Coloured Petri Nets

Coloured Petri nets [31, 30, 32, 36] is a modelling language aimed at systems in which concurrency, communication, synchronisation and resource sharing play an important role. As apposed to simple place/transition nets do coloured Petri nets provide structured tokens in the form of so called colours and provide significant increase in the expressiveness and compactness of models. Coloured Petri nets also have an intuitive graphical representation which makes it easy to see the basic structure of the net and to understand how individual processes interact with each other. The nets allow the modeller to make much more concise and manageable descriptions than with ordinary Petri nets, thus coloured Petri nets are called high-level nets.

---

[1]A multi-set or bag, like a set, is a collection of elements over some domains. However, unlike a set, bags allow multiple occurrences of elements. In set theory, an element is either a member of a set or not a member of a set. In bag theory, an element may be in a bag zero times (not in the bag), one time or several numbers of times.

[2]http://www.informatik.uni-hamburg.de/TGI/PetriNets/ (Visited May 2005)

Each place in a coloured Petri net is associated with a *colour set* (a type), which determines the kind of data the place may contain.

A state of a coloured Petri net is called a *marking*. A marking, describes how coloured tokens are distributed among the places in a net at a specific point of net execution. A token has or even is a colour (a value) from a colour set, and a token can only be present on a place if the colour is from the defined colour set in the place. A place may have several tokens with the same colour. A colour set can be quite complex, for example union colour sets make it possible for a place to contain more than one kind of tokens. Figure 3 shows a simple coloured Petri net. The marking of a place is written at the upper right of the place. The black dot notation for tokens is seldom used in coloured Petri nets, instead the marking is shown in the upper right of places by a number, i.e. the number of a specific colour, followed by the colour itself.

The actions in a coloured Petri net are, like ordinary Petri nets (Section 2.2), represented by transitions. Transitions and places are connected by arcs. A transition can fire when it is enabled. A transition T is enabled if and only if there is at least one token in each input place of T. When firing a transition, tokens are removed from each of the input places of T and tokens are generated in each of the output places of T, i.e. a transition occurs. This means that there must not be any closer relationship between the removed token in one place and the added token at another place. So saying that tokens move from place to place are not entirely correct according to Petri net formalism, however for our intuitive interpretation of the model we sometimes think of the token as being moved from place to place [31]. The numbers of tokens added or removed by the enabled transition are determined by *arc expressions*. The arc expressions contain variables, and transitions are enabled in bindings of variables.

Figure 3 show a simple example coloured Petri net. The net diverse net inscriptions are explained in Appendix A. The net in Figure 3 contain one transition, `Login`, which is surrounded by tree places; `Username`, `Password` and `Access`. The arc expressions around the transition `Login` in Figure 3 contain the variables `username` and `password`, ranging over the colour set `STRING`. Thus the possible bindings of the transition Login are `username` $\rightarrow$ `Administrator`, `password` $\rightarrow$ `AdminPassword` and `password` $\rightarrow$ `WrongPassword` ($\rightarrow$ is read "bound to").

Transitions in coloured Petri nets may also have *guards*. The guard is a Boolean expression[3] or a list of Boolean expressions. It may have variables in exactly the same way that the arc expressions have. A guard is written at the upper left corner of the transition. In Figure 3 a guard at the transition `Login` is only enabled if the tokens from the places `Username` and `Password` has the colour `Administrator` and `AdminPassword`. The purpose of the guard is to define an additional constraint to the coloured Petri which must be fulfilled before the transition is enabled.

Now the formal definition of a coloured Petri net [31]:

A non-hierarchical[4] coloured Petri net is a tuple CPN = $(\Sigma, P, T, A, N, C, G, E, I)$ satisfying the requirements below:

- Σ is a finite set of non-empty types, called *colour sets*.

---

[3]An expression which evaluates to either true or false

[4]Hierarchical coloured Petri nets are not covered by this thesis. Hierarchical coloured Petri nets that make it possible to construct very large nets that combine smaller nets, see [31, 30].
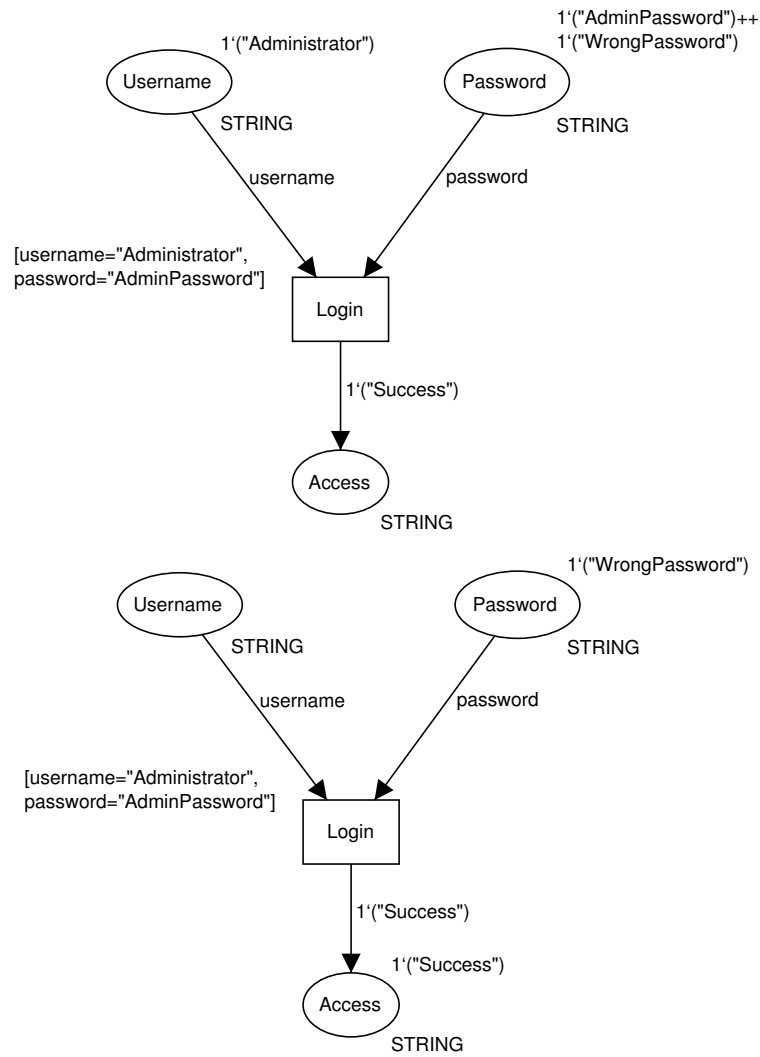
Figure 3: Coloured Petri net example. Before and after execution.

- P is a finite set of *places*.

- T is a finite set of *transitions*.

- A is a finite set of arcs such that:
  $$P \cap T = P \cap A = T \cap A = \varnothing$$

- N is a *node* function. It is defined from A into $P \times T \cup T \times P$.

- C is a *colour* function. It is defined from P into Σ.

- G is a *guard* function. It is defined from T into expressions such that:
  $$\forall t \in T : [Type(G(t)) = \mathbb{B} \wedge Type(Var(G(t))) \subseteq \Sigma]$$

- E is a *arc expression* function. It is defined from A into expressions such that:
  $$\forall a \in A : [Type(E(a)) = C(p(a))_{multi-set} \wedge Type(Var(E(a))) \subseteq \Sigma]$$ where $p(a)$ is the place of $N(a)$.

8

- I is an *initialisation* function. It is defined from P into closed expressions such that:
  $$\forall p \in P : [\mathsf{Type}(I(p)) = C(p)_{\mathrm{multi-set}}].$$

For a more complete formal explanation of coloured Petri nets see [31].

Coloured Petri nets are used in many areas [32], mainly in systems in which communication, synchronisation and resource sharing play an important role. Because coloured Petri nets make use of all the strengths of ordinary Petri nets it can be used as a substitute that enables much more detail in the modelling. Some examples of use are in modelling are network protocols, distributed databases, electronic chip design and more. A vast collection of other usage of coloured Petri nets can be found at the web pages of the CPN group at the University of Aarhus, Denmark[5].

2.3.1 Timed Coloured Petri Nets

Timed coloured Petri nets is an extension to coloured Petri nets, by Jensen [30], that deals with time and timing. It has a timing mechanism where time is associated with tokens, i.e. time stamps.

The formal definition of timed coloured Petri nets [30]:

A *timed* non-hierarchical coloured Petri net is a tuple $\mathsf{TCPN} = (\mathsf{CPN}, R, r_0)$ such that:

- Coloured Petri net satisfied the requirements of a non-hierarchical coloured Petri net as defined in Section 2.3 – when in arc expression function and the initialisation function we allow the type of $E(a)$ and $I(p)$ to be a timed or an un-timed multi-set over $C(p(a))$ and $C(p)$, respectively.

- R is a set of *time values*, also called *time stamps*. It is a subset of $\mathbb{R}$ closed under $+$ and containing 0.

- r_0 is an element of R, called the *start time*.

Timed coloured Petri nets introduce a *global clock*. The clock represents the *model time*, also called time stamp. The time stamp describes the earliest model time at which the token can be consumed, that is be removed by an occurring transition. The time stamps are shown as a time list associated with each colour of a colour set. The execution of a timed coloured Petri net is driven by time. The system remains at one time until no more transitions are enabled at the current model time. We can say that a state of a timed coloured Petri net consists of a marking, and the model time (the global clock). Timed coloured Petri nets can also contain colour sets which are un-timed, thus a net can contain both timed and un-timed colour sets and the timing concept can be added to already modelled systems in coloured Petri nets.

The time can be manipulated by transitions and/or arc expressions under execution. Figure 4 show an example timed coloured Petri net before and after firing of the transition `Login`, see Appendix A for a explanation of the net inscriptions. The time is manipulated by the transaction `Login`. The bottom part shows the login net after firing of the transition, as we can see the token success has a time stamp of 10 time units. The global clock will then also be 10, given that it started at 0. Arc expression in output arcs can

---

[5]`http://www.daimi.au.dk/CPnets/` (Visited May 2005)
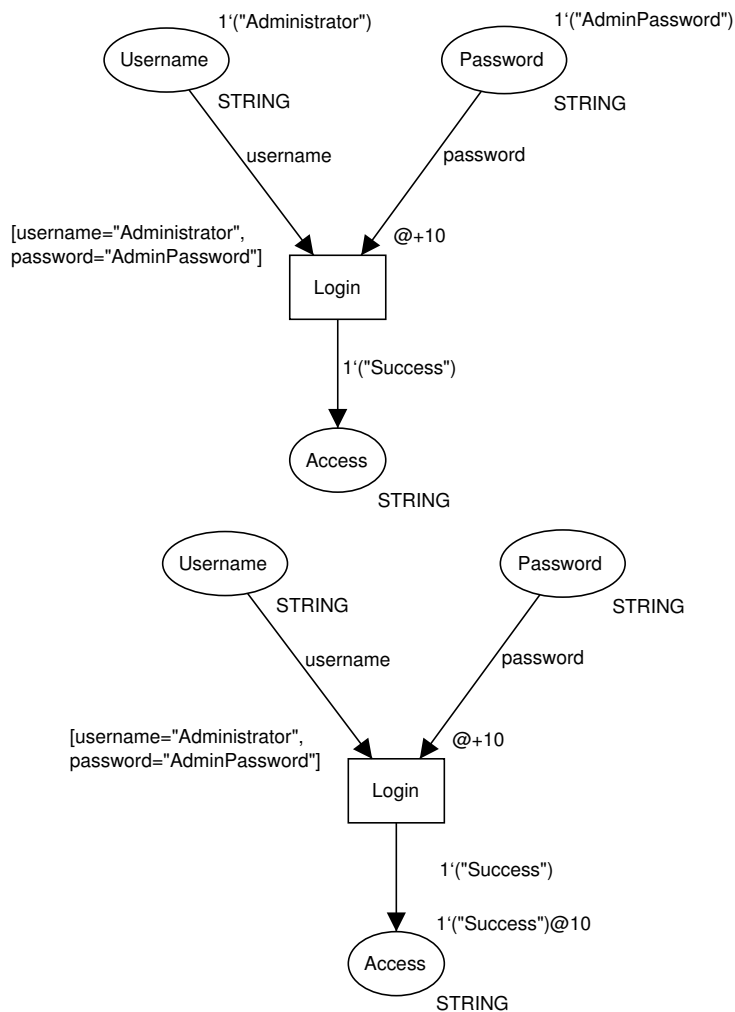
Figure 4: Timed coloured Petri net example. Before and after execution.

also manipulate with the token time stamps. For a more complete formal explanation of timed coloured Petri nets see [30].

Usage of timed coloured Petri nets are seen for example in Mortensen *et al.* research on capacity planning of web servers using timed coloured Petri nets [51]. With the use of timed coloured Petri nets performance and capability optimalisation can be modelled for a HTTP web server. A vast collection of other usage of timed coloured Petri nets can be found at the web pages of the CPN group[6].

An important observation to make about timed coloured Petri nets is that all time stamps are point-values. Therefore it is not easy to take uncertainty or fuzziness into account, e.g. if a transaction does not always consume the same amount of time. Interval timed coloured Petri nets can take this into account, if necessary, in the model.

---

[6]`http://www.daimi.au.dk/CPnets/intro/example_indu.html` (Visited May 2005)

2.3.2 Interval Timed Coloured Petri Nets

In [72] van der Aalst introduces interval timed coloured Petri nets. This is an alternative formalism to timed coloured Petri nets which represent transition time delays as *closed intervals*. Like with the timed coloured Petri nets presented above, the tokens in interval timed coloured Petri nets are time stamped and coloured. Arc expressions and guards function in same way as with coloured Petri nets.

One of the major differences between timed coloured Petri nets and interval timed coloured Petri nets lies with the transition delays. In interval timed coloured Petri nets the transitions are represented by closed intervals. Transition output arcs can be assigned with a time delay interval. The intervals provide a mechanism to model uncertainty and nondeterminism in individual transitions.

The enabling time of an event or step is the maximum of all timestamps of the tokens to be consumed. The model time is defined as the minimum of all enabling times. Thus the model time advances only when an event happens. Transitions are eager to fire, meaning they will fire as soon as possible. Thus the transition with smallest enabling time will fire first, just as with timed coloured Petri nets.

Once a step occurs the new tokens are put into the output places. Each of the tokens will have a timestamp calculated by adding the delay value which is required to fall into the delay interval defined at the output arc. Figure 5 show an interval timed coloured Petri net before and after firing of the Login transition, see Appendix A for a explanation of the net inscriptions. The closed interval on the output arc is $[8, 12]$. The actual delay after the transition Login is 9.4, this could of course be any time value between 8 and 12.

The definition of the interval is as follows [72]:

> TS is the the **time set**, $TS = \{x \in \mathbb{R}|x \geq 0\}$, i.e. the set of all non-negative reals.
> $INT = \{[y, z] \in TS \times TS|y \leq z\}$, represent the set of all closed intervals. If $x \in TS$ and $[y, z] \in INT$, then $x \in [y, z]$ if and only if $y \leq x \leq z$.

Finally the formal definition of a interval timed coloured Petri net is [72]:

> A *interval* timed non-hierarchical coloured Petri net is a five tuple $ITCPN = (\Sigma, P, T, C, F)$ satisfying the following requirements:
>
> - Σ is a finite set of types, called colour sets.
>
> - P is a finite set of places.
>
> - T is a finite set of transitions.
>
> - C is a colour function. It is defined from P into Σ, i.e. $C \in P \rightarrow \Sigma$.
>
> - $CT = \{\langle p, v \rangle | p \in P \wedge v \in C(p)\}$ is the set of all possible coloured tokens.
>
> - F is the transition function. It is defined from T into functions. If $t \in T$, then:[7]
> $$F(t) \in CT_{multi-set} \nrightarrow (CT \times INT)_{multi-set}$$

For a more complete formal explanation of interval timed coloured Petri nets see [72]. Usage of interval-timed coloured Petri nets are often found in modelling of real-

---

[7] $A \nrightarrow B$ denotes the set of all partial functions from A to B.
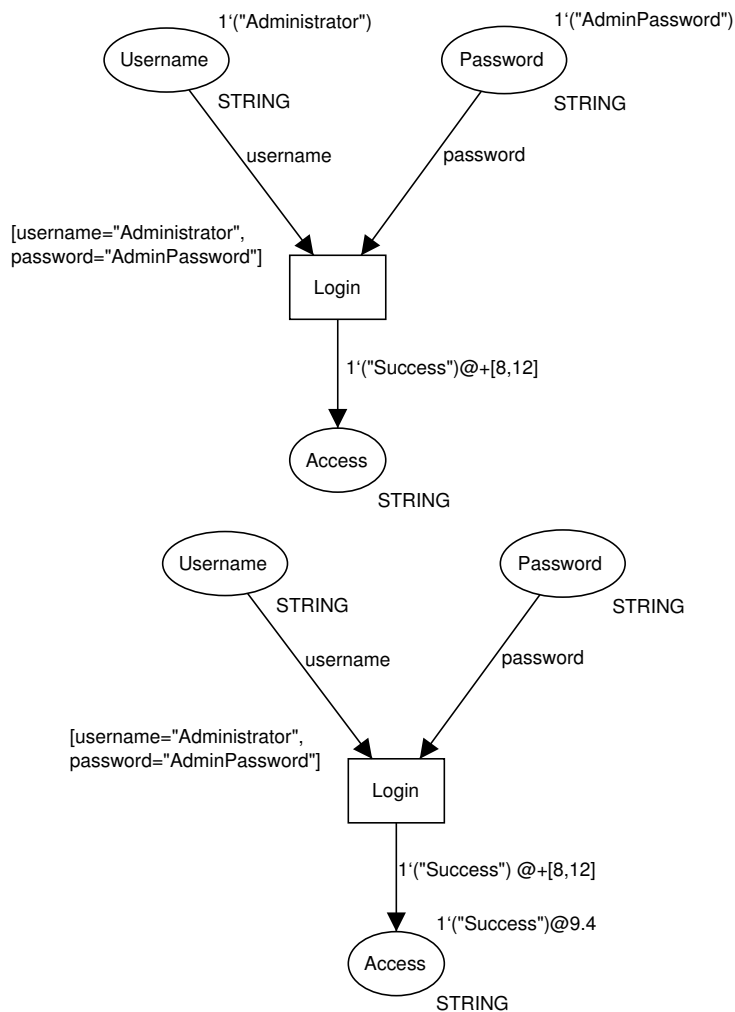
Figure 5: Interval timed coloured Petri net example. Before and after execution.

time systems. van der Aalst and Odijk use interval timed coloured Petri nets in [73] for an analysis of railway stations and their operating schedules. This is an example where the modelled system requires some fuzziness, which are the trains will not always operate on schedule.

A time interval Petri net, very similar to interval timed coloured Petri nets, is used by Bulitko and Wilkins in [9] for real time decision making for shipboard damage control. Here they simulate how fire spreads through a ship at the same time as decisions of compartment flooding etc. is done by humans and automated systems.

The main benefit this thesis will utilise with interval timed- as opposed to timed-coloured Petri nets are that they allow more fuzziness in timing, e.g. in distributed systems where it is difficult to have synchronised clocks.

3   Penetration Testing

Penetration testing can not prove security to a system [14, 77] or as Geer and Harthorne puts it [24]:

> Because the list of potential insecurities is unknowable and hence innumerable, no penetration tester can prove security, just as no doctor can prove that you are without occult disease.

Penetration testing can only provide a limited bird-eye perspective of current system security [8]. The effectiveness of penetration testing depends largely on the skill and experience of the testers. To get good valid results from a penetration test, the tester and his/hers methods and knowledge must be at its best. In spite of these requirements, penetration testers that follow a methodology can become more effective in their use of resources and provide more valid results [24].

Many other engineering disciplines also rely on failure data to improve their designs, which is what penetration testing can be compared to. Just imagine what would happen if engineering and design did not learn from disasters like space travels that goes wrong, bridges that collapse, oil platform disasters and so on. Information system engineering also utilise this way of improving security in systems [49] through penetration testing and similar vulnerability testing methods. Penetration testing is about learning from security attacks and help stopping them and similar ones from happening in information systems.

3.1   Current Penetration Testing Methodologies

Using a methodology is beneficial when doing penetration testing. Penetration testing depend on many ad-hoc mechanisms for identifying flaws in attacks or tests, a structured approach can therefore benefit both the testers and recources used under the test. An additional benefit is that test results with good structure are easier to re-use in the future to ensure that no regressions occur [68]. Throughout the literature written about penetration testing, most methodologies are similar [1, 17, 18, 24, 27, 38, 46, 47]. Each step in the methodologies have different names, but each with similar meanings. Some of the most used methodologies for penetration testing are now described.

Most approaches are similar to how an attacker would attack a system. A classical example of such approaches is used in Farmer and Venemas seminal paper "Improving security of your site by breaking into it" [19]. SANS Institutes[1] guidelines for developing penetration "rules of behaviour" [1] propose a similar penetration testing methodology:

1. Discovery

---

[1] http://www.sans.org/ (Visited June 2005)

2. Enumeration

3. Vulnerability mapping

4. Exploitation

This is the typical method presented for penetration testing in the current security literature [38]. Books like [35, 44] base penetration testing on this kind of methodology, with only small differences. Differences like dividing step 4 into gain access and privileged escalation or/and diving step 1 into initial reconnaissance and service determination are used, e.g. in [46, 47]. Two different penetration testing standards are now presented.

3.1.1 OSSTMM
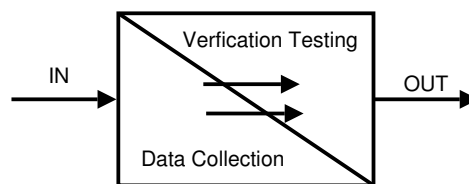
Figure 6: OSSTMM overall penetration testing methodology [27]

The Open-Source Security Testing Methodology Manual (OSSTMM)[27] cover how to do security testing. The manual is not just specified on penetration testing but also other aspects of security testing like risk assessment, security policy review, physical security and more. The manual introduce a security testing methodology called Risk Assessment Values (RAV) where key dimensions like time and frequency are included so that a test not only become a "security snapshot" that become invalid when new vulnerabilities are discovered or when configurations change. OSSTMM is a popular methodology used for security testing. The overall methodology is presented in Figure 6.

The methodology is not highly specific or strict. The OSSTMM have different modules that utilise the same overall methodology. A test begins with an input (e.g. a host address), and then collects data about the input, apply testing and at last gets an output that may become a new input to test further.

3.1.2 NIST SP 800-42
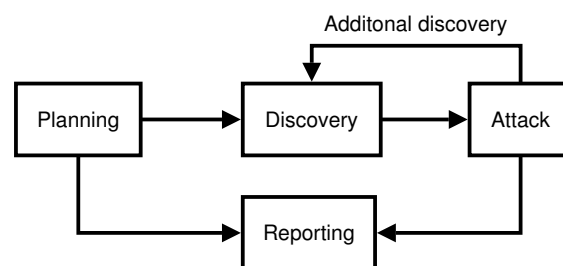
Figure 7: NIST SP 800-42 overall penetration testing methodology [17]

National Institute of Standards and Technology (NIST) have a publication called Guideline on Network Security Testing [17], which like OSSTMM is a methodology for security testing. It does not just cover penetration testing but also how to make security

testing a routine and a part of network and system operations. Chapter 3 of SP 800-42 is specific about penetration testing. The penetration testing methodology is presented in Figure 7.

The NIST SP 800-42 uses a four phase methodology with planning-, discovery-, attack- and a reporting phase. The attack phase is done iteratively with the discovery phase since an attack can reveal more vulnerabilities in the system, e.g. privileged escalation possibilities. The reporting phase is done simultaneously with the other three phases during the penetration test.

3.2 Tools and Exploit Code

To help the penetration testing process, tools and exploit code are necessary. Completely manual testing in modern complex information systems are not possible within the recourses that normally is used for penetration testing.

Books are written about tools and methods used in penetration testing, e.g. [35, 44]. They explain the usage of many tools and propose toolkits for the penetration tester. Pitfalls of some tool categories are also discussed. Examples are denial of service tools that may have undesirable effects on the target and other tools that have bugs which may unintentionally harm the target.

Different tools are widely available, and the Internet is the main source for penetration testing tools and exploits codes. Open source security testing software, commercial tools and information on hacker sites are available, e.g. at packet storm[2], K-OTik[3], malware.com[4], @stake[5] and more.

Exploit code are also often build for proof-of-concept in reported vulnerabilities in vulnerability databases. Mitre's CVE is a common namespace for all vulnerabilities and exploits, where vulnerabilities can be searched and found at their web site[6]. The NIST I-cat project also provide database interface over the web that allows users to browse and search for vulnerabilities by platform, intent and more with the CVE format [7]. Bugtraq[8] is another vulnerability database that provides descriptions of known vulnerabilities in information systems. A problem with vulnerability databases alone as an information source in penetration testing is that detailed information about the hows and whys of vulnerabilities and their exploitations are often lacking in such databases [65].

Not only independent tools and exploit code are used for penetration testing. There are also software suites that include several tools and exploit code. Fully automated tools also exist, like an on-line tool that SPI Dynamics[9] has build. This tool called WebInspect is specifically designed for penetration testing Web-based applications. In [23], Garfinkel discuss this tool and similar tools and their pros and cons. Garfinkel shows that such tools must be used with caution and the validity of the results are not always very good.

---

[2]http://www.packetstormsecurity.org (Visited May 2005)

[3]http://www.frsirt.com/ (Visited May 2005)

[4]http://www.malware.com/ (Visited May 2005)

[5]http://www.atstake.com (Visited May 2005)

[6]Common Vulnerabilities and Exposures. The MITRE Corporation. http://cve.mitre.org (Visited May 2005)

[7]ICAT Metabase. NIST. http://icat.nist.gov (Visited May 2005)

[8]Bugtraq Vulnerability Database www.securityfocus.com (Visited May 2005)

[9]http://www.spidynamics.com/ (Visited May 2005)

Workbenches like Core Impact[10], Canvas[11] and Metasploit[12] are other types of popular tools used to help the penetration testing process. With these tools the penetration tester has more control over the process. This kind of tools may be said to be semi-automated.

3.2.1 Scanners

Perhaps the most important set of tools for penetration testing are scanners. Port scanners are used for exploring alive machines and network topologies through different kinds of network sweeping techniques (ICMP sweeps, UDP sweeps, TCP sweeps and broadcast sweeps), and machine services on alive hosts through service port scans (TCP connect scans, SYN scans, ACK scans FIN scans, XMAS scans, Null scans, Idle scans, UDP scans) [39].

Vulnerability scanners are slightly different from other scanners. A vulnerability scanner uses a vulnerability database containing hundreds of known vulnerabilities, which it scans for. SATAN (Security Administrator Tool for Analyzing Networks) [22] is one of the first vulnerability scanners from the work of Farmer and Venema [19]. Nessus[13] [15] is an example of a current reputable widely used open-source vulnerability scanner. The scanner systematically engages the target in an attempt to assess where it is vulnerable to attack. Nessus also has the possibility for scripting specially grafted tests or attacks through NASL (Nessus Attack Scripting Language). Such scripting possibilities are very useful in an penetration test where special attacks may be necessary to be build or when a combination of flaws are needed.

Vulnerability scanning alone relies almost entirely on automated scanning scripts. Penetration testing relies more on the ingenuity of the penetration tester. The human element enables the tester to go beyond the simple discovery of vulnerabilities. Individual flaws are combined to illustrate the potential damage to the system [39]. Vulnerabilities that result from interaction among system components that include not single flaws, but also customised program elements and emergent properties such as timing and load behaviour that can arise only in certain hardware and software configurations.

3.2.2 Tool Limitations

Tools used in penetration testing, like vulnerability scanners, have limitations [74]. Relying solely on automated tool will often lead to a false sense of security [21]. The reality of vulnerability scanners on the market are that they are massively flawed [62]. Maybe the main reason for this is that if they had tested thoroughly for every reported vulnerability, they would only crash computers and damage networks. That is not a good sales product, so the scanners must act carefully. The golden mean that the tools rely upon to be usable lead to problems. It is important to remember that scanners and other tools have flaws, just as any other type of software. The problems with the results by current tools are [71]:

False Positives: A false positive is a vulnerability that is reported to exist, but does not.

---

[10]Core Impact is a penetration testing workbench developed by Core Security Technologies `http://www.coresecurity.com` (Visited May 2005)

[11]Canvas is a penetration testing workbench developed by Immunity `http://www.immunitysec.com` (Visited May 2005)

[12]Metasploit is a framework for developing, testing, and using exploit code `http://www.metasploit.com` (Visited May 2005)

[13]`http://www.nessus.org` (Visited May 2005)

The method that the tools rely upon can not guarantee that a certain hypothesised vulnerability exists. But the test for the vulnerability needs to be constructed sufficiently well that false positives are not reported. The problem with a false positive is that it wastes a security testers resources and time in analysing whether the problem really exists or not. If complete confidence is placed in the security assessment tool, then resources could be wasted in testing and verifying flaws that do not exist.

False Negatives: A false negative is a vulnerability that exists, but is not reported by the tool. This can happen even if the tool's documentation or vulnerability database state that the vulnerability is checked for. False negatives are more dangerous than false positives as it gives the testers a false sense of security in a vulnerable system.

Inconsistent Results: Inconsistent results occur if we run a tool against the same system several times and do not get the same results. The danger with inconsistent results is that a tester never can be sure whether the tool needs to be run several more times to ensure that he or she has found all the vulnerabilities the tool is capable of finding. Not only the tool is to blame for inconsistent results, the environment where the tools are ran can cause this, e.g. because of network load.

These problems arise from combinations of limitations in the tools:

Software bugs: All software has bugs. The testers should be aware of this and not place too much trust in the tools.

Poor data collection: The data collected by the tool may not be sufficient or appropriate to conclude whether a vulnerability exist or not. Another source for poor data collection is the location in the network that the test is done, sufficient data are not always available.

Poor data analysis: Making inference about existence of a vulnerability even with enough data collected is not easy, and the data analysis techniques in tools can be poor.

Insufficiency of tests: Tools may claim that they test for the existence of a vulnerability, but does not carry out several possible tests for that vulnerability. And of course some possible vulnerability tests that exist can be missing in the tool used.

Incomplete architecture: Some vulnerabilities must be tested from several locations and coordination between locations. Few of the current vulnerability tools use more that a single location in the network to generate tests from.

Not only do these technical problems arise when using different kinds of tools. Tools, e.g vulnerability scanners, do not help in building an attack for the penetration tester, this must be done manually. When a custom attack is needed to be build the tools also have shortcomings. For example custom vulnerabilities which may arise in system logics are close to impossible to find with tools that rely on finding known or at best specific classes of vulnerabilities. Building manual attack hypotheses is therefore often essential.

With the help of methodologies, attack modelling and human judgement many of these limitations can be reduced.

3.3 Flaw Hypothesis Methodology

The first work done on penetration testing in the early 1970s resulted in the flaw hypothesis methodology. It was developed by Richard Linde [42] and Clark Weissman [76]. The question behind this methodology was; how does one test for flaws that may lead to penetration and eventual control over the entire system [43]? This methodology and its ideas are used in almost every approach to penetration testing. RFC 2828 [63] defines the flaw hypothesis methodology as:

> An evaluation or attack technique in which specifications and documentation for a system are analysed to hypothesise flaws in the system. The list of hypothetical flaws is prioritised on the basis of the estimated probability that a flaw exists and, assuming it does, on the ease of exploiting it and the extent of control or compromise it would provide. The prioritised list is used to direct a penetration test or attack against the system.

The classic example of the flaw hypothesis methodology usage is an article by Attanasio *et al.* [3]. Here a team of testers use the flaw hypothesis methodology to find flaws in the VM/370[14] operating system. Here the source code of the operating system was the only available tool for the penetration testers. The thorough penetration test generated 880 flaw hypotheses where 76 of them were studied in detail and 35 flaw hypotheses were confirmed in the operating system. During the process the penetration testers clearly benfited from following the flaw hypothesis methodology for saving resouces and providing structure.

The flaw hypothesis methodology has been refined and has become a key method for performing penetration testing. Its ideas are found in most of the current penetration testing methodologies, as the ones described in Section 3.1. Because this methodology is used in this thesis, a more detailed description of the methodology is now presented.

3.3.1 Flaw Hypothesis Methodology Theory

The methodology introduces the word "flaw". It is a subtle distinction between a flaw and vulnerability. Apprehend a flaw as a specific occurrence of a specific vulnerability, i.e. a penetration test discover a flaw, which then can be generalised based on the vulnerability that was used to hypothesize the flaw. The terms are often used interchangeable.

Bishop presents the Flaw Hypothesis Methodology in a four step method in [6] which is equal to Weissman and Linde's original methodology presented in [42, 77, 78]. Bishop's presentation of the methodology will be used and further refined in this thesis:

1. Information gathering.

2. Flaw hypothesis.

3. Flaw testing.

4. Flaw generalisation.

There are some steps that are natural to add in this presentation, namely a goal definition and flaw elimination. Originally goal definition is left out of the listing of the methodology, but is presented as a background study by Weissmann [78]. Since goal definition is an important part of testing it is added to our listing. Flaw elimination is a

---

[14]IBM Virtual Machine Facility/370

natural part that is done after flaws have been verified in a penetration test. This step was also added as an extra step by Weissmann in 1995 [78].

The following list briefly explains each step in the complete flaw hypothesis methodology [42, 77, 78]:

1. *Goal definition*. The test starts with setting the scope, establishing ground rules and objectives, and defining the purpose of the test. Overall goals may be time limit of the test, number of flaws to be found, etc.

2. *Information gathering*. The system's functioning, design, implementation, operating procedures, and its use are examined as thoroughly as possible. Limited by the information available in the test scenario. A firm understanding of the target system is important.

3. *Flaw hypothesis*. The knowledge from the former step is used together with knowledge of vulnerabilities in other similar systems to hypothesize flaws of the system under study.

4. *Flaw testing*. The hypothesised flaws are tested. If the flaw does not exist or is unexploitable, the tester go back to step 2. If the flaw is confirmed, they proceed to the next step.

5. *Flaw generalisation*. The testers try to generalise the vulnerability and find similar ones. Other hypotheses may be yielded from successful tests. The new understanding is used in an iterative process back to step 2.

6. *Flaw elimination*. This step proposes how the found flaws can be patched or fixed, so that they cannot be exploited again. Here the testers must have in mind that fixing a flaw may introduce additional flaws.
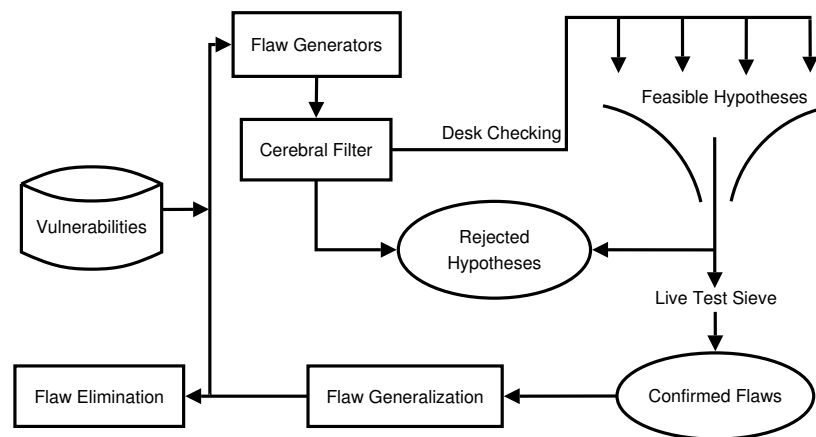
Figure 8: The flaw hypothesis methodology [77, 78]

Figure 8 is a reproduction of the original Figures by Weissman with slight modifications. A collection of vulnerabilities is used to generate flaws, this is typically done with the use of a vulnerability database [71]. Other information is also used to generate flaws like system design documentation, source code, user documentation, and results of unit and integration testing. The level and amount of information depends on what is avail-

able under the test, e.g. if the test is white, grey[15], or black box. This is the second step, information gathering. The flaw generator is then used to hypothesize flaws. The cerebral filter applies human judgement to the hypothesized flaws. These filters evaluate and rate each flaw in terms of its probable existence and how significant it is. Now we are in the flaw hypotheses step. The feasible flaws from the cerebral filter are desk checked and tested live in the live test sieve if necessary. Here the mapping of the flaw into a test happens, which can be a difficult task. This is the flaw testing step. Flaws that are found unlikely or non-existent are rejected, preferably before live testing. Now we enter the flaw generalisation step. Confirmed flaws in the tested system are generalised, that is an assessment of the found flaws is done, the reasons why they exist are explored and the flaws are analysed for patterns that may be repeated other places in the system. The flaw elimination stage considers the generalised flaws and recommends ways to repair the flaws, which is flaw elimination. Another possible outcome of this step are generation of new flaws, these are used to further hypothesis flaws.

Any penetration testing process is unavoidably dependent upon the flaw hypothesis methodology model [45]. We can see many similarities with other methods used in penetration testing, as discussed in the beginning of this Chapter.

3.3.2 Methodology Limitations

The job of following the steps in the flaw hypothesis methodology is hard work and can fast become over-complex.

The traditional method of hypothesising flaws is with the use of brainstorming of some kind. Brainstorming is the method proposed in the original flaw hypothesis methodology [42, 77, 78]. The expertise using the target system and experience in security testing is important to yield good results [29]. Using brainstorming is important in any penetration test, but systematising and visualising the attacks in a more formal manner are useful.

Gupta and Gligor, [25], argue that the flaw hypotheses in the flaw hypothesis methodology are generated in an ad-hoc manner, this because there is generally no systematic ways to generate and test flaw hypotheses are available in the methodology. Knowing the ways to reach and exploit the flaw is nontrivial. The knowledge and experience of the testers are vital. Gupta and Gligor argue that the abstraction at which the methodology is meant to be used is too high. The flaw hypothesis methodology can be said to only provide high level guidance for derivation of actual penetration attempts. Defined attack models and tools to help the process will make it easier to use the flaw hypothesis methodology. In fact in the same article by Gupta and Gligor, they propose a theory on how to formalise the notion of penetration-resistant systems.

The flaw generalisation step makes inferences about the existence of a vulnerability based on the results on the test for a flaw. Flaws found are often tested manually, especially to verify results that may be returned by used network security assessment tools. Relying on the results from such tools can give false positives, false negatives and inconsistent results, as discussed in Section 3.2, that may reduce the reliability and validity of a penetration test.

The coloured Petri net attack modelling approach presented in Chapter 5 will try to

---

[15]Grey box testing represent a test with system knowledge in the area between a fully black or white box testing. Most penetration tests is grey box [24].

address some of these limitations.

3.4 Fault Tree Analysis Based Methodologies

This Section briefly presents an alternative methodological approach to the flaw hypothesis methodology, which is largely based on fault tree analysis [53]. Today attack trees [61] and threat trees [68] are widely used, both are descendent from fault tree analysis. The tree structure is helpful in organising penetration tests, and is considered as an alternative to the flaw hypothesis methodology, however they both do have properties in common. Trees provide a goal oriented methodology to system penetration testing, mostly focused on black box testing.

3.4.1 Fault Tree Analysis

Fault trees was originally developed for system analysis of the Minuteman strategic missile system [40] in 1962. Fault trees are widely used in dependability analysis [53]. A fault tree consists of all sequences of individual component failures, which could make a system stop functioning. The root node of a fault tree is often referred to as system failure. Each level in a fault has events that are divided into combinations of sub events using logical gates. The detail level is then extended from parent nodes and at the end a complete tree present all the possible ways to system failure.

Fault tree analysis have been used in many different systems [53], e.g. in complex systems like space shuttle design. Attack trees can be seen as a refinement of fault tree analysis for attack modelling in penetration testing.

3.4.2 Attack Trees

Attack trees offer a goal-oriented perspective on the attacks [61]. Threat trees [68] are treated as the same as attack trees in this thesis, the distinction between them are small. It can be difficult to limit the trees to just discussing threats and not drift away into specific attacks [48]. The attack tree formalism is mostly used in penetration testing, where the system attacks are the main focus.

In an attack tree the goal of the test is the root node. Thus attack trees are mostly modelled as a top-down approach. Sub-goals are then represented at the next level. This is repeated until a detailed tree is created on how to attack the system under scrutiny.
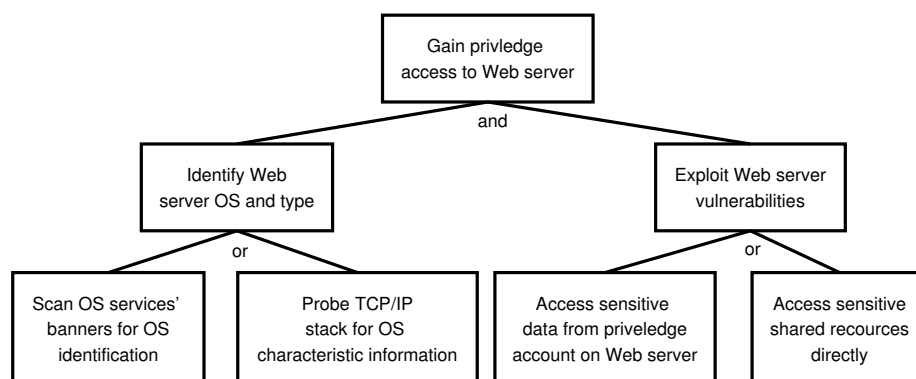
Figure 9: Attack tree example

A small attack tree is described in Figure 9. Here an attack of a web server to gain priv-

ilege access is sketched. As we can see from Figure 9 the tree is build with disjunctive (or) and conjunctive (and) node relationships. Disjunctive means that the attack or action is accomplished if any of the actions described by its children are accomplished. Conjunctive nodes however mean that both child actions or attacks must be accomplished before the attack can be done successfully.

Attack trees have been refined and applied to many different penetration testing cases [48, 49, 70]. Examples of refinement are assigning values to the leafs of the tree, e.g. attack cost and probability of success weighting, which can be propagated up the tree, thus minimum cost or maximum probability of reaching the root node can be calculated.

4   Related Work

This Chapter present related work in the topic of attack modelling. The fault tree based methodologies presented in Section 3.4 offer a graph based attack model, however this methodology is not closely related to the coloured Petri net attack modelling presented in this thesis. However some research that refine the attack tree approach are presented below. There are different kinds of specific attack graphs that are more closely related to this thesis' approach. The third and closest related graphical attack model is done with different kinds of Petri nets. All these graphical models are more or less in family, that is they use different kind of graphs to visualise an attack. There are also other attack models, but these three models represent the most relevant for our research.

Attack modelling is not only used in penetration testing. Other areas of usage are in intrusion detection and computer forensics. Intrusion detection is the area where attack modelling is closely related to how it is done in penetration testing, mostly because intrusion detection and penetration testing both focus on how the attack is performed and what is done under the attack. The different relevant research are now presented.

4.1   Tree Based Attack Modelling

In Section 3.4 we describe the attack tree methodology for penetration testing. The methodology uses a graphical tree to show how the penetration tester can approach an attack or penetration attempt to a system, it is therefore somewhat related to our coloured Petri net based attack modelling approach. However the detail level of which the trees are meant to be modelled are not equal to the coloured Petri net approach, for example the actual behaviour of the attackers and the functioning of the attack are not modelled in trees. Some refinements of attack tree modelling have been done to more closely model the attack process and the attacker's actions. Daley *et al.* [13] in 2000 presented an extention to the attack tree paradigm that supports incident correlation, analysis, and prediction, but there are no possibilities in their model to simulate or execute attacks.

The attack tree structure has limitations in modelling sequence or dependency between conjunctive and disjunctive nodes, e.g. it is not always clear in what order nodes must be reached or done. Also there are often good reasons for nodes to be both disjunctively and conjunctively connected. Sequence and dependencies can be better visualised by other types of graph modelling. The attack trees ability to model sequence and dependency is a problem when closely modelling information system attacks, as McDermott [45] points out.

The structure of the attack tree can be difficult to edit and extend dynamically. Adding a conjunctive branch to an existing disjunctive node can require restructuring existing

child nodes, which can be very inconvenient and confusing. The structure of a tree also make it difficult to model cycles [57], e.g. reuse of graph strucures in attacks.

Another difficulty with attack trees is that both actions and resulting states are represented in a single node. This can lead to unclear node labels [65]. In Figure 9 back in Section 3.4 the root node gain privileged access to web server express that an attacker both gain capabilities and has to perform some action achieve it. However it can be hard to clearly define both the actions and capabilities in one node, or even differentiating between them. Attack trees and fault trees also miss the possibility to simulate attacks.

4.2 Attack Graph Modelling

Another way to model attacks is with use of simple directed graphs. One approach to this has been developed by Philips and Swiler [57] in 1998. They use a graph based approach to help in a network vulnerability analysis similar to a penetration test. Their weighted graph focuses on finding the easiest attacks in a network. It is meant to go beyond typical scanning tools that check laundry lists of services or conditions to find vulnerabilities. The graph type in use is a simple graph with one type of node. It is used in a similar way as attack trees, where the root node is the goal and sub nodes are subgoals. The edges represent transitions between the nodes. The graph is primarily aimed for use in automated network analysis of large networks, using a database of known attacks. Graphs can help in identifying linked attacks that are potentially more harmful than individual attacks and it can model cyles in attacks. The models are however missing possibilities to model states and concurrency in attacks, and a possibility to simulate attacks, i.e. execute models.

4.3 Petri Net Attack Modelling

The main source for Petri net attack modelling comes from McDermott [45], published in the year of 2000. He identifies limitations and disadvantages in attack tree modelling and propose the use of Petri net attack modelling, the nets are called attack nets.
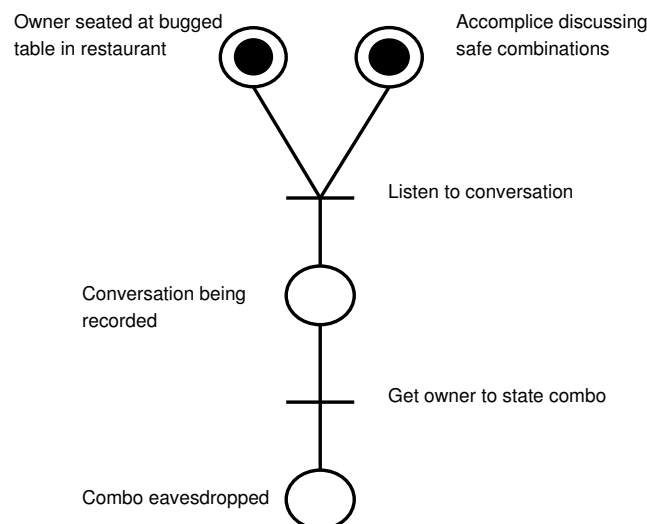

Figure 10: Attack net example [45]

McDermott uses simple disjunctive Petri nets as a mechanism for structuring the pen-

24

etration testing process. The attack nets represent larger-scale states of the system under attack. In this model, attack steps are represented by places similar to nodes in attack trees. Transitions are used for modelling the attackers actions which extends the expressiveness of this model compared to attack trees. Figure 10 show a simple example taken from McDermott's article [45]. As we can see the tokens represent current control states of the attack, that is tokens represent a marking (state) of the attack, and the movement of the tokens represent every step the penetration tester must take in order to achieve his objective. The net uses the Petri net or place/transition net formalism without additional constraints (as described in Section 2.2). The transition `Listen to conversation` in Figure 10 will only fire if both input places have a token present, this implies that only then can the attacker successfully record the conversation in progress and hopefully get the owners safe combinations.

The attack net is used as a part of penetration testing with the flaw hypothesis methodology where the attack net is developed during the flaw hypothesis step in the methodology. The nets are used to design the attacks. The development is done iteratively and McDermott recommends building initial subnets and then complete the attack net by combining these initial subnets. The subnets are single commands or actions that take a product or system into an undesirable state, represented by two places connected by a single transition.

The mechanism in the attack net provides the tester with the ability to model

- concurrency and attack progress in the form of tokens

- intermediate and final objectives as places, and

- commands or inputs as transitions.

As note by McDermott, the attack net mechanism is not intended to model actual behaviour of attackers and does not provide the requisite level of detail for such a process. This thesis will further refine the McDermott attack net mechanism.

Another example of usage of Petri nets in attack modelling is Steffan and Shcumachers published work from 2002 on collaborative attack modelling [65]. They present a Wiki Wiki Web system[1] to enable groups of persons to collaborate on the same attack models. The models make use of conditions (corresponding to places in Petri nets) and transitions, both different types of web pages in the Wiki Wiki Web system. Arcs are simply hyperlinks between the Wiki Wiki Web pages. The Petri net provide a solid framework for their collaborative attack model. However they do not use features in Petri nets like tokens and execution possibilities.

4.3.1 Coloured Petri Net Attack Modelling

Literature on usage of high-level Petri nets, like coloured Petri nets, to model attacks is scarce. However some studies have been done.

A small article by Zhou *et al.* [80] published in 2003 discuss coloured Petri net based attack modelling. They identify limits in attack tree modelling and explain how to map an attack tree to coloured Petri nets. They also argue that coloured Petri net attack models can be beneficial for identifying controlling functions against modelled attacks. This can be used for intrusion detection systems to allow active response and defence.

Kumar and Spafford [37] and Helmer *et al.* [26] uses coloured Petri nets to model in-

---

[1]The WikiWikiWeb `http://c2.com/cgi/wiki?WikiWikiWeb` (Visited May 2005)

trusion detection for intrusion detection systems. Kumar and Spafford developed a model with pattern matching for intrusion detection that uses coloured Petri net attack patterns in 1994. Helmer *et al.* presented their multi-agent intrusion detection system which uses a coloured Petri net design to detect attacks in 2002. Agents are modelled as places or transitions. Hierarchical coloured Petri net models are used for correlation of individual attacks that detect intrusions. Token colours contain placeholders for the objects of an attack, e.g. target and source address. Transitions use logical conditions to make use of these placeholders. The coloured Petri nets are designed from fault trees (see Section 3.4) and must be build according to a framework so it can be implemented into the intrusion detection system. A difference found in attack modelling within the intrusion detection domain as opposed to in penetration testing, is that intrusion detection mostly focus on classes of attacks and not specific attack scenarios.

Stephenson, in [66, 67] from 2003/2004, uses coloured Petri nets to model attacks for use in computer forensics. E.g. to model how a worm or virus infection has been spread through an enterprise network structure. This approach is not too relevant for attack modelling in penetration testing.

5 Using Coloured Petri Nets in Penetration Testing

The penetration testing approach presented here is based on the concept of coloured Petri nets as a methodical way of hypothesising and analysing attacks on an information system and the ideas of the flaw hypothesis methodology as a guideline for the penetration testing process. The steps in the flaw hypothesis methodology, see Section 3.3, are used and coloured Petri nets are designed to visualise and simulate hypotheses of system vulnerabilities. Results from vulnerability assessment tools (see Section 3.2), like Nessus, are meant to be used to both find individual vulnerabilities and help with the process of verifying the hypothesised vulnerabilities, e.g. through NASL (see Section 3.2.1).

5.1 The Overall Model

Figure 11 shows where in the original flaw hypothesis methodology Figure by Weissman [77, 78] the coloured Petri net will be developed.
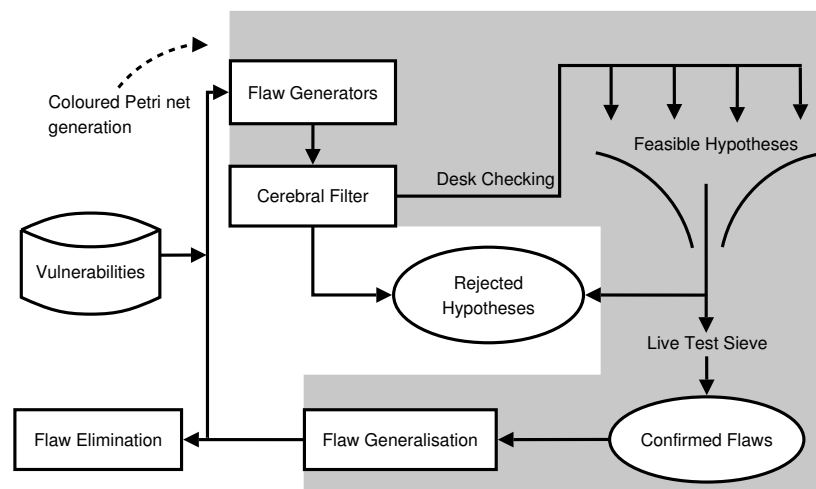
Figure 11: The flaw hypothesis methodology with coloured Petri net generation (marked with gray background)

Figure 12 shows our use of coloured Petri nets as an attack model together with the flaw hypothesis methodology steps. This approach is similar to McDermott's high level organisation in [45]. The flaw hypothesis methodology is done iteratively and the tester will have to do the flaw hypothesis steps several times when found flaws are investigated. This is also what Linde [42] and Weissman [76, 77, 78] proposed in the original flaw hypothesis methodology without a graphical attack model. During the iterative process

the coloured Petri net attack model are created to help understand, control and simulate the hypothesised system attack.

In the iterations of the methodology the information gathering step is the most important, which is true in any penetration test, without target knowledge it is almost impossible to find a valid vulnerability. In flaw hypothesising, flaw testing and flaw generalisation it can always happen that we need more information and thus we must iterate back to the information gathering step. Even back to other steps if that is more accommodated. The coloured Petri net attack model will be developed within the steps.
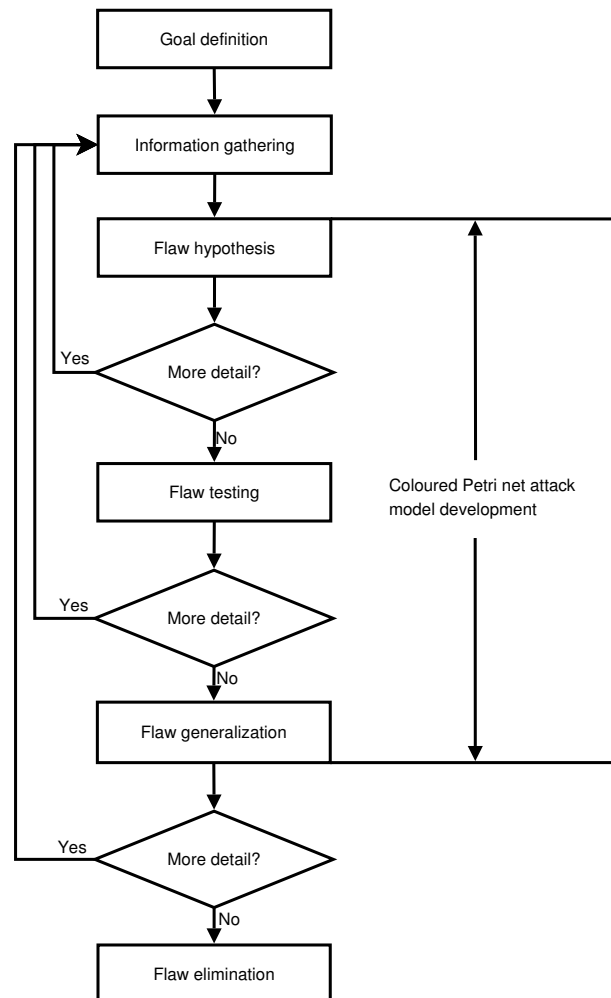
Figure 12: Overall methodology for using coloured Petri nets in penetration testing

5.2 Motivation for Coloured Petri Net Attack Modelling

Attack modelling is based on taking the viewpoint of an attacker. This is also true in penetration testing. Attack modelling together with penetration testing give possibilities of modelling advanced attack scenarios and have close to full control over the test in progress. Especially when modelling attacks with coloured Petri nets we gain the advantage of modelling attacks where the mental image capacity of the penetration tester is breached.

The main quality of an attack model is that it must characterise every step of attacks accurately and point out how the attacking process continues [70]. The model will help analyse how an attack might be accomplished through attack paths [68]. The attack path is a route from one condition to another. Different methods exist for modelling attacks, see chapther 4. However in this thesis coloured Petri nets are explored for the modelling purpose, which has mechanisms for detailed modelling.

5.3   Coloured Petri Net Attack Model

Petri net as an attack model are both state and action oriented, which is clearly advantageous when modelling computer attacks. Places represent attack or system states. Transitions represent actions, commands or attacks. Put in another way the places are the passive components, while transitions represent the active components in the model. Tokens show the progress of the attack and carry colours that are used under execution of the net.

McDermott [45] and Zhou *et al.* [80] both specify two different arc/transition relationships used in their Petri net based attack models. The two different basic node relationships in Petri nets are disjunctive and conjunctive. Figure 13 show these relationships modelled in Petri nets. Disjunctive relationships means that one of the former places must be reached by the token before state change can happen or the transition can be fired, or that one of the output places receives tokens after firing. Conjunctive means that both of the former places must be visited before the state change can happen, or that both output places of a transition receives tokens. Both similar to what is used in attack trees, as shown in Section 3.4. Figure 13 show these relationships.
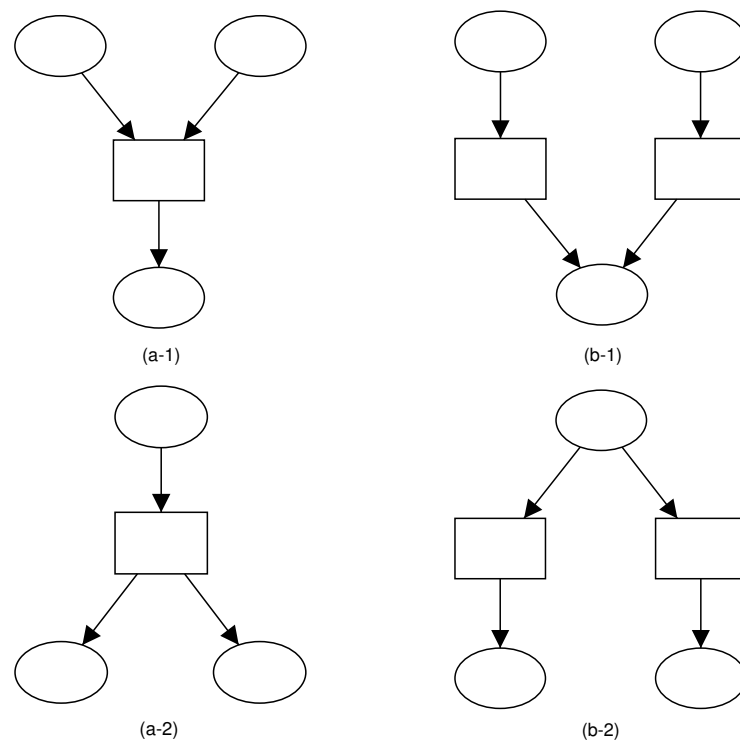
Figure 13: Disjunctive (b-1,b-2) and conjunctive (a-1,a-2) Petri net relationships

By adding colour to Petri nets, more detail can be inserted into a net based attack model. Because each place has different colour sets (types), modelling of attacks can be done in much greater detail. The colour of the tokens in one place may differ, and is not limited to a single value. A simple example that shows this can be with the declarations as follows (see Appendix A for net inscription and declaration explanation):

```
(* type (colour sets) definitions *)
colour SambaAttack = with smbBID7294 | smbBID7107;
colour SambaAttackSelected = SambaAttack;

(* variable declarations *)
var attack : SambaAttack;
```

This shows an simple attack on a Samba[1] server. The server have been found vulnerable to two different vulnerabilities; a buffer overflow (Bugtraq ID 7294[2]) and a race condition (Bugtraq ID 7101[3]). Figure 14 show the initial stages of how this attack can be modelled.
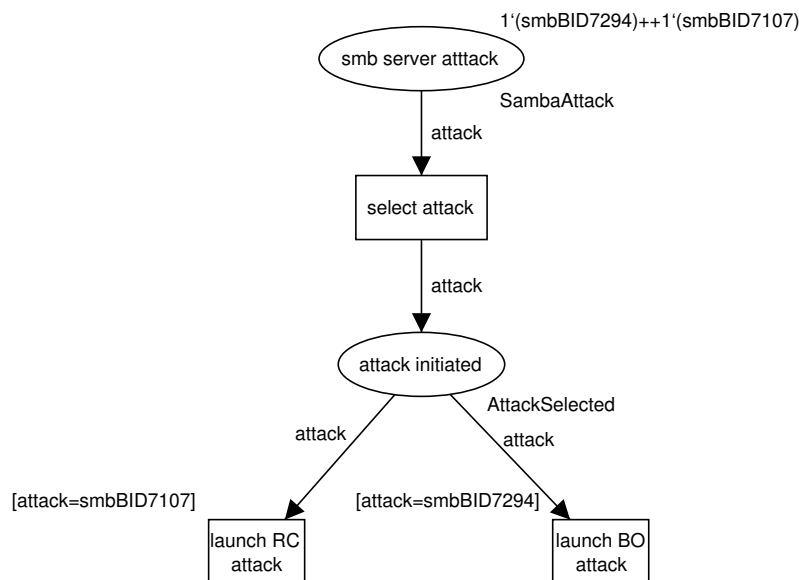
Figure 14: Coloured Petri net example for Samba attack

The place `smb server attack` has the colour set `SambaAttack` with the possible colours `smbBID7294` and `smbBID7107`. Of course the naming of these definitions can be just as the penetration testers wish, the most important is that they are good mnemonic names that are easy to understand [31]. The colour sets can hold different objects of an attack, e.g. attack type, target host, source host and others, or they may also hold objects that are not directly a part of the attack, e.g. process states, system status information and others. The colours used here represent attack type or what vulnerability that is going to be exploited.

---

[1] Samba provide file and print services to SMB/CIFS clients, see http://www.samba.org (Visited May 2005)
[2] http://securityfocus.com/bid/7294 (Visited May 2005)
[3] http://securityfocus.com/bid/7107 (Visited May 2005)

Different actions or processes can easily be modelled within the coloured Petri net, e.g. the attacker's actions and the relevant workings of the attacked system.

The two tokens in the above example, one with the colour `smbBID7294` and the other `smbBID7107` can take different directions in the coloured Petri net and change in both amount and colour. This can be manipulated by arc expressions and transition guards (see Section 2.3). The arc expressions can control which tokens that are removed from input places and which is added to output places. The transition guards are used to control the enabling by Boolean expressions. In Figure 14, guards are used on the transitions `launch RC attack` and `launch BO attack`, to make sure that the tokens are removed and added to the correct attack path in the coloured Petri net. The last tree nodes, two transitions and one place, represent a disjunctive relationship where tokens take one direction, not both, thus guards or arc expressions are used to constrain correct tokens for each transition.

5.3.1 Timed Coloured Petri Nets

Using timing is a feature in coloured Petri nets that can be useful in attack modelling. Timing does appear in different computer vulnerabilities and timed coloured Petri nets can model such attacks. The tokens in a timed coloured Petri net can be time stamped, see Section 2.3.1.

Timing is useful as an additional constraint in attacks where order is important, e.g. when an attack is done by multiple hosts and time is used to arrange the order, or other types of collaborative attack scenarios. However the most useful area of the timing is when an attack must happen between exact interleaving in a system operation time, e.g. local race conditions and other attacks that require time synchronisation of events. For local attacks, timed coloured Petri nets works great for modelling.

When the timing attacks are used in distributed environments it can become difficult to model with timed coloured Petri nets, this because of the variation in time often found when dealing with multiple agents in networked environments. Then intervals in the modelled time can be used.

5.3.2 Interval Timed Coloured Petri Nets

The time intervals provide a mechanism to model uncertainty and nondeterminism in individual transitions. Timing-dependent attacks, particularly executed by multiple agents in a distributed environment, can be modelled in interval timed coloured Petri nets. Examples of relevant attacks are race conditions and resource contention attacks in distributed enviroments. Network latency and load conditions can provide significant distortions in the timing and sequencing of events and actions in order to successfully exploit vulnerability.

A simple example is presented in Figure 15, Appendix A explin the inscriptions. A race between the injection of a forged package and a true response usually happens in a spoofing situation. The attacker must send a forged response package to a client, before the real server respond. Then the attackers forged package hopefully will be accepted as a real server response by the client, that is if the system has a flawed or breakable server authentication mechanism. As we can se from the time intervals the injection will not always be successful, e.g. if the server response is sent between time values 2 and 3 the attacker may fail if the injection consume more time, load conditions and network latency can be one of the causes behind the nondeterminism in the communication.
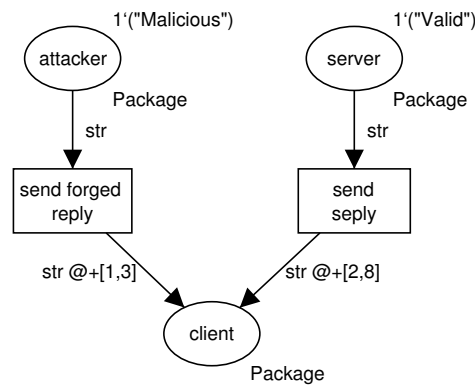
Figure 15: Interval timed coloured Petri net example for Spoofing

The time intervals provide a straight forward mechanism for both simulation and execution of the time intervals found in timing attacks. However interval timed coloured Petri nets do use the simplified assumption of a single global time and point-value time representations, which is not entirely appropriate for distributed where such cannot be determined. However, for the purposes of modelling described in this thesis, the interval time properties provide sufficient abstraction even for fuzziness of timing in multi-agent distributed environments when used in simulation.

5.3.3   Development Stages

In the limited experience in attack modelling in this thesis, the following describe our recommended method of developing the differnet coloured Petri net attack models, based on the recommendations by Jensen [31]:

1. *Start by identifying the most important components of the attack.* This can be done by listing objects, processes, states and actions that are vital to attack. It can also be beneficial to categorise these components, e.g. into objects, processes, states and actions.

2. *Consider the purpose of the model and determine adequate level of detail.* As many actions may consist of a number of sub actions, we have to decide if these sub actions are relevant to model as separate actions. E.g. preparation actions done by an attacker may not be necessary to model in great detail.

3. *Try to find good mnemonic names for objects, processes, states and actions.* All nodes in the net should be easy to understand and possible to distinguish. It becomes much easier to model the attack if the name of the nodes makes the penetration tester immediately remember what the nodes represent.

4. *Do not attempt to cover all aspects of the attack in the first version of the model.* The detail should come naturally through the iterations in the flaw hypothesis methodology (see Section 5.1). Starting with a simplified version of the attack makes it easier to see where the detail in model should be.

5. *Choose one and one part of the processes in the attack and try to make an isolated net for this process.* Represent each state by a place and each possible change from one state to another by a transition. States that are identical and has the same logical

properties can be uncovered and designed together.

6. *Use the net structure to model control and the net inscriptions to model data manipulations.* It can be a good idea to use the net structures in a similar way as the control structure of an ordinary programming language, see Figure 16 for how to model some useful control structures. Also always having the disjunctive and conjunctive relationships in mind is useful when modelling attacks, see Figure 13.

Figure 16: Petri net control structures

7. *Distinguish between different kinds of tokens.* The tokens can represent many different phenomenas in a net, e.g. some tokens may represent physical actions while others represent shared data.
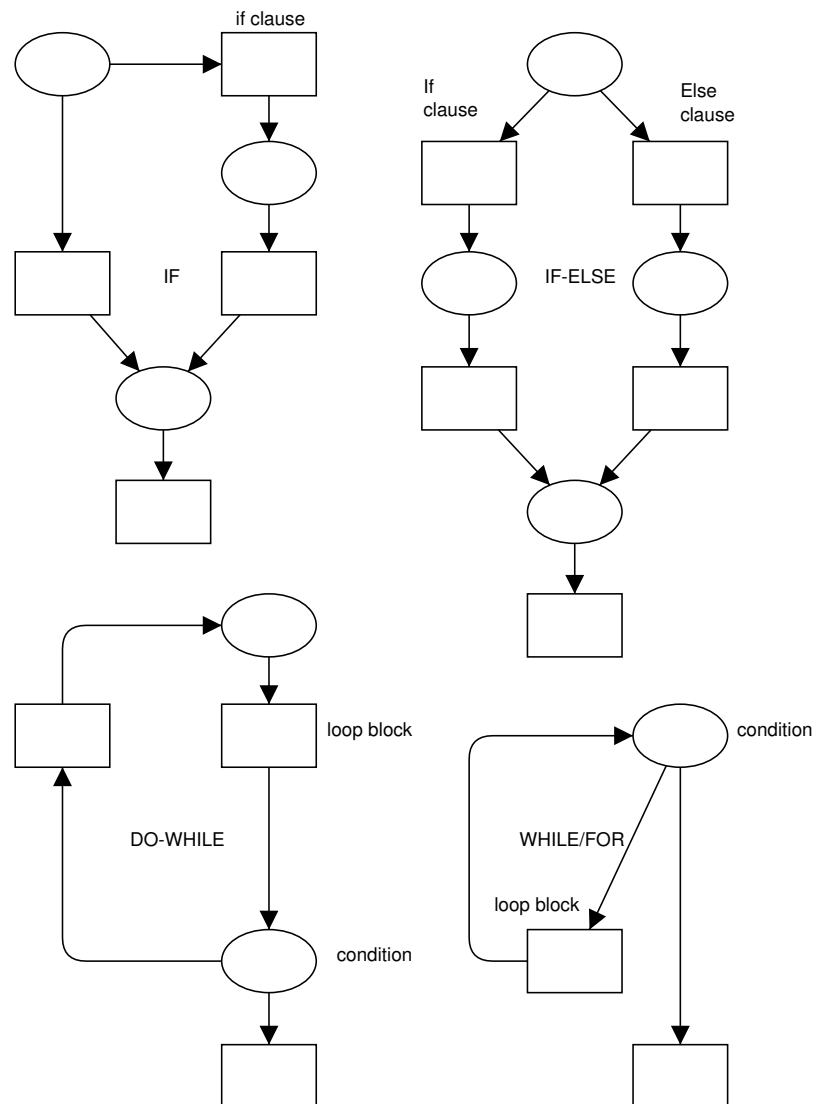
8. *Use different kinds of colour sets.* There are many possible colour sets that can be

used in coloured Petri nets, e.g. indexed sets, Cartesian products and lists. Enumeration colour sets are used to name objects by means of mnemonic names. Indexed colour sets are similar, but we can have a class of objects which can be denoted by class name and object number. Product and record colour sets are used to represent sets of attributes, e.g. the source IP, destination IP, and the data of a message. Union colour sets are used when the same place may contain more than one kind of tokens. List colour sets can be used in the same way as arrays. CPN ML[4], which is the modeling language used in coloured Petri, based on standard ML[5] , explain the complete notations.

9. *Augment the individual process net by describing how the process communicates/interacts with other processes.* This can be done by merging transitions or places. It can be necessary duplicate both places and transition to make it work.

10. *Investigate whether there are classes of similar processes.* When this is the case it is often convenient represent these by a single net, which has a token for each process. The colour of the token identifies the process, while the position tells the state of the process or attack.

11. *Combine the subnets of individual processes to a large attack model.* At the end, that is the last iterations of the flaw hypothesis methodology, we combine loose subnets into the complete attack model. To do this it is possible to use hierarchical coloured Petri nets, however this is not covered by this thesis and are mostly used when the modelled system are very large and complex. Simply connecting the subnets with conjunctive and disjunctive relationships in mind can build the complete attack scenarios from subnets of individual processes. These connections can be done with equal transitions or places.

12. *Making a coloured Petri net attack model is very similar to the construction of a program.* Qualities of good programming are also desirable qualities of coloured Petri net modelling [36].

All these steps represent good guidelines for coloured Petri net attack model development. They are not meant to be followed strictly, but they hopefully make the attack model development within the flaw hypothesis methodology more unconstrained.

5.4 Main Benefits

The main benefits with using coloured Petri nets combined with the flaw hypothesis methodology in penetration testing are:

- The model are graphically made and presented. An attack is intuitive to understand when represented graphically in a coloured Petri net.

- Coloured Petri nets have well-defined semantics which unambiguously defines the behaviour of an attack modelled.

- We can model concurrency and attack progress with tokens.

- Coloured Petri nets builds upon well defined mathematics and a programming

---

[4]Coloured Petri Net Modelling Language, see `http://wiki.daimi.au.dk/cpntools-help/cpn_ml.wiki` (Visited May 2005)

[5]The Standard ML Basis Library `http://www.standardml.org/Basis/` (Visited May 2005)

language which make it possible to model diverse attack scenarios.

- Coloured Petri nets model both states and actions. This is in contrast to other graphical attack models and system description languages which describe states and actions the same way or only one of them.

- Intermediate and final stages and states of the attack are modelled as places.

- Commands, inputs or actions are modelled as transitions.

- Individual nets can be reused in larger attack scenarios.

- Coloured Petri nets are stable towards changes in the model. Different subnets describing different sequential processes can be combined into larger coloured Petri nets.

- Timed coloured Petri nets are useful for modelling penetration attempts that require timing.

- Interval timed coloured Petri nets can introduce fuzziness into the model, this helps us to model more advanced timed attack scenarios.

- The model can be executed. Simulations make it possible to debug large attack scenarios while the attack is being hypothesised and the complete attack is clearly visualised through net execution.

- Coloured Petri nets refine the flaw hypothesis methodology.

6   Scenarios

This Chapter will present three penetration scenarios that use coloured Petri nets for attack modelling and the flaw hypothesis methodology as penetration testing methodology, as described in Chapter 5. The reason for these three scenarios is to illustrate the benefits of using the proposed approach with the coloured Petri nets. One scenario without timing, one with ordinary timing, and the last with interval timing. The scenarios are not live penetration tests, they are arranged cases that are chosen to demonstrate the approach. They are not meant to be quantitative experiments, this has not been done in the thesis work, as described in Section 1.6.

Some assumptions are made regarding information available during the penetration test scenarios, the focus is on the key elements of the attacks that are useful for demonstrating the theory. The assumptions are stated in each scenario.

All verification testing is done in a controlled environment. A public beta version of VMware Workstation 5[1] is used as platform for various hosts. To design and simulate the coloured Petri net attack models, CPNtools[2] [4, 58] have been used. The vulnerability scanner Nessus [15] have also been used to compare and verify results. Other necessary software used are all based on open source.

6.1   Scenario 1 – SQL Injection

This scenario is based on typical web application input validation vulnerability. The scenario study is similar in many of today's web application vulnerabilities that are reported daily at Bugtraq and other vulnerability reporting sites. It is a simple attack that use coloured Petri nets as an attack model.

Here an input validation vulnerability in PHP-Nuke[3] version 6.0 are exploited. The input validation vulnerability has been patched in newer versions of PHP-Nuke.

Goal Definition

The overall goal is to gain administrative privileges to the content management system. The penetration tester has an ordinary user account to the system, meaning he can post comments etc. to articles presented in the system.

Information Gathering

By visiting the web site in question much information can be gathered. The content management system in use is the open source application PHP-Nuke. Therefore docu-

---

[1]VMware Workstation is a software for running virtual machines and networks, see `http://www.vmware.com` (Visited May 2005)

[2]CPNtools are a reputable tool for editing, simulating and analysing coloured Petri nets, available at `http://wiki.daimi.au.dk/cpntools` (Visited May 2005)

[3]PHP-Nuke is a popular open source content management system, available at `http://phpnuke.org/` (Visited May 2005)

mentation and source code is freely available on the Internet. In this scenario, searching vulnerability databases are useful. PHP-Nuke, like most web applications, has many vulnerabilities reported.

The version of PHP-Nuke used is version 6.0. This version is vulnerable to SQL injection in the search module of the web application (Bugtraq ID 6887[4]). This vulnerability may be exploited to cause sensitive information to be disclosed to a remote attacker.

From a simple code review and web searching we can determine that each user's login credentials are stored in a SQL database. The passwords are stored using a MD5 hash function. For session handling and ease-of-use, the user credentials (username and MD5 hashed password) are stored in a client-side cookie, as many web applications do.

Flaw Hypothesis

This is the step where flaws are hypothesised and attacks are modelled using coloured Petri nets. This step is done iteratively with the flaw testing and generalisation step until enough detail is modelled in the coloured Petri net, as described in Chapter 5.

There are some obvious approaches that may be possible to reach the overall goal of this penetration study; brute-force guess administrator credentials, elude the authentication and/or identification mechanisms, or acquire valid administrator credentials. The former method seems possible through the reported vulnerability found during the information gathering. If the password hash of a privileged user can be extracted from the backend database we can build a cookie with administrator credentials and successfully reach the overall goal. So the operative flaw is that an attacker can Figure out an administrator's password hash through SQL injection [28], and then build a valid client-side cookie, that permits the attacker to login as the administrator.

Starting by hypothesising some key places and transitions are useful, including control states that are important to reach during the attack:

Key places:	Key transitions:
`administrator username`	`query database`
`password hash`	`login with administrator credentials`
`valid administrator credentials`	
`privileged access`	

Then we hypothesise initial subnets from single commands or actions that can be used under the attack. Most individual actions can be divided into such subnets. These can be modelled as (the smallest possible nonempty) disjoint Petri net [45], that is two places connected by a single transition. In a real world scenario with several penetration testers, such nets can be developed by individual team members or in brain storming sessions. The idea is that these subnets then can be correlated in complete attack coloured Petri nets, as described in seciton 5.3.3.

Subnet 1:

- `Valid admin credentials`

- `Build and use admin cookie`

- `Privileged access`

Subnet 2:

- `Hex char to test`

---

[4]`http://www.securityfocus.com/bid/6887/` (Visited May 2005)

- `Query database`

- `Retrieved password hash character`

Now when key places, transitions and some initial subnets are hypothesised, we can start to develop a coloured Petri net for the complete attack. Figure 17 show the complete hypothesised coloured Petri net.
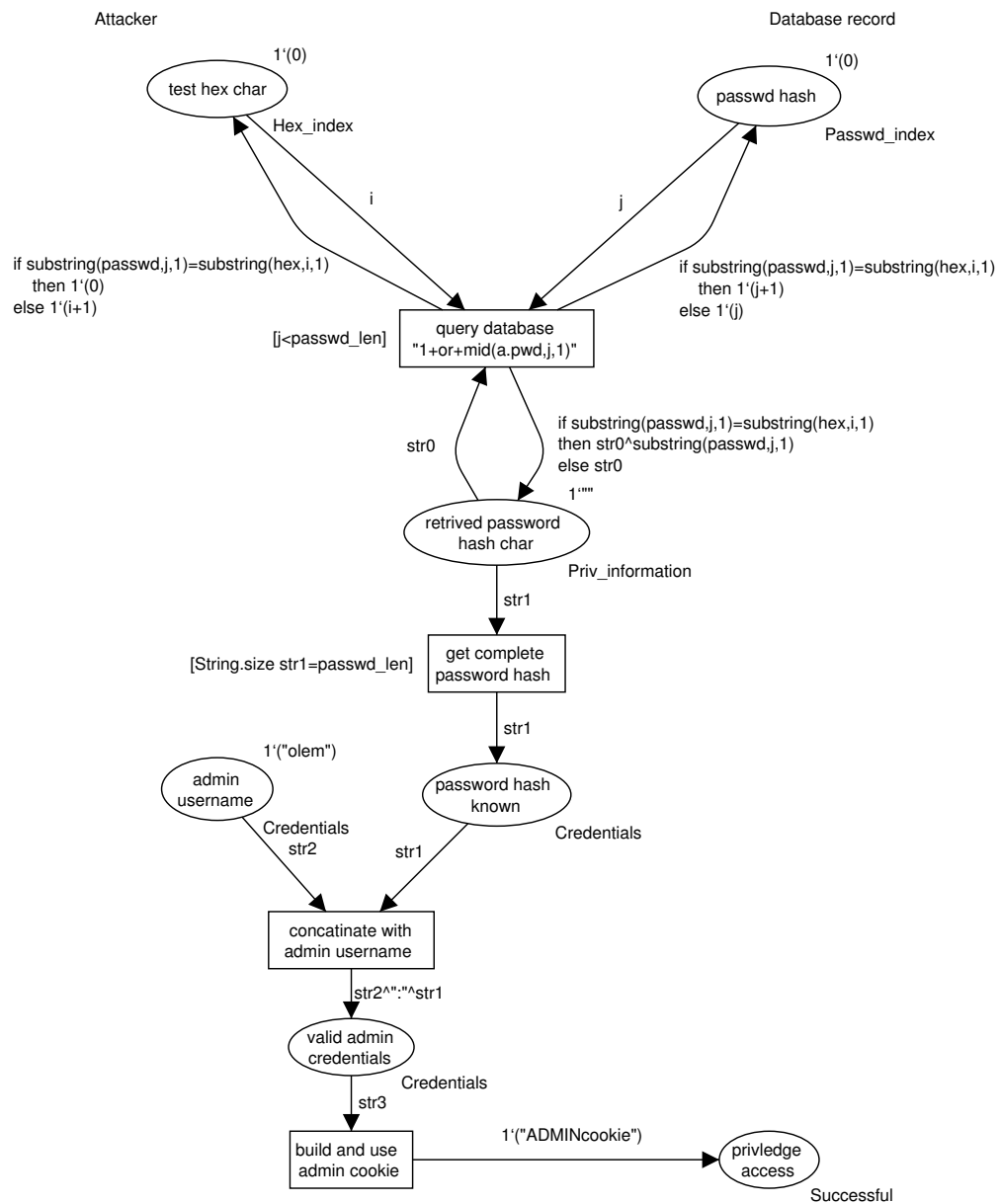

Figure 17: Scenario 1 – Coloured Petri net attack model

The global declarations of the attack model are as follows (see Appendix A for syntax explanation):

```
// type (colour sets) definitions
colour Hex_index = int;
```

```
colour Passwd_index = int;
colour Priv_information = string;
colour Credentials = string;
colour Successful = string;

// variable declarations
var str0 : Priv_information;
var str1, str2, str3 : Credentials;

// value declarations
val hex = "0123456789abcdef";
val passwd  = "b3414baa6a1890b23290cd2e2d267be"
```

The main working of the attack is presented by the four upper nodes in Figure 17, which is a conjunctive relationship. The place `test hex char` contain an index to a hexadecimal character the attacker whish to test, associated with the colour set `Hex_index` and an initial token with the colour 0 (an integer), which is an index to the hexadecimal character 0.

The place `passwd hash` is the password record in the database which the attack wish to read, its initial token is an index to the first character in the password. The attack starts by guessing the first character in the password.

The transition `query database` represent the SQL injection query where the attacker can use a specially crafted SQL statement to compare the hexadecimal character 0 with the first hexadecimal character in the secret password hash. If these characters are alike the attacker will get a positive result from the query, i.e. a result from the search engine, and know one character of the password hash. This is done by the arc expressions on the output arcs of `query database`, with a simple `if-then-else` control structure.

The place `retrived password char` with the colour set `Priv_information` contain an initial token with an empty string colour. If the guessed hex character is equal to the password character the token will be concatenated with the password character. As the attacker keep on guessing/trying hexadecimal characters the complete password hash slowly appear at the place `retrieved password char`. When the attacker has guessed all 32 hex characters (128 bit MD5 hash) password correct the guard at the transition `query database` will stop the transition firing.

The transition `get complete password` will fire when a token with the complete password string is at the place `retrieved password char`. Then when both the place `admin username` and `password hash` has a token, the two tokens, with the password and username as colour, are concatenated in the output arc of the transition `concatenate with admin username`.

Now we have valid admin credentials and can build the administrator cookie and use it in a web browser, i.e. reach the place `privilege access` with the colour set `Successful`.

Flaw Testing

Using the hypothesised coloured Petri net we can now execute an attack and verify the hypothesised flaw to achieve administrative privileges in the web application. Doing the same through a script or manually against the real web application produce the same

results as the hypothesised coloured Petri net attack.

Flaw generalisation

The nature of this flaw comes from bad or lacking input validation. This can probably also be found in other places in the application as well.

Another flaw or at least weakness is the way that the credentials are used in the web application. The use of MD5 hashing of the password has small effect since we can use this hash together with a username in a cookie. Another user's login cookie can be used by an adversary that can gain access without ever knowing the clear text password.

Flaw Elimination

The main flaw is the lack of input validation. All client input must be checked before it is allowed. Newer versions of the application have better input validation, so keeping updated are important.

Using a more secure type of authentication may be desirable. A keyed hash based MAC function in the client-server side authentication scheme, e.g. with a HMAC, can make the credential handling on the client more secure. Moreover a more secure authentication mechanism is called for.

6.1.1   Conclusions and Observations on Scenario 1

This scenario is not an advance attack, but it illustrates some of the benefits of using coloured Petri nets. The detail level necessary to illustrate the attack is available within the coloured Petri net semantics. Concurrency of the attackers actions and the attacked system are also shown. The tokens in the net make it easy to both build the attack and visualise how it should be done in live testing.

In this kind of attack timing is not an issue and there are not any difficult concurrent activities happing, which is where coloured Petri nets has additional advantages. The complexity of the attack alone is not too high, and can be done without attack modelling like this. However this scenario shows the working of a real attack inside a coloured Petri net. Because of the detail level, we clearly see the workings of the net. The execution of the net clearly show what actions the penetration testers have to do to reach the test goal. The net structure also opens for re-use in other penetration tests which require the same mechanisms or as a part of a larger attack model.

6.2   Scenario 2 – Consistency of Condition Checks

Consistency of condition checks or Time-of-Check,Time-Of-Use (TOCTOU) is making sure that conditions (or validations) that existed before a system call is called are the same when the system call is actually performed [43]. Lack of such checks may lead to race conditions [75]. When exploiting these types of attacks timing is important. The most common way of exploiting is automating an attack, run it multiple times, and adjusting timing to hit just the right interleaving.

On the UNIX platform these vulnerabilities appear frequently, the most common ones are symbolic link[5] attacks. These attacks are often used by attackers to gain or use other users privileges locally on a UNIX system. The typical TOCTOU vulnerability is a program

---

[5]A symbolic link is a file or directory that points to a different file or directory.

running EUID[6] of another user, preferably root, for the duration of the race condition exists.

This scenario show how we can use coloured Petri nets to model such an attack. This scenario is based on a historical scenario of a TOCTOU vulnerability, described by Bishop and Dilger [7]. This attack is a local attack on a vulnerable version of `passwd` in UNIX. `passwd` is a utility program in UNIX that allows someone to change a password entry, usually their own.

Goal Definition

The goal is to be able to login to the UNIX computer as another user in the system.

Information Gathering

We know that the `passwd` program is vulnerable to a symbolic link attack, e.g. through a Nessus vulnerability scan. A precondition for the attack is a penetration tester with local, nonprivileged shell access to the UNIX computer. This access could have been gained through another attack during a larger penetration test study. This is however irrelevant to this scenario, since its main mission is to show how to use timed coloured Petri net in an attack that requires timing of events.

Flaw Hypothesis

The underlying flaw hypothesis here is the ability to masquerade as another user in the system. Investigating the flawed program, `passwd`, the following processing steps can either be deducted or hypothesised:

1. Open the password file, read it, and retrieve the entry for the running user.

2. Create and open a temporary file (called `ptmp`) in the same directory as the password file.

3. Open the password file, copy its contents into `ptmp`, and update modified information.

4. Close password file and `ptmp`, then rename `ptmp` to be the password file.

The attacker will try to execute his/hers symbolic link attack between the processing steps. Since we have no way to cause the `passwd` program to stop between each step, we must time an attack. Modelling the `passwd` steps (transitions) in a coloured Petri net is straight forward, see the right part of Figure 18. A more detailed model of these process steps are of course possbile, but keeping the detail level sufficent and not to complex are benifical when modeling, as described in Section 5.3.3.

The attacker must do some initial preparation to exploit the vulnerability:

1. Create a password directory.

2. Create a `.rhost`[7] file in that directory.

3. Insert login credentials into `.rhost`

4. Make a symbolic link that links to the password directory.

---

[6]Effective User Identification, e.g. the `passwd` process runs with real-UID = attacker and effective-UID = root so it will be able to access `/etc/passwd` and change the password for the user "attacker" when "attacker" runs the `passwd` program.

[7]`rlogin` is a UNIX software utility that allows users to log in on another host via a network, the `.rhost` file, located in each system users' home directory, contain the login credentials of a spesific `rlogin` user.

These steps are just done for preparation and can be modelled as one transition in the coloured Petri net, with two conjunctive places as input, which is the fake login credentials and shell access to the computer.

Now we hypothesis some key places and transitions for the attackers actions:

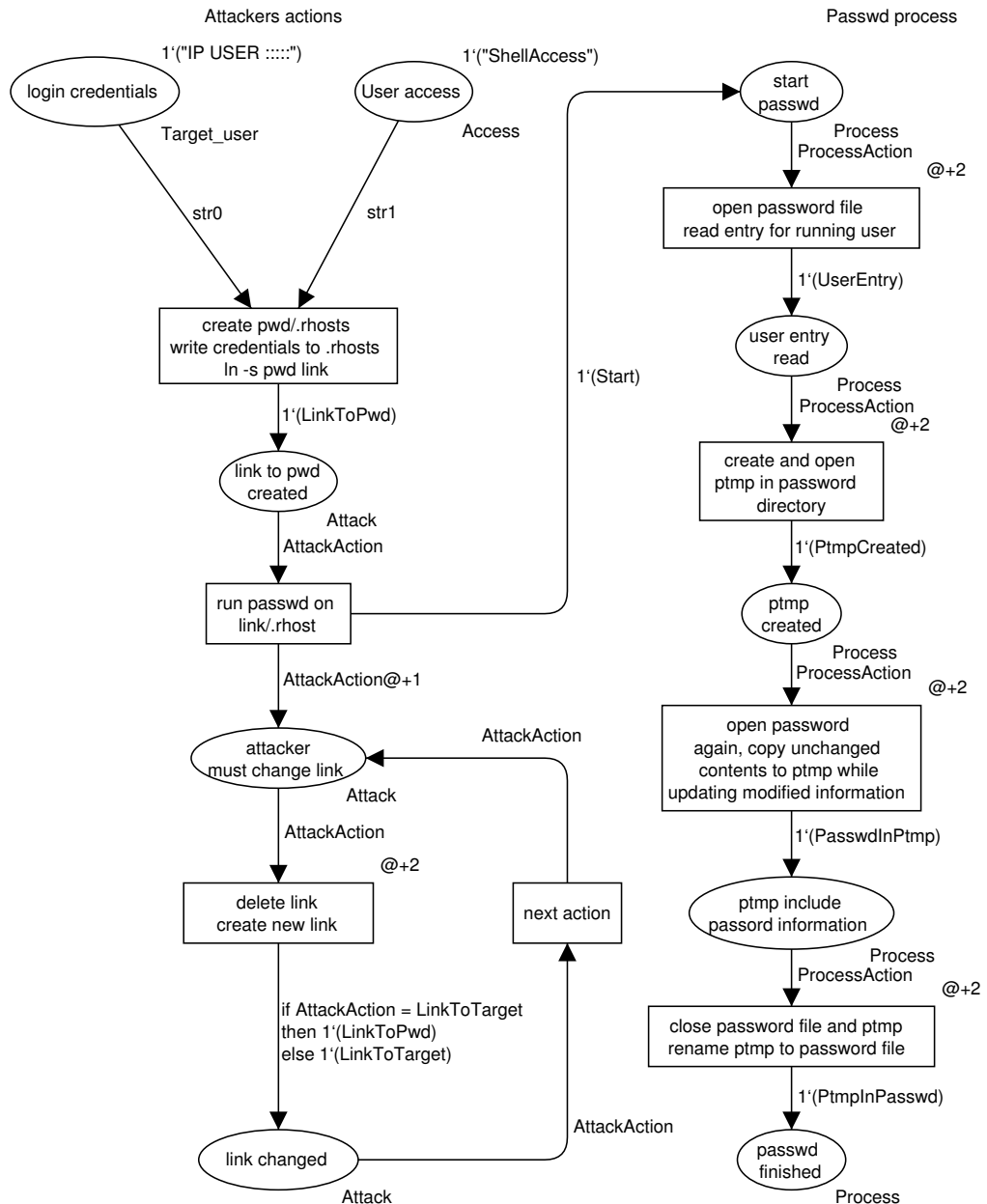| Key places:	Key transitions:
`attacker must change link`	`delete link and create new link`
`link changed`	

Figure 18: Scenario 2 – Timed coloured Petri net attack model

43

Figure 18 show the complete attack model, which use the declarations below (see Appendix A for syntax explanation):

```
// type (colour sets) definitions
colour Target_user = string;
colour Access = string;
colour Attack = with LinkToTarget | LinkToPwd timed;
colour Process = with Start | UserEntry | PtmpCreated |
PasswdInPtmp | PtmpInPasswd timed;

// variable declarations
var str0 : Target_user;
var str1 : Access;
var AttackAction : Attack;
var ProcessAction : Process;
```

There are two colour sets in the coloured Petri net which are timed, `Attack` and `Process`. Both colour sets use the `with` syntax where we can define a list of colours to be used. The symbolic link state are represented by the colour set `Attack`, while the system state is represented by the colour set `Process`. For illustrative purposes, a simplified assumption of the net in Figure 18 is that each transition or step in the `passwd` process consume the same amount of CPU time. The actions of the attacker are then modelled as a parallel net within the same time window.

The TOCTOU or symbolic link attack starts by starting the `passwd` process on the link to the attackers `pwd` directory. Then when the process action has a token with the colour `UserEntry` at the place `user entry read` the attacker must change the symbolic link to point to the target directory. Since the net are timed the token coloured `UserEntry` will have a time stamp of 2 and the token coloured `LinkToPwd` at the place `attacker must change link` have a time stamp of 1. This happens because the input arc of the place `attacker must change link` adds 1 time unit and the transition `open password file read entry for running user` uses 2 time units. The differences in time stamps imply that the token with lowest time stamp will fire first, thus the transition `delete link create new link` will be enabled first. This is how the timed coloured Petri net continues, the model remain at a specific model time until no more transitions are enabled at the current model time. The model time increases as the tokens change time stamp. The attacker must change the symbolic link between every transition in the `passwd` process. This attack will end when the model time has reached 7. Then the `passwd` process is finished and the attacker has successfully changed the target users `.rhost` file by exploiting the lack of proper condition checks in `passwd`.

Flaw Testing

With the attack model, we can easily build attack code that uses the same interleavings that are necessary to reach the time windows. Firstly we have to know how much time the `passwd` process consumes and then we can make a simple attack script that use the time interleaving from the attack model. It will of course be necessary with some minor manual time justifications in the exploit code.

When the `.rhost` file is successfully changed the attacker can log in as the target user to the UNIX machine, thus the goal is reached.

Flaw Generalisation

This vulnerability comes from a typical TOCTOU flawed UNIX program. Similar symbolic link attacks appear in many programs in UNIX systems. The model can be used in other similar TOCTOU flaws, with only small justifications.

In this attack we focused on writing over some other user's `.rhost` file so we could login as that user. The vulnerability could be used just as easily to write over some other system file.

Flaw Elimination

Updating the `passwd` process will eliminate this specific vulnerability. To help avoid similar TOCTOU problems in other applications we can try to avoid file system calls that takes a filename as input, instead use file handler or a file descriptor.

The main recommendation for avoiding this kind of local vulnerabilities is to keep software updated and properly patched at all time, especially programs that run with EUID as another user.

6.2.1   Conclusions and Observations on Scenario 2

This local penetration testing exercise demonstrate the possibilities and limitations of modelling timed attacks using timed coloured Petri nets. The complexity of such attacks done manually can become difficult to control if several conditions must be true at one time, as can be seen clearly in the attack described here, even though only two main concurrent processes were present. Other, similar attacks may have three or more concurrent processes, e.g. attacker actions, a process that writes to a file, and another that reads the file. Modelling such attacks in a timed coloured Petri net helps the penetration test to become successful without too many manual and time-consuming operations and interventions.

A problem, however, is obvious when vulnerabilities like this appear in a distributed environment, i.e. not on a single computer where one may use or at least model using a single global time. For distributed systems, there not only is no such global time (at most a partial order), but also a number of variations owing e.g. to load and latency conditions within the distributed system. In such an environment, timed coloured Petri nets are inadequate since one must take the uncertainties immanent in the model into account both for the modelling and the simulation/execution stages.

The following scenario therefore demonstrates the use of interval-timed coloured Petri nets.

6.3   Scenario 3 – Race Condition

This is a hypothetical scenario that illustrate the use of interval timed coloured Petri nets for modelling and execution of a race condition-type attack. This type of attack is not to often exploited by attackers or penetration testers, but similar vulnerabilities exist. Application systems constructed in the form of multi-tiered architectures can have race conditions just as any other application, but they can be very difficult to exploit remotely in an effective manner.

The attack scenario is an attack on a distributed system, i.e. a multi-tier e-commerce system, where time is not fully synchronised and an unintended race condition appear. We assume a three-tier electronic commerce site, with a back end database system and an intermediate application layer, running on multiprocessor or at least a multithreading

system. Figure 19 shows the multi-tiered system, which has a design similar to many large scale e-commerce systems in use today. There are several distributed processes that can read and write to a database in the database back end. These processes write and update different records in the database tables, some times even to the same records concurrently. This require a locking mechanism, so that a process should not be able to read or write to a spesific database record at the same time that another process writes to this record, e.g. with row locking or table locking for exclusive writing [11].
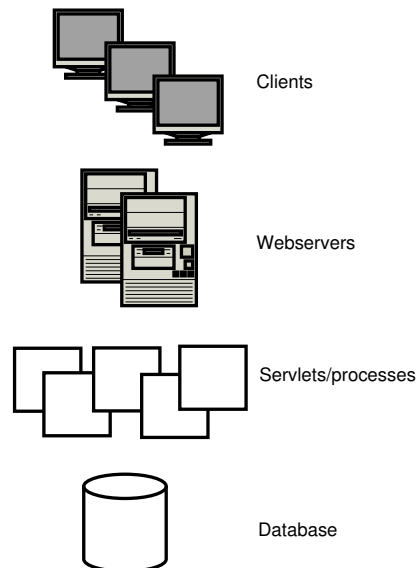
Figure 19: Multi tier e-commerce system

Goal Definition

The objective of the penetration test is to modify back end data, specifically modifying order data of another customer's order.

Information Gathering

It has been ascertained that all connections to the application layer are secured via SSL/TLS [16, 60], with server certificate authentication and username and password client authentication, i.e. without mutual SSL/TLS authentication on the transport layer. This approach is used by almost every e-commerce site today. A strong perimeter defence, e.g. a firewall mechanism, separates the presentation layer (client side) from the application and back end layers effectively.

We assume that an attacker is able to obtain data from customer identification cookies from legimate customers, as is commonly accomplished through e.g. cross-site scripting [10]. The penetration tester has, moreover, also identified a construction rule for legimate transaction identifiers, in many electronic commerce applications these are simple sequence numbers and therefore trivial to predict. This is assumed done before or as a former part of this penetration test.

The penetration tester can either through deduction from existing commerce back ends or study of design documentation identify the final step in performing a purchase, that is the confirmation steps on the part of the customer. These steps are done after a

customer has signed in, been authenticated onto the commerce site, selected the merchandise, and confirmed shipping and payment details by communicating with both internal and external databases and services.

When the customer has confirmed his order, the final confirmation steps in the system are done, without customer interaction, as follows:

1. *Inventory update*. In this step, an update of the inventory database table is performed.

2. *Payment confirmation*. A final confirmation of customer credit standing is sent to the banking service, e.g. confirming withdrawal from already reserved credit. Then the payment is recorded in the local credit database table.

3. *Shipping registration*. The shipping details are written to the shipping information database table.

4. *Order fulfilment*. Based on the previous information, the shipping process pickup data and shipping addresses are collated and sent to the shipping agent.

Flaw Hypothesis

The operative flaw hypothesis is that developers have, in order to obtain maximum throughput, in the various databases accesses during the course of the transaction, are performed with locking at each stage, but not across multiple stages of the above transaction.

An attack is hypothesised where an authenticated attacker is able to inject data with valid customer and transaction identification into the application layer, to perform modification to the shipping information database table after the completion of the third stage and prior to the commencement of the fourth stage, resulting in goods being diverted to an attacker's address of choice. In this attack scenario it is not necessary for the attacker to learn payment information details of a customer, or to compromise the back end. Thus the hypothetical flaw lies in the application logic and its implementation in accessing the database back end layer. The attacker will simply inject a database write with customer identification, transaction identification and shipping details just after the system does this, and by this changes the address that the shipping department will read for finally shipping of the order.

We assume that the attacker has an injection channel (modelled separately) into the database communication between step three and four. The SQL injection itself are not modelled in this hypothetical scenario, the lack of input validation that the injection is based in can for example be similar to the flaw exploited in scenario 1 (see Section 6.1). The channel can be a specially crafted (timing independent) HTTP POST request [20] that utilize the information gained from victim customer. Since the attacker is authenticated as a valid customer to the SSL/TLS application, no traffic protection or perimeter defence will be likely to see this injection as a malicious act.

The interval timed coloured Petri net attack model will focus primary on the application layer and with the timing dependency on the progress through the four confirmation steps.

Each of the four final confirmation steps, as described above, is framed by a beginning and ending transition associated with a time interval, along with similar time interval for intermediate processing between the steps, e.g. simulating how much time it can take to

acquire a table or row lock.

We hypothesis the following places and transitions for the attack:

Key places:	Key transitions:

`confirmed order`
`inventory table`
`credit table`
`shipping table`
`shipping department`
	`injection channel`
	`request inventory update`
	`update inventory`
	`request payment fulfilment`
	`write payment`
	`request shipping registration`
	`write shipping`
	`request fulfilment`
	`read shipping details`

The attack model is shown in Figure 20, that uses the global declarations as follows (see Appendix A for syntax explanation):

```
// type (colour sets) definitions
colour UID = int; // User identification
colour TXID = int; // Transaction identification
colour Prod= string;
colour Price = string;
colour Addr = string;
colour Inventory = product UID*TXID*Prod timed;
colour Credit = product UID*TXID*Price timed;
colour Shipping = product UID*TXID*Addr timed;
colour SendShipping = product UID*TXID timed;
colour Order = product UID*TXID*Prod*Price*Addr timed;
colour DBres = string;
colour Lock = string timed;

// variable declarations
var k,m : UID;
var l,n : TXID;
var str0 : Prod;
var str1 : Price;
var str2 : Addr;
var lck : Lock;
var res : DBres;
```

This interval timed coloured Petri net uses a product colour sets [31] (`Inventory`, `Credit`, `Shipping`, `SendShipping`, `Order`), which combines other colour sets to represent the information written to the back end database. These product colour sets and the colour set `Lock` are timed.

Prior to the attack steps shown in the attack model in Figure 20, the customer has finished and confirmed his order. The attack models the system confirmation steps in parallel with the attacker's injection action. The place `confirmed order` represents the system state where a customer has confirmed his order in the presentation layer. From this point in time the penetration tester must time his injection. The timing will vary because of the distributed nature of the system, hence the interval timing. The first transaction, `req inventory update`, will update the inventory table in the back end database. The entire
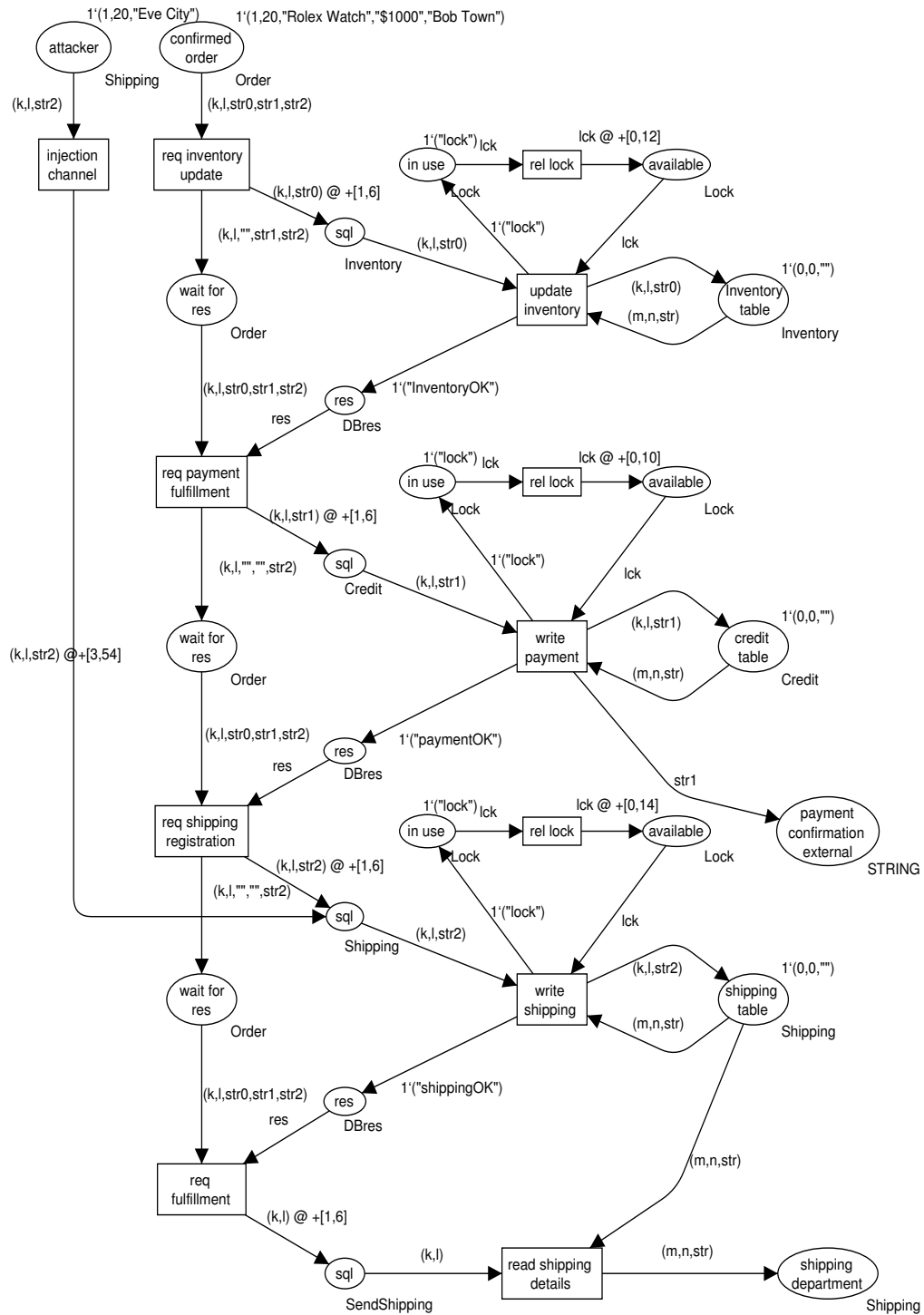
Figure 20: Scenario 3 – Interval timed coloured Petri net attack model

transaction is modelled to consume time, which is between 1 and 6 ([a, b]) time units (here: milliseconds). The minimum time, 1, represents how long the entire transaction may take if there are no delays. Acquiring the database table lock is modelled to take

between 0 and 12 ($[c, d]$) time units, depending on database load. The minimum time of 0 is chosen because the lock may be available immediately for the transaction `update inventory`. The transaction will not fire before there is at least one token in all input places. The fuzzy timing is represented by closed intervals in the interval timed coloured Petri net. The first confirmation step – inventory update of a customer order – can by this use all between 1 and 18 ($[a + c, b + d]$) time units in the attack model.

The second confirmation step, payment confirmation, is modelled similarly to the former. However, this step uses a slightly different timing, simulating that the loads on the credit database table are different from the more heavily loaded inventory table. At the same time as the payment is written to the credit table, a payment confirmation is sent to the external financial service, confirming credit withdrawal. In the next confirmation step the address is written to the shipping database table. It is here that the race condition the penetration tester is trying to exploit appears. The `req fulfillment` transition will wait for the results from the database table write to finish, and then the shipping fulfilment will read from the shipping table and finish the order. The transaction `read shipping details` will in most database systems acquire a shared read lock that prevents the row or table to be updated at the same time as another client attempts to read. However, because of the time window that appear between the write to the database table and the read from this database table by the two last fulfilment processes (and an imprudent lapse of the surrounding lock), the attacker can gain a exclusive write lock before the transaction `read shipping details` commences reading.

Flaw Testing

Immediately after the customer has placed his order, the attacker will inject the changed shipping address, leaving the remaining data untouched. The available time window for the attack appears after the shipping details has been written to the database shipping table and the final fulfilment step is initiated. Thus, if the attacker can inject a new record into the shipping table before the transaction `read shipping details` is fired, the attack is successful. By calculating the minimum and maximum time this could take, we can obtain an interval where the injection is most likely to be successful. This interval must start from the starting point, which is from where the customer confirms his order in the presentation layer, to the maximum time it can take for the customers shipping details to be written to the database table. In the attack model this time interval can be calculated to be in the closed time interval $[3, 54]$. In this time interval the attack will be most likely to succeed. This means however that the attack not always will succeed, but it is likely to succeed after a small number of (automated) attempts.

Flaw Generalisation

The main flaw hypothesis investigated here is the race condition situation between the confirmation steps. The small time window which appears between the release of a lock and the next confirmation step make the injection possible. Another underlying flaw in the system that becomes clear from this penetration test is the weak user identification mechanism. The system design has weaknesses in separating user that has been authenticated against the server by username and password and subsequently through a cookie mechanism. The internal processes assume that a message with specific customer identification always comes from the correct client. This assumption can be imprudent, especially when the transaction identification can be intercepted or guessed by an ad-

versary, meaning that an adversary may have all information that identifies an order in the back end database. The flaw of being able to guess transient transaction identifiers in combination with the customer identifiers make this vulnerability (present in quite a few shopping-cart type applications) dangerous.

Flaw Elimination

If the system had not released the database table or row lock between each stage in the final confirmation steps, the attacker would not be capable of injecting arbitrary information and thereby change a customer's order that has already been properly approved by the customer, application and external systems. However, for performance reasons, and because of the distributed nature of the system, such locks are frequently not used in such a conservative way. A downgrade from the exclusive write lock to a read lock could help in the two last steps of the confirmation transaction, but this is again difficult to realise in a distributed environment. In addition, the elimination of all HTTP POST injections would also stop this attack, but in complex multi tier e-commerce systems like this, it is difficult to prevent all such injection possibilities and may again result in significant performance penalties.

6.3.1 Conclusions and Observations on Scenario 3

This attack scenario has demonstrated the use of an interval timed coloured Petri net for modelling a moderately complex time-dependent vulnerability and exploits of such vulnerabilities. Even in such a limited scenario, however, the benefits of a more rigid modelling approach and the ability to perform a round-trip modelling, simulation, and execution flaw hypothesis penetration test have become apparent.

This attack show that the combination of many system flaws can lead to a dangerous system vulnerabilty. The principal system flaws are exploited:

- weak customer identification and authentication within the system

- easy to guess and predict transaction identificators

- SQL injection flaw through the presentation layer

- race condition between internal processes, because of the lock releasing between final confirmation steps.

The flaw hypothesis methodology together with the interval timed coloured Petri net attack model make this attack easier to understand, visualise and apply in a real world scenario. The closed intervals add a nice feature to coloured Petri nets that can be used when timing is necessary in penetration testing. However getting the intervals right according to the attacked system can be difficult. Without thorough knowledge of the e-commerce system attacked in the above example the timing can be difficult to know correctly. Some time benchmarking of the system before modelling may be necessary.

7   Conclusion

Penetration testing tools and high-level penetration testing mehtodologies alone do not solve the problem of finding all types of computer attacks in our modern complex information systems, thus the ingenuity of the penetraiton testers are important. This thesis has described a mechanism for the modelling, partial analysis, and automatic execution of attacks, based on different types of coloured Petri nets. This has been done together with a penetration testing model, the flaw hypothesis methodology, that together provide a detailed way of hypothesising and modelling system attacks in penetration testing. The coloured Petri nets provide a detail level that makes modelling of multi-stage and multi-agent attacks with and without time constraints possible. This help penetration testers to model attacks that are mentally difficult to hypothesis and visualise. Using the coloured Petri net attack models togheter with the flaw hypothesis methodology make it possible to model custom attack scenarios graphically before attacks are tested live, and by this helping out with designing attack hypotheses and the difficult step of mapping attack hypotheses into real system attacks. Therefore the colured Petri nets can be a valuable modelling tool for the penetration tester.

Coloured Petri nets have been found to have many features of modelling that adapt well into attack modelling. Coloured tokens and Petri net structure can model concurrent actions. With the use of guards, arc expressions, timing, and conjunctive and disjunctive relationships we can easily control the progress of an attack with tokens and thereby model and simulate complex attack scenarios before they are tested live. In network-based environments, where TOCTOU- and race condition- type attacks are difficult to identify, interval timed coloured Petri nets provide a mechanism to model the fuzziness in timing that appear in such distributed environments.

Coloured Petri nets are very powerful in modelling computer attacks, but they can be complicated. A penetration tester must not only have adequate knowledge in the penetration testing domain, but also have knowledge in modelling coloured Petri nets. However with the help of coloured Petri net design tools and a methodical way of developing the nets in the penetration testing process, this limitation should be outweighed by the coloured Petri nets ability to model complex attack scenarios.

This thesis do not conclude the research on the coloured Petri net penetration testing approach, further research and refinement are therefore called for.

8 Future Work

The chosen topics, penetration testing and Petri nets, are both large areas of research and much have been done in both categorise separately. However using both areas together have not been greatly explored yet. Much of the research done in the area of coloured Petri nets can refine the results from this study and may be used in other areas of computer security.

Following this thesis, the next step should be large-scale live penetration testing or quantitative penetration testing with the proposed approach. The results from this could be further refinement of this thesis work. This study could also introduce the use of hierarchical coloured Petri nets for large scale attack models. New theory maybe necessary, e.g. with the use of other research results available in Petri net research, to further adapt the approach. This can be the next step in proving or rejecting the proposals done in this thesis.

Another area of research is to make a Petri net application tool designed for the penetration tester. This tool could be build with basis in the CPNtools[1] and the standard Petri net markup language [5]. The main feature missing in the tools available today are the support for interval timed coloured Petri nets.

Proposing a structured method for building attack scripts from coloured Petri net attack models are also open for research. In other areas of Petri net usage such mechanisms do exist, e.g. looking into Mortensen work on automatic code generation from coloured Petri nets [50].

State space analysis methods [30] for attack models in coloured Petri nets are also a possible next step from this thesis work. However the need for such thorough analysis in attack models used in Penetration testing are not entirely clear at this point.

[1] http://wiki.daimi.au.dk/cpntools/cpntools.wiki (Visited June 2005)

Bibliography

[1] SANS Institute 2001. Guidelines for Developing Penetration Rules of Behavior. `http://www.sans.org/rr/papers/42/259.pdf` Visited May 2005, 2001.

[2] Brad Arkin, Scott Stender, and Gary McGraw. Software Penetration Testing. *IEEE Security & Privacy*, 3(1):84–87, January/February 2005.

[3] Clement R. Attanasio, Peter W. Markstein, and Ray J. Phillips. Penetrating an operating system: a study of VM/370 integrity. *IBM Systems Journal*, 15(1):102–116, 1976.

[4] Michel Beaudouin-Lafon, Wendy E. Mackay, Peter Anderson, Paul Janecek, Mads Jensen, Michael Lassen, Kasper Lund, Kjeld Mortensen, Stephanie Munck, Anne Ratzer, Kartrine Ravn, Søren Christensen, and Kurt Jensen. CPN/Tools: A Post-WIMP Interface for Editing and Simulating Coloured Petri Nets. In J.-M. Colom and M. Koutny, editors, *Proceedings of the 22nd International Petri Net Conference (ICATPN 2001)*, volume 2075 of *Lecture Notes in Computer Science*, pages 71–80, Newcastle upon Tyne, England, June 2001. Springer-Verlag.

[5] Jonathan Billington, Søren Christensen, Kees van Hee, Ekkart Kindler, Olaf Kummer, Laure Petrucci, Reinier Post, Christian Stehno, and Michael Weber. The Petri Net Markup Language: Concepts, Technology, and Tools. In W.v.d. Aalst and E. Best, editors, *Proceedings of the 24th International Conference on the Application and Theory of Petri Nets (ICATPN 2003)*, volume 2679 of *Lecture Notes in Computer Science*, pages 483–505, Eindhoven, The Netherlands, August 2003. Springer-Verlag.

[6] Matt Bishop. *Computer Security Art and Science*. Addison Wesley, 2003.

[7] Matt Bishop and Michael Dilger. Checking for Race Conditions in File Accesses. *Computing Systems*, 9(2):131–152, 1996.

[8] Rahmat Budiarto, Sureswaran Ramadas, Azman Samsudin, and Salah Noor. Development of penetration testing model for increasing network security. In *Information and Communication Technologies: From Theory to Applications*, pages 563–564, April 2004.

[9] Vadim Bulitko and David Wilkins. Real-time Decision Making For Shipboard Damage Control. In *Proceedings of the AAAI/KDD/UAI-2002 Joint Workshop on Real-Time Decision Support and Diagnosis Systems*, pages 37–46. AAAI Press, 2002.

[10] CGI Security. The Cross Site Scripting FAQ. `http://www.cgisecurity.com/articles/xss-faq.shtml` Visited June 2005, August 2003.

[11] Thomas Connolly and Carolyn Begg. *Database Systems: A Practical Approach to Design, Implentation, and Management*. Addison Wesley, 3 edition, 2002.

[12] Ole Martin Dahl and Stephen D. Wolthusen. Modeling and Execution of Complex Attack Scenarios using Interval Timed Colored Petri Nets. In *ACSAC '05: Proceedings of the 21st Annual Computer Security Applications Conference*, Tucson, Arizona, USA, December 2005. Submitted for publication.

[13] Kristopher Daley, Ryan Larson, and Jerald Dawkins. A Structural Framework for Modeling Multi-Stage Network Attacks. In *Proceedings of the International Conference on Parallel Processing Workshops (ICPPW'02)*, pages 1530–2016, 2000.

[14] Tina Darmohray. hacking for fun and profit. *;login*, April 2003.

[15] Renaud Deraison, Noam Rathaus, HD Moore, Raven Alder, George Theall, Andy Johnston, and Jimmy Alderson. *Nessus Network Auditing*. Syngress, Rockland, MA, USA, 2004.

[16] Tim Dierks and Christopher Allen. The TLS Protocol Version 1.0. RFC 2246 (Proposed Standard), January 1999. Updated by RFC 3546.

[17] Computer Security Division. NIST Special Publication 800-42: Guideline on Network Security Testing, August 2003.

[18] Deborah D. Downs and Ranwa Haddad. Penetration Testing – The Gold Standard for Security Rating and Ranking. In *Workshop on Information-Security-System Rating and Ranking*, March 2001.

[19] Dan Farmer and Wietse Venema. Improving Security of Your Site by Breaking Into It. http://www.alw.nih.gov/Security/Docs/admin-guide-to-cracking.101.html Visited May 2005, December 1993. Visited 2005.

[20] Roy T. Fielding, James Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul J. Leach, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Updated by RFC 2817.

[21] Jane Frankland. Automated Penetration Testing – False Sense of Security. *eBCVG IT Security*, August 2004.

[22] Martin Freiss. *Protecting Networks with Satan*. O'Reilly Press, 1997.

[23] Simson Garfinkel. Under Attack: Can your systems really benefit from penetration testing? http://www.csoonline.com/read/100103/machine.html Visited May 2005, October 2003. Visited 2004.

[24] Daniel Geer and John Harthorne. Penetration Testing: A Duet. In *Proceedings of the 18th Annual Computer Security Applications Conference (ACSAC 2002)*, pages 185–198, Las Vegas, NV, USA, December 2002. IEEE Press.

[25] Sarbari Gupta and Virgil D. Gligor. Towards a Theory of Penetration-Resistant Systems and Its Applications. *Journal of Computer Security*, 1(2):133–158, 1992.

[26] Guy Helmer, Johnny Wong, Mark Slagell, Vasant Honavar, Les Miller, Yanxin Wang, and Robyn Lutz. Software Fault Tree and Colored Petri Net Based Specification, Design and Impementation of Agent-Based Intrusion Detection. Technical report, Iowa State University, Ames, IA, USA, June 2002.

[27] Pete Herzog. *OSSTMM 2.1.: Open-Source Security Testing Methology Manual*. ISECOM Institute for Security and Open Methologies, August 2003.

[28] Sverre H. Huseby. *Innocent Code: A Security Wake-Up Call for Web Programmers*. Wiley, 2004.

[29] Cynthia E. Irvine. Security: Where Testing Fails. *ITEA Journal*, pages 53–57, June 2000.

[30] Kurt Jensen. *Coloured Petri Nets: Analysis Methods (Volume 2)*. Monographs in Theoretical Computer Science. Springer Verlag, Heidelberg, Germany, 1997.

[31] Kurt Jensen. *Coloured Petri Nets: Basic Concepts (Volume 1)*. Monographs in Theoretical Computer Science. Springer Verlag, Heidelberg, Germany, 1997.

[32] Kurt Jensen. *Coloured Petri Nets: Practical Use (Volume 3)*. Monographs in Theoretical Computer Science. Springer Verlag, Heidelberg, Germany, 1997.

[33] Paul A. Karger and Roger R. Schell. Multics Security Evaluation: Vulnerability Analysis. Technical Report ESD-TR-74-193, Vol. II, Information Systems Technology Application Office, Deputy for Command and Management Systems, Electronic Systems Division (AFSC), L. G. Hanscom AFB, MA, USA, June 1974.

[34] Paul A. Karger and Roger R. Schell. Thirty Years Later: Lessons from the Multics Security Evaluation. In *ACSAC '02: Proceedings of the 18th Annual Computer Security Applications Conference*, Washington, DC, USA, 2002. IEEE Computer Society.

[35] Thomas J. Klevinsky, Scott Laliberte, and Ajay Gupta. *Hack I.T.: Security Through Penetration Testing*. Addison-Wesley Professional, February 2002.

[36] Lars M. Kristensen, Søren Christensen, and Kurt Jensen. The practitioner's guide to coloured Petri nets. *International Journal on Software Tools for Technology Transfer (STTT)*, 2(2):98–132, December 1998.

[37] Sandeep Kumar and Eugene H. Spafford. A Pattern-Matching Model for Instrusion Detection. In *In Proceedings of the National Computer Security Conference*, pages 11–21, Baltimore, MD, USA, October 1994.

[38] George Kurtz and Chris Prosise. Penetration Testing Exposed. *Information Security Magazine*, September 2000. `www.infosecuritymag.com` Visited May 2005.

[39] Kevin Lam, David LeBlanc, and Ben Smith. *Assessing Network Security*. Microsoft Press, 2004.

[40] W. Lee, D. Grosh, and F. Tillman. Fault tree analysis, methods, and applications. *IEEE Transactions on Reliability*, 34(3):194–203, August 1985.

[41] Woo Jin Lee, Sung Deok Cha, and Yong Rae Kwon. Integration and analysis of use cases using modular Petri nets in requirements engineering. *IEEE Transactions on Software Engineering*, 24(12):1115–1130, December 1998.

[42] Richard R. Linde. Operating System Penetration. In *Proceedings of the AFIPS National Computer Conference*, volume 44 of *AFIPS Conference Proceedings*, pages 361–368, Anaheim, CA, USA, May 1975. AFIPS Press.

[43] Daniel Lowry Lough. *A Taxonomy of Computer Attacks with Application to Wireless Networks*. Doctor of Philosophy in Computer Engineering, Faculty of the Virginia Polytechnic Institute and State University, Blacksburg, Virginia, April 2001.

[44] Stuart McClure, Joel Scambray, and George Kurtz. *Hacking Exposed: Network Security Secrets & Solutions, Fourth Edition*. McGraw-Hill Osborne Media, 4 edition, February 2003.

[45] John P. McDermott. Attack net penetration testing. In *Proceedings of the 2000 workshop on New security paradigms*, pages 15–21. ACM Press, October 2000.

[46] Paul Midian. Perspectives on Penetration Testing – Black Box vs. White Box. *Network Security*, 2002(11):10–12, November 2002.

[47] Paul Midian. An Insight White Paper – Penetration Testing. *Insight Consulting*, January 2004. `www.insight.co.uk` Visited May 2005.

[48] Fredrik Moberg. Security analysis of an information system using an attack tree-based methodology. Master's thesis, Chalmers University of Technology, November 2000.

[49] Andrew P. Moore, Robert J. Ellison, and Richard C. Linger. Attack Modeling for Information Security and Survivability. Technical report, CMU/SEI-2001-TN-001, Carnegie Mellon Universtiy, March 2001.

[50] Kjeld H. Mortensen. Automatic Code Generation Method Based on Coloured Petri Net Models Applied on an Access Control System. In M. Nielsen and D. Simpson, editors, *Proceedings of Application and Theory of Petri Nets 2000: 21st International Conference*, volume 1825 of *Lecture Notes in Computer Science*, Aarhus, Denmark, June 2000. Springer-Verlag.

[51] Kjeld H. Mortensen, Søren Christensen, Lars M. Kristensen, , and Jan S. Thomasen. Capacity Planning of Web Servers using Timed Hierarchical Coloured Petri Nets. In *In Proceedings of HP Openview University Association (HP-OVUA'99) 6th Plenary Workshop*, 1999.

[52] Tadao Murata. Petri Nets: Properties, Analysis and Applications. In *Proceedings of the IEEE*, volume 77, pages 541–580, April 1989.

[53] NASA. Fault Tree Analysis: A Bibliography, July 2000.

[54] Andrew Odlyzko. Economics, Psychology, and Sociology of Security. In *Proceedings of the 7th International Conference of Financial Cryptography*, volume 2742 of *Lecture Notes in Computer Science*, pages 182–189. Springer-Verlag, 2003.

[55] James Lyle Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981.

[56] Carl A. Petri. *Kommunikation mit Automaten*. PhD thesis, Technische Hochschule Darmstadt, Darmstadt, Germany, 1962. Published in Schriften des Instituts für Instrumentelle Mathematik 3,1-128, University of Bonn, Germany.

[57] Cynthia Phillips and Laura Painton Swiler. A graph-based system for network-vulnerability analysis. In *NSPW '98: Proceedings of the 1998 workshop on New security paradigms*, pages 71–79, Charlottesville, Virginia, United States, 1998. ACM Press.

[58] Anne Vinter Ratzer, Lisa Wells, Henry Michael Lassen, Mads Laursen, Jacob Frank Qvortrup, Martin Stig Stissing, Michael Westergaard, Søren Christensen, and Kurt Jensen. CPN Tools for Editing, Simulating and Analysing Coloured Petri Nets. In W.v.d. Aalst and E. Best, editors, *Proceedings of the 24th International Conference on the Application and Theory of Petri Nets (ICATPN 2003)*, volume 2679 of *Lecture Notes in Computer Science*, pages 450–462, Eindhoven, The Netherlands, June 2003. Springer-Verlag.

[59] Wolfgang Reisig. *Petri Nets: An Introduction*. Monographs in Theoretical Computer Science. Springer Verlag, Heidelberg, Germany, 1985.

[60] Eric Rescorla. HTTP Over TLS. RFC 2818 (Informational), May 2000.

[61] Bruce Schneier. Attack Trees. *Dr. Dobb's Journal*, 24(12):21–29, December 1999.

[62] Bruce Schneier. *Secrets and Lies, Digital Security in a Networked World*. Wiley Computer Publishing, 2000.

[63] Robert W. Shirey. Internet Security Glossary. RFC 2828 (Informational), May 2000.

[64] Bryan Smith, William Yurcik, and David Doss. Ethical Hacking: The Security Justification Redux. In *IEEE International Symposium on Technology and Society (ISTAS)*, Raleigh NC USA, June 2002.

[65] Jan Steffan and Markus Schumacher. Collaborative attack modeling. In *SAC '02: Proceedings of the 2002 ACM symposium on Applied computing*, pages 253–259. ACM Press, March 2002.

[66] Peter Stephenson. Modeling of Post-Incident Root Cause Analysis. *International Journal of Digital Evidence*, 2(2), 2003.

[67] Peter Stephenson. The Application of Formal Methods to Root Cause Analysis 0f Digital Incidents. *International Journal of Digital Evidence*, 3(1), 2004.

[68] Frank Swiderski and Window Snyder. *Threat Modeling*. Microsoft Press, 2004.

[69] Herbert H. Thompson. Application Penetration Testing. *IEEE Security & Privacy*, 3(1):66–69, January/February 2005.

[70] Terry Tidwell, Ryan Larson, Kenneth Fitch, and John Hale. Modeling Internet Attacks. In *Proceedings of the 2001 IEEE Workshop on Information Assurance and Security*, United States Military Academy, West Point, NY, June 2001.

[71] Mahesh V. Tripunitara and Partha Dutta. Security Assessment of IP-Based Networks: A Holistic Approach. In *Ninth Annual Conference of the Internet Society (INET'99)*, San Jose, CA, June 1999.

[72] Wil M. P. van der Aalst. Interval Timed Coloured Petri Nets and their Analysis. In Marco Ajmone Marsan, editor, *Proceedings of Application and Theory of Petri Nets 1993, 14th International Conference*, volume 691 of *Lecture Notes in Computer Science*, pages 453–472, Chicago, IL, USA, June 1993. Springer–Verlag.

[73] Wil M. P. van der Aalst and M. A. Odijk. Analysis of Railway Stations by Means of Interval Timed Coloured Petri Nets. In *Real-Time Systems*, volume 9, pages 1–23. Kluwer Academic Publisher, 1994.

[74] Hein S. Venter and Jan H. P. Eloff. Assessment Of Vulnerability Scanners. *Network Security*, pages 11–16, February 2003.

[75] John Viega and Gary McGraw. *Building Secure Software: How to avoid security problems the right way*. Addison-Wesley, 2002.

[76] Clark Weissman. System Security Analysis/Certification Methodology and Results. Technical Report SP-3728, System Development Corporation, Santa Monica, CA, USA, October 1973.

[77] Clark Weissman. Security Penetration Testing Guideline, U.S. Navy Handbook on Security Certification. Technical Report TM-8889/000/00, Paramax Systems Corp., Camarillo, CA, USA, December 1992. Prepared under contract to the U.S. Naval Research Laboratory.

[78] Clark Weissman. Penetration Testing. In Abrams and Jajoida and Podell, editor, *Information Security: An Integrated Collection of Essays*, pages 269–296. IEEE Computer Society Press, 1995.

[79] Nick Wingfield. It Takes a Hacker. *Wall Street Journal*, March 2002.

[80] Shijie Zhou, Zhiguang Qin, Feng Zhang, Xianfeng Zhang, Wei Chen, and Jinde Liu. Colored Petri Net Based Attack Modeling. In *Proceedings of Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing: 9th International Conference (RSFD-GrC 2003)*, volume 2639 of *Lecture Notes in Computer Science*, pages 715–718, Chongqing, China, May 2003. Springer-Verlag GmbH.

A   Used Declarations and Net Inscriptions

This appendix briefly explain the main syntax used in the net inscriptions for the coloured Petri net attack models in the thesis. The inscriptions explained here are used in explanatory examples found in chapter 2 and 5, and in the penetration testing scenarios described in chapter 6. A more thorough explanation of the syntax and other possible declarations in the coloured Petri net modelling language can be found in [31], at the CPNtools web site[1] and in the standard ML basis library[2].

Colour Sets

Each colour set declaration introduce a new colour set, whose elements are called colours. Thus each colour set declaration implicitly declares a set of constants [31]. The basic coloured Petri net ML colour sets used in the thesis are:

```
colour AA = int;
colour BB = string;
```

The first colour set,`AA`, can have integers as colours and the second, `BB`, can have text strings as colours. A subset colour set, i.e. a subset of a basic colour set, used in this thesis are:

```
colour CC = with "string1" | "string2";
```

The colour set `CC` can have two colours; the colours `string1` and `string2`. The last colour set inscription used in this thesis are:

```
colour DD = product AA*BB;
```

The colour set declare all pairs (`a`,`b`) where a $\in$ `AA` and b $\in$ `BB`. The `product` syntax can inevitably be used in combination of more than two previously declared colour sets. This inscription uses colour sets from already declared colour sets by means of a built-in colour set constructor, several other built-in colour set constructors exist.

Timed Colour Sets

All colour sets can be timed and are declared as follows:

```
colour EE = .... timed;
```

A colour set is timed by appending the keyword timed to the end of its declaration. At least one colour set must be timed in order to run a simulation with time.

---

[1] `http://wiki.daimi.au.dk/cpntools-help/cpn_ml.wiki` (Visited May 2005)
[2] `http://www.standardml.org/Basis/` (Visited May 2005)

Variables and Values

Each variable declaration introduce one or more variables, with a type which must be an already declared colour set. These variables are used in guards and in arc expressions. Variables are declared as follows:

```
var a : AA;
var b1, b2 : BB;
```

This mean that the variable `a` can contain colours from the colour set `AA`, i.e. integers. The variables `b1` and `b2` can contain colours from the colour set `BB`, i.e. strings.

Value declarations introduces a constant, with a type which must be an already declared colour set. In the thesis only simple string constants are used:

```
val FF = "text string";
```

The declared constants can be used by arc expressions and guards.

Net Expressions

Net expressions are built from a number of variables, values and colours. Used by arc expressions, guards and initialisation functions. The expressions used in the coloured Petri nets in this thesis can be explained by the following examples:

```
x=q
(x,i)
1'(x,i)
if x=q then 1'(q,i+1) else empty
substring(b1,i,1)
b1^b2
```

The first expression above simply compares a variable `x` and a constant `q`, evaluating to true or false. The second expression show a tuple-construction in an operation, where `x` and `i` are variables. `1'(x,i)` show these variables with the constant 1, where ' (read "n of") and the tuple-construction are operations. The if-then-else control structure use two variables, `x` and `i`, and the constants `q`, `1` and empty. ', $=$, $+$, the tuple-construction and the if-else construction are operations. The `substring` operation extract a substring of length 1 starting at position `i` in `b1`, first position is 0. The last operation, `b1^b2`, concatenate the strings `b1` and `b2`

The expressions used that has to do with time in arc expressions and guards can be explained by the following examples:

```
string @ t
a @ + j
b2 @ + [l,m]
```

A colour followed by an @ symbol and a integer, e.g. the time stamp `t` above, denote a time stamped colour. The expression `a @ + j` mean that an expression add the integer time delay `j` to each of the colours in variable `a`, the expression returns a timed colour. The last timed expression is the same however, the time added is represented by a closed interval `[l,m]`, meaning that the timed colour returned will be in between `t` and `m`.

B   Paper Submitted for Publication

Modeling and Execution of Complex Attack Scenarios using Interval Timed Colored Petri Nets

Ole Martin Dahl
Department of Computer Science
Gjøvik University College
Gjøvik, Norway
ole.dahl@hig.no

Stephen D. Wolthusen
Department of Computer Science
Gjøvik University College
Gjøvik, Norway
swolthusen@ieee.org

Abstract

The commonly used flaw hypothesis model (FHM) for performing penetration tests provides only limited, high-level guidance for the derivation of actual penetration attempts. In this paper, a mechanism for the systematic modeling, simulation, and exploitation of complex multi-stage and multi-agent vulnerabilities in networked and distributed systems based on stochastic and interval-timed colored Petri nets is described and analyzed through case studies elucidating several properties of Petri net variants and their suitability to modeling this type of attack.

1. Introduction

The design and deployment of large-scale internet-worked systems such as multi-tier database and electronic commerce systems, particularly when consisting of a large number of independent off-the-shelf components rarely proceeds from a precise specification or model of system behavior. In addition to vulnerabilities in individual components of such systems, however, this can also result in vulnerabilities that result from interactions among system components that include not only the off-the-shelf elements but also customized program elements and emergent properties such as timing and load behavior that can arise only in certain hardware and software configurations.

While basic configuration errors and implementation flaws in individual components such as host operating systems and network components can be identified both locally through debuggers, in-circuit emulators, custom device drivers and proxies and in part also remotely using automated tools such as fuzzers, port scanners, and network sniffers as well as tool suites such as Nessus [3], Metasploit, and Core Impact based on research and knowledge about such relatively standardized components, there are obvious limits to re-using such common attack scenarios for large composite systems that may well be unique.

Given the above, the problem of assurance for larger systems for which no formal specification and proof of security models and implementations existed [11, 10] has mainly been approached through custom penetration test exercises, i.e. by attempting to demonstrate the identification of flaws that could be found with a fixed but arbitrary work factor by skilled individuals [18, 21]. However, such testing clearly neither demonstrates the absence of critical defects or vulnerabilities nor does it necessarily even provide a satisfactory predictive value for the work factor of an (unknown) adversary, as is amply demonstrated by continuous discovery of vulnerabilities in off-the shelf components such as standard operating systems.

In addition to the issue of overall complexity of large information systems, application systems – particularly those exposed to clients – provide multiple potential vulnerability areas [9, 31]. Not only do such systems by definition also provide access paths to adversaries, but such systems are typically customized from off-the-shelf components or contain elements of customization or site-specific components. Moreover, unlike vulnerabilities in operating systems or other standard application programs, vulnerabilities in application systems can often be associated with quantifiable monetary risks (e.g. through embezzlement, pilferage, or by denial of service leading to losses in revenue and reputation), leading to increased interest in penetration testing and securing application systems. However, as is frequently the case in the absence of codified scientific or engineering practices and doctrine, this type of penetration testing has mainly been characterized as an art [9]. In this paper we argue that the modeling and analysis of expected and desired system behavior, in parallel with the modeling and analysis of complex multistage attacks can be performed effectively within a framework provided by Petri nets, which has been demonstrated to scale to very large environments.

It should be noted that Petri nets, particularly the variants discussed in this paper, are suited to both the modeling of desired or required system behavior and its limitations and also to the modeling, simulation, and ultimately execution of attacks as well. Unfortunately, the information flowing into these two types of models are somewhat disparate and it is not immediately possible to derive usable attack models from system models and vice versa in the general case. Therefore, the focus of this paper is solely on the perspective of attack modeling.

To this end, section 2 briefly reviews the background of pertinent Petri net variants, while section 3 provides an overview of penetration testing models currently in use as well as the positioning of Petri net-based attack models within a variant on a commonly found generic penetration testing methodology. Section 4 reviews two case studies applying colored and interval timed colored Petri nets for modeling, simulation, and execution of attack scenarios. Subsequently, section 5 briefly reviews related work in the area of complex attack modeling, and section 6 provides an outlook on ongoing and future research.

2. Petri Nets and Interval Timed Colored Petri Nets

Petri nets were originally introduced as a modeling technique for concurrent systems and provide a rich and graphically intuitive approach for modeling, simulation, and execution [25, 24, 26].

Elementary Petri nets can be considered as bipartite graphs with distinct node types that consist of vertices (places and transitions) and edges (arcs) connecting these. Arcs are further subdivided into input arcs, which connect places with transitions and output arcs, which start at a transition and end at a place. Places can contain tokens; the current state of the modeled system (referred to as a marking) is given by the number and type of tokens in each place. Transitions model activities through firing when enabled and thereby inducing changes in the marking. Several types of Petri nets can be distinguished by the level of information associated with individual tokens which can range from simple boolean information to structured tokens.

Colored Petri nets provide structured tokens in the form of so-called colors and provide a significant increase in the expressiveness and compactness of models [15, 14, 16]. To support time-dependent environments such as real-time systems or multi-agent interactions such as those considered in this paper, a number of extensions to both general Petri nets and colored Petri nets have been proposed, including time interval colored Petri nets [15] and stochastic colored Petri nets [38].

Timed colored Petri nets (TCPN) introduced by Jensen [14] uses a global clock and time stamps associated with to-

kens. The time stamp represent an aditional constraint in CPNs which describes the earlist model time at which tokens can be consumed by an occuring transition.

Interval timed colored Petri nets (ITCPN) were introduced by van der Aalst [32] and subsequently applied in a number of areas, particularly in real-time control systems [33]. In this model, time stamps are associated with tokens in addition to the token color and the firing delay of transitions is specified by bounded time intervals, providing a mechanism to model uncertainty and nondeterminism in individual transitions that cannot be captured adequately in models providing deterministic delay models. Formally, an ITCPN can be defined as follows:

Definition 1 *An ITCPN is a 5-tuple* (Σ, P, T, C, F) *such that*

1. Σ *is a finite set of types; the types in* Σ *are also referred to as colors, where multiple colors can be associated with a given type.*
2. P *is a finite set of places, and*
3. T *is a finite set of transitions such that* $P \cap T = \emptyset$.
4. C *is a color function* $C : P \mapsto \Sigma$,
5. $CT = \{(p, v) | p \in P \wedge v \in C(p)\}$ *is the set of all possible colored tokens, and*
6. F *is the transition function such that given a time set* $TS = \{x \in \mathbb{R} | x \geq 0\}$ *and the set of all closed intervals* $INT = \{[y, z] \in TS \times TS | y \leq z\}$, *the multiset (denoted by the superscript* X°) *of consumed tokens and the multiset of produced tokens for a transition* t *is mapped by* $F(t)$:

$$F : CT^\circ \nrightarrow (CT \times INT)^\circ$$

$domF(t)$ is the condition under which a given transition $t \in T$ *is enabled.*

The following definition provides a rudimentary description of the dynamic semantics of ITCPN:

Definition 2 *The state of a ITCPN* (Σ, P, T, C, F) *is defined as a multiset of colored tokens, such that the state space is*

$$S = (CT \times TS)^\circ.$$

The marking of an ITCPN in state $s \in S$ *is the token distribution* $M(s) \in CT^\circ$ *such that*

$$M(s) = \sum_{\langle \langle p, v \rangle, x \rangle \in CT \times TS} s(\langle \langle p, v \rangle, x \rangle) \langle p, v \rangle$$

.

An event in an ITCPN is a 3-tuple $\langle t, b_{in}, b_{out} \rangle$ *representing the firing of transition* t, *removal of tokens in* b_{in}

and addition of tokens in b_{out} where the event set E is given as

$$E = T \times (CT \times TS)^\circ \times (CT \times TS)^\circ$$

.

A relation on tokens $b \lhd b'$ is defined as

$$b \lhd b' \iff \begin{cases} b = \emptyset \wedge b' = \emptyset \quad \text{or} \\ \exists \langle \langle p, v \rangle x \rangle \in b \exists \langle \langle p, v \rangle [y, z] \rangle \in b' \bullet \\ (x \in [y, z]) \wedge \\ (b - \langle \langle p, v \rangle x \rangle) \lhd (b' - \langle \langle p, v \rangle [y, z] \rangle) \end{cases}$$

An event $\langle t, b_{in}, b_{out} \rangle \in E$ is enabled in a state $s \in S$ if and only if the following conditions are met:

1. $b_{in} \leq s$,

2. $M(b_{in}) \in dom F(t)$, and

3. $b_{out} \lhd F(t)(M(b_{in}))$

It should be noted that a token residing in a given place $p \in P$ must have a value v such that $v \in C(p)$, i.e. a token color must match one of the place colors. For a detailed description of the dynamic behavior model of ITCPN refer to [32].

ITCPN uses the simplified assumption of a single global time and point-value time representations, which is not entirely appropriate for distributed systems where such cannot be determined. However, for the purposes of modeling described in this paper, the interval time properties provide sufficient abstraction even for multi-agent attack environments when used in simulation. Both this simplification and the semantics for ITCPN described in this sections imply a limited suitability for analytical purposes such as exhaustive reachability graphs; this would require a severe restriction in the supported semantics [32]. However, as the main purpose of the mechanisms described in this paper lie in the simulation and execution of generated models, the simplicity and effectiveness of the semantics described here appear to outweigh this drawback.

3. Penetration Testing Models

While ad-hoc mechanisms for identifying flaws and exploiting such flaws in attacks or penetration tests are still prevalent, some of the earliest results in the area of penetration testing introduced a structured approach that was also used in practice early on, with a primary focus on operating system penetration testing [35, 21, 8, 7]. The Flaw Hypothesis Model (FHM) by Linde and Weissman has been refined since [36, 2, 37] and is widely used as a mechanism for structuring penetration-type testing, primarily in the area of certification and accreditation. It consists of four interrelated and iterative stages:

1. *Flaw generation*: Generation of hypothetical or suspected flaws in the system under test.
2. *Flaw confirmation*: Classification of flaw hypotheses as true, false, or untested.
3. *Flaw generalization*: Attempt at identifying common underlying weaknesses from confirmed flaws and generalization into flaw classes.
4. *Flaw elimination*: Removal or mitigation of flaws through external controls.

The steps in the FHM are solely based on heuristics and do not lend themselves to automation to a significant extent. In particular, the flaw generation step is confined to perusal of all documentation and source code available (depending on whether a red-team or blue-team penetration test is intended) and informal techniques for the creation of attack or flaw scenarios.

An alternate methodology, based largely on fault tree analysis, originally developed for systems analysis by Bell Telephone Laboratories for use on the Minuteman strategic missile system [20], is the use of tree-based mechanisms [27, 28]. This methodology, similar to FHM, provides a mechanism for structuring and guiding an informal and heuristic-based approach to the identification and confirmation of flaws and attack scenarios.

McDermott proposed the use of simple disjunctive Petri nets as a mechanism for structuring the penetration testing process, called attack nets [23]. Attack nets represent larger-scale states of the system under attack as places and model operations such as events and data flows as transitions within the attack net with multiple tokens representing the steps an attacker must take in order to achieve his objective (e.g. control over the system under attack). In this model, the constraints on transitions provide the requisite coordination in case multiple tokens are required for a given action.

This mechanism provides testers with the ability to model

- concurrency and attack progress in the form of tokens
- intermediate and final objectives as places, and
- commands or inputs as transitions.

As noted by McDermott, the attack net mechanism is not intended to model actual behavior of attackers and does not provide the requisite level of detail that would be necessary for such a process.

3.1. Interval Timed Colored Petri Nets

We argue that the attack net model can be further refined to address the challenges of identifying several classes of flaws in penetration testing that cannot be identified with justifiable effort using the previously described informal heuristics.

To this end, we extend the attack net mechanism by Mc-Dermott to provide a level of detail sufficient for the simulation and execution of attacks as part of an overall methodology such as the flaw hypothesis methodology. This mechanism is, however, not intended to supplant the larger, holistic methodologies such as the FHM, Attack Trees, or related approaches such as the InfoSec Assessment Methodology as defined by the U.S. National Security Agency.

One of the key benefits of Petri nets in addition to their graphical nature and hence their appeal to visual comprehension and to some extent intuition is that well-defined semantics exist for Petri nets which permit the seamless integration of modeling, simulation, and execution. It is particularly in the latter two elements that the results reported in this paper extend the approach found in [23]. Based on the overall methodical framework of the FHM, Petri nets and particularly ITCPN can be used for the stepwise refinement in the identification, exploration, and analysis of attacks and flaws within information systems.

In the basic case, Petri nets (not necessarily with time components) can be used to structure attack elements, including documenting the need for concurrent actions, reactions by the system under attack and its operators, and the codification of preconditions for attack steps to proceed; in this, even basic binary Petri nets provide a mechanism for effectively modeling of multi-agent attacks and the interactions between multiple attacking agents and defenders.

This mechanism can subsequently be refined considerably by using colored Petri nets (CPN) for including details on the attack steps to be conducted; the context provided by the token color as well as other annotations provides sufficient information to permit both the simulation and, more importantly, a direct mapping onto an execution element for an attack.

The execution mechanism itself can e.g. be constituted by a Nessus module or custom-written code element. However, the general framework for sequencing the attack components and providing information gained in the course of the attack to other components is retained within the general CPN simulation tool. This provides a significant benefit both in the structure and the reusability of individual attack elements.

Moreover, the Petri net mechanism lends itself well to a hierarchical structuring of attacks as well as the factoring and parameterization of attack components for reuse. If a sufficient level of detail is provided, a CPN model can also be used for a direct reachability analysis for certain classes and vulnerabilities, thereby automating at least in part one of the steps that is typically based on heuristics within the FHM. However, both the computational complexity of such reachability analyses and the level of detail required for the creation of a CPN model suitable for such analysis clearly delineate the limits of the approach [17].

The class of attacks this paper is primarily concerned with, however, is not immediately amenable to reachability analysis. Timing-dependent attacks, particularly executed by multiple agents in a networked environment and typically also executed against multiple targets or networked components of a target system constitute a large class of attacks that are promising to attackers since some of the fault conditions that render such systems susceptible to attacks are difficult to eliminate through static analysis and software engineering practices and may in some cases be ephemeral properties of system configurations that were not anticipated by system designers even if such systems were designed with due diligence. Examples of relevant classes include TOCTOU (time of check to time of use) attacks, race conditions and resource contention attacks.

While such vulnerabilities are frequently identified and both exploited and corrected for local systems (e.g. in case of temporary file or lock creation), they are significantly mode difficult to identify in distributed environments where network latency and load conditions can provide significant distortions in the timing and even sequencing of events and actions relevant for the successful conclusion of an attack. Attackers therefore are forced to rely mainly on brute force type approaches to identify vulnerabilities even if sufficient knowledge of the system under attack such as component source code is available. This, in combination with the effort required to construct multiple-agent type attacks has presumably limited the success in identifying significant numbers of vulnerable systems and configurations.

Moreover, since the systems exhibiting this type of timing-based vulnerabilities are frequently exposed to public networks and must let application network traffic pass through network defense mechanisms (e.g. in case of web services or electronic commerce sites), threat mitigation strategies such as limiting network traffic is ineffective if the attack can be framed in terms of legitimate or legitimate-appearing traffic following application semantics. In addition, the systems susceptible to these timing-based attacks are also typically custom-built with significant configurations and custom software added to underlying standard components such as web servers, databases, and programming environments, therefore making the elimination of this class of attack through generic defect or threat removal becomes exceedingly difficult.

The addition of time intervals to the model of transitions and events in Petri networks in the form of ITCPN provides a simple, elegant, and powerful mechanism to model such timing relations, even in the presence of uncertainty over the precise nature of the relation (as may be the case even when performing full white-box penetration testing).

While the ability to perform static reachability analysis on larger-scale systems is even more limited than in the case

of CPN without timing elements [29], the interval timed net nevertheless provides a natural mechanism for both simulation and execution of the time intervals through the sampling of the time intervals (which may be further supplemented by approaches and semantics commonly found in stochastic Petri nets [22]). It should be noted that while the Petri net framework provides partial automation of such attacks, it is still necessary particularly for larger networks to guide the simulation and execution using heuristics since exhaustive sampling of all permutations even over a suitably constrained partial reachability tree may well exceed the time available even for automated penetration testing. Nevertheless, the benefits of automation in conjunction with the use of the general Petri net modeling tool framework for reusability and refinement of attack models including hierarchical models at different levels of detail in modeling previously mentioned provide significant benefits over the de novo creation of attack scripts.

4. Scenarios

The following section demonstrates the types of attack scenarios of primary interest for this methodology in the form of case studies, beginning with an elementary net.

4.1. Consistency of Condition Checks

The class of Time-of-Check-to-Time-Of-Use (TOCTOU) vulnerabilities provides a large number of opportunities for attackers not only for operating systems but particularly also in case of application systems. While such attacks can occur in case of logical flaws, the more common root cause is a race condition (see also section 4.2). In the latter case, timing becomes relevant. The most common way of exploiting such vulnerabilities, particularly for local exploits, is to automate an attack and run it multiple times in a brute force approach, adjusting timing to hit just the right interleaving of operations. On the UNIX platform, the canonical example of application TOCTOU vulnerabilities are symbolic link attacks. A TOCTOU vulnerability in a program running EUID[1] of another user, preferably `root`, for the duration of the race condition.

The following penetration test study show how we can use TCPN to model such an attack. This case is based on a historical case of a TOCTOU vulnerability of `passwd(1)`, described by Bishop and Dilger in [4]. Precondition for the attack is a penetration tester with local, nonprivileged shell access. The underlying flaw hypothesis here is the ability to masquerade as another user in the system. Investigating

---

1 Effective User Identification, e.g. the `passwd` process runs with the real UID of `attacker` and effective UID of `root` so it will be able to access `/etc/passwd` and change password for user `attacker`.

the flawed program, `passwd`, the following steps can be either deduced or hypothesized:

1. Open password file, read it, retrieve the entry for the running user
2. Create and open a temporary file (called `ptmp`) in the same directory as the password file.
3. Open password file, copy contents into `ptmp`, update modified information.
4. Close password file and `ptmp`, then rename `ptmp` to be the password file.

Without the ability to halt `passwd` between each step, a timed attack model becomes relevant. Modeling the `passwd` steps (transitions) in a colored Petri net is straightforward, see the right of figure 1 on page 8.

The attacker must do some initial preparation to exploit the vulnerability:

1. Create a password directory.

2. Create a `.rhost` file in that directory.

3. Insert login credentials into `.rhost`

4. Make a symbolic link that links to the password directory.

These steps are just done for preparation and can be modeled as one transition in the colored Petri net, with two conjunctive places as input, which is the fake login credentials and shell access to the computer.

There are two color sets in the TCPN which are timed, `Attack` and `Process`. The symbolic link state are represented through the color set `Attack`, while the system state is represented by the color set `Process`. For illustrative purposes, a simplified assumption of the net in figure 1 is that each transition or step in the `passwd` process consumes the same amount of CPU time. The actions of the attacker are then modeled as a parallel net within the same time window.

The TOCTOU or symbolic link attack starts by starting the `passwd` process on the link to the attackers `pwd` directory. Then when the process action has a token with the color `UserEntry` at the place `user entry read` the attacker must change the symbolic link to point to the target directory. Since the net are timed the token colored `UserEntry` will have a time stamp of 2 and the token colored `LinkToPwd` at the place `attacker must change link` have a time stamp of 1. This happens because the input arc of the place `attacker must change link` adds 1 time unit and the transition `open password file read entry for running user` uses 2 time units. The differences in time stamps imply that the token with lowest time stamp will fire first, thus the transition `delete link create new link`

will be enabled first. This is how the TCPN continues, the model remains at a specific time until no more transitions are enabled at the current model time. The model time increases as the tokens change time stamp. The attacker must change the symbolic link between every transition in the `passwd` process. This attack will end when the model time has reached 7. Then the `passwd` process is finished and the attacker has successfully changed the target users `.rhost` file.

This local penetration testing exercise demonstrate the possibilities and limitations of modeling timed attacks using TCPNs. The complexity of such attacks done manually can become difficult to control if several conditions must be true at any one time, as can be seen clearly in the attack described here, even though only two main concurrent processes were present. Other, similar attacks may have three or more concurrent processes, e.g. attacker actions, a process that writes to a file, and another that reads the file. Modeling such attacks in a TCPN helps the penetration test to become successful without too many manual and time-consuming operations and interventions. A problem, however, is obvious when vulnerabilities like this appear in a distributed environment, i.e. not on a single computer where one may use or at least model using a single global time. For distributed systems, there not only is no such global time (at most a partial order), but also a number of variations owing e.g. to load and latency conditions within the distributed system. In such an environment, TCPN are inadequate since one must take the uncertainties immanent in the model into account both for the modeling and the simulation/execution stages.

The following section therefore briefly demonstrates the use of interval-timed colored Petri nets.

4.2. Race Condition Attack

Race conditions, particularly in application systems constructed in the form of multi-tiered architectures, are both common and difficult to exploit remotely in an efficient manner. To illustrate the use of ITCPN for modeling and execution of a race condition-type attack, we assume a three-tier electronic commerce site in the following scenario (this scenario is a composite of multiple typical application-layer attacks and does not correspond to a single existing system). Without loss of generality, we assume a back end database system and an intermediate application layer (e.g. running on web servers with servlets), running on multiprocessor or at least multithreading systems.

The objective of the penetration test is to modify back end data, in this case to modify the shipping data of a customer's order. It is not necessary for the attacker to compromise either the application or back end layer for this attack to succeed.

We further assume that, during the information gathering phase, it has been ascertained that all connections to the application layer are secured via a TLS channel and that the firewalling mechanism separating the presentation layer (client side) from the application and back end layers are effective. However, an attacker is able to obtain data from customer identification cookies from legitimate customers, as is commonly accomplished through e.g. cross-site scripting. The penetration tester has, moreover, also identified a construction rule for legitimate transaction identifiers[2].

Subsequently, the penetration tester can either through deduction from existing commerce back ends or through study of design documents identify the final step in performing a purchase, namely the confirmation on the part of the customer (after having signed onto the commerce site, selected the merchandise and entered shipping and payment details) and the recording of purchase data with both internal and external databases and services.

The elements of the order confirmation transaction thus obtained are as follows:

Inventory update In this step, an update of the inventory database table is performed.

Payment confirmation A final confirmation of customer credit standing is performed by an banking service and, on success of this step, the payment is recorded in the local credit database.

Shipping processing The shipping details are written to the shipping information database table.

Order fulfillment Based on the previous information, merchandise pickup data and shipping addresses are collated and sent to the shipping agent.

The operative flaw hypothesis is that developers have, in order to obtain maximum throughput, the various databases accesses during the course of the transaction, are performed with row-locking at each stage, but not across multiple stages of the above transaction. Only the entire transaction can be rolled back in case of a failure at an individual stage.

An attacker able to inject data with valid customer and transaction identification to the application layer as described above can perform modifications to the shipping information database table after the completion of the third stage and prior to commencement of the fourth stage, resulting in goods being diverted to an attacker's address of choice. In this type of attack scenario it is not necessary for the attacker to learn payment information details of a customer, or to compromise the back end; the hypothetical flaw lies solely within the application logic and its implementation in accessing the database back end.

To model an application-level penetration, an ITCPN can therefore be constructed that focuses the primary level of

―――――――――――――――

2 In many electronic commerce applications these are simple sequence numbers and therefore trivial to predict.

detail and timing dependency on the progress through the transaction stages outlined above. Each of the four steps described above is framed by a beginning and ending transition associated with a time interval, along with a similar time interval for intermediate processing between the steps. The same structure can then be used – with different colors – for a transaction initiated but not completed by the attacker.

To simplify the network sufficiently so it will fit on a single page, we assume that the attacker has an injection channel (modeled separately) into the database communication between steps three and four. Such an attack can be a (timing independent) HTTP POST request that utilizes the information gained as described above. Since the attacker is authenticated as a valid customer through the TLS layer, perimeter defenses are unlikely to register this injection as a malicious act.

Figure 2 on page 9 shows the attack model. This interval timed colored Petri net uses product color sets (`Inventory`, `Credit`, `Shipping`, `SendShipping`, `Ordr`), which combines other color sets (strings and integers) to represent the information written to the back end database. These product color sets and the color set `Lock` are timed.

Prior to the attack steps shown in the attack model in figure 2 on page 9, the customer has finished and confirmed his order. The attack models the system confirmation steps in parallel with the attacker's injection action. The place `confirmed order` represents the system state where a customer has confirmed his order in the presentation layer. From this point in time the penetration tester must time his injection. The timing will vary because of the distributed nature of the system, hence the interval timing. The first transaction, `req inventory update`, will update the inventory table in the back end database. The entire transaction is modeled to consume time, that is between 1 and 6 ($[a, b]$) time units (here: milliseconds). The minimum time, 1, represents how long the entire transaction may take if there are no delays. Acquiring the database table lock is modeled to take between 0 and 12 ($[c, d]$) time units, depending on database load. The minimum time of 0 is chosen because the lock may be available immediately for the transaction `update inventory`. The transaction will not fire before there is at least one token in all input places. The fuzzy timing is represented by closed intervals in the interval timed colored Petri net. The first confirmation step – inventory update of a customer order – can by this use all between 1 and 18 ($[a + c, b + d]$) time units in the attack model.

The second confirmation step, payment confirmation, is modeled similarly to the former. However, this step uses a slightly different timing, simulating that the loads on the credit database table are different from the more heavily loaded inventory table. At the same time as the payment is written to the credit table, a payment confirmation is sent to the external financial service, confirming credit withdrawal. In the next confirmation step the address is written to the shipping database table. It is here that the race condition the penetration tester is trying to exploit appears. The `req fulfillment` transition will wait for the results from the database table write to finish, and then the shipping fulfillment will read from the shipping table and finish the order. The transaction `read shipping details` will in most database systems acquire a shared read lock that prevents the row or table to be updated at the same time as another client attempts to read. However, because of the time window that appear between the write to the database table and the read from this database table by the two last fulfillment processes (and an imprudent lapse of the surrounding lock), the attacker can gain a exclusive write lock before the transaction `read shipping details` commences reading.

Immediately after the customer has placed his order, the attacker will inject the changed shipping address, leaving the remaining data untouched. The available time window for the attack appears after the shipping details has been written to the shipping database table and the final fulfillment step is initiated. Thus, if the attacker can inject a new record into the database shipping table before the transaction `read shipping details` is fired, the attack is successful. By calculating the minimum and maximum time this could take, we can obtain an interval where the injection is most likely to be successful. This interval must start from the starting point, that is from where the customer confirms his order in the presentation layer, to the maximum time it can take for the customers shipping details to be written to the database table. In the attack model this time interval can be calculated to be in the closed time interval $[3, 60]$. In this time interval the attack will be most likely to succeed. This means however that the attack will not always will succeed, but it is likely to succeed after a small number of (automated) attempts. The main flaw hypothesis investigated here is the race condition situation between the confirmation steps. The small time window which appears between the release of a lock and the next confirmation step make the injection possible. Another underlying flaw in the system that becomes clear from this penetration test is the weak user identification mechanism. The system design has a weaknesses in separating user that has been authenticated against the server by username and password and subsequently through a cookie mechanism. The internal processes assume that a message with a specific customer identification always comes from the correct client. This assumption can be imprudent, especially when the transaction identification can be intercepted or guessed by an adversary, meaning that an adversary may have all information
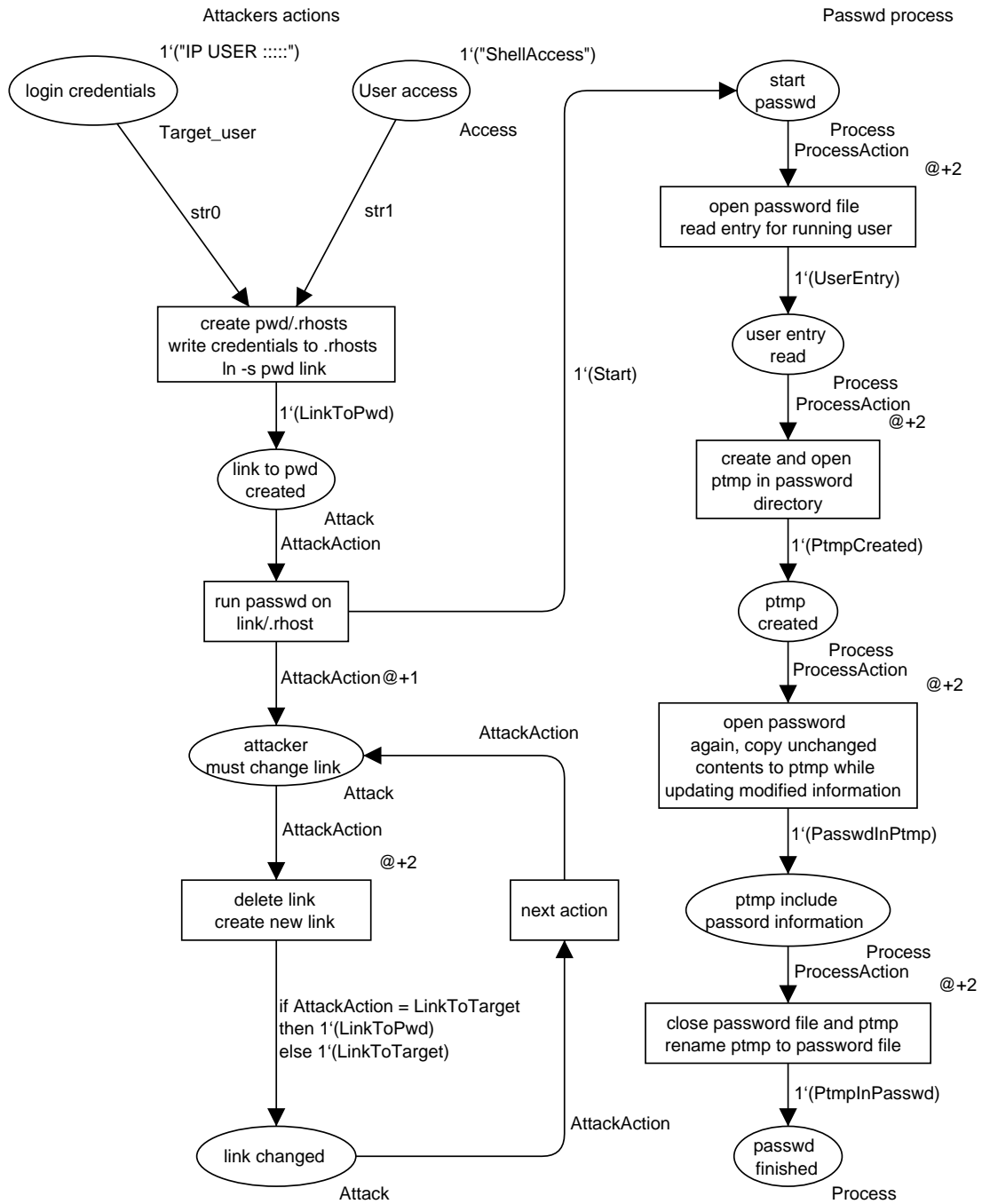
Figure 1. Symbolic link attack. Time constraints are in upper right-hand corner in transitions and after @ symbol in arcs. Color sets are in lower right-hand corner, initial markings in upper right-hand corner of places.
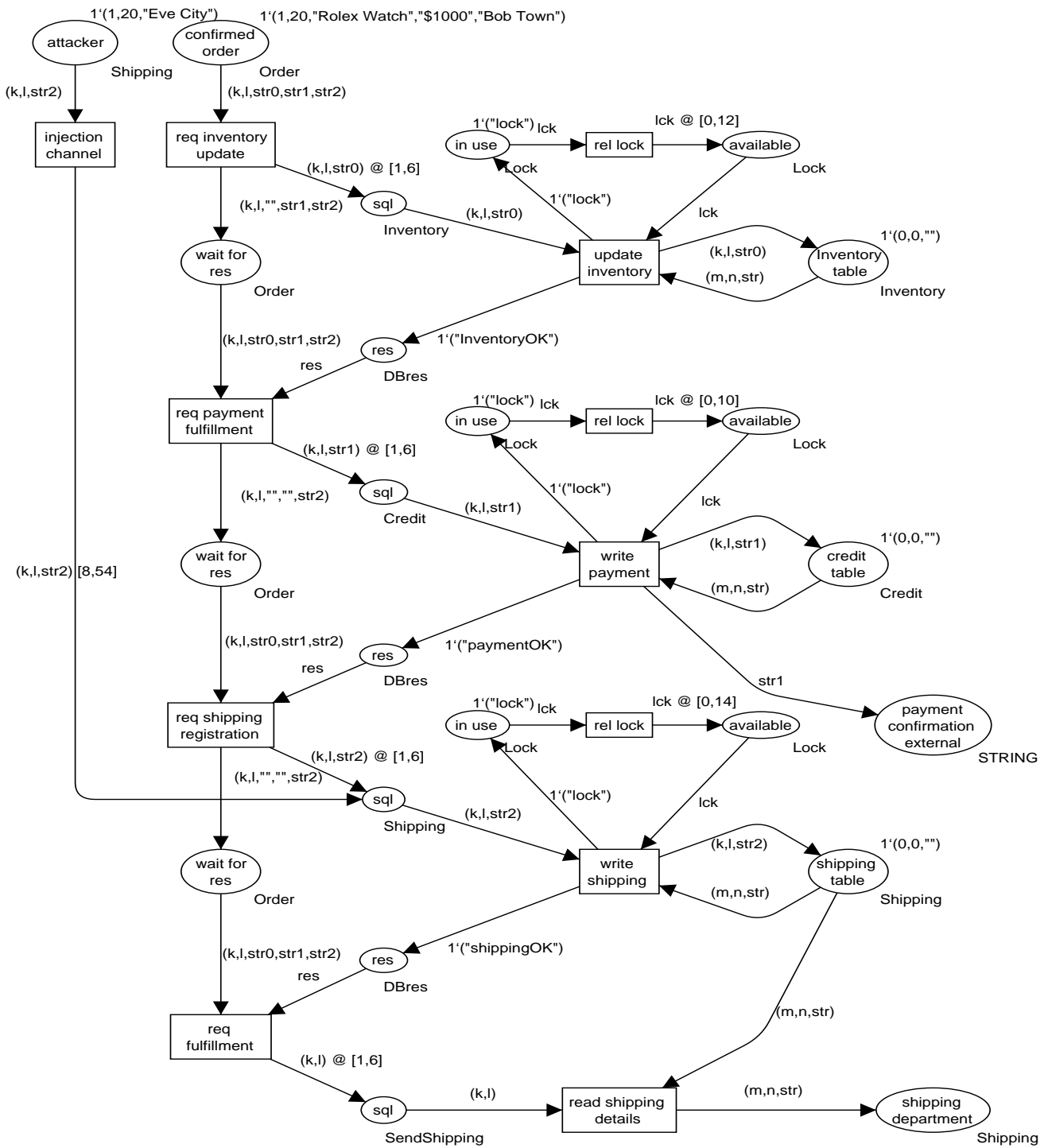
Figure 2. Race condition attack. The interval timing constraints are denoted as closed intervals in the arc expressions of output arcs, i.e. an @ followed by the interval.

that identifies an order in the back end database. The flaw of being able to guess transient transaction identifiers in combination with the customer identifiers make this vulnerability (present in quite a few shopping-cart type applications) dangerous. If the system had not released the database table or row lock between each stage in the final confirmation steps, the attacker would not be capable of injecting arbitrary information and thereby change a customer's order that has already been properly approved by the customer, application and external systems. However, for performance reasons, and because of the distributed nature of the system, such locks are frequently not used in such a conservative way. A downgrade from the exclusive write lock to a read lock could help in the two last steps of the confirmation transaction, but this is again difficult to realize in a distributed environment. In addition, the elimination of all HTTP POST injections would also stop this attack, but in complex multi tier e-commerce systems like this it is difficult prevent all such injection possibilities and may again result in significant performance penalties.

This attack scenario has demonstrated the use of a interval timed CPN for modeling a moderately complex time-dependent vulnerability and exploits of such vulnerabilities. Even in such a limited scenario, however, the benefits of a more rigid modeling approach and the ability to perform a round-trip modeling, simulation, and execution flaw hypothesis penetration test have become apparent.

5. Related Work

Tool-based flaw identification mechanisms for penetration testing were investigated in parallel with the development of penetration testing methodology for operating systems [13, 5, 1], but were not pursued further. The main use of tool-based mechanisms currently is in the identification of network topologies, system services, and network host configurations [34]; the confirmation of flaws' existence is also facilitated by tools such as Nessus, Metasploit, and Core Impact.

As noted in section 3, the use of Petri networks for the description and modeling of abstract attack approaches was proposed by McDermott [23]. Subsequently, Steffan and Schumacher [30] proposed a similarly abstract knowledge-sharing approach to attack modeling that combines the free-form mechanisms of Wiki publishing systems with conditional transitions as may be found in Petri nets, albeit without formalized semantics.

Petri nets and Colored Petri nets have, however, been used extensively in the design and implementation of intrusion detection systems such as those by Kumar and Spafford [19] and Helmer *et al.* [12] along with related formalisms based on timed finite state machines such as the approach

proposed by Chang *et al.* [6], which may be considered as complementary to the approach described in this paper.

6. Conclusion and Future Work

This paper has described a mechanism for the modeling, partial analysis, and automatic execution of multi-agent, multi-stage attacks based on interval timed colored Petri nets. The ability to construct partial and hierarchical models using an intuitive yet mathematically sound mechanism and a graphically oriented toolkit permits the exploration of several types of attacks, particularly based on TOCTOU and race conditions, which are difficult to identify in network-based environments. By coupling an execution mechanism based on sampling interval times and parameterizing attack code fragments, such attacks can be automatically conducted based on the model constructed once the iterative development of attack scenarios has reached a sufficient level of specificity. Moreover, both attack fragments and elements of attack scenarios can be reused either in part or as components of larger-scale attacks.

Such mechanisms should provide the ability to expose vulnerabilities particularly for network-based application systems that are difficult to identify in the course of manual, heuristic-driven penetration testing given the effort required in manually constructing attack scripts that are specific to a given application system and which may not be transferable to other sites without major modification or complete revision.

Future work will include investigations into the use of generalized stochastic Petri nets Monte Carlo-based sampling strategies for coverage of the state space; in addition, the composition of partial attack models from heterogeneous sources in which data structures and data types need to be reconciled for use in larger-scale ITCPN requires further investigation to permit the use of this attack and penetration testing model for collaborative efforts. Another subject of future work that should prove useful is the integration of multiple subnets (and, where necessary, their homogenization through renaming and intermediate networks to reconcile naming and token cardinality conflicts) and the semi-automated documentation of full FHM roundtrip analyses that so far need to be conducted manually, leading to less than consistent documentation of the penetration testing processes.

References

[1] R. P. Abbott. Security Analysis and Enhancement of Computer Operating Systems. Technical Report NBSIR 76-1042, National Bureau of Standards ICST, Gaithersburg, MD, USA, Apr. 1976.

[2] M. D. Abrams, S. Jajodia, and H. J. Podell, editors. *Information Security: An Integrated Collection of Essays*. IEEE Press, 1995.

[3] J. Beale, H. Meer, R. Temmingh, C. van der Walt, and R. Deraison. *Nessus Network Auditing*. Syngress, Rockland, MA, USA, 2004.

[4] M. Bishop and M. Dilger. Checking for Race Conditions in File Accesses. *Computing Systems*, 9(2):131–152, 1996.

[5] J. Carlstedt, R. Bisbey, and G. Popek. Pattern-Directed Protection Evaluation. Technical Report ISI/RR-75-31, University of Southern California Information Sciences Institute, Marina del Rey, CA, USA, June 1975.

[6] H.-Y. Chang, S. F. Wu, and Y. F. Jou. Real-Time Protocol Analysis for Detecting Link-State Routing Protocol Attacks. *ACM Transactions on Information and System Security*, 4(1):1–36, Feb. 2001.

[7] L. M. Galie and R. R. Linde. Security Analysis of the IBM VS2/R3 Operating System Scientific Computer. Technical Report TM-WD-7203/000/00, System Development Corp., Washington D.C., USA, Jan. 1976.

[8] L. M. Galie, R. R. Linde, and K. R. Wilson. Security Analysis of the Texas Instruments Advanced Scientific Computer. Technical Report TM-WD-6505/000/00, System Development Corp., Washington D.C., USA, June 1975.

[9] D. Geer and J. Harthorne. Penetration Testing: A Duet. In *Proceedings of the 18th Annual Computer Security Applications Conference (ACSAC 2002)*, pages 185–198, Las Vegas, NV, USA, Dec. 2002. IEEE Press.

[10] S. Gupta and V. D. Gligor. Experience with a Penetration Analysis Method and Tool. In *Proceedings of the 15th National Computer Security Conference*, pages 165–183, Baltimore, MD, USA, Oct. 1992.

[11] S. Gupta and V. D. Gligor. Towards a Theory of Penetration-Resistant Systems and Its Applications. *Journal of Computer Security*, 1(2):133–158, 1992.

[12] G. Helmer, J. Wong, M. Slagell, V. Honavar, L. Miller, Y. Wang, and R. Lutz. Software Fault Tree and Colored Petri Net Based Specification, Design and Implementation of Agent-Based Intrusion Detection Systems. Technical report, Iowa State University, Ames, IA, USA, June 2002.

[13] D. Hollingsworth, S. Glaseman, and M. Hopwood. Security Test and Evaluation Tools: An Approach to Operating System Security Analysis. Technical Report P-5298, RAND Corporation, Santa Monica, CA, USA, Sept. 1974.

[14] K. Jensen. *Coloured Petri Nets: Analysis Methods (Volume 2)*. Monographs in Theoretical Computer Science. Springer Verlag, Heidelberg, Germany, 1997.

[15] K. Jensen. *Coloured Petri Nets: Basic Concepts (Volume 1)*. Monographs in Theoretical Computer Science. Springer Verlag, Heidelberg, Germany, 1997.

[16] K. Jensen. *Coloured Petri Nets: Practical Use (Volume 3)*. Monographs in Theoretical Computer Science. Springer Verlag, Heidelberg, Germany, 1997.

[17] E. Y. T. Juan, J. J. P. Tsai, and T. Murata. Compositional Verification of Concurrent Systems Using Petri-Net-Based Condensation Rules. *ACM Transactions on Programming Languages and Systems*, 20(5):917–979, Sept. 1998.

[18] P. A. Karger and R. R. Schell. Multics Security Evaluation: Vulnerability Analysis. Technical Report ESD-TR-74-193, Vol. II, Information Systems Technology Application Office, Deputy for Command and Management Systems, Electronic Systems Division (AFSC), L. G. Hanscom AFB, MA, USA, June 1974.

[19] S. Kumar and E. H. Spafford. A Pattern-Matching Model for Instrusion Detection. In *In Proceedings of the National Computer Security Conference*, pages 11–21, Baltimore, MD, USA, Oct. 1994.

[20] W. Lee, D. Grosh, and F. Tillman. Fault tree analysis, methods, and applications. *IEEE Transactions on Reliability*, 34(3):194–203, Aug. 1985.

[21] R. R. Linde. Operating System Penetration. In *Proceedings of the AFIPS National Computer Conference*, volume 44 of *AFIPS Conference Proceedings*, pages 361–368, Anaheim, CA, USA, May 1975. AFIPS Press.

[22] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets* . John Wiley & Sons, New York, NY, USA, 1995.

[23] J. P. McDermott. Attack Net Penetration Testing. In *Proceedings of the 2000 Workshop on New Security Paradigms (NSPW 2000)*, pages 15–21, Ballycotton, Ireland, Oct. 2000.

[24] J. L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981.

[25] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Technische Hochschule Darmstadt, Darmstadt, Germany, 1962. Published in Schriften des Instituts für Instrumentelle Mathematik 3,1-128, University of Bonn, Germany.

[26] W. Reisig. *Petri Nets: An Introduction*. Monographs in Theoretical Computer Science. Springer Verlag, Heidelberg, Germany, 1985.

[27] C. Salter, O. S. Saydjari, B. Schneier, and J. Wallner. Toward a Secure System Engineering Methodolgy. In *Proceedings of the 1998 Workshop on New Security Paradigms (NSPW 1998)*, pages 2–10, Charlottesville, VA, USA, Sept. 1998.

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

[29] R. H. Sloan and U. Buy. Stubborn Sets for Real-Time Petri Nets. *Formal Methods in System Design*, 11(1):23–40, July 1997.

[30] J. Steffan and M. Schumacher. Collaborative Attack Modeling. In *SAC '02: Proceedings of the 2002 ACM Symposium on Applied Computing*, pages 253–259, Madrid, Spain, Mar. 2002. ACM Press.

[31] H. H. Thompson. Application Penetration Testing. *IEEE Security & Privacy*, 3(1):66–69, Jan./Feb. 2005.

[32] W. M. P. van der Aalst. Interval Timed Coloured Petri Nets and their Analysis. In M. A. Marsan, editor, *Proceedings of Application and Theory of Petri Nets 1993, 14th International Conference*, volume 691 of *Lecture Notes in Computer Science*, pages 453–472, Chicago, IL, USA, June 1993. Springer-Verlag.

[33] W. M. P. van der Aalst. Analysis of Railway Stations by Means of Interval Timed Coloured Petri Nets. *Real-Time Systems*, 9(3):241–263, Nov. 1995.

[34] J. Wack, M. Tracy, and M. Souppaya. Guideline on Network Security Testing. Technical Report Special Publication SP 800-42, U.S. National Institute of Standards and Technology, Gaithersburg, MD, USA, Oct. 2003.

[35] C. Weissman. System Security Analysis/Certification Methodology and Results. Technical Report SP-3728, System Development Corp., Santa Monica, CA, USA, Oct. 1973.

[36] C. Weissman. Security Penetration Testing Guideline, U.S. Navy Handbook on Security Certification. Technical Report TM-8889/000/00, Paramax Systems Corp., Camarillo, CA, USA, Dec. 1992. Prepared under contract to the U.S. Naval Research Laboratory.

[37] C. Weissman. Penetration Testing. In Abrams et al. [2], pages 269–296.

[38] A. Zenie. Colored Stochastic Petri Nets. In *Proceedings of the International Workshop on Timed Petri Nets*, pages 262–271, Torino, Italy, July 1985.