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 [17, 20]. 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, 30]. 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 [24, 23, 25].

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 [14, 15, 15]. 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 [14] and stochastic colored Petri nets [37].

Interval timed colored Petri nets (ITCPN) were introduced by van der Aalst [31] and subsequently applied in a number of areas, particularly in real-time control systems [32]. 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 $P \cap T = \varnothing$.*
4. *C is a color function $C : P \mapsto \Sigma$,*
5. *$CT = \{(p, v) \mid p \in P \land v \in C(p)\}$ is the set of all possible colored tokens, and*
6. *F is the transition function such that given a time set $TS = \{x \in \mathbb{R} \mid x \geq 0\}$ and the set of all closed intervals $INT = \{[y, z] \in TS \times TS \mid y \leq z\}$, the multiset (denoted by the superscript X°) of consumed tokens and the multiset of produced tokens for a transition t is mapped by $F(t)$:*

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

   $\operatorname{dom} F(t)$ 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

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

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 [31].

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 [31]. 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 [34, 20, 8, 7]. The Flaw Hypothesis Model (FHM) by Linde and Weissman has been refined since [35, 2, 36] 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 [19], is the use of tree-based mechanisms [26, 27]. 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 [22]. 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 McDermott 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 [22]. 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 [16].

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 [28], 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 [21]). 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 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.

[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`.

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 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

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

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 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 ap-

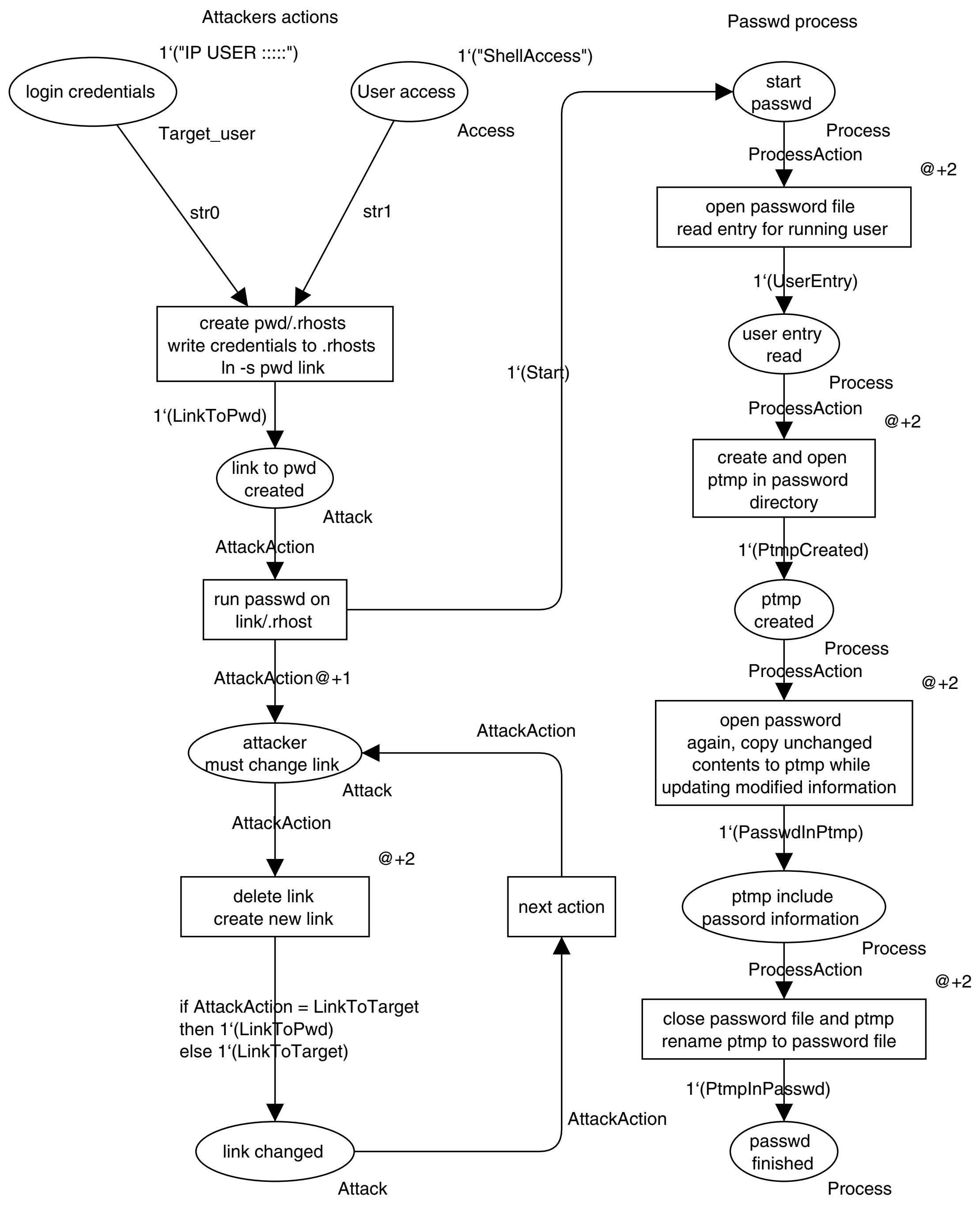

Figure 1. Symbolic link attack. Time constraints are in upper right-hand corner in transitions and after @ symbol in arcs. Color sets are in lower right-hand corner, initial markings in upper right-hand corner of places.

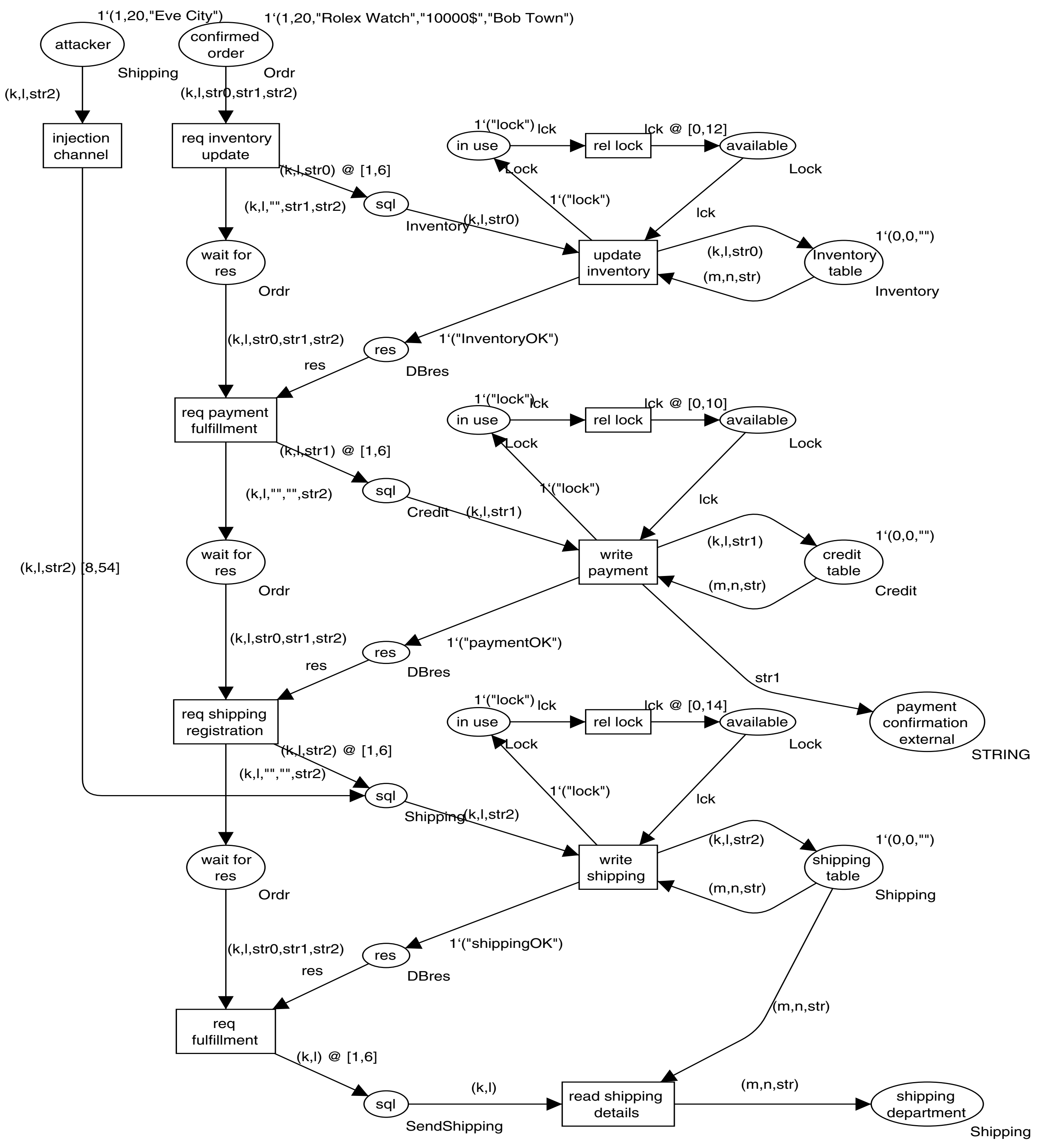

Figure 2. Race condition attack. Semantics: The interval timing constraints are denoted as closed intervals in the arc expressions of output arcs, i.e. an @ followed by the interval.

plications) 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 and network host configurations [33]; 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 3. Subsequently, Steffan and Schumacher [29] 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 [18] 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: Basic Concepts (Volume 1)*. Monographs in Theoretical Computer Science. Springer Verlag, Heidelberg, Germany, 1997.

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

[16] 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.

[17] 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.

[18] 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.

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

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.

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

[26] 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.

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

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

[29] 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.

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

[31] 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.

[32] 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.

[33] 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.

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

[35] 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.

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

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