Canadian Journal of Pure and Applied Sciences
Vol. 9, No. 3, pp. 3629-3634, October 2015
Online ISSN: 1920-3853; Print ISSN: 1715-9997
Available online at www.cjpas.net

RSPS: MODELING AND VERIFICATION OF A REAL-TIME SECURITY PROTECTION SERVICE USING MODEL CHECKING

*Alireza Souri[1], Solmaz Abdollahizad[2], Majid Samad Zamini[2] and Adalat Safarkhanlou[1]
[1] Department of Computer Engineering, Hadishahr Branch, Islamic Azad University, Hadishahr, Iran
[2] Department of Computer Engineering, Sardroud Branch, Islamic Azad University, Sardroud, Iran

ABSTRACT

In this paper a real-time security protection model for scanning the security files is presented. In this model, a workflow mechanism is presented for real-time scanning Dynamic Link Library files. A specification relation between the proposed model and the Kripke structure is presented that enables the verification of the system specifications. By presenting the appropriate formal semantics, we discuss that how labeling functions permits navigating information and specifications of the security system. We illustrated expected properties of the system which can be verified and specified by using temporal logic. So, we defined satisfaction relations for verifying the system specifications. We also described how some of expected properties of the system are verified. Finally, we implemented some properties of proposed model in NuSMV model checker. The verification results show that our proposed real-time security protection model is reachable, deadlock free and fair.

Keywords: Real-time security protection service, dynamic link library, model checking, NuSMV.

INTRODUCTION

When an application or some executable files access to a dynamic link library (DLL) file without specifying an exactly path to find it, the windows operation system searches the needed DLL file in a set of predetermined paths. If there is a destructive file of homonymous whit this DLL file in one of the predetermined paths, the application runs destructive file undesirably. So, correctness of the scanning operations in real-time protection service is very important. For increasing performance of real-time protection service, all files are scanned for viruses or unwanted programs, irrespective of their content and their file extension. These DLL files may be called by CPU in first fetch or middle fetch of the paths. As a destructive file of homonymous whit these DLL files may be entered in fetching of paths before running main file of application and the virus can enable in the system. So, the real-time protection service should scan files before opening, reading and executing and after writing that this procedure cause antivirus prevents to influx of each malware and infected file to windows the system directory.

Currently, antivirus systems (Szor, 2005) have an essential situation in software development. Every computer needs to a security software for protecting and maintaining its data and applications. Now days, several attacks (Wang *et al*., 2011) are happened to critical systems, bank servers and military systems via Viruses and Malwares. Information maintenance and prevention from unauthorized data access is main reasons for using antimalware against attacks and destroying data which has been occurred by invasive malware (Zhang *et al*., 2010) widely and suddenly. Du To some specific problems, verifying the security applications such as the antivirus systems are very important and essential in Security Discussion (Schneider, 2000) of computer systems. Of course, computer viruses (Singh and Lakhotia, 2002), Spywares (Filiol, 2010), Trojans, Worms (Sellke *et al*., 2008) and other new malwares debut every day.

In this paper, we present a model for a Real-time Security Protection Service (RSPS) which has all of important properties of antivirus applications and these properties are important for verification (Zhiqiao *et al*., 2012). The proposed model has focused on maintaining secure state of the system. We convert the proposed model to a Kripke structure (Edmund *et al*., 1999) by using formal verification (Schlipf *et al*., 1997).

Formal methods, supporting tools and theory can help to the analysis, design and verification of the security-related and internet security protocols used over open networks and distributed systems. The most commonly followed techniques for the application of formal methods for the ex-post analysis and verification of internet security and

*Corresponding author e-mail: alirezasouri.research@gmail.com

antivirus systems as the analysis approach are reviewed (Gritzalis *et al*., 1999; Yasinsac and Childs, 2005), followed by the examination of robustness principles and application limitations. Formal verification and model checking techniques can be used for automatically analyzing antivirus and security systems (Gritzalis *et al*., 1999). Antivirus model is used in designing antivirus software and verification of its functional properties**.** Formalization and verifying a security model using formal verification methods is a new idea in software development. In following, we show some related works in this topic.

Morales *et al*. (2006) presented a formal model of virus transformation that enables variation traceability using four antivirus solutions for handheld devices. They tested proposed formal model of antivirus software under attack of some viruses. They presented formalized antivirus model for testing some solutions against virus attacks.

Another study, Andronick *et al*. (2005) considered a new approach for verification of a smart card embedded operating system. They proved a C source program against supplementary annotations and generated a high-level formal model of the annotated C program that was used to verifying certain global security properties. This paper is focused on modeling smart card security in embedded source codes. Heitmeyer *et al*. (2008) presented verifying a system's high-level security properties. Their approach is focused on computer security by using antivirus systems rather than security properties of software systems. Safarkhanlou *et al*. (2015) proposed an antivirus protection service which has two protection modes: PC protection and Internet Protection. They modeled the proposed service using a state transition diagram. They verified proposed model using NuSMV model checker. Recently, Souri and Navimipour (2013) proposed an adapted resource discovery approach to address multi-attribute queries in grid computing. They presented a behavioral model for their proposed approach that separate into data gathering, discovery and control behaviors. So, they used to Kripke structure for modeling these behaviors and verify their behavioral models by using NuSMV model checker.

In the present study first we discuss a security protection service according to dynamic link library (DLL) functions. Also, this study presents how states of the security protection service convert to a state chart model. By using formal verification techniques, we describe how the model of the system is verified by using Computation Tree Logic (CTL) rules. So, we formalize some example of specification rules the paths. Following this the proposed model implementation is presented using NuSMV model checker. Finally, conclusion and future work are provided.

Real-time Security Protection Service

In this section, we present a real-time protection service and describe how the model of security protection service is formulated. Then by using formulated model, we present a Kripke structure for antivirus model. A Kripke structure is sufficiently indicative many aspects of the system behaviors which are important for reasoning about verifying the systems (Clarke *et al*., 1999).

One of the important functions in DLL files accessibility is DLLMain function. This function is an optional method of entry into a DLL. Also, this function has some methods for providing access to DLL files. We present one of the main methods for scanning created paths on real-time protection service. By using this function, it is calling by the system when processes and threads are initialized and terminated. Also, these call use to the LoadLibrary and FreeLibrary functions that allow executable files access to its specific .dll file in LoadLibrary and FreeLibrary. The DllMain takes two parameters for itself: *hinstDLL* and *dwReason*.*hinstDLL* is the base address of the DLL and *dwReason* specifies a flag indicating why the DLL entry-point function is being called.

dwReason parameter can be set to one of the following values:

- *DLL_PROCESS_ATTACH*: this value shows that the DLL is being loaded into the virtual address of the current process as a result of a call to **LoadLibrary**.
- *DLL_THREAD_ATTACH*: this value shows that the current process creates a thread. When this procedure occurs, the system calls the entry-point function of all DLLs attached to the process. The call is made in the context of the new thread.
- *DLL_THREAD_DETACH*: this value shows that a thread is exiting cleanly. The call is made in the context of the exiting thread.
- *DLL_PROCESS_DETACH*: this value shows that the DLL is being unloaded from the virtual address space of the calling process as a result of either a process exit or a call to **FreeLibrary**.

In the figure 1, we analyze some specifications of the real-time protection service by using satisfaction relations of Dllmain function.

Definition 1. A state structure is a 4-tuple St = (Q,I, e, T, P) where Q is a finite set of states, I is an initial state; e is finite set of events; T is a transition relation according to his relation: $\alpha(s)$ $\beta(s)$ such that two states $\alpha(s)$ and $\beta(s)$ create a transition relation between themselves by using event e_i, P is the state-labeling function (Schneider, 2004).

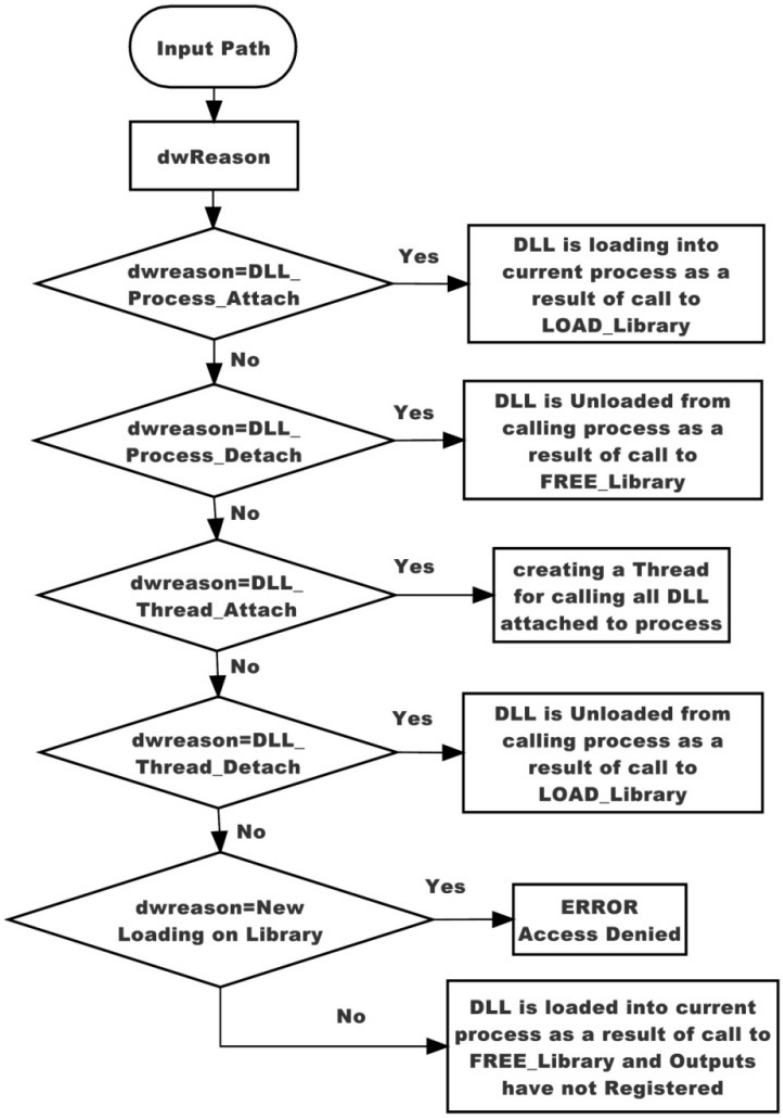

Fig. 1. Calling DLL by using Dllmain function.

A DllMain structure is a 4-tuple DM = (F, V, A, Cr) where F is set of functions, Vis set of values of *dwReason* parameter, A is set of actions, Cr is calling results of *dwReason*. Values of each tuple are as follows:

F= {LoadLibrary, FreeLibrary}

V= {Dll_Process_Attach, Dll_Process_Detach, Dll_Thread_Attach, Dll_Thread_Detach, New_Loading, Other_Valuse}

A = {Loaded, Unloaded, Outputs_not_Registered, New_Thread}

Cr = {Access, Access_Denied}

We obtain label function L for creating satisfaction relation on DllMain structure

$p \models: (V, f \times a) \rightarrow \{True, False\}$ where there is a *True* proposition for satisfaction of $L_{(Cr)}$iff V f = true. We show Boolean relations as follow:

V×Cr = {(Dll_Process_Attach, Access), (Dll_Process_Detach, Access), (Dll_Thread_Attach, Access), (Dll_Thread_Detach, Access), (New_Loading, Access_Denied), (Other_Valuse, Access)}

f×a = {(LoadLibrary, Loaded), (FreeLibrary, Unloaded), (FreeLibrary, New_Thread), (LoadLibrary, Unloaded), (FreeLibrary, Outputs_not_Registered)}

For using formal verification techniques (Wang *et al*., 2012), we need some formal semantics which have been obtained from expected behaviors of the system and temporal logic. To achieve formal semantics, we use Kripke structures (Qianchuan and Krogh, 2001). We define temporal rules for verifying the specification of the system by using CTL in satisfaction relation (Clarke *et al*., 1999). CTL formulas are divided to two categories:

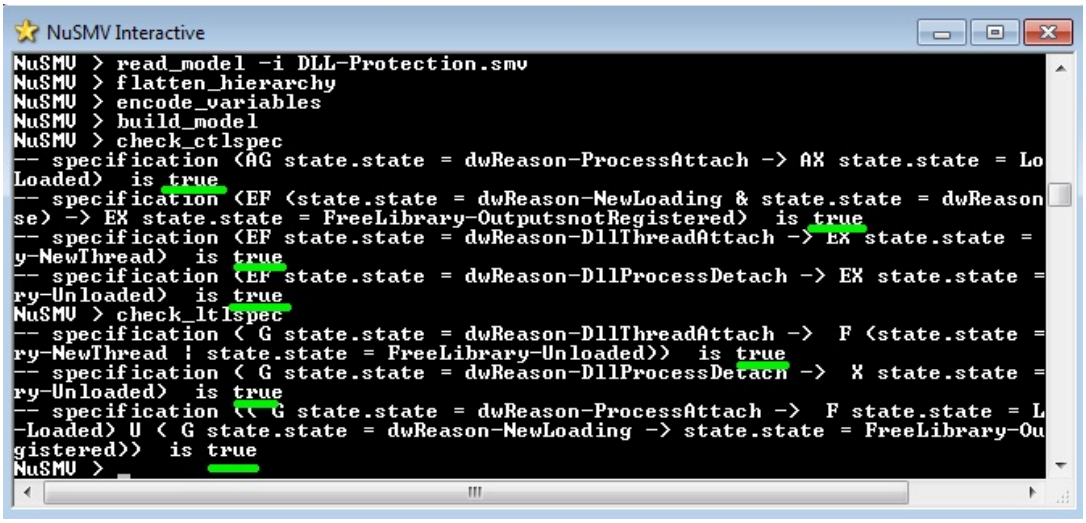

Fig. 2. Verification of CTL properties by using NuSMV model checker.

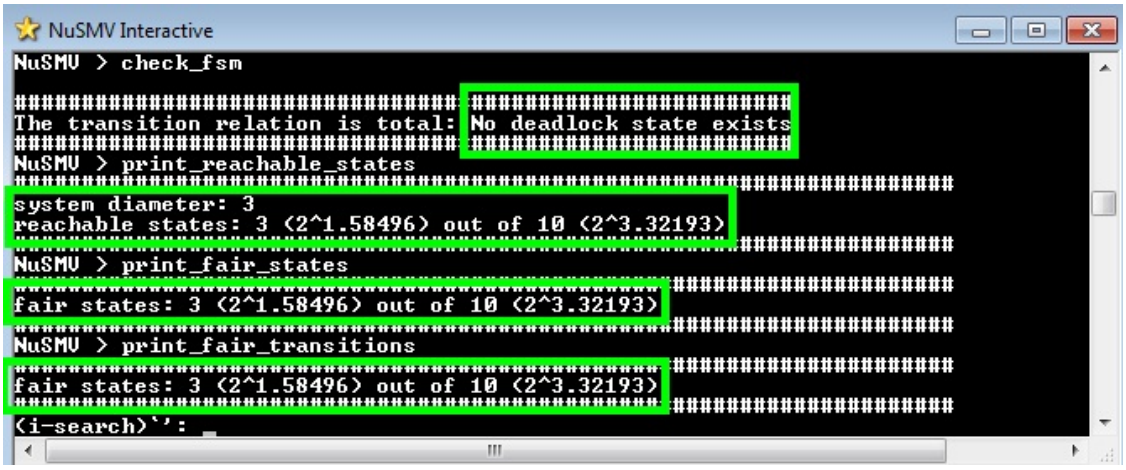

Fig. 3. Checking reachability and fairness of proposed model.

state formulae and *path formulae*. By using verification techniques we can specify that the specifications of model are satisfied or not. We can see these procedures in the next section. Now, we analyze some specification of real-time protection service by using CTL rules:

SPE 1: **AG (***dwReason*.Dll_Process_Attach) →**AX** (LoadLibrary.Loaded)

For this formula we have (Dll_Process_Attach, Access) ∈ V×Cr and (LoadLibrary, Loaded) ∈ f×a. So, *L*(*Access*)= {Dll_Process_Attach∈V, Loaded∈A | Dll_Process_Attach LoadLibrary=*True*}. Then, after accessing *dwReason* to LoadLibrary, a specific dll loaded into current process. In final *L*(*Access*) occurred after calling Dll_Process_Attach. In this result, the formula ***SPE 1*** is satisfied.

SPE 2: **EF**(¬(*dwReason*.New_Loading) ∧ (*dwReason*. Other_Valuse)) →**EX** (FreeLibrary. Outputs_not_Registered)

For all of the paths, when *dwReason*runs new loading operations for calling LoadLibrary function, the antivirus prevent from this loading and accessing to LoadLibrary. Also, if *dwReason* don't call New_Loading and *dwReason*call Other_Valuse, then dll loaded into current process after accessing to FreeLibrar. But produced outputs have not registered in FreeLibrary. We have¬ (New_Loading, Access_Denied) = (Other Valuse,

Access) ∈ V×Cr and (FreeLibrary.Outputs_not_Registered) ∈f×a. Also, we can show:

When (*dwReason*.New_Loading) $\subseteq L_{(Access_Denied)}$then ¬ (*dwReason*.New_Loading)$\subseteq$ $L_{(Access)}$ and we know (*dwReason*. Other_Valuse) $\subseteq$ $L_{(Access)}$. So, following relation is *True*:
¬(*dwReason*.New_Loading)∧ (*dwReason*.Other_ Valuse)$\subseteq L_{(Access)}$.
So, $L_{(Access)}$= {New_Loading, Other_Valuse∈ V, Outputs_not_Registered∈A | ¬New_Loading∧Other_Valuse FreeL ibrary}. In this result, the formula ***SPE 2***is satisfied.

Table 1. Verification results for proposed model.

Property	Result	Time (s)	Memory (KB)	Temporal Language
AG (dwReason-ProcessAttach) -> AX (LoadLibrary-Loaded)	Satisfiable	85.332	46,236	CTL
EF ((dwReason-NewLoading) & (dwReason-OtherValuse)) ->EX (FreeLibrary-OutputsnotRegistered)	Satisfiable	11.778	12,952	CTL
EF (dwReason-DllThreadAttach) -> EX (FreeLibrary-NewThread)	Satisfiable	26.785	37,792	CTL
EF (dwReason-DllProcessDetach) -> EX (FreeLibrary-Unloaded)	Satisfiable	19.935	17,792	LTL
G(dwReason-DllThreadAttach) -> F ((FreeLibrary-NewThread)|(FreeLibrary-Unloaded))	Satisfiable	16.597	46,956	LTL
(G(dwReason-ProcessAttach) -> F (LoadLibrary-Loaded)) U (G(dwReason-NewLoading) -> (FreeLibrary-OutputsnotRegistered))	Satisfiable	65.332	89,136	LTL

SPE 3: **EF**(*dwReason.*Dll_Thread_Attach**)** →**EX** (FreeLibrary.New_Thread)
For this formula, there is a path which when *dwReason*= Dll_Thread_Attach then current process is creating a thread. When this occurs, the system calls the entry function of all DLLs attached to the process. The call is made in the context of the new thread. So, we have (Dll_Thread_Attach, Access) ∈ V×Cr and (FreeLibrary.New_Thread)∈f×a. We show $L_{(Access)}$ = {Dll_Thread_Attach∈V, New_Thread∈A | Dll_Thread_Attach FreeLibrary = *True*}. In final, $L_{(Access)}$ occurred after calling Dll_Thread_Attach. In this result, the formula ***SPE 3*** is satisfied.

SPE 4: **EF (***dwReason*.Dll_Process_Detach) →**EX** (FreeLibrary.Unloaded)

For this formula, there is a path that when *dwReason* = Dll_Process_Detach then DLL is being unloaded from calling process as a call to **FreeLibrary**. So, we have (Dll_Process_Detach, Access) ∈ V×Cr and (FreeLibrary.Unloaded)∈f×a. We can see $L_{(Access)}$= {Dll_Process_Detach∈V,Unloaded∈A | Dll_Process_Detach FreeLibrary = *True*}. In final, $L_{(Access)}$ occurred after exit calling Dll_Process_Detach. In this result, the formula ***SPE 4*** is satisfied.
Finally, after analyzing some CTL formulas in security status service and real-time protection service, we showed how the expected specifications of the security protection service verified. In the next section, the procedure of verifying the proposed models is shown.

Implementation
In this section, the following commands are used in NuSMV model checker. First, we have to read the SMV program by name DLL-Protection.smv then flatten the hierarchy, encode the model variables and build our model. Figure 2 shows the results of the model checking of CTL and LTL properties by using NuSMV model checker. In the implementation, our proposed model detected successfully some specifications described in the above section (shown by Green color).

Using *check_fsm* command, we can check the deadlock problem in finite state machine of our proposed model as a performance evaluation. In figure 3, we showed that the proposed model has not deadlocked (by green line). These results specify that our proposed model is reachable, deadlock free and fair in the states and the transitions.

Moreover, table 1 shows the evaluation results to verify the total number of properties in Antivirus model which are obtained by NuSMV model checker tool.

CONCLUSION

In this paper, we modeled a security protection service in DLL functions. We showed that how model of security protection service has formulated using formal verification techniques. A specification relation between the model and the Kripke structure which enables the conditions for verifying specifications of the system is presented. By presenting the appropriate formal semantics, we showed that how labeling functions permits navigating information and specifications of the security

protection service. We illustrated expected properties of the system which can verify and specified by using CTL and LTL. So, we defined satisfaction relations for verifying the system specifications. We also described how some of expected properties of the system are verified. Also, we could find suitable relations between the system specifications rules. Finally, we implemented some properties of proposed model in NuSMV model checker. In the future work, we will analyze verifications results on behavioral modeling of the antivirus systems and find correct relations between formal verifications and CTL and LTL logics in the security protocols.

REFERENCES

Andronick, J., Chetali, B. and Paulin-Mohring, C. 2005. Formal Verification of Security Properties of Smart Card Embedded Source Code. In: Formal Methods. Eds. Fitzgerald, J., Hayes, I. and Tarlecki, FM. Springer Berlin Heidelberg. 3582:302-317.

Clarke, EM., Grumberg, O. and Peled, DA. 1999. Model checking. MIT Press.

Edmund, M., Clarke, J., Grumberg, O. and Peled, DA. 1999. Model checking: MIT Press.

Filiol, E. 2010. Viruses and Malware. In: Handbook of Information and Communication Security. Eds. P. Stavroulakis, P. and M. Stamp, M. Springer Berlin Heidelberg. 747-769.

Gritzalis, S., Spinellis, D. and Georgiadis, P. 1999. Security protocols over open networks and distributed systems: formal methods for their analysis, design, and verification. *C*omputer Communications. 22(8):697-709. doi: http://dx.doi.org/10.1016/S0140-3664(99)00030-4.

Heitmeyer, CL., Archer, MM., Leonard, EI. and McLean, JD. 2008. Applying Formal Methods to a Certifiably Secure Software System. Software Engineering, IEEE Transactions on. 34(1):82-98. doi: 10.1109/TSE.2007.70772

Morales, J., Clarke, P., Deng, Y. and Golam Kibria, BM. 2006. Testing and evaluating virus detectors for handheld devices. Journal in Computer Virology. 2(2):135-147. doi: 10.1007/s11416-006-0024-y

Qianchuan, Z. and Krogh, BH. 2001. Formal verification of Statecharts using finite-state model checkers. [Paper presented at the American Control Conference, 2001]. Proceedings of the 2001.

Safarkhanlou, A., Souri, A., Norouzi, M. and Sardroud, S. EH. 2015. Formalizing and Verification of an Antivirus Protection Service using Model Checking. Procedia Computer Science. 57:1324-1331. doi: http://dx.doi.org/10.1016/j.procs.2015.07.443

Schlipf, T., Buechner, T., Fritz, R., Helms, M. and Koehl, J. 1997. Formal verification made easy. IBM Journal of Research and Development. 41(4.5):567-576. doi: 10.1147/rd.414.0567.

Schneider, FB. 2000. Enforceable security policies. ACM Trans. Inf. Syst. Secur. 3(1):30-50. doi: 10.1145/353323.353382.

Schneider, K. 2004. Verification of Reactive Systems: Formal Methods and Algorithms: Springer Verlag.

Sellke, SH., Shroff, NB. and Bagchi, S. 2008. Modeling and Automated Containment of Worms. Dependable and Secure Computing, IEEE Transactions on. 5(2):71-86. doi: 10.1109/TDSC.2007.70230.

Singh, PK. and Lakhotia, A. 2002. Analysis and detection of computer viruses and worms: an annotated bibliography. SIGPLAN Not. 37(2):29-35. doi: 10.1145/568600.568608.

Souri, A. and Navimipour, NJ. 2013. Behavioral Modeling and Formal Verification of Resource Discovery in Grid Computing. Expert Systems with Applications(0). doi: 10.1016/j.eswa.2013.11.042.

Szor, P. 2005. The Art of Computer Virus Research and Defense: Addison-Wesley Professional.

Wang, W., Zhang, PT., Tan, Y. and He, XG. 2011. Animmune local concentration based virus detection approach. Journal of Zhejiang University Science. 12(6):443-454. doi: 10.1631/jzus.C1000445.

Wang, ZX., Wang, JM., Zhu, XC. and Wen, LJ. 2012. Verification of workflow nets with transition conditions. Journal of Zhejiang University Science. 13(7):483-509. doi: 10.1631/jzus.C1100364.

Yasinsac, A. and Childs, J. 2005. Formal analysis of modern security protocols. Information Sciences. 171(1–3):189-211. doi: http://dx.doi.org/10.1016/j.ins.2004.03.021.

Zhang, XS., Chen, T., Zheng, J. and Li, H. 2010. Proactive worm propagation modeling and analysis in unstructured peer-to-peer networks. Journal of Zhejiang University Science. 11(2):119-129. doi: 10.1631/jzus.C0910488.

Zhiqiao, W., Kwong, CK., Tang, J. and Chan, JWK. 2012. Integrated model for software component selection with simultaneous consideration of implementation and verification. Computers and Operations Research. 39(12):3376-3393. doi: http://dx.doi.org/10.1016/j.cor.2012.04.020.

Received: May 1, 2015; Revised: Aug 18, 2015; Accepted: Sept 2, 2015