Chapter

Pull-Type Security Patch Management in Intrusion Tolerant Systems: Modeling and Analysis

Junjun Zheng, Hiroyuki Okamura and Tadashi Dohi

Abstract

In this chapter, we introduce a stochastic framework to evaluate the system availability of an intrusion tolerant system (ITS), where the system undergoes patch management with a periodic vulnerability checking strategy, i.e., pull-type patch management. In particular, a composite stochastic reward net (SRN) is developed to capture the overall system behaviors, including vulnerability discovery, intrusion tolerance, and reactive maintenance operations. Furthermore, two kinds of availability criteria, the interval availability and the steady-state availability of the system, are formulated by applying the phase-type (PH) approximation to solve the Markov regenerative process (MRGP) model derived from the composite SRN. Numerical experiments are conducted to investigate the effects of the vulnerability checking interval on the system availability.

Keywords: intrusion tolerance system, security patch management, vulnerability checking, interval availability, steady-state availability, stochastic reward net, Markov regenerative process, phase expansion

1. Introduction

Computer systems face an increased number of security threats, which exploit the system's potential vulnerability to breach computer security, eventually causing possible damages such as information leakage and economic losses. Software testing is important for ensuring a program's quality, but it is acceptable that perfect software is impossible to achieve. For example, software vulnerabilities are discovered and disclosed continuously, even though developers carefully execute software testing in the development phase [1]. Online vulnerability databases such as MITRE Corporation's Common Vulnerabilities and Exposures (CVE) list[1] and Open Source Vulnerability Database (OSVDB)[2] have reported a vast number of vulnerabilities for recent years. According to CVE, 69,417 vulnerabilities were discovered in web applications over the years 1999–2015 [2]. Due to the existence of vulnerabilities, the risk to cyber

[1] http://www.cve.mitre.org

[2] http://www.osvdb.org

security becomes more significant, and the tricks of attacks also become cleverer and more sophisticated [3]. That means how to guarantee computer security against malicious attacks is a challenging task.

Computer security generally has three attributes; that is, confidentiality, integrity, and availability (CIA) [4]. Two typical techniques, i.e., intrusion detection [5] and intrusion tolerance [6], have been developed and well studied to protect the CIA. Intrusion detection is traditionally used to prevent intrusion as a proactive barrier by monitoring the system behavior. For example, misuse detection is to find the detection signature and anomaly detection is to predict the system's anomaly by comparing normal profiles. Nevertheless, unfortunately, intrusion detection is not still efficient enough to prevent recent and sophisticated malicious attacks. On the other hand, intrusion tolerance is practical to keep the correct services even under attack by masking intrusion based on fault-tolerant techniques for software faults. Some well-known intrusion tolerant systems (ITSs) are, for instance, the SITAR (scalable intrusion tolerant architecture) [7], a concrete ITS architecture using COTS (commercial-off-the-shelf) distributed servers, the BFT-WS [8], a Byzantine fault-tolerant framework for web services providers, and the virtual machine (VM) based ITS, a multistage ITS in virtualized computing environments [9–11].

However, there is no doubt that the most efficient way to ensure computer security is to apply a patch to fix the vulnerable system before a malicious attack occurs. The problem now in patch management from the user's perspective is when to apply the patch because the system may stop while the patch is applied. Even for ITSs, it is essential to decide on an appropriate patch management strategy. Some literature studies have considered such a security patch management from the user's perspective. For example, Kansal et al. [12] presented a generalized framework to identify the optimal patch applying strategy and its minimum cost when the level of system reliability is retained. Uemura et al. [13] focused on typical DoS (denial-of-service) attacks for SITAR and formulated the optimal security patch management policy via semi-Markov models in terms of system availability. In [13], a push-type patch management was considered; that is, the vulnerability information was pushed to a client whenever a new vulnerability was discovered. In the push-type patch management, a patch can be applied just after release. But in fact, for open software projects, such as Apache httpd server, the users need to check the vulnerability information by themselves; that is, pull-type patch management. Therefore, this chapter considers the security patch management of SITAR architecture and discusses the pull-type patch management strategies.

In this chapter, based on two availability measures, we reveal the effect of the number of checking on the system availabilities. More specifically, we develop a composite stochastic reward net (SRN) model [14] with the following four submodules: a vulnerability model to describe the vulnerability discovery process, an intrusion tolerance model to capture the system behaviors under reactive defense strategies after the occurrence of a security failure, a clock model to control the periodic checking interval, and a maintenance model to adopt the preventive and corrective actions for security threats. Also, the phase-type (PH) expansion approach is applied to analyze the Markov regenerative process (MRGP) derived from the SRN to evaluate two kinds of system availabilities. The stationary analysis of MRGP is generally achieved by employing an embedded Markov chain (EMC) approach based on Markov renewal theory [15–18]. Despite this, it is relatively difficult for transient cases. Besides, for the situation where the state in MRGP has multiple competitive transitions timed with generally distributed firing time (GEN transition), it is difficult to analyze the MRGP through Markov renewal equations since it is difficult to use the discretization and supplementary variable method [19]. Therefore, in this chapter, we seek to bridge this gap by developing the solution with PH expansion [19, 20], which is to replace general distributions in MRGP with approximate PH distributions and reduce the original MRGP to an approximate continuous-time Markov chain (CTMC). The accuracy of PH approximation has been validated in [20]. In particular, this chapter utilizes PH expansion of MRGP based on the Kronecker representation.

The remaining part of this chapter is organized as follows. In Section 2, we introduce an overview of an ITS and describe its composite SRN. Section 3 presents the performance analysis through MRGP analysis and PH-expansion CTMC analysis. In particular, the system's interval availability and steady-state availability under patch management are formulated. In Section 4, we present evaluation results. The conclusion and future work are given in Section 5.

2. Intrusion-tolerant system

2.1 System architecture

Consider an intrusion-tolerant architecture as in **Figure 1**, which is the SITAR architecture [7]. In this figure, the part within the denoted box is regarded as an intrusion tolerant architecture that enables us to build intrusion-tolerant servers out of the existing intrusion vulnerable servers $S_1, S_2, \ldots, S_i$. The architecture consists of five critical components: proxy server, acceptance monitor, ballot monitor, adaptive

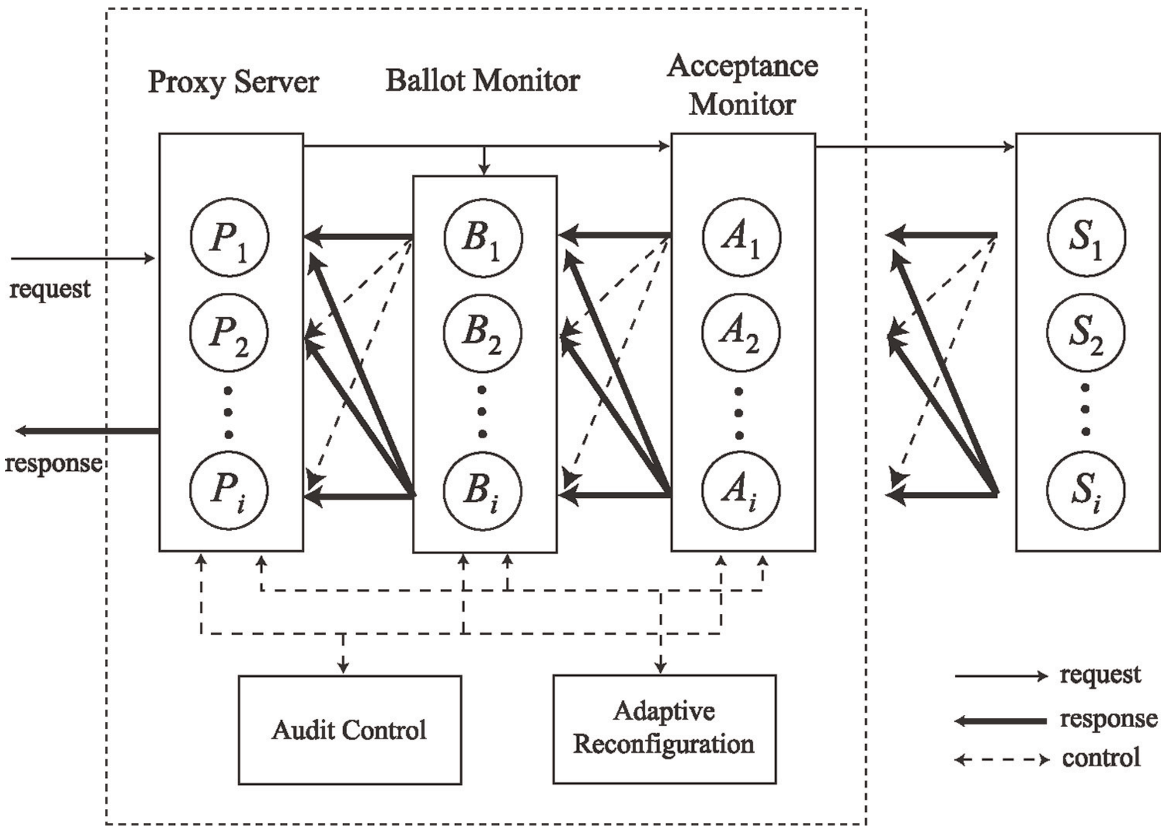


Figure 1.
Intrusion-tolerant architecture.

reconfiguration module, and audit control module. P_i, B_i, and A_i in the functional blocks are the logical functions to be executed to satisfy a given service request.

The proxy servers act as public access points for the services provided. When a request from remote client arrives at one of the proxy servers depending on the service needs, the proxy servers forward the request to one or more COTS servers based on the current intrusion-tolerant strategy. After receiving the COTS servers' responses, the acceptance monitors apply certain validity check to these responses and then forward them along with a check result indication to the ballot monitors. Besides, the acceptance monitors detect the signs of compromised servers and produce intrusion triggers for the adaptive reconfiguration module. The ballot monitors make a final response by either a simple majority voting or Byzantine agreement process and then forward the final response to the proxy servers to be delivered to the remote client.

The audit control module monitors the behaviors of all the other components in the system, by verifying their audit logs. When intrusion is detected, the corresponding information will be sent to the adaptive reconfiguration module. The adaptive reconfiguration module receives intrusion trigger information from all other modules, evaluates the threats, the tolerance objectives, and the cost/performance impact, and finally generates new configurations for the system.

2.2 System behavior

2.2.1 Intrusion tolerance scheme

The system becomes vulnerable once the vulnerability in servers $S_1, S_2, \dots, S_i$ is disclosed. In this state, the system may encounter security threats that exploit discovered vulnerabilities. When a malicious attack arrives, the system moves to the active attack state and attempts to detect the intrusion threat. If the threat is detected successfully, the system begins to diagnose the detected threat and then tries to mask the compromised part; otherwise, security failure occurs and then a recovery process, namely corrective maintenance, is conducted. The system becomes normal again after the recovery ends.

For the case where the intrusion threat is detected successfully and the masking of compromised parts succeeds, the system can continually provide services to users after a minor fix in the background. Once the masking fails, several corrective inspections are tried in parallel with services. If a fatal system error is inspected, the system fails and becomes unavailable. In such a case, a recovery operation is executed to fix a fatal system error. The system goes back to the normal again after the completion of the recovery operation. If a fatal system error is not found, the system can keep servicing with a degraded performance if the attack's damage is not so large, or move to a fail-secure state otherwise, in which the system stops servicing to users. In either case, the system becomes normal after removing the system secure errors.

On the other hand, the system applies security patches if preventive maintenance (i.e., security patch application) is triggered before the attack. After completing the preventive maintenance, the state becomes normal.

2.2.2 Periodic vulnerability checking strategy

Maintenance strategies aim to prevent malicious attacks by executing the security patch application. This chapter considers pull-type patch management with a periodic

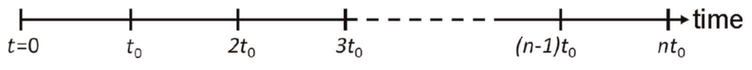


Figure 2.
Periodic vulnerability checking points.

vulnerability checking strategy. **Figure 2** illustrates the periodic checking points for discovered vulnerabilities. The length of one checking interval is given by t_0, and the time points $t_0, 2t_0, \dots, nt_0$ are checking points for deciding whether to implement patches or not. At these checking points, if discovered vulnerabilities exist, the system stops providing services and executes a patch application. Otherwise, the system continues to provide services. The pull-type patch management with a periodic vulnerability checking strategy is described as follows.

Apply the security patch if discovered vulnerabilities exist in the system at the checking points. The length of the checking interval is denoted by $t_0\ (>0)$.

2.3 Stochastic reward net

The SRN is a highly representative model, consisting of: place P, represented by circle; transition T, represented by box; directed arcs, connecting places and transitions; and token(s). A transition is enabled if all of its input places have at least one token. When a transition is enabled, it may be fired to remove one token from each input place and create one token at each output place. Places may be marked by an integer number of tokens. The overall state of a system is represented by a vector consisting of the markings on each place. In SPN, there may exist the following types of transitions; (i) IMM transition (immediate, i.e., they fire in zero time); (ii) EXP transition (timed with exponentially distributed firing time); and (iii) GEN transition (timed with generally distributed firing time). In general, the IMM transition, EXP transition, and GEN transition are often expressed by a thin black bar, a white box, and a thick black bar, respectively. When more than two transitions are enabled simultaneously, guard functions are added to these transitions to control the firing sequence. A transition with a guard function occurs when the value of the guard function is evaluated to be true. The SRN can capture common characteristics of computer systems such as concurrency, synchronization, and sequencing, so it is widely used for stochastic modeling.

In this chapter, we present an SRN with the following submodules for the aforementioned ITS:

1. Vulnerability model, which depicts the vulnerability discovery process.

2. Intrusion tolerance model, which determines the system operation after a security threat occurs.

3. Clock model, which controls the checking interval.

4. Maintenance model, which describes the preventive and corrective actions for security threats.

Figure 3 depicts the composite SRN of the ITS with the pull-type patch management described in 2.2.2.

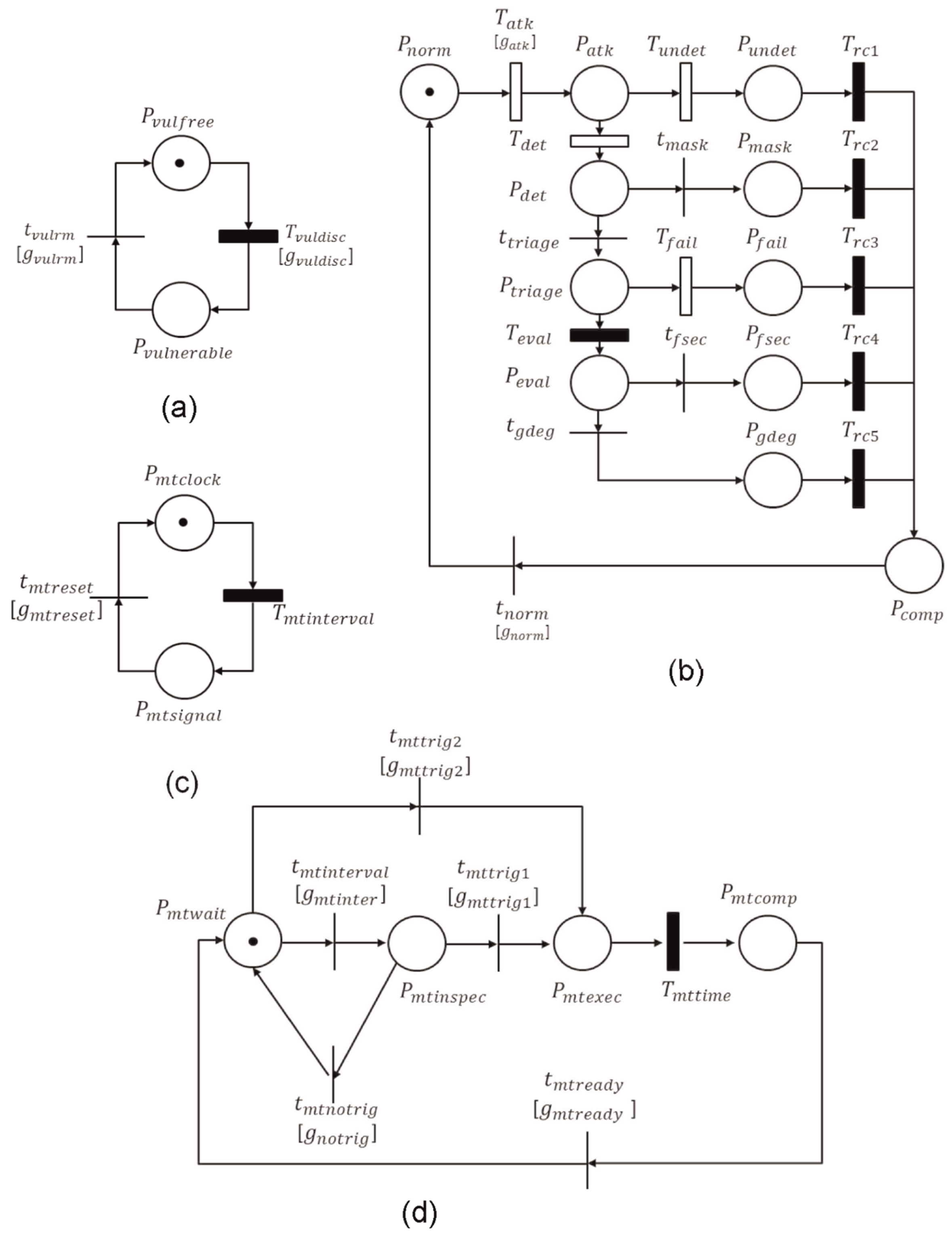


Figure 3.
Composite SRN for the ITS (a) Vulnerability model, (b) intrusion tolerance model, (c) clock model, and (d) maintenance model.

2.3.1 Vulnerability model

Figure 3a depicts an SRN of the vulnerability discovery process. As in **Figure 3a**, the model has two place ($P_{vulfree}$ and $P_{vulnerable}$), one IMM transition (t_{vulrm}) and one EXP transition ($T_{vuldisc}$). A token in $P_{vulfree}$ denotes that the system is vulnerability-free, i.e., no vulnerability has been discovered. When $T_{vuldisc}$ fires, one token is removed from $P_{vulfree}$

and put in $P_{vulnerable}$, which means that the vulnerability is discovered, and the system becomes vulnerable. Once the value of the guard function of t_{vulrm} is true (i.e., the system is under patch application), the system returns the vulnerability-free state immediately.

2.3.2 Intrusion tolerance model

Figure 3b presents an SRN of the intrusion tolerance model, which determines the system operation after a security threat occurs. In this figure, GEN transitions with the generally distributed firing times (represented by thick black bars) are used. Each place and corresponding transition represent the status of progress of an intrusion tolerant process and are given as **Table 1**.

Node	Description
P_{norm}	The system is in a normal state.
P_{atk}	Threat has occurred in the system. The system attempts to detect the threat.
P_{undet}	Threat cannot be detected. The security failure occurs due to the attack and the system is forced to undergo recovery processes.
P_{det}	Threat has been detected. The system begins to diagnosis the detected threat.
P_{mask}	The compromised part is being masked. Concretely, the system provides services to users, though minor errors causing threat are being fixed in the background.
P_{triage}	Threat triage state. Several corrective inspections are tried in parallel with services.
P_{fail}	The system fails and starts a recovery operation to fix a fatal system error.
P_{eval}	The damage of attack is being evaluated.
P_{fsec}	The system becomes fail-secure. The system stops servicing to users and applies recovery operation.
P_{gdeg}	The system keeps servicing while the quality of service is degraded.
P_{comp}	The recovery operation is completed.
T_{atk}	The system is attacked by adversary.
T_{undet}	The threat is undetected.
T_{det}	The threat is detected.
t_{mask}	The compromised part is masked.
t_{triage}	Threat triage begins.
T_{fail}	The system fails.
T_{eval}	The damage of attack is evaluated.
t_{fsec}	The system becomes fail-secure.
t_{gdeg}	The system degrades.
T_{rc1}	The system is in recovery process regarding detection failure.
T_{rc2}	The system is in recovery process regarding masking.
T_{rc3}	The system is in recovery process regarding system failure.
T_{rc4}	The system is in recovery process regarding fail-secure.
T_{rc5}	The system is in recovery process regarding graceful degradation.

Table 1.
*Places and transitions in SPN for intrusion tolerance model (see **Figure 3b**).*

2.3.3 Clock and maintenance models

In this chapter, the security patch application is regarded as the maintenance action. **Figure 3d** and **c** describe the maintenance model and its clock model. As in **Figure 3c**, the clock model controls the checking interval; that is, if a checking point is reached, the transition $T_{mtinterval}$, corresponding to the checking interval t_0, fires, then the token in $P_{mtclock}$ is removed, and a token is put into $P_{mtsignal}$. Upon confirmation that the maintenance model has received the signal of reaching a checking point (i.e., $\#\left(P_{mtinspec}\right) = 1$), the clock is reset with transition $t_{mtreset}$ immediately. On the other hand, from **Figure 3d**, we see that the maintenance model contains four places, one GEN transition, five IMM transitions, and one token in P_{mtwait}, indicating that the system is waiting for a maintenance operation. Besides, a token in $P_{mtinspec}$ represents that the system is checking whether to execute a patch application; once there exists discovered vulnerabilities at the checking point (i.e., the guard function $g_{mttring1}$ is true), the system performs patch application; otherwise, the system continues to wait for the next checking point. A token in P_{mtexec} means that the system is carrying out the maintenance, and the time spent is given by transition T_{mttime}. A token in P_{mtcomp} says that a maintenance is completed, and then the system goes back to the normal state with transition t_{norm} in **Figure 3b** and becomes ready for the next maintenance chance through transition $t_{mtready}$. Note that transition $t_{mttrig2}$ indicates the maintenance triggered due to a security threat.

In these above SRNs, the guard functions are shown in **Table 2**, which determine the enabled timing and are given by the interrelationships between the transition and the corresponding places. A marking of composite SRN is given by a vector that represents the number of tokens for all the places and provides the state of ITS. Actually, the composite SRN can be described by the underlying stochastic process, called MRGP [21], and analyzed by using MRGP analysis based on Markov renewal theory [15, 16]. The MRGP is one of the favored techniques for modeling system behavior with non-Markovian processes, can adequately represent more complex

	Guard function
$g_{vuldisc}$	$\#(P_{mtwait}) = 1$
g_{vulrm}	$\#(P_{mtexec}) = 1$
g_{atk}	$\#(P_{vulnerable}) = 1$
g_{norm}	$\#\left(P_{mtcomp}\right) = 1$
$g_{mtreset}$	$\#\left(P_{mtinspec}\right) = 1\ \#(P_{mtexec}) = 1$
$g_{mtinter}$	$\#\left(P_{mtsignal}\right) = 1$
g_{notrig}	$(\#(P_{norm}) = 0\ \#\left(P_{vulfree}\right) = 1)\ \&\&\ \#(P_{mtclock}) = 1$
$g_{mttrig1}$	$\#(P_{norm}) = 1\ \&\&\ \#(P_{vulnerable}) = 1\ \&\&\ \#(P_{mtclock}) = 1$
$g_{mttrig2}$	$\#\left(P_{comp}\right) = 1$
$g_{mtready}$	$\#(P_{norm}) = 1$

Table 2.
Enabling functions in the composite SRN.

software intrusion tolerant process and maintenance actions, and has been successfully applied in several modeling analyses [16–19].

3. Performance analysis

The performance criteria of interest in this chapter are the interval availability and the steady-state availability of the system, which require the state probabilities of MRGP derived according to the analysis of composite SRN described in 2.3 by using JSPetriNet software package[3]. The MRGP model of ITS is depicted in **Figure 4**. In this figure, the solid lines denote the GEN transitions, whereas the dashed ones denote EXP transitions. In particular, all states except $\mathcal{S}^G_{mtint}$ have two competitive GEN transitions. In such a case, it is difficult to obtain the state probabilities of MRGP through Markov renewal equations because it is hard to use the discretization and supplementary variable method [19]. This chapter, therefore, considers the solution with phase-type (PH) expansion for analyzing the MRGP model of the ITS. Also, in this chapter, we utilize the PH expansion of MRGP based on the Kronecker representation.

3.1 PH approximation

The phase expansion, alternatively PH approximation, is the technique by using PH distribution, which is defined as the probability distribution of the absorbing time in a CTMC with absorbing states. The PH distribution is practical, since it can approximate any probability distribution with high precision. To take benefit from this property, an approximate CTMC can be obtained by replacing probability distribution with PH distributions. Without loss of generality, the infinitesimal generator $\boldsymbol{Q}$ of CTMC is assumed to be partitioned as follows:

$$\boldsymbol{Q} = \begin{pmatrix} \boldsymbol{T} & \boldsymbol{\xi} \\ \boldsymbol{0} & 0 \end{pmatrix}, \tag{1}$$

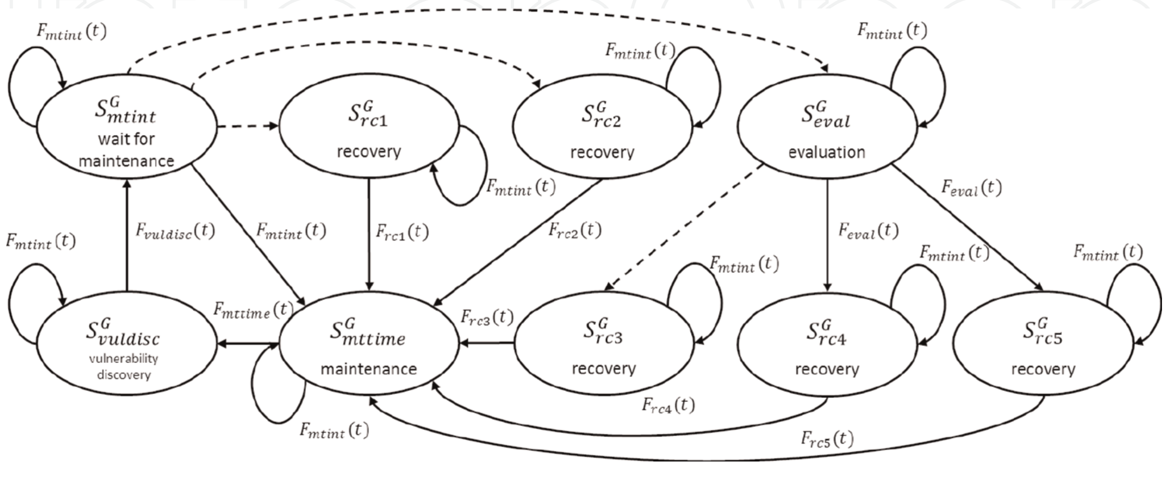


Figure 4.
State transition diagram of ITS with periodic vulnerability checking strategy.

[3] https://github.com/okamumu/JSPetriNet

where $\boldsymbol{T}$ and $\boldsymbol{\xi}$ correspond to transition rates among transient states and the exit rates from transient states to the absorbing state, respectively. Let $\boldsymbol{\alpha}$ be an initial probability vector over the transient states. Then, the cumulative distribution function (c.d.f.) of a PH-distributed variable with representation $(\boldsymbol{\alpha}, \boldsymbol{T})$ and its associated probability density function (p.d.f.) are represented by

$$F_{PH}(t) = 1 - \boldsymbol{\alpha} \exp(\boldsymbol{T}t)\mathbf{1}, \quad f_{PH}(t) = \boldsymbol{\alpha} \exp(\boldsymbol{T}t)\boldsymbol{\xi}, \tag{2}$$

where $\mathbf{1}$ is a column vector whose elements are all 1. Note that the transient states are called *phases*, and the exit rate vector is given by $\boldsymbol{\xi} = -\boldsymbol{T}\mathbf{1}$, according to the property of CTMC. In particular, the accuracy of approximation depends on the number of phases.

In the MRGP shown as in **Figure 4**, the state space is divided into nine classes (more details on MRGP state classification is referred to [18]);

- $\mathcal{S}^G_{mtint}$, consisting of the states where only GEN transition, $T_{mtinterval}$ is enabled.
- $\mathcal{S}^G_{rc1}$, consisting of the states where both GEN transitions, T_{rc1} and $T_{mtinterval}$, are enabled.
- $\mathcal{S}^G_{rc2}$, consisting of the states where both GEN transitions, T_{rc2} and $T_{mtinterval}$, are enabled.
- $\mathcal{S}^G_{rc3}$, consisting of the states where both GEN transitions, T_{rc3} and $T_{mtinterval}$, are enabled.
- $\mathcal{S}^G_{rc4}$, consisting of the states where both GEN transitions, T_{rc4} and $T_{mtinterval}$, are enabled.
- $\mathcal{S}^G_{rc5}$, consisting of the states where both GEN transitions, T_{rc5} and $T_{mtinterval}$, are enabled.
- $\mathcal{S}^G_{eval}$, consisting of the states where both GEN transitions, T_{eval} and $T_{mtinterval}$, are enabled.
- $\mathcal{S}^G_{mttime}$, consisting of the states where both GEN transitions, T_{mttime} and $T_{mtinterval}$, are enabled.
- $\mathcal{S}^G_{vuldisc}$, consisting of the states where both GEN transitions, $T_{vuldisc}$ and $T_{mtinterval}$, are enabled.

The general distributions of GEN transitions, T_x, $x \in \{mtint, rc1, rc2, rc3, rc4, rc5, eval, mttime, vuldisc\}$ are given by $F_x(t)$. In particular, we denote t_0 as the length of one checking interval, following the constant distribution:

$$F_{mtint}(t) = \begin{cases} 0 & t < t_0, \\ 1 & t \geq t_0. \end{cases} \tag{3}$$

That means, the checking interval t_0 is deterministic.

In this chapter, the general distributions are approximated by the following PH distributions:

$$\begin{aligned}
&F_{rc1}(t) \approx 1 - \alpha_1 \exp(\boldsymbol{T}_1 t)\mathbf{1}_1, \quad f_{rc1}(t) \approx \alpha_1 \exp(\boldsymbol{T}_1 t)\boldsymbol{\xi}_1,\\
&F_{rc2}(t) \approx 1 - \alpha_2 \exp(\boldsymbol{T}_2 t)\mathbf{1}_2, \quad f_{rc2}(t) \approx \alpha_2 \exp(\boldsymbol{T}_2 t)\boldsymbol{\xi}_2,\\
&F_{rc3}(t) \approx 1 - \alpha_3 \exp(\boldsymbol{T}_3 t)\mathbf{1}_3, \quad f_{rc3}(t) \approx \alpha_3 \exp(\boldsymbol{T}_3 t)\boldsymbol{\xi}_3,\\
&F_{rc4}(t) \approx 1 - \alpha_4 \exp(\boldsymbol{T}_4 t)\mathbf{1}_4, \quad f_{rc4}(t) \approx \alpha_4 \exp(\boldsymbol{T}_4 t)\boldsymbol{\xi}_4,\\
&F_{rc5}(t) \approx 1 - \alpha_5 \exp(\boldsymbol{T}_5 t)\mathbf{1}_5, \quad f_{rc5}(t) \approx \alpha_5 \exp(\boldsymbol{T}_5 t)\boldsymbol{\xi}_5,\\
&F_{eval}(t) \approx 1 - \alpha_e \exp(\boldsymbol{T}_e t)\mathbf{1}_e, \quad f_{eval}(t) \approx \alpha_e \exp(\boldsymbol{T}_e t)\boldsymbol{\xi}_e,\\
&F_{mttime}(t) \approx 1 - \alpha_m \exp(\boldsymbol{T}_m t)\mathbf{1}_m, \quad f_{mttime}(t) \approx \alpha_m \exp(\boldsymbol{T}_m t)\boldsymbol{\xi}_m,\\
&F_{vuldisc}(t) \approx 1 - \alpha_v \exp(\boldsymbol{T}_v t)\mathbf{1}_v, \quad f_{vuldisc}(t) \approx \alpha_v \exp(\boldsymbol{T}_v t)\boldsymbol{\xi}_v,
\end{aligned} \tag{4}$$

where $\mathbf{1}_1$, $\mathbf{1}_2$, $\mathbf{1}_3$, $\mathbf{1}_4$, $\mathbf{1}_5$, $\mathbf{1}_e$, $\mathbf{1}_t$, and $\mathbf{1}_v$ are the 1's column vectors, and

$$\begin{aligned}
&\boldsymbol{\xi}_1 = -\boldsymbol{T}_1\mathbf{1}_1, \quad \boldsymbol{\xi}_2 = -\boldsymbol{T}_2\mathbf{1}_2, \quad \boldsymbol{\xi}_3 = -\boldsymbol{T}_3\mathbf{1}_3,\\
&\boldsymbol{\xi}_4 = -\boldsymbol{T}_4\mathbf{1}_4, \quad \boldsymbol{\xi}_5 = -\boldsymbol{T}_5\mathbf{1}_5, \quad \boldsymbol{\xi}_e = -\boldsymbol{T}_e\mathbf{1}_e,\\
&\boldsymbol{\xi}_m = -\boldsymbol{T}_m\mathbf{1}_m, \quad \boldsymbol{\xi}_v = -\boldsymbol{T}_v\mathbf{1}_v.
\end{aligned} \tag{5}$$

Let $\mathbf{Q}_{x,x}$, $x \in \{\ (i)\ mtint,\ (1)\ rc1,\ (2)\ rc2,\ (3)\ rc3,\ (4)\ rc4,\ (5)\ rc5,\ (e)\ eval,\ (m)\ mttime,\ (v)\ vuldisc\}$ be the infinitesimal generator matrix of non-regenerative transitions of $\mathcal{S}_x^G$. The CTMC transition rate matrix from $\mathcal{S}_x^G$ to $\mathcal{S}_y^G$ is denoted by $\mathbf{Q}_{x,y}$. On the other hand, $\boldsymbol{A}_{x,y}^k$ denote the regenerative transitions from $\mathcal{S}_x^G$ to $\mathcal{S}_y^G$ triggered by transition T_k with probability $F_k(t)$, $k \in \{mtint, rc1, rc2, rc3, rc4, rc5, eval, mttime, vuldisc\}$.

Then by taking account of one checking interval t_0, the MRGP process during this interval can be approximated by the CTMC with the following infinitesimal generator as in Eq. (6), in which $\otimes$ and $\oplus$ are Kronecker product and sum. Apparently, the transition probability triggered by transition $T_{mtinterval}$ in **Figure 3c** with probability $F_{mtint}(t)$ is given by Eq. (7). In this equation, $\boldsymbol{I}$ is an identity matrix.

$$\mathbf{Q} = \begin{pmatrix}
\mathbf{Q}_{i,i} & \mathbf{Q}_{i,1}\otimes\boldsymbol{\alpha}_1 & \mathbf{Q}_{i,2}\otimes\boldsymbol{\alpha}_2 & & & & \mathbf{Q}_{i,e}\otimes\boldsymbol{\alpha}_e & & \\
 & \mathbf{Q}_{1,1}\oplus\boldsymbol{T}_1 & & & & & & & \boldsymbol{A}_{1,m}^{rc1}\otimes(\boldsymbol{\xi}_1\alpha_m)\\
 & & \mathbf{Q}_{2,2}\oplus\boldsymbol{T}_2 & & & & & & \boldsymbol{A}_{2,m}^{rc2}\otimes(\boldsymbol{\xi}_2\alpha_m)\\
 & & & \mathbf{Q}_{3,3}\oplus\boldsymbol{T}_3 & & & & & \boldsymbol{A}_{3,m}^{rc3}\otimes(\boldsymbol{\xi}_3\alpha_m)\\
 & & & & \mathbf{Q}_{4,4}\oplus\boldsymbol{T}_4 & & & & \boldsymbol{A}_{4,m}^{rc4}\otimes(\boldsymbol{\xi}_4\alpha_m)\\
 & & & & & \mathbf{Q}_{5,5}\oplus\boldsymbol{T}_5 & & & \boldsymbol{A}_{5,m}^{rc5}\otimes(\boldsymbol{\xi}_5\alpha_m)\\
 & & & \mathbf{Q}_{e,3}\otimes(\mathbf{1}_e\boldsymbol{\alpha}_3) & \boldsymbol{A}_{e,4}^{eval}\otimes(\boldsymbol{\xi}_e\boldsymbol{\alpha}_4) & \boldsymbol{A}_{e,5}^{eval}\otimes(\boldsymbol{\xi}_e\boldsymbol{\alpha}_5) & \mathbf{Q}_{e,e}\oplus\boldsymbol{T}_e & & \\
\boldsymbol{A}_{v,i}^{vuldisc}\otimes\boldsymbol{\xi}_v & & & & & & & \mathbf{Q}_{v,v}\oplus\boldsymbol{T}_v & \\
 & & & & & & & \boldsymbol{A}_{m,v}^{mttime}\otimes(\boldsymbol{\xi}_m\alpha_v) & \mathbf{Q}_{m,m}\oplus\boldsymbol{T}_m
\end{pmatrix}. \tag{6}$$

$$\boldsymbol{P} = \begin{pmatrix}
\boldsymbol{A}_{i,i}^{mtint} & & & & & & & & \boldsymbol{A}_{i,m}^{mtint}\otimes\boldsymbol{\alpha}_m\\
 & \boldsymbol{A}_{1,1}^{mtint}\otimes\boldsymbol{I} & & & & & & & \\
 & & \boldsymbol{A}_{2,2}^{mtint}\otimes\boldsymbol{I} & & & & & & \\
 & & & \boldsymbol{A}_{3,3}^{mtint}\otimes\boldsymbol{I} & & & & & \\
 & & & & \boldsymbol{A}_{4,4}^{mtint}\otimes\boldsymbol{I} & & & & \\
 & & & & & \boldsymbol{A}_{5,5}^{mtint}\otimes\boldsymbol{I} & & & \\
 & & & & & & \boldsymbol{A}_{e,e}^{mtint}\otimes\boldsymbol{I} & & \\
 & & & & & & & \boldsymbol{A}_{v,v}^{mtint}\otimes\boldsymbol{I} & \\
 & & & & & & & & \boldsymbol{A}_{m,m}^{mtint}\otimes\boldsymbol{I}
\end{pmatrix}. \tag{7}$$

We next consider the checking point when the transition $T_{mtinterval}$ fires with the probability $F_{mtint}(t)$, then the underlying process is actually an EMC with only one subspace that consists of the states where only GEN transition $T_{mtinterval}$ is enabled. Thus, the transition matrix on this regeneration point regarding $F_{mtint}(t)$ is given by

$$\boldsymbol{P}^{EMC} = \exp(\boldsymbol{Q}t_0)\boldsymbol{P}. \tag{8}$$

3.2 Availability measures

It is well known that availability is an important metric commonly used to assess the performance of repairable systems by considering both the reliability and maintainability properties of computer systems. There exist many classifications and definitions of availability, and they are used for different system environments properly. For example, when the system has a long lifetime, the steady-state availability [22] is appropriate to represent the system performance. On the other hand, when one wishes to ensure the system performance for a specific time period, the interval availability [23, 24] may be chosen to present the proportion of time during a mission or time period that the system is available for use. In this chapter, we focus on two availability criteria: interval availability and steady-state availability of the system. The interval availability is defined as the expected fraction of a given interval of time that the system is operational and is appropriate when one wishes to ensure the system availability for a specific time period. On the other hand, the steady-state availability is the limiting availability and is appropriate when the targeted system is continuously operated for a long time.

3.2.1 Interval availability

Let $\boldsymbol{\pi}_0$ denote the initial probability vector of the PH-expanded CTMC. Without loss of generality, it is assumed that the system starts at time $t = 0$. For the time interval $(0, nt_0]$, the interval availability is given by

$$\begin{aligned} A_{in}^{(n)} = \frac{1}{nt_0}(&\boldsymbol{\pi}_0 + \boldsymbol{\pi}_0\boldsymbol{P}^{EMC} + \boldsymbol{\pi}_0\boldsymbol{P}^{EMC2} + \\ &\cdots + \boldsymbol{\pi}_0\boldsymbol{P}^{EMC(n-1)}) \textstyle\int_0^{t_0} \exp(Qs)ds\boldsymbol{r}. \end{aligned} \tag{9}$$

In the above equation, $\boldsymbol{r}$ is the reward vector of the PH-expanded CTMC, and defined as

$$\boldsymbol{r} = \begin{pmatrix} \boldsymbol{r}_{mtint} \\ \boldsymbol{r}_{rc1} \otimes \boldsymbol{1}_1 \\ \boldsymbol{r}_{rc2} \otimes \boldsymbol{1}_2 \\ \boldsymbol{r}_{rc3} \otimes \boldsymbol{1}_3 \\ \boldsymbol{r}_{rc4} \otimes \boldsymbol{1}_4 \\ \boldsymbol{r}_{rc5} \otimes \boldsymbol{1}_5 \\ \boldsymbol{r}_{eval} \otimes \boldsymbol{1}_e \\ \boldsymbol{r}_{vuldisc} \otimes \boldsymbol{1}_v \\ \boldsymbol{r}_{mttime} \otimes \boldsymbol{1}_m \end{pmatrix}, \tag{10}$$

where $\boldsymbol{r}_i$ is the reward vector of system states belonging to corresponding subspace. For example, the interval availability within the first checking interval becomes

$$A_{in}^{(1)} = \frac{1}{t_0}\boldsymbol{\pi}_0\int_0^{t_0} \exp(Qs)ds\boldsymbol{r}. \tag{11}$$

3.2.2 Steady-state availability

Using Eq. (8), the steady-state probability distribution $\boldsymbol{\pi}^{EMC} = (\boldsymbol{\pi}_{mtint}^{EMC}, \boldsymbol{\pi}_{rc1}^{EMC}, \boldsymbol{\pi}_{rc2}^{EMC}, \boldsymbol{\pi}_{rc3}^{EMC}, \boldsymbol{\pi}_{rc4}^{EMC}, \boldsymbol{\pi}_{rc5}^{EMC}, \boldsymbol{\pi}_{eval}^{EMC}, \boldsymbol{\pi}_{vuldisc}^{EMC}, \boldsymbol{\pi}_{mttime}^{EMC})$ can be computed by solving the following linear equation:

$$\boldsymbol{\pi}^{EMC} = \boldsymbol{\pi}^{EMC}\boldsymbol{P}^{EMC}, \quad \boldsymbol{\pi}^{EMC}\mathbf{1} = 1, \tag{12}$$

where $\mathbf{1}$ is a column vector whose elements are 1.

Finally, we obtain the steady-state availability of the system:

$$A_{ss} = \boldsymbol{\pi}^{EMC}\boldsymbol{r}. \tag{13}$$

4. Numerical experiments

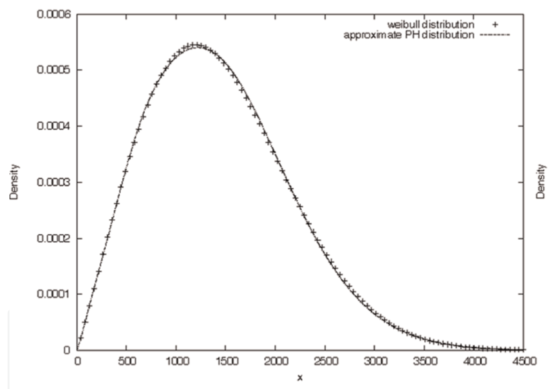
This section evaluates the interval availability and steady-state availability of the system, where the system undergoes the pull-type patch management with a periodic vulnerability checking strategy. **Table 3** gives the parameters for EXP transitions in

Parameter	Description	Value [hrs.]
$1/T_{atk}.rate$	Mean time to complete an intrusion	1200
$1/T_{undet}.rate$	Mean time passed since detection start and detection failure	8
$1/T_{det}.rate$	Mean time to detect an intrusion	12
$1/T_{fail}.rate$	Mean time to failure of a triage	6

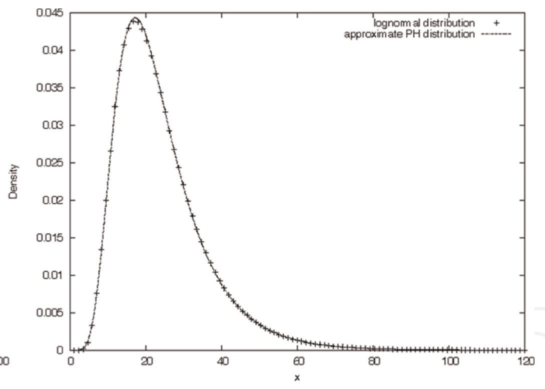
Table 3.
Model parameters.

Notation	Transition	Distribution	Mean [hrs.]	CV
$F_{vuldisc}(t)$	$\mathcal{S}_{vuldisc}^{G}$ to $\mathcal{S}_{mtint}^{G}$	Weibull	1440	0.5
$F_{rc1}(t)$	$\mathcal{S}_{rc1}^{G}$ to $\mathcal{S}_{mttime}^{G}$	Lognormal	24	0.5
F_{rc2}	$\mathcal{S}_{rc2}^{G}$ to $\mathcal{S}_{mttime}^{G}$	Lognormal	12	0.5
$F_{rc3}(t)$	$\mathcal{S}_{rc3}^{G}$ to $\mathcal{S}_{mttime}^{G}$	Lognormal	48	0.5
$F_{rc4}(t)$	$\mathcal{S}_{rc4}^{G}$ to $\mathcal{S}_{mttime}^{G}$	Lognormal	30	0.5
$F_{rc5}(t)$	$\mathcal{S}_{rc5}^{G}$ to $\mathcal{S}_{mttime}^{G}$	Lognormal	40	0.5
$F_{eval}(t)$	$\mathcal{S}_{eval}^{G}$ to $\mathcal{S}_{rc4}^{G}$ $(\mathcal{S}_{rc5}^{G})$	Lognormal	8	0.5
$F_{mttime}(t)$	$\mathcal{S}_{mttime}^{G}$ to $\mathcal{S}_{mtint}^{G}$	Lognormal	10	0.5

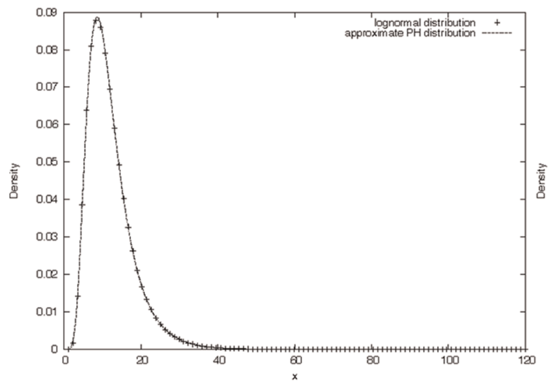
Table 4.
Probability distributions of GEN transitions.



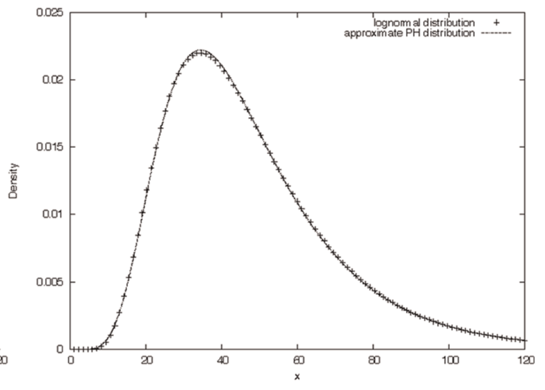
(a)



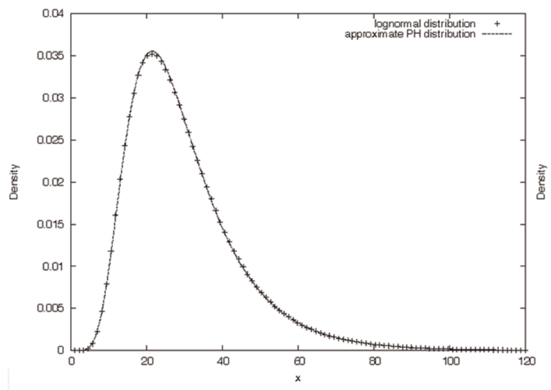
(b)



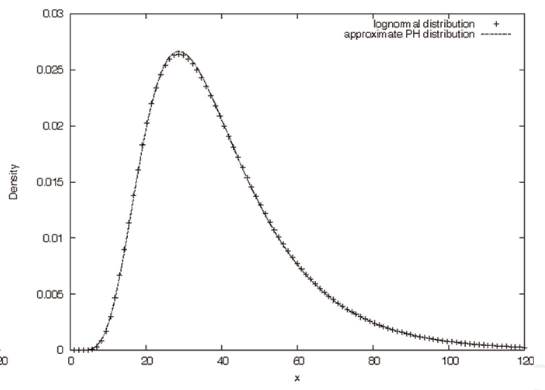
(c)



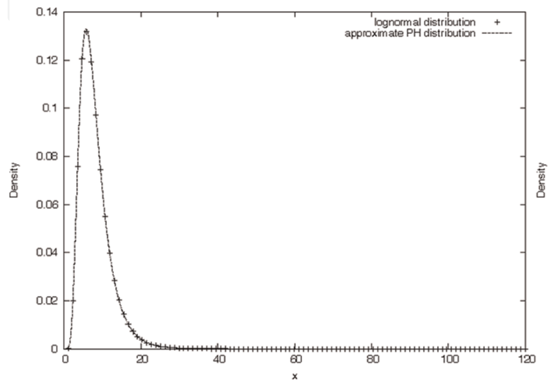
(d)



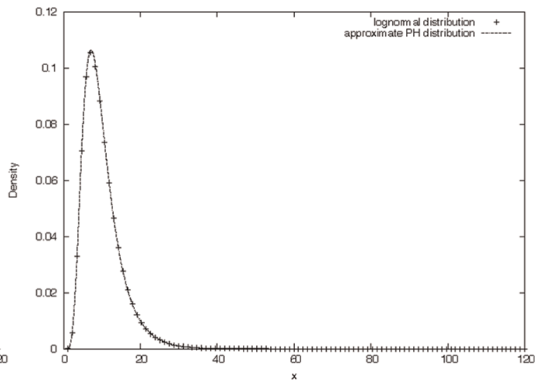
(e)



(f)



(g)



(h)

Figure 5.
Approximate PH distributions ((a) $F_{vuldisc}(t)$, (b) $F_{rc1}(t)$, (c) $F_{rc2}(t)$, (d) $F_{rc3}(t)$, (e) $F_{rc4}(t)$, (f) $F_{rc5}(t)$, (g) $F_{eval}(t)$, (h) $F_{mttime}(t)$).

Figure 3. The probability distributions for GEN transitions $T_{vuldisc}$, T_{rc1}, T_{rc2}, T_{rc3}, T_{rc4}, T_{rc5}, T_{eval}, and T_{mttime} are given in **Table 4**, where the columns of "Mean" and CV represent the mean time and the coefficient of variation, respectively.

Figure 5a–h draw the original probability distributions for GEN transitions and the approximate PH distributions with 10 phases. These figures indicate that the PH distributions are accurate enough to approximate the general distributions.

To investigate the effect of the number of checking, we consider the number of checking during 1 year, N, varying from 4 to 36. For example, in the case of $N = 4$, the length of one checking interval is 3 months. In the case of $N = 36$, the length of one checking interval is about 10 days.

Figure 6 depicts the interval availability of the system, which increases monotonically as the number of checking, N, increases. In particular, the interval availability increases sharply when the number of checking is very small. In such a case, the length of one checking interval decreased remarkably; for example, when $N = 4$, it

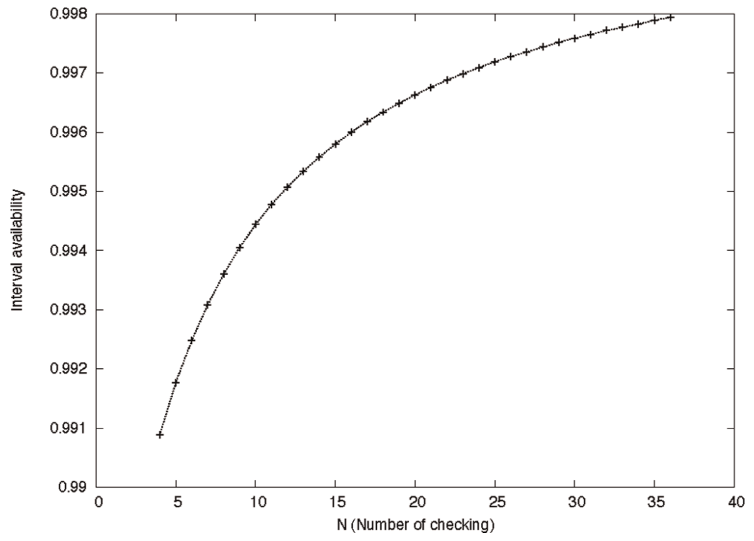


Figure 6.
Sensitivity of the number of checking on the interval availability.

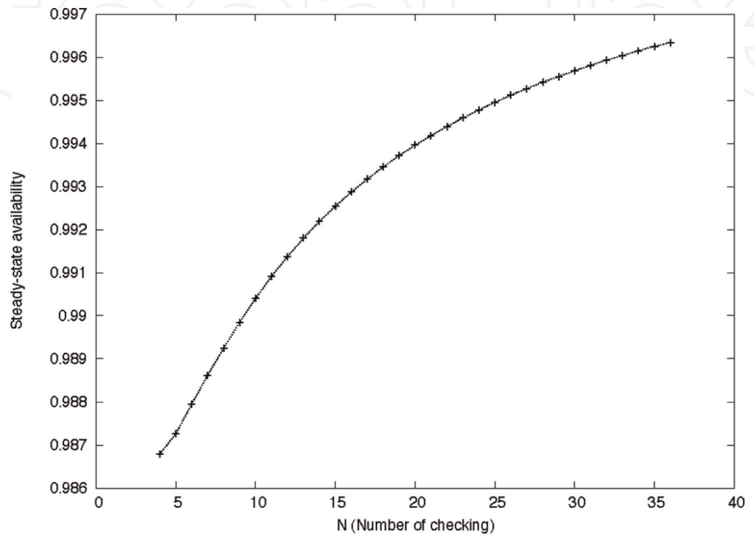


Figure 7.
Sensitivity of the number of checking on the steady-state availability.

N	t_0 [days]	Interval availability	Steady-state availability
4	91.3	0.99088	0.98679
5	73.0	0.99177	0.98726
6	60.8	0.99248	0.98795
7	52.1	0.99308	0.98862
8	45.6	0.99360	0.98925
9	40.6	0.99405	0.98985
10	36.5	0.99444	0.99040
11	33.2	0.99478	0.99091
12	30.4	0.99507	0.99137
13	28.1	0.99534	0.99180
14	26.1	0.99558	0.99219
15	24.3	0.99580	0.99254
16	22.8	0.99600	0.99287
17	21.5	0.99618	0.99317
18	20.3	0.99634	0.99345
19	19.2	0.99649	0.99372
20	18.3	0.99663	0.99396
21	17.4	0.99676	0.99418
22	16.6	0.99688	0.99439
23	15.9	0.99699	0.99459
24	15.2	0.99709	0.99478
25	14.6	0.99719	0.99495
26	14.0	0.99728	0.99512
27	13.5	0.99736	0.99527
28	13.0	0.99744	0.99542
29	12.6	0.99752	0.99555
30	12.2	0.99759	0.99569
31	11.8	0.99765	0.99581
32	11.4	0.99772	0.99593
33	11.1	0.99777	0.99604
34	10.7	0.99783	0.99615
35	10.4	0.99789	0.99625
36	10.1	0.99794	0.99634

Table 5.
The number of checking per year and its corresponding length of checking interval and availabilities.

takes almost 3 months to execute a checking operation, whereas the checking interval reduces to 2.4 months in the case of $N = 5$. However, when N increases from 35 to 36, the checking interval almost does not change. Besides, it is intuitively obvious that a shorter checking interval generally brings higher availability. Therefore, when N is a small value, the interval availability is very sensitive to the change in the value of N.

On the other hand, the steady-state availability of the system is shown in **Figure 7**. From this figure, it is found that the steady-state availability also increases as the number of checking, N, increases. Furthermore, more details on the experimental results about the number of checking per year and its corresponding length of one checking interval and availabilities are referred to **Table 5**. From this table, we can see that for any given N, the interval availability is higher than the steady-state availability.

5. Conclusion and future work

In this chapter, we presented a stochastic model to evaluate the system availability of an ITS, where the system undergoes the patch management with a periodic vulnerability checking strategy; that is, pull-type patch management. Concretely, a composite SRN model was developed to capture the overall system behaviors, including vulnerability discovery, intrusion tolerance, and reactive maintenance. Two kinds of availability criteria, the interval and steady-state availabilities, were formulated by using phase expansion. In numerical experiments, we evaluated the effect of the checking number on the system availability, and the results imply that when the checking number is small (a long checking interval), the variation in the checking number brings an significant effect into the interval availability. In addition, both interval availability and steady-state availability increase monotonically as the number of checking increases. We have also validated the accuracy of the PH approximation with 10 phases.

The chapter aims is to present a method for formulating the system availability from both transient and stationary points of view and evaluate the effect of the number of checking on the system availability. Nevertheless, it is actually well known that one of the main issues in the design of security patch management is to determine the optimal length of checking interval bringing the optimal trade-off between system performance and checking cost. For example, if the checking interval is too short, the system availability will be high, but the total checking cost will be very high. On the other hand, if the checking interval is too long, a discovered vulnerability may be exploited by malicious attacks, which decreases the system availability; in this case, the checking cost can be reduced, but the total cost due to security failures will be high. Therefore, it will be interesting, as one of future directions, to find the optimal checking number (i.e., optimal checking policy) by the consideration of both system performance and maintenance cost.

Acknowledgements

This chapter is an extension of work originally reported at the 2018 42nd IEEE International Conference on Computer Software & Applications (COMPSAC'18) [25]. Moreover, this work was supported by JSPS KAKENHI Grant Number 21 K17742.

Conflict of interest

The authors declare no conflict of interest.

Nomenclature

ITS	Intrusion tolerant system
SRN	Stochastic reward net
PH	Phase-type
MRGP	Markov regenerative process
CIA	Confidentiality, integrity, and availability
SITAR	Scalable intrusion tolerant architecture
COTS	Commercial-off-the-shelf
VM	Virtual machine
DoS	Denial-of-service
EMC	Embedded Markov chain
CTMC	Continuous-time Markov chain
GEN	Generally distributed
EXP	Exponentially distributed
c.d.f.	Cumulative distribution function
p.d.f.	Probability density function
CV	Coefficient of variation

Author details

Junjun Zheng[1]*, Hiroyuki Okamura[2] and Tadashi Dohi[2]

1 Department of Information Science and Engineering, Ritsumeikan University, Kusatsu, Japan

2 Graduate School of Advanced Science and Engineering, Hiroshima University, Hiroshima, Japan

*Address all correspondence to: jzheng@fc.ritsumei.ac.jp

References

[1] Arora A, Krishnan R, Telang R, Yang Y. An empirical analysis of software vendors' patch release behavior: Impact of vulnerability disclosure. Information Systems Research. 2010;**21**(1):115-132

[2] Abunadi I, Alenezi M. An empirical investigation of security vulnerabilities with web applications. Journal of Universal Computer Science. 2016; **22**(4):537-551

[3] Khan YI, Al-Shaer E, Rauf U. Cyber resilience-by-construction: Modeling, measuring & verifying. In: Proceedings of 2015 Workshop on Automated Decision Making for Active Cyber Defense. Denver, Colorado, USA: ACM; 2015. pp. 9-14

[4] Jansen W. Directions in Security Metrics Research. Darby, PA, USA: DIANE Publishing Co; 2010

[5] Mukkamala S, Janoski G, Sung A. Intrusion detection using neural networks and support vector machines. In: Proceedings of 2002 International Joint Conference on Neural Networks (IJCNN'02). Honolulu, HI, USA; 2002. pp. 1702-1707

[6] Stavridou V, Dutertre B, Riemenschneider RA, Saidi H. Intrusion tolerant software architectures. In: Proceedings of Darpa Information Survivability Conference and Exposition (DISCEX II'01). Anaheim, California, USA: IEEE; 2001. pp. 230-241

[7] Wang F, Gong F, Sargor C, Goševa-Popstojanova K, Trivedi KS, Jou F. SITAR: A scalable intrusion-tolerant architecture for distributed services. In: Proceedings of the 2nd Annual IEEE Systems, Man and Cybernetics Information Assurance Workshop (SMC-IAW'01). New York, USA: IEEE; 2001

[8] Zhao W. BFT-WS: A Byzantine fault tolerance framework for web services. In: Proceeding of the 11th International IEEE EDOC Conference Workshop (EDOC'07). Annapolis, MD, USA: IEEE; 2007. pp. 89-96

[9] Junior VS, Lung LC, Correia M, Fraga JDS, Lau J. Intrusion tolerant services through virtualization: A shared memory approach. In: Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA'10). Perth, Australia: IEEE; 2010. pp. 768-774

[10] Lau J, Barreto L, Fraga JDS. An infrastructure based in virtualization for intrusion tolerant services. In: Proceedings of the 19th IEEE International Conference on Web Services (ICWS'12). Honolulu, HI, USA: IEEE; 2012. pp. 170-177

[11] Zheng J, Okamura H, Dohi T. Survivability analysis of VM-based Intrusion tolerant systems. IEICE Transactions on Information and Systems. 2015;**E-98**(12):2082-2090

[12] Kansal Y, Kapur PK, Kumar D. Assessing optimal patch release time for vulnerable software systems. In: Proceedings of 2016 International Conference on Innovation and Challenges in Cyber Security. Greater Noida, India: IEEE; 2016. pp. 308-314

[13] Uemura T, Dohi T, Kaio N. Availability analysis of an Intrusion tolerant distributed server system with preventive maintenance. IEEE Transactions on Reliability. 2010;**59**(1): 18-29

[14] Wang D, Madan BB, Trivedi KS. Security Analysis of SITAR intrusion tolerance system. In: Proceedings of the

2003 ACM Workshop Survivable and Self-regenerative Systems: in association with 10th ACM Conference on Computer and Communications Security (SSRS'03). Fairfax, VA, USA: ACM; 2003. pp. 23-32

[15] Çinlar E. Introduction to Stochastic Processes. Englewood Cliffs, NJ, USA: Prentice-Hall Inc; 1975

[16] Fricks R, Telek M, Puliafito A, Trivedi KS. Markov renewal theory applied to performability evaluation. In: Bagchi K, Zobrist G, editors. State-of-the Art in Performance Modeling and Simulation. Modeling and Simulation of Advanced Computer Systems: Applications and Systems. Amsterdam, The Netherlands: Gordon and Breach Publishers; 1998. pp. 193-236

[17] Garg S, Pfening S, Puliafito A, Telek M, Trivedi KS. Analysis of preventive maintenance in transaction based software systems. IEEE Transactions on Computers. 1998;**47**(1): 96-107

[18] Zheng J, Okamura H, Li L, Dohi T. A comprehensive evaluation of software rejuvenation policies for transaction systems with MarMarkov arrival. IEEE Transactions on Reliability. 2017;**66**(4): 1157-1177

[19] Okamura H, Yamamoto K, Dohi T. Transient analysis of software rejuvenation policies in virtualized system: phase-type expansion approach. Quality Technology & Quantitative Management. 2014;**11**(3):335-351

[20] Okamura H, Dohi T. A phase expansion approach for transient analysis of software rejuvenation model. In: Proceedings of the 8th International Workshop on Software Aging and Rejuvenation (WoSAR'16). Ottawa, Canada: IEEE; 2016. pp. 98-103

[21] Choi H, Kulkarni VG, Trivedi KS. Markov regenerative stochastic Petri nets. Performance Evaluation. 1994;**20**: 337-357

[22] Hosford JE. Measures of dependability. Operations Research. 1960;**8**(1):53-64

[23] Rubino G, Sericola B. Interval availability analysis using operational periods. Performance Evaluation. 1992; **14**(3–4):257-272

[24] Smith M, Aven T, Dekker R, van der Duyn Schouten FA. A survey on the interval availability distribution of failure prone systems. In: Advances in Safety and Reliability: Proceedings of ESREL'97. Oxford: Elsevier; 1997. pp. 1727-1737

[25] Zheng J, Okamura H, Dohi T. A pull-type security patch management of an intrusion tolerant system under a periodic vulnerability checking strategy. In: Proceedings of the 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC'18). Tokyo, Japan: IEEE; 2018. pp. 630-635