Journée **S**écurité des **S**ystèmes & **S**ûreté des **L**ogiciels

3SL
conjointement à RenPar'20, SympA'14, et CFSE 8.

10 mai 2011

Préface

Les fautes, qu'elles soient de nature accidentelle, intentionelle ou encore malicieuse, menacent la survie de n'importe quel réseau, système, matériel ou logiciel embarqué. Comment peut-on créer un système sécurisé et résilient ? Quelle technique de conception et de développement appliquer ? Comment évaluer et valider ces environnements ? Mais encore, les données manipulées par les logiciels sont-elles protégées par les systèmes d'exploitation, composants logiciels ou protocoles mis en jeux ? Quelles sont les conséquences prévisibles lors d'une panne de tels systèmes dits sécurisés ou résilients ?

Ces préocupations ont mené à l'organisation de la journée 3SL. Cette journée se veut un forum de discussion dans lequel des orateurs présentent les problèmes auxquels ils s'attaquent. Les organisateurs pensent ainsi pouvoir créer des opportunités pour les participants afin que de nouvelles collaborations puissent émerger dans le domaine de la sécurité et de la sureté de fonctionnement. Au sein d'une communauté française s'intéressant de plus en plus à la sécurité des systèmes et à la sureté de fonctionnement logicielle, cet événement renforce les échanges tout en fournissant aux participants de nouvelles opportunités.

Se trouvent ci-après les actes des résumés des interventions de cette journée organisée sous la forme de quatre sessions couvrant les domaines de la sécurité système, des preuves et vérifications de contrainte, des attaques et contremesures, des systèmes résilients.

Plusieurs personnes ont contribué au développement du programme technique de 3SL. Tout d'abord, nous voudrions remercier les auteurs pour avoir contribué au cœur de ce qui forme la journée 3SL. Nous remercions le comité de programme pour le temps et les efforts dévolus à la relecture ; Cédric Tedeschi pour avoir intégré 3SL au triumvirat Renpar, Sympa, CFSE et Catherine Tallon pour le design du site Web de 3SL.

En espérant que vous appréciez la journée 3SL.

Preface

Accidental, intentional, malicious faults together threaten the survability of one another network, system, embedded hardware and software. How can we build secure and resilient systems? What design and development are involved? How should these environments be evaluated and validated? Are the data manipulated by software well protected by operating systems, software components or protocols? And, in case of failure of the designed secure or resilient system, what are the expected consequences?

Those are majors concerns that lead the organisation of the 3SL day.

The 3SL day is envisionned as an open discussion-forum wherein orators present to the community the problems they address. Organizers had in mind that it might give the oportunity to the participants to build new collaborations within the security and dependability fields. The French community that works on the security of systems and the dependability of software is growing. The 3SL organizers hopes that this event reinforces exchanges while giving new opportunities to participants.

Hereafter is provided the research on secured system and dependable software that have been presented. Extended abstracts are organised into four sessions covering topics including secure system, proof and verification, attacks and countermeasures and resilient system.

Several individuals contributed to the development of technical program of 3SL. First and foremost, we would like to thank the authors for contributing to the core of 3SL. We also would like to thank the program committee for their time and effort reviewing and contributing to the reviewing process; Cédric Tedeschi for his work integrating 3SL into the overall conference frame, and behind the scenes, Catherine Tallon for her design of the 3SL Web site.

We hope that you enjoy the 3SL day and are enabled to participate to many discussions.

Comité de programme (program committee)

Conference Program

Testing micro-kernel syscalls to discover vulnerabilities

Amaury Gauthier*, Clément Mazin*, Julien Iguchi-Cartigny* and Jean-Louis Lanet*
*Smart Secure Devices Team — Xlim Laboratories
University of Limoges
France
{amaury.gauthier, clement.mazin, julien.cartigny, jean-louis.lanet}@xlim.fr

***Abstract*—Virtual machine monitor is a hot topic in the embedded community. Most of the current hypervisors for embedded devices use the paravirtualization technique which runs well on small processor without virtualization instructions. This is the case of the OKL4 kernel which is based on the L4 micro-kernel and implements among other the Linux kernel as guest OS.**

We introduce our method based on fuzzing and constraints solving for testing the security of OKL4. We have chosen to focus on the lower level OKL4 interface usable from an external actor: the system calls API. Because all operating system components use directly or indirectly these system calls, a minor flaw at this level can impact in chain the entire system including a virtualized kernel.

I. Introduction

Virtualization is an attractive technology which brings a lot of flexibility whatever the application domain is. But this flexibility has implication on the system security. Embedded virtual machine monitors are still young, and currently, the security impact of a vulnerability in this software piece is not known. This fact motivated our work on the security analysis of these components.

We begin this paper with a section which explains the main arguments in favor of embedded virtualization in a mobile phone context. Then, we present a new grammar which enables the formalization of the OKL4 system calls. Finally, we talk about the use of these models to test the system calls.

II. Virtualization in mobile phone

The first argument used by Companies which develop mobile virtual machine monitors is the cost reduction. Indeed, we can use a hypervisor to make current operating systems run seamlessly on new hardware components with only modifications to the hypervisor architecture dependent code. Depending on the hypervisor architecture, we only need to write a hypervisor driver if we change or add a new device on the system. All operating systems running on top of the hypervisor will be able to use directly the new device functionalities. A hypervisor also enables the possibility to run a real time operating system dedicated to radio communication and a "user experience" operating system on the same system on chip. Nowadays, the majority of mobile phones use two systems on chip when the user experience part of the phone is slightly advanced.

There is also the software update management problem which appeared with the success of smart phone. It can be resolved at least in part by delegating update management to the hypervisor.

Finally, Companies are more and more concerned by the assets management. A hypervisor can be used to build a trusted chain at low level and enforces a Company policies when employees use the same mobile phone for unrelated context (e.g. business and personal context). Hence, a Company can use hypervisor capabilities to protect their assets and enforce a Company security policies.

III. Testing the OKL4 micro-kernel

There is little work focused on the hypervisor security. But this domain is very close to the more developed operating system security field. Several approaches exist to test the kernel of an operating system like statical analysis, fuzzing or formal verification[1]. We choose to base our work on the fuzzing technique.

A. System calls formalization for OKL4

In OKL4, a system call is a function taking at least one arguments and providing one or more results. There are two types of input and output arguments: standard parameters and virtual registers.

Standard parameters are like normal arguments of any function in a simple program and virtual registers are objects which are associated to each thread in the system. The user can interact with them with "getter" and "setter" functions. They can be mapped directly on processor registers and must be set before running a system call.

Some conditions must also be verified before running a system call to ensure its success. The system should be in a precise state described in the OKL4 Micro-kernel Reference Manual[2].

After a system call, some results are returned. There is always a *Result* parameter which indicates if the system call is successful or not. If the *Result* is set to false, the *ErrorCode* parameter provides some information about the error. The different errors are described in [2]. In case of a successful system call, the manual describes in which state OKL4 should be after this call.

B. A grammar for system calls testing

In order to enhance test generation, we have designed a grammar to model OKL4 system calls. This grammar allows to model precisely a system call: its execution context, its parameters, their constraints and the results of the system call.

Then, the data stored in these models can be used to perform computation. These models have two goals. Firstly, they will be used as inputs for the space test generation algorithm.

Secondly, they will be used to automatically configure the fuzzer.

As an example, we have modeled the OKL4 *IPC* system call. This system call is used when two processes want to exchange data.

```
Name Ipc
Args
  Name 'TargetIN'
  Type 'ThreadID'
  VirtualRegister 'Parameter0'
  Constraint '=NilThread OR
    in(SysState.ExistingThread)'
  Name 'SourceIN'
  Type 'ThreadID'
  VirtualRegister 'Parameter1'
  Constraint '=WaitNotify OR =AnyThread OR
              =ThreadID OR =NilThread'
  Name 'AcceptorIN'
  Type 'Acceptor'
  VirtualRegister 'Acceptor'
  Constraint '=True OR =False'
  Name 'TagIN'
  Type 'MessageTag'
  VirtualRegister 'MessageDate0'
  Constraint ''
InputRegisters
  Name'Data1IN'
  Type 'Word'
  VirtualRegister 'MessageData1'
  Constraint ''
Output
  Name 'ReplyCapOUT'
  Type 'ThreadID'
  VirtualRegister 'Result0'
  Constraint 'TargetIN=NilThread => undefined'
  Name 'SenderSpaceOUT'
  Type 'SpaceID'
  VirtualRegister 'SenderSpace'
  Constraint 'TargetIN=NilThread => undefined'
  Name 'ErrorCodeOUT'
  Type 'ErrorCode'
  VirtualRegister 'ErrorCode'
  Constraint '=NoPartner OR = InvalidPartner OR
              =MessageOverflow OR =IpcRejected OR
              =IpcConcelled OR =IpcAborted,
              TargetIN=NilThread => undefined'
  Name 'TagOUT'
  Type 'MessageTag'
  VirtualRegister 'MessageData0'
  Constraint 'TargetIN=NilThread
   => TagOUT.Untyped=0'
  Name 'DataOUT'
  Type 'Word'
  VirtualRegister 'MessageData0'
  Constraint ''
Success
  'TagOUT.E=False AND ErrorCode=undefined,
   TargetIN=NilThread AND SourceIN=NilThread
    => null_syscall,
   SourceIN in(WaitNotify,AnyThread,ThreadId)
    => ReplyCapOutSysState.ExistingThread'
Failure
  'TargetIn.NotWaiting => ErrorCode=NoPartner,
   TargetIn notIn(SysState.ExistingThread)
    =>  ErroCode=InvalidPartnerr,
   TagIN.Untyped>MaxMessageData'
```

A constraint like *TargetIN=NilThread => undefined* means that if *TargetIN* value is *NilThread*, then the value of *ReplyCapOUT* will be undefined.

IV. How to use our system calls model to generate useful fuzzer outputs

From the above model, we can compute some properties. The set of properties allows the generation of each valid system calls. But this set can not be used as fuzzing input directly because the input domain is too large to be evaluated in a reasonable time.

But, we can use the *Constraint* property of the model in a manner to remove some testing values from this first set and therefore reduce it. Indeed, this property allows the elimination of all values which are not valid. We just need to keep some values which are out of these bounds to effectively test verifications operated by the kernel at system call invocation.

The main algorithm of our method is defined in the next pseudo-code fragment. It takes as parameter a system call formalized with the grammar and the system state. This last argument enables the computation of constraints which use the operating system state to restrict the set of values. The function returns a set of values to be tested on the system call in addition to the expected values.

```
computeValues(Grammar syscall, SystemState state)
              : output{
  foreach syscall.Args,
          syscall.InputRegisters as arg{
    space <- computeConstraint(arg.Constraint,
                               state)
    foreach space as value{
      output[arg] <- value
      output[arg] <- lowerBound(value)
      output[arg] <- upperBound(value)
    }
  }
  output[Success] <-
      computeConstraint(syscall.Success, state)
  output[Failure] <-
      computeConstraint(syscall.Failure, state)
  foreach syscall.Output as out {
    output[Success] <-
        computeConstraint(out.Constraint, state)
    output[Failure] <-
        computeConstraint(out.Constraint, state)
  }
}
```

V. Future works

This is an on going work, so it remains a lot of work to make this method fully functional and automatic.

When this goal will be reached, we will work on the portability of this method to make it usable on other micro-kernels and then we will try to apply it to more traditional virtual machine monitors like Xen[3].

References

[1] G. Klein, J. Andronick, K. Elphinstone, G. Heiser, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood, "seL4: Formal verification of an OS kernel," *Communications of the ACM*, vol. 53, no. 6, pp. 107–115, Jun 2010.

[2] O. K. Labs, "Okl4 microkernel reference manual."

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*. New York, NY, USA: ACM, 2003, pp. 164–177.

A. Gauthier is a second year PhD student, in the Xlim laboratories at University of Limoges, working in the Smart Secure Devices team on the security of hypervisor for embedded devices. His research interests include operating system security and hypervisors for embedded systems.

Évaluation des performances des hyperviseurs pour l'avionique

Maxime Lastera[1,2], Eric Alata[1,2], Jean Arlat[1,2], Yves Deswarte[1,2], Bertrand Leconte[3], David Powell[1,2]
[1] CNRS ; LAAS ; 7 avenue du Colonel Roche, F-31077 Toulouse, France
[2] Université de Toulouse ; UPS, INSA, INP, ISAE, UT1, UTM, LAAS ; F-31077 Toulouse Cedex 4, France
[3] EYMI2 ; BP M3020 ; AIRBUS Operations SAS ; 316 route de Bayonne, F-31060 Toulouse Cedex ; France

***Résumé*—La technologie de virtualisation facilite la mise en œuvre d'architectures redondantes (voire diversifiées), afin de tolérer différentes classes de fautes et ainsi d'atteindre les objectifs de sûreté de fonctionnement visés. Cet article propose une méthodologie d'évaluation d'hyperviseurs afin d'analyser l'impact de la virtualisation sur la performance temporelle d'exécution. Plusieurs tests (CPU, mémoire, réseaux) ont été effectués, et des mesures du temps d'exécution ont été collectées sur différentes configurations mettant en jeu différents hyperviseurs.**

I. Introduction

Le développement de systèmes critiques est soumis à de fortes exigences de validation et de vérification. Ces exigences ont un impact important sur les coûts de développement de ces systèmes. Par ailleurs, les composants pris sur étagère (*commercial off-the-shelf* ou COTS) offrent de nombreuses fonctionnalités tout en permettant de réduire ces coûts de développement. En revanche, ces composants n'ont pas été conçus dans le but de satisfaire les exigences liées au domaine avionique. C'est le cas des systèmes d'exploitation. Ces logiciels sont non seulement susceptibles d'être affectés par des fautes de conception accidentelles, mais sont aussi vulnérables à des attaques. Dans le domaine de l'avionique, les applications critiques sont généralement totalement séparées du monde ouvert, afin d'éviter toute interaction qui pourrait corrompre les systèmes embarqués à bord de l'avion. Cependant, les nouvelles générations d'avions ont besoin de plus d'interactions avec les systèmes au sol pour offrir des services étendus, et il est nécessaire de sécuriser les communications vis-à-vis des flux d'information potentiellement dangereux. Dans une précédente étude [1], l'utilisation de la virtualisation a été proposée pour assurer la fiabilité des applications critiques tout en permettant une communication bidirectionnelle entre les systèmes critiques embarqués dans l'avion et les systèmes moins critiques au sol. Afin d'analyser l'impact de l'utilisation de la virtualisation, nous avons ainsi développé un banc de test permettant de mesurer de façon précise des temps d'exécution d'architectures à base d'hyperviseurs. Plus précisément, nous nous intéressons à la capacité d'un hyperviseur à ne pas dégrader les ressources mises à la disposition des machines virtuelles. Dans cette étude, différentes configurations ont été expérimentées, allant du cas d'une machine dépourvue de système d'exploitation à celui d'une architecture complète avec un hyperviseur et un système d'exploitation exécuté dans une machine virtuelle.

La suite de cet article est organisée en cinq sections. La Section II décrit brièvement les travaux connexes à cette étude et met nos travaux en perspective dans ce contexte. La Section III présente brièvement le banc de test développé. La Section IV décrit le programme de test choisi pour l'évaluation de la mémoire. Ensuite, dans la Section V, nous détaillons les premiers résultats de l'évaluation. Enfin, la Section VI conclut cet article et présente les perspectives à envisager.

II. Travaux connexes

L'évaluation des hyperviseurs a fait l'objet de plusieurs travaux. Dans [2], les auteurs ont comparé la dégradation de l'exécution d'une application entre une machine physique et différentes solutions de virtualisation. Ils utilisent le benchmark Ubench afin de réaliser leurs mesures. La machine physique est utilisée comme référence pour évaluer les différences entre les hyperviseurs considérés. Dans [3], différentes techniques de virtualisation sont comparées en prenant comme référence une machine physique munie d'un système d'exploitation non virtualisé. Dans ce cas c'est le benchmark VMmark qui est utilisé pour évaluer les performances. Le benchmark NAS est utilisé dans [4] afin de comparer les performances entre l'hyperviseur Xen et une solution VMware. Les auteurs étudient notamment l'influence du nombre d'unités de traitement (CPUs) virtuelles affectées à un hyperviseur. Les algorithmes RandomWriter et Sort sont utilisés dans [5] pour évaluer les performances d'un réseau de machines avec un hyperviseur. Dans ces différentes études, la mesure du temps d'exécution se fait de manière logicielle, notamment dans [5] la commande *top* est utilisée. Pourtant, dans un environnement virtualisé, l'horloge de l'hyperviseur est différente de l'horloge de la machine virtuelle. Aussi, un mécanisme de synchronisation doit être mis en place pour assurer l'ordonnancement de la machine virtuelle par l'hyperviseur. Or, d'une solution de virtualisation à une autre, ces mécanismes peuvent ne pas être identiques et de ce fait biaiser les mesures effectuées pour les temps d'exécution. Notre démarche se démarque de ces études par l'utilisation d'un périphérique matériel, indépendant de la configuration à évaluer. Cela nous permet à la fois de réaliser des mesures et d'observer le comportement des différentes configurations.

III. Banc de test

Les mesures collectées doivent permettre de faciliter le choix d'un hyperviseur adapté aux exigences de performance. A cette fin des programmes de test distincts ont été définis pour surcharger différents composants matériels : mémoire, carte réseau et processeur. Un environnement de mesure spécifique a été développé pour permettre d'évaluer les temps d'exécution de ces programmes. Ces programmes sont exécutés

sur les différentes configurations de l'architecture, de manière incrémentale. Tout d'abord, nous utilisons une configuration minimaliste permettant uniquement un accès matériel, cette configuration est nommé « Pépin ». Ensuite, nous avons choisi deux versions du noyau Linux correspondant à celles utilisées par les hyperviseurs considérés. Il s'agit du noyau 2.6.32 utilisé par l'hyperviseur Xen et du noyau 2.6.34 utilisé par l'hyperviseur Qubes. Ces configurations testées sont nommées respectivement « 2.6.32 », « 2.6.34 », « Xen » et « Qubes » (*cf.* table 1). Nous avons utilisé une machine virtuelle exécutant un système d'exploitation pour représenter la dernière configuration. Une machine virtuelle est exécutée sur l'hyperviseur Xen et nommée « Xen-vm » et une autre machine virtuelle est exécutée par l'hyperviseur Qubes, nommée « Qubes-vm ». Ces deux machines virtuelles utilisent le même système d'exploitation, une distribution Fedora 13 avec un noyau 2.6.34.

IV. Les programmes de test

Nous souhaitons évaluer la dégradation des hyperviseurs vis-à-vis des performances de plusieurs types d'applications qui peuvent être déployés dans l'environnement. Par conséquent, nous avons développé des programmes types qui sollicitent chacun un composant particulier du matériel (CPU, mémoire et réseau) [1]. Les applications sont développées de la manière suivante, écriture de *un* sur un bit du port parallèle avant le lancement de l'application puis écriture de *zéro* sur le même bit du port parallèle à la fin de l'exécution, ainsi le temps qui s'écoule sur le palier haut mesuré à l'oscilloscope représente le temps d'exécution de la charge. Par manque de place, dans la suite, nous ne traiterons que du test relatif aux transferts mémoire. Le programme de test mémoire est réalisé en langage assembleur pour nous permettre de maîtriser les instructions exécutées par l'application. Le test mémoire doit permettre d'identifier si l'hyperviseur exploite pleinement les capacités du bus FSB (*Front-Side Bus*). Ce dernier permet de connecter le processeur à la mémoire. Dans la section suivante nous présentons les résultats du test mémoire.

V. Résultats

La taille mémoire à copier a été définie à 10Mo et répétée dix fois afin de mettre en défaut les différents caches. La copie mémoire représente donc un volume de 100Mo. La taille de l'échantillon d'étude a été fixée à mille expériences. Les histogrammes de la figure 1 correspondent aux valeurs moyennes calculées sur mille expériences pour chacune des configurations. L'échelle temporelle est exprimée en millisecondes. La table I présente les valeurs extrêmes correspondant à un intervalle de confiance à 95% pour les données que nous avons analysées. Nous observons que la configuration « Pépin » n'est pas optimisée, en effet elle a été conçue seulement pour accéder au port parallèle, aucune mise en place de cache n'est prévue dans cette configuration. Il existe entre les deux versions du noyau une différence qui peut s'expliquer par les améliorations apportées lors des changements de version du noyau Linux. Au niveau des hyperviseurs, Xen se comporte comme la configuration « Pépin », ce qui suggère qu'il n'implémente pas le mécanisme de mise en cache pour améliorer les transferts mémoire. Il a été conçu pour assurer la mise en place de techniques de virtualisation et d'isolation. En effet, la performance de la configuration « Xen-vm » est du même ordre que celle des systèmes d'exploitation non virtualisés (noyau 2.6.32 et noyau 2.6.34). L'hyperviseur Qubes suit une autre hypothèse. Il semble mettre en place des mécanismes de cache pour améliorer le transfert mémoire, car il obtient le meilleur résultat du test. Cependant la gestion de sa machine virtuelle est moins performante vis-à-vis de Xen. En effet, sur le test considéré, la configuration « Qubes-vm » est moins performante que la configuration « Xen-vm ».

1. Un programme complexe peut être vu comme la somme pondérée de ces trois applications types. Par conséquent, la dégradation de performance sera une somme pondérée de ces trois applications.

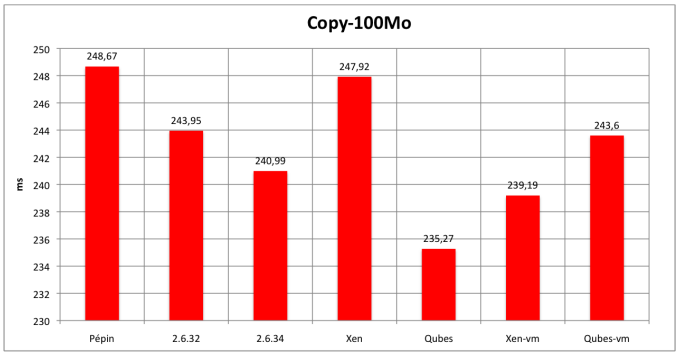

Figure 1. Histogramme comparatif

Configuration	Pépin	2.6.32	2.6.34	Xen	Qubes	Xen-vm	Qubes-vm
Borne sup.	248,71	244,59	241,01	247,94	235,3	239,21	243,63
Borne inf.	248,64	243,3	240,97	247,90	235,24	239,17	243,57

Table I
Borne de l'intervalle de confiance à 95%

VI. Conclusion

Cet article présente une méthodologie d'évaluation des performances de différents hyperviseurs. L'utilisation d'un accès matériel tel que le port parallèle permet d'être indépendant des configurations à évaluer. Les résultats ont montré que les hyperviseurs ont chacun leur politique d'ordonnancement vis-à-vis de leur machine virtuelle. L'hyperviseur Xen semble privilégier sa machine virtuelle au dépend de ses propres performances alors que l'hyperviseur Qubes privilégie ses propres performances. Dans la suite de nos travaux nous essaierons d'appliquer cette méthodologie à d'autre hyperviseurs tels que OKL4 microvisor ou PolyXene.

Références

[1] Y. Laarouchi, "Sécurités (immunité et innocuité) des architectures ouvertes à niveaux de criticité multiples : application en avionique," Thèse de Doctorat, LAAS-CNRS, INSA de Toulouse, Nov. 2009.

[2] X. Xu, F. Zhou, J. Wan, and Y. Jiang, "Quantifying Performance Properties of Virtual Machine," in *International Symposium on Information Science and Engineering (ISISE'08)*, vol. 1, 2008, pp. 24–28.

[3] R. McDougall and J. Anderson, "Virtualization Performance : Perspectives and Challenges Ahead," *ACM SIGOPS Operating Systems Review*, vol. 44, pp. 40–56, Dec. 2010.

[4] D. Bhukya, S. Ramachandram, and A. R. Sony, "Evaluating Performance of Sequential Programs in Virtual Machine Environments Using Design of Experiment," in *IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, 2010, pp. 1–4.

[5] M. Kontagora and H. Gonzalez-Velez, "Benchmarking a MapReduce Environment on a Full Virtualisation Platform," in *Intl. Conf. on Complex, Intelligent and Software Intensive Systems (CISIS)*, 2010, pp. 433–438.

Construire des abstractions par preuve pour générer des tests

P.-C. Bué, J. Julliand, P.-A. Masson
LIFC - Université de Franche-Comté - 16, route de Gray F-25030 Besançon Cedex, France
Email: {bue, julliand, masson}@lifc.univ-fcomte.fr

I. Motivations

Dans une approche de test à partir de modèles (*Model-Based Testing*) [1], [2], [3], un modèle est rédigé et validé séparément d'une implantation, à partir d'un document initial de spécification. Des tests sont calculés à partir du modèle, en utilisant des critères de couverture de celui-ci. En confrontant l'implantation aux tests issus du modèle, on peut détecter des non conformités de l'implantation par rapport au modèle.

Mais la mise en œuvre de ces techniques de test reste difficile pour des systèmes de taille industrielle, en raison de la très grande taille de l'espace d'états de leurs modèles. On peut alors calculer et utiliser des abstractions de ces modèles pour faire face à ce problème. Notre intention est de réduire la taille du modèle à partir duquel sont générés les tests. L'objectif est de maîtriser l'explosion combinatoire du nombre d'états, de transitions ou de chemins qui sont les critères de couverture de test habituels utilisés sur ce type de modèle.

Mais abstraire un modèle initial pose deux problèmes, son coût de calcul et sa relation, en termes de correction et de précision, avec le modèle initial et le critère d'évaluation. Deux sortes d'abstractions sont envisageables, soit une sur-approximation, soit une sous-approximation au sens où l'ensemble des éxécutions de l'abstraction est respectivement, soit un sur-ensemble, soit un sous-ensemble des exécutions du modèle initial. Une sur-approximation est adaptée à la vérification des propriétés de sûreté qui sont préservées : si "quelque-chose de mauvais n'arrive pas sur un plus grand nombre d'exécutions, il n'arrivera pas sur le modèle initial". Par contre, la génération de tests à partir d'une sur-approximation nécessite de vérifier que chaque test issu de l'abstraction est bien une exécution du système initial. Au contraire, les tests issus d'une sous-approximation sont tous exécutables sur le modèle initial. Mais se pose alors un problème de précision : le nombre de tests issus d'une sous-approximation couvre-t-il de manière suffisante le système ? Finalement, pour la génération de tests, les deux approches sont possibles, mais le calcul de sur-approximation nécessite une phase de filtrage des tests réellement exécutables alors que le calcul de sous-approximation peut engendrer un nombre insuffisant de tests dû à une trop faible précision.

Ce résumé aborde cette problématique en décrivant une méthode de calcul de sur et de sous-approximations. Le problème à résoudre est de trouver le meilleur compromis entre la qualité de la couverture de test et les performances de calcul des tests. Les techniques de sur-approximation ont un coût en temps de calcul important dû à la phase de filtrage par instanciation des tests qui consiste à réaliser une recherche combinatoire. Le taux de couverture dépend de la réussite de cette phase d'instanciation. Par contre, les méthodes par sous-approximation par concrétisation sont plus efficaces puisqu'elles ne nécessitent pas de phase de filtrage et d'instanciation. Mais le taux de couverture dépend de la capacité à engendrer des graphes d'états concrets connexes.

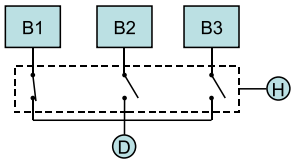

Fig. 1. Schéma du Système Electrique

Nous présentons un algorithme de calcul d'abstraction qui prend en entrée, outre le modèle à abstraire, un ensemble de n prédicats d'abstraction [4]. Il calcule une relation de transitions symboliques sur-approximée Δ entre les 2^n états symboliques en évaluant des conditions établissant l'existence de transitions entre ces états. Il calcule une sous-approximation Δ^c en instanciant ces transitions à la volée en s'inspirant des travaux de [5]. L'ensemble des exécutions concrètes partant des états initiaux constituent la sous-approximation. Cet algorithme repose sur l'utilisation de trois fonctions primitives de preuve : un calcul de plus faible précondition, de plus forte postcondition et une évaluation de satisfiabilité (par un solveur SAT). Nous avons implanté notre méthode en utilisant des solvers SAT SMT, en l'occurrence Z3 [6].

II. Exemple

Nous présentons un exemple pour illustrer nos propos. C'est un système réactif de contrôle de l'alimentation électrique d'un dispositif D alimenté par l'une des trois batteries B_1, B_2, B_3 comme le montre la figure 1. Un interrupteur connecte (ou non) une batterie B_i à l'appareil D. Une horloge H envoie périodiquement un signal de commutation, c'est à dire un ordre de changement de la batterie alimentant D. Le fonctionnement du système doit satisfaire les trois exigences suivantes:

- Req_1 : il n'y a pas de court-circuit,
- Req_2 : l'appareil est constamment alimenté,
- Req_3 : lorsque l'horloge envoie une commande de commutation, l'interrupteur fermé est modifié.

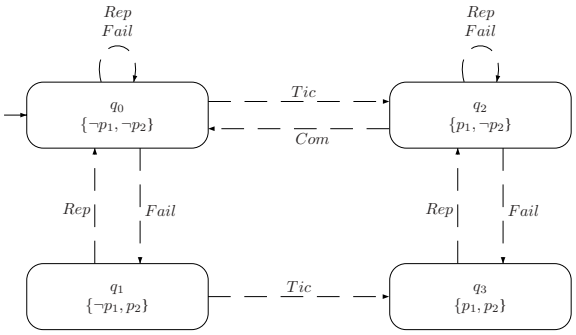

Fig. 2. Relation Δ du système électrique

Mais les batteries peuvent tomber en panne. Quand la batterie qui alimente D tombe en panne, le système effectue une commutation exceptionnelle pour satisfaire l'exigence Req_2. Les batteries en panne sont remplacées par un service de maintenance. Nous supposons que ce service travaille suffisamment rapidement pour qu'il n'y ait jamais trois batteries en panne simultanément. Quand deux batteries sont en panne, l'exigence Req_3 est relâchée et les ordres de commutation de l'horloge ne sont plus pris en compte.

Par exemple, ce processus d'abstraction donne la relation de transition Δ representée dans Fig. 2.

III. Application du calcul d'abstractions à la génération de tests

Notre intention est d'utiliser les abstractions produites selon l' algorithme présenté précédemment pour la génération de tests. Nous présentons deux processus de génération de tests illustrés dans la figure 3, le premier $\mathcal{P}_{sur}$ utilise la sur-approximation Δ et le second $\mathcal{P}_{sous}$ la sous-approximation Δ^c.

La figure 3 présente les grandes lignes de ces processus. Les deux premières étapes sont communes aux deux processus. L'abstraction AM est synchronisée avec un objectif de test OT pour cibler les exécutions décrites par l'objectif. Les tests sont extraits du produit synchronisé PS à partir d'un critère de sélection structurel comme la couverture des transitions et/ou des états. Si l'abstraction était la sous-approximation, le processus $\mathcal{P}_{sous}$ est terminé car les tests TI sont définis au niveau du modèle M qui a été abstrait. Si l'abstraction était la sur-approximation, le processus $\mathcal{P}_{sur}$ se poursuit par l'instanciation des tests TAS qui sont transformés du niveau de l'abstraction AM au niveau du modèle M.

Sur l'exemple, l'objectif de test est d'observer les comportements lorsque la batterie qui alimente le système tombe en panne et lorsqu'il n'y a plus qu'une seule batterie en fonctionnement. On extrait les deux prédicats suivants de cet objectif de test : p_1=*déclenchement d'une commutation* et p_2=*une seule batterie est en fonctionnement*. Dans [7], nous avons décrit le processus $\mathcal{P}_{sur}$ et défini une méthode pour extraire l'ensemble de prédicats à partir de l'objectif de test OT et du modèle comportemental M.

IV. Conclusion et perspectives

Nous avons présenté dans cet article un algorithme de génération d'abstraction à partir d'un ensemble de prédicats et

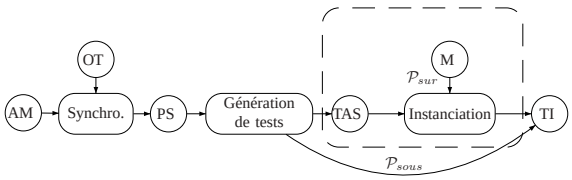

Fig. 3. Génération de tests à partir d'une abstraction et d'un objectif de test

son application à la génération de tests. Cet algorithme calcule d'une part une sur-approximation constituée par l'ensemble des transitions symboliques potentiellement déclenchables par l'ensemble des opérations présentes dans le système. Il calcule d'autre part, à la volée, une concrétisation de cette relation de transition qui est une sous-approximation. Nous présentons l'utilisation de ces approximations pour générer automatiquement des tests de deux manières. La première extrait des exécutions potentielles de la sur-approximation et tente de les instancier *a posteriori*. La seconde extrait directement des tests de la sous-approximation. Nous comparons les résultats obtenus sur des exemples selon plusieurs critères, notamment le temps de calcul des tests et les taux de couverture des abstractions.

***Keywords*-Abstraction, Résolution de contraintes, Preuve, Génération de tests à partir de modèles**

References

[1] B. Beizer, *Black-Box Testing: Techniques for Functional Testing of Software and Systems*. New York, USA: John Wiley & Sons, 1995.

[2] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, Eds., *Model-Based Testing of Reactive Systems*, ser. LNCS. Springer, 2005, vol. 3472.

[3] M. Utting and B. Legeard, *Practical Model-Based Testing*. Morgan Kaufmann, 2006.

[4] S. Graf and H. Saïdi, "Construction of abstract state graphs with pvs," in *CAV'97*, ser. LNCS, vol. 1254, 1997, pp. 72–83.

[5] C. S. Pasareanu, R. Pelánek, and W. Visser, "Predicate abstraction with under-approximation refinement," *CoRR*, vol. abs/cs/0701140, 2007.

[6] L. de Moura and N. Bjørner, "An efficient smt solver," in *TACAS'08*, ser. LNCS, vol. 4963, 2008, pp. 337–340.

[7] F. Bouquet, P.-C. Bué, J. Julliand, and P.-A. Masson, "Test generation based on abstraction and test purposes to complement structural tests," in *A-MOST'10, 6th int. Workshop on Advances in Model Based Testing*, ser. IEEE proceedings of ICST'2010, Paris, France, Apr. 2010.

V. Biographies

Jacques Julliand et Pierre-Alain Masson sont respectivement professeur et maître de conférences au laboratoire d'informatique de l'Université de Franche-Comté. Leurs thèmes de recherche sont la vérification et la validation de systèmes critiques. Les mots-clés de leur activité sont : Vérification de propriétés de logique temporelle, génération de tests à partir de modèles, expression et test de propriétés de sécurité, test guidé par les propriétés et abstraction de modèles pour le test.

Pierre-Christophe Bué est doctorant au laboratoire d'informatique de l'Université de Franche-Comté. Il travaille sur une approche de génération de tests utilisant des objectifs de test dynamiques et des abstractions qui sur-approximent ou sous-approximent des modèles comportementaux.

Modeling Distributed Real-Time Systems using Adaptive Petri Nets

Olivier BALDELLON* Jean-Charles FABRE Matthieu ROY
CNRS ; LAAS ; 7 avenue du colonel Roche, F-31077 Toulouse Cedex 4, France
Université de Toulouse ; UPS, INSA, INP, ISAE ; UT1, UTM, LAAS ; F-31077 Toulouse Cedex 4, France

***Abstract*—Industrial systems, like aircrafts or cars, are based on real-time distributed networks and require high level of robustness, reliability and adaptability. This paper first introduces our approach to implement a distributed monitor of real-time properties, and then introduce a new formalism, *adaptive Petri nets*, that will allow to model such complex, distributed and real-time systems.**

***Keywords*—Distributed Systems Models, Petri Net, Real Time**

I. INTRODUCTION

On-line monitoring of complex, distributed and real-time systems is a highly complex task that, to our knowledge, has not yet been fully tackled. On-line monitoring of applications (distributed or not) in a centralized way has been an active research domain [5][6] since a few years now, but these approaches cannot handle the case of distributed and real-time systems, and the following two challenges must be addressed to be able to provide online monitoring in the general case :

- the first challenge concerns *property modeling*, *i.e.*, the ability to formally reason on distributed systems with timing constraints. Such formalism should take into account (i) temporal aspects, (ii) distribution and (iii) evolvability. Moreover, for a formalism to be useable by human users, it should permit to express systems' feature in a hierarchical way, to prevent overwhelming complexity of description.
- the second challenge concerns the transformation of formal properties into a *runtime support*, i.e., the "compilation" of properties in a runnable format. The important points for such a transformation are (i) it should maximize offline work to reduce overhead at runtime, (ii) it must provide a distributed scheme for verification of system wide properties, and (iii) it has to provide the most possible efficient way (both in computation and in network load) to verify a given property.

II. OUR APPROACH

Our approach can be described in three steps represented in Figure 1. Each step builds some tools that will allow to "compile" a model into a distributed monitor. The first step is to introduce a new formalism that will allow to describe complex real-time distributed services. The second step will transform (with three operations) the model to adapt it for the last step, that consists in the deployment and distributed execution of the transformed model.

The introduction of a new formal tool that can be used to describe the system is the main subject of this paper.

This work has been supported by the french national agency (ANR) under contract ANR-BLAN-SIMI10-LS-100618-6-01.

∗ Corresponding author: olivier.baldellon@laas.fr

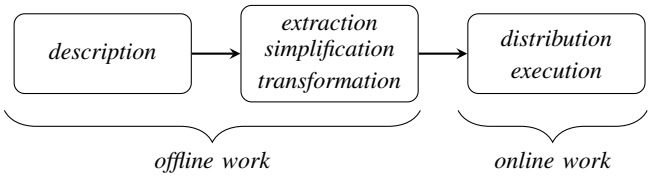

Figure 1. Our approach

This formal approach, called *adaptive Petri net*, is a scalable and hierarchical way to define complex systems that can be modified and adapted.

Then, some transformation work has to be done offline on such a system. The distributed system is a network made of nodes connected together, but because an adaptive Petri net represents a whole system and could be huge, and because we don't want every node to check the whole system, we will extract some parts of this Petri net, a part being a subsystem that a subset of nodes will monitor. This operation is called *extraction*.

The next operation will consist in simplifying the system's parts obtained from the previous operation, for example by abstracting a whole set of places and transitions by a simple place; such an operation is called *simplification*. It could be useful is some nodes are supposed to monitor the global behavior of the system, but do not need to monitor every single operation.

Then we need to transform this piece of adaptive Petri net: as the original adaptive Petri net only describes properties we want to check, we need indeed to transform it to obtain a new model that prevents and/or avoids faults as soon of possible.

The last step will be to deploy and execute the parts to create a distributed real-time monitoring system.

Another approach based on Petri net connected together can be found in [4], [1], but this work do neither consider real-time systems, nor degradations.

III. COMPONENTS AND ARC TIME PETRI NETS

A bottom-up approach will be used to describe our system. The component notion will be the first that will be introduced. A component is the more basic part of the system and is described by arc time Petri net [3]. An arc time Petri net is a classical Petri net where arcs between place and transition are labelled by a time interval (*cf.* Figure 2).

We suppose that the reader is familiar with the Petri net concept. The main difference between classical Petri nets is that every time a token appears in a place, a new timer starts. Time passes at the same speed in all timers of every token. A token can fire a transition if and only if the value of the timer

is contained in the time interval of the arc. For example, in Figure 2, the first token can fire the transition after a waiting time between 3 and 5 unit. It easy to see that if the token waits more than 5 it will not be able to fire the transition anymore. In this case the token is said to be *dead*.

To summarize, there are two main rules: *the discrete rule* corresponding to the firings of transitions, and *the continuous rule* corresponding to the passing of time. For a more formal description of Arc time Petri nets, and a comparison with other model time Petri net, the reader can look at [2].

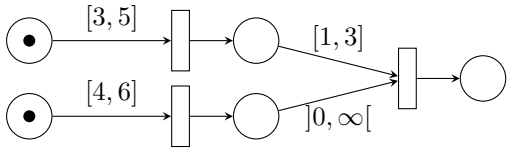

Figure 2. An arc time Petri net

IV. Degradable components

A degradable component is a sequence of components $(d_0, d_1, d_2, \ldots, d_n)$ where d_0 is the nominal mode (with no degradation) of the degradable component and the different d_i represent the degraded modes.

We assume that nominal mode and degraded modes have the same interface. The interface of a Petri net is the set of its sources and sinks. The set of sources, the input interface, is the set of place where no transition points to (as i_1 and i_2 in Figure 3). Similarly, the set of sinks, the output interface, is the set of place pointing to no transition (as o_1 in Figure 3).

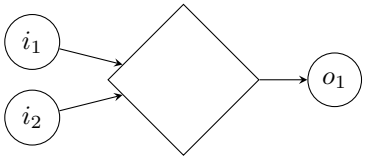

Figure 3. A degradable component (e.g., the example of Figure 2)

V. Adaptative Petri Nets

This section shows how a complex system architecture can be described by connecting degradable components. Let us consider the following example: one client and one server. The client can be seen as a set of three components (c_1, c_2, c_3), the server provide three services (a read one r, a write server w_1 and w_2 and a deliver service c). We can first connect with a function f_c the three components c_1, c_2 and c_3 to obtain the client; we can then connect the two operations w_1 and w_2 to obtain the *write* subsystem; the server can be obtained by connecting the three services together with a f_s function to obtain the server. To obtain the whole system we just have to connect the server and the client with an appropriate function.

Due to space limitations, the formal definition of a connection function cannot be described here. The above system can be described by a tree as shown in Figure 4. We call such a tree an *adaptative Petri net*.

In the tree, every branch is a degradable system composed of degradable components. For example if we consider the write subsystem (w_1 and w_2), d_0 represent the default mode (all

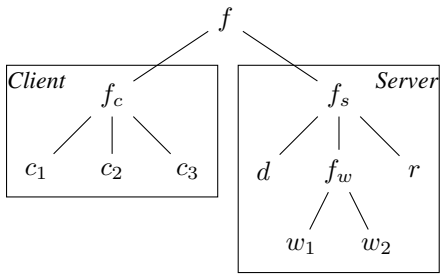

Figure 4. An Adaptative Petri net

information is logged and replicated), d_1 could represent the degraded mode with no replication, d_2 the one with replication but less information logged, d_3 less information logged and no replication and at last d_4 the write service is stopped. In the general case, all d_i are degraded modes of d_0, but d_{i+1} doesn't have to be a degraded mode of d_i.

In a more formal approach, an adaptive Petri net is a tree whose leaves are degradable components and nodes are connecting functions. This approach allows us to associate a degradable system to each subtree.

Conclusion

This paper described a formal approach based on trees and Petri nets to model real-time embedded systems. Our current work include the definition of a clear semantics for such a model. Intuitively, we need to explain (1) how such a model should be executed, (2) when and how it has to be degraded and (3) when and how it must recover, in order to provide a complete solution for resilient systems.

Degradations rules are not defined by the model but by the designers. Degradations rules must reflect priorities between components — which one is the less important and can be degraded to preserve a more important one — and adaptive capabilities of a system: if one component has to be changed, which part of the system do we have to degrade to keep it in a consistent state?

References

[1] A. Benveniste, S. Haar, E. Fabre, and C. Jard. Distributed monitoring of concurrent and asynchronous systems. *CONCUR 2003-Concurrency Theory*, pages 1–26, 2003.

[2] M. Boyer and O. H. Roux. On the compared expressiveness of arc, place and transition time Petri nets. *Fundamenta Informaticae*, 88(3):225–249, 2008.

[3] H.M. Hanisch. Analysis of place/transition nets with timed arcs and its application to batch process control. *Application and Theory of Petri Nets 1993*, pages 282–299, 1993.

[4] A. Madalinski and E. Fabre. Modular construction of finite and complete prefixes of Petri net unfoldings. *Fundamenta Informaticae*, 95(1):219–244, 2009.

[5] T. Robert, J.C. Fabre, and M. Roy. On-line monitoring of real time applications for early error detection. In *14th IEEE Pacific Rim International Symposium on Dependable Computing*, pages 24–31. IEEE, 2008.

[6] W. Zhou, O. Sokolsky, B. T. Loo, and I. Lee. DMaC: Distributed Monitoring and Checking. *Lecture Notes in Computer Science*, 5779:184, 2009.

Olivier Baldellon Alumnus of the Rennes extension of the *École Normale Supérieure de Cachan* in the computer science department owns a MsC from University of Rennes 1/IRISA, where he worked on distributed calculability and consensus with Michel Raynal. He is currently doing a PhD supervised by Matthieu Roy and Jean-Charles Fabre on formal tools to implement a distributed real-time monitoring system.

Définition de règles de sécurité-innocuité vérifiables en ligne pour des systèmes autonomes critiques

Amina Mekki-Mokhtar*‡, Jean-Paul Blanquart†, Jérémie Guiochet*‡ et David Powell*‡

* LAAS-CNRS, 7 Avenue du colonel Roche, 31077 Toulouse, France
‡ Université de Toulouse, UPS, INSA, INP, ISAE ; IUT, UTM, LAAS, Toulouse, France
{prenom.nom@laas.fr}

† EADS Astrium, 31 rue des cosmonautes, 31402 Toulouse, France
{jean-paul.blanquart@astrium.eads.net}

***Résumé*—Le développement des systèmes décisionnels a permis de rendre les systèmes réactifs de plus en plus autonomes et l'émergence de nouvelles applications dans des domaines tels que la robotique de service. En revanche, les défaillances éventuelles dans ces nouvelles applications peuvent avoir des conséquences catastrophiques. Afin d'assurer la sécurité-innocuité de tels systèmes, nous proposons dans cet article un processus de génération des règles de sécurité vérifiables en ligne implémentables dans un moniteur de sécurité indépendant.**

I. Introduction

Les progrès de l'intelligence artificielle et plus particulièrement des systèmes décisionnels font que les systèmes réactifs sont de plus en plus autonomes, et apparaissent maintenant dans des domaines comme la robotique de service. Cependant, ces systèmes sont critiques : leur défaillance peut entraîner des conséquences catastrophiques. Une technique privilégiée visant à assurer la sécurité des systèmes critiques, malgré la présence éventuelle de fautes de conception résiduelles ou l'occurrence de situations dangereuses non-prévues, consiste en la mise en place d'une surveillance au moyen d'un « moniteur de sécurité ». De tels moniteurs sont présentés dans la littérature sous différentes appellations comme : *safety manager* [10], *safety monitor* [9], *checker* [5], *guardian agent* [2], *safety bag* [6], etc. Les moniteurs de sécurité concernés par notre étude sont des mécanismes qui vérifient en ligne un ensemble de conditions de déclenchement de règles de sécurité ; si une de ces conditions est violée, une ou plusieurs actions de sécurité doivent être enclenchées afin de maintenir le système dans un état sûr. La sûreté de fonctionnement globale dépend, entre autres, de l'efficacité de ces règles. S'il existe de nombreux systèmes de sécurité de ce type, il n'existe en revanche que peu de travaux décrivant des méthodes permettant de déterminer les règles de sécurité. Nos travaux répondent à cette problématique en se situant dans le contexte des systèmes autonomes critiques où les règles de sécurité à mettre en oeuvre peuvent être complexes et différentes suivant les tâches qu'effectue le système fonctionnel.

Notre approche s'appuie sur la mise en place d'un processus systématique de génération de règles de sécurité à partir d'une analyse de risques basée sur une méthode intégrant la technique HAZOP (HAZard OPerability) [4] et le langage de modélisation UML (*Unified Modeling Language*) [8]. Le langage formel TPTL (*Timed Propositional Temporal Language*) [1] a été choisi afin d'exprimer ces règles dont certaines sont temps réel. Enfin, nous proposons nos perspectives d'implémentation de l'ensemble de ces règles au sein d'un moniteur de sécurité.

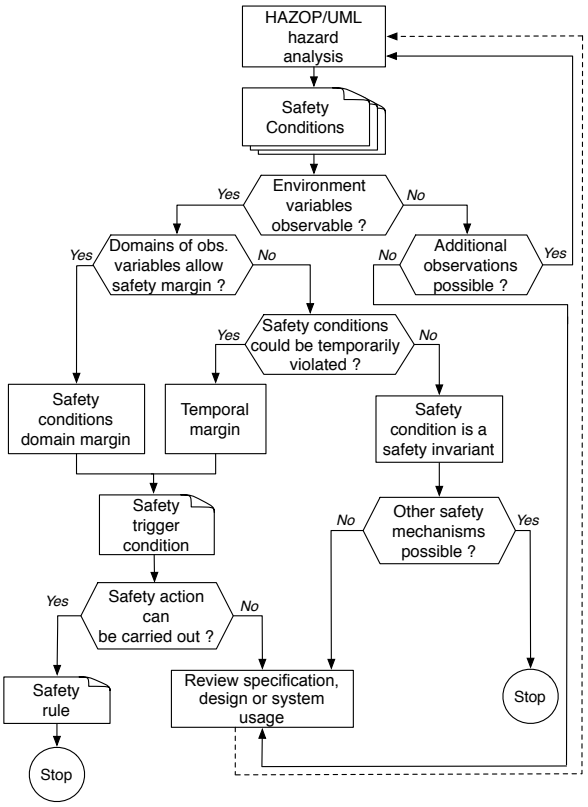

Figure 1. Processus de génération des règles de sécurité.

II. Vue générale du processus de détermination de règles de sécurité

Une vue générale du processus proposé est présentée Figure 1. L'objectif est de générer les règles de sécurité qui seront exécutées au sein du moniteur de sécurité. La première étape de notre processus est l'analyse de risques HAZOP/UML [3], [7], qui permet d'obtenir une liste de

comportements dangereux du système et une analyse informelle des causes potentielles et des conséquences (stockées dans des tables appelées tables de « déviations »). À partir de ces tables, une liste de *conditions de sécurité* est extraite. Nous définissons une condition de sécurité comme étant : *«une condition suffisante afin d'éviter une situation dangereuse»*. Chaque condition de sécurité est analysée selon les variables environnementales qu'elle contient. Si ces variables environnementales ne sont pas observables, des dispositifs d'observation additionnels devront être envisagés. Dans le cas contraire, nous essayons d'appliquer des marges de sécurité sur les valeurs des variables observables afin de définir une *condition de déclenchement* d'une *règle de sécurité* lorsque cela est possible. Cela revient à détecter une situation, que l'on qualifie de dangereuse, telle qu'il est encore possible d'entreprendre des actions (dites « actions de sécurité ») permettant de ramener le système dans un état sûr. Si les domaines des valeurs des variables observables ne permettent pas de définir des marges de sécurité, la possibilité de spécifier une marge temporelle est étudiée, en considérant que la condition de sécurité pourrait être temporairement violée. Si cela est possible, alors la dernière étape consiste à définir les actions à entreprendre pour ramener le système surveillé dans un état sûr. Dans le cas contraire, les conditions de sécurité doivent être considérées comme des *invariants de sécurité* et ne pourront pas servir à définir des règles de sécurité exécutables par le moniteur de sécurité. Les différentes sorties de ce processus sont : un ensemble de règles de sécurité, des invariants de sécurité qui devront être traités par d'autres dispositifs, comme des « interverouillages » par exemple, ou bien l'identification de la nécessité de revoir la spécification, la conception ou l'utilisation du système surveillé afin de permettre la mise en oeuvre d'une surveillance efficace.

III. Application au cas d'un robot d'aide à la déambulation

MIRAS (*Multimodal Interactive Robot for Assistance in Strolling*) est un projet développé conjointement par ISIR (Institut des Systèmes Intelligents et Robotiques), ROBOSOFT et LAAS-CNRS avec la participation de plusieurs hôpitaux de l'Île de France et de Toulouse. Ce projet a pour objectif de développer un robot semi-autonome d'aide à la déambulation. Il apportera une aide à la mobilité et la surveillance de l'état physiologique des personnes âgées atteintes de troubles de la marche et d'orientation. Il leur permettra aussi une assistance pour se lever et s'asseoir. Le processus de détermination des règles de sécurité a été appliqué et a permis d'identifier et de spécifier en langage formel TPTL plus de trente règles de sécurité. Nous travaillons actuellement sur l'exploitation de ces règles.

IV. Conclusions et perspectives

Notre objectif est de répondre au manque existant quant à la spécification de règles de sécurité implémentables au sein d'un moniteur indépendant. Notre approche propose un processus systématique pour la génération de ces règles de sécurité en se basant sur une analyse de risques du système surveillé. Nous proposons ainsi un ensemble de concepts et de formalismes afin d'exprimer les conditions de sécurité, conditions de déclenchement de sécurité et actions de sécurité. Ces concepts ont été appliqués lors de l'étude d'un cas concret de robot d'aide à la déambulation. La détermination des marges de sécurité est une étape cruciale. Elle nécessite la collaboration entre les concepteurs du système fonctionnel et les experts en sûreté de fonctionnement afin de trouver le bon équilibre entre sécurité et disponibilité du système surveillé, l'un étant souvent au détriment de l'autre. Le formalisme des marges de sécurité et notamment les conditions d'existence de ces marges reste un sujet sur lequel nous travaillons actuellement. Puis se posera la question des niveaux de l'architecture du système où seront effectuées l'observation et la réaction. Ces différentes réflexions permettront la spécification d'un *moniteur en ligne* qui implémentera les règles de sécurité.

Références

[1] Rajeev Alur and Thomas A. Henzinger. A really temporal logic. *J. ACM*, 41 : 181–203, January 1994.

[2] J. Fox and S. Das. *Safe and Sound - Artificial Intelligence in Hazardous Applications*. AAAI Press - The MIT Press, 2000.

[3] Jeremie Guiochet, Damien Martin-Guillerez, and David Powell. Experience with model-based user-centered risk assessment for service robots. In *IEEE International Symposium on High-Assurance Systems Engineering (HASE2010)*, pages 104–113, San Jose, CA, USA, 2010. IEEE Computer Society.

[4] IEC61882. Hazard and operability studies (HAZOP studies) - application guide. International Electrotechnical Commission, 2001.

[5] F. Ingrand and F. Py. Online execution control checking for autonomous systems. In *Proceedings of the 7th International Conference on Intelligent Autonomous Systems (IAS-7)*, Marina del Rey, California, USA, 2002.

[6] Peter Klein. The safety-bag expert system in the electronic railway interlocking system Elektra. *Expert Systems with Applications*, 3(4) : 499 – 506, 1991.

[7] Damien Martin-Guillerez, Jérémie Guiochet, and David Powell. Experience with a model-based safety analysis process for an autonomous service robot. In *IARP Workshop on Technical Challenges for Dependable Robots in Human Environments (DRHE 2010), Toulouse, France*, pages 1–8, 2010.

[8] OMG. 2nd revised submission to OMG RFP ad/00-09-02 - Unified Modeling Language : Superstructure - version 2.0. Object Management Group, 2003.

[9] S. Roderick, B. Roberts, E. Atkins, and D. Akin. The Ranger Robotic satellite servicer and its autonomous software-based safety system. *IEEE Intelligent Systems*, 19(5) : 12–19, 2004.

[10] D. Seward, C. Pace, R. Morrey, and I. Sommerville. Safety analysis of autonomous excavator functionality. *Reliability Engineering and System Safety*, 70(1) : 29 – 39, 2000.

Amina MEKKI MOKHTAR Après l'obtention de son diplôme d'ingénieur en informatique spécialité informatique industrielle, elle a intégré le master sécurité des systèmes d'information de l'université Paris Est Val de Marne où elle a pris goût aux aspects formels de la sécurité informatique. Puis, elle a rejoint en novembre 2009 l'équipe Tolérance aux fautes et Sûreté de Fonctionnement du LAAS-CNRS comme doctorante contractuelle. Dans le cadre de ses recherches, elle s'intéresse à l'élaboration de moniteurs de sécurité indépendants afin d'assurer la sécurité-innocuité des systèmes autonomes critiques.

Attaques par entrée-sortie et contremesures

Fernand Lone Sang*†, Vincent Nicomette*† et Yves Deswarte*†
*CNRS ; LAAS ; 7 avenue du colonel Roche, F-31077 Toulouse Cedex 4, France
†Université de Toulouse ; UPS, INSA, INP, ISAE ; UT1, UTM, LAAS ; F-31077 Toulouse Cedex 4, France

***Résumé*—Il devient aujourd'hui de plus en plus difficile de protéger efficacement les systèmes informatiques. Au fur et à mesure de l'accroissement de leur complexité, la surface d'attaque de ces systèmes s'est élargie et le nombre d'attaques perpétrées et réussies ne cessent de croître, en dépit des nombreux mécanismes de protection mis en place. Les attaques peuvent être réalisées par du logiciel s'exécutant sur le processeur, mais peuvent également reposer sur l'exploitation des entrées-sorties. Cet article s'intéresse à cette dernière catégorie d'attaques. Nous dressons un état de l'art de telles attaques sur les systèmes informatiques conçus autour de l'architecture Intel x86, notamment les ordinateurs personnels (PC) et, plus récemment, certains *System-On-Chip* (SoC). Nous détaillons ensuite quelques technologies matérielles qui permettent de s'en protéger et nous discutons finalement de leurs limites respectives.**

I. Introduction

De nos jours, les systèmes informatiques font partie intégrante de notre quotidien. Nous les utilisons, par exemple, pour travailler, pour échanger des informations, ou pour effectuer des achats. Malheureusement, ces systèmes sont encore régulièrement victimes d'attaques réussies, malgré les nombreux mécanismes de protection mis en place, en particulier dans l'architecture Intel x86. Les attaques peuvent être réalisées par du logiciel malveillant (*maliciel*) s'exécutant sur le processeur, en exploitant des erreurs d'implémentation matérielle ou logicielle (ex. un débordement de tampon). Mais elles peuvent aussi reposer sur l'exploitation des mécanismes d'entrée-sortie (E/S). C'est ce type d'attaques que nous étudions dans cet article. Nous commençons par rappeler quelques concepts propres à l'architecture Intel x86 en section II. Nous établissons ensuite un rapide état de l'art des attaques par E/S en section III, et nous présentons quelques technologies matérielles qui permettent de s'en protéger en section IV. Finalement, la section V conclut notre article et présente les perspectives envisagées.

II. Quelques éléments d'architecture

Une architecture matérielle Intel x86 typique se compose généralement d'un processeur et d'un *chipset*. Ce dernier se charge d'interconnecter le processeur, la mémoire principale et les différents contrôleurs d'E/S. Il se base pour cela sur deux puces distinctes, le *northbridge* et le *southbridge*. Le *northbridge* se charge d'interconnecter le processeur, la mémoire principale, les contrôleurs d'E/S nécessitant une bande passante importante (ex. la carte graphique) et le *southbridge*, chargé, lui, de relier les contrôleurs d'E/S nécessitant des ressources moindres (ex. les contrôleurs USB). Pour permettre au processeur de dialoguer avec les différents composants du système, l'architecture Intel x86 définit plusieurs espaces d'adressage. Nous nous focalisons, dans la suite de cet article, sur l'espace d'adressage mémoire dans lequel est projeté la mémoire principale ainsi que certains registres des contrôleurs d'E/S, voire même leur mémoire interne. On parle alors de *Memory-Mapped I/O* (MMIO)[1]. Grâce au *chipset*, l'accès à ces différents emplacements s'effectue de manière transparente. Le processeur accède à cet espace par l'intermédiare d'une *Memory Management Unit* (MMU). Il est également possible d'y accéder depuis un contrôleur d'E/S via le mécanisme matériel dit *Direct Memory Access* (DMA). La section qui suit effectue un état de l'art des attaques qui s'appuient sur les entrées-sorties, en particulier le mécanisme de DMA.

III. Classes d'attaques par les entrées-sorties

Il existe de nombreuses attaques par les E/S utilisant les mécanismes de DMA. Cette section détaille certaines de ces attaques, classées en fonction des éléments qu'elles ciblent.

A. Attaques DMA visant la mémoire principale

La plupart des attaques DMA existantes ciblent la mémoire principale. Elles cherchent à violer les propriétés de sécurité (intégrité et confidentialité) des composants logiciels chargés en mémoire ou des données manipulées par ceux-ci. Dans certaines circonstances, il est légitime d'utiliser les fonctionnalités d'un contrôleur d'E/S pour lire la mémoire principale. Tribble [1] et Copilot [2], des contrôleurs PCI dédiés à l'*inforensique* (investigation informatique légale), sont des exemples d'usage légitime. Dans des cas moins légitimes, le DMA peut être exploité pour corrompre la mémoire principale. L. Duflot [3] a montré par exemple qu'un code malveillant, s'exécutant avec peu de privilèges, peut mettre en place des transferts DMA depuis un contrôleur USB UHCI afin d'acquérir plus de privilèges. L'exécution d'un code malveillant sur le système attaqué n'est pas toujours nécessaire et un accès physique peut suffire. M. Dornseif [4], [5] et A. Boileau [6] ont montré qu'un périphérique FireWire (ex. un iPod modifié) peut accéder à la totalité de la mémoire principale au travers du contrôleur FireWire. Par la suite, plusieurs preuves de concept d'attaques au travers du bus FireWire ont été publiées : D.R. Piegdon [7] s'est intéressé, par exemple, aux techniques de compromission des systèmes Linux tandis que D. Aumaitre [8] s'est concentré sur les systèmes Windows. Finalement, L. Duflot *et al.* [9] ont montré récemment qu'un accès DMA peut être commandé à distance, suite à l'exploitation d'une vulnérabilité dans le *firmware* d'une carte réseau. G. Delugré [10] a étendu leur travaux en développant des outils de rétro-ingénierie pour étudier le *firmware* qu'ils ont exploité et a démontré que ce dernier aurait pu être modifié de façon à intégrer un *rootkit* indétectable.

[1]Précisons que l'architecture Intel x86 définit un autre mode d'E/S appelé *Programmed I/O* qui n'est plus guère utilisé de nos jours.

B. Attaques DMA visant les contrôleurs d'E/S

Certaines attaques par E/S ciblent des registres ou la mémoire interne des contrôleurs d'E/S. De telles attaques cherchent à exploiter directement les ressources fournies par le matériel, tout en contournant les mécanismes de protection de la mémoire principale. Afin de souligner que ces attaques s'effectuent directement entre contrôleurs d'E/S, nous désignons de telles attaques par « attaques DMA *peer-to-peer* ». M. Dornseif a probablement été l'un des premiers à démontrer la faisabilité de telles attaques. Pour illustrer ses travaux publiés dans [4], il a développé, sur des *chipsets* d'ancienne génération, une preuve de concept d'attaque DMA dans laquelle il modifie le *framebuffer* d'une carte graphique depuis un iPod connecté via le bus FireWire. Nous avons développé une preuve de concept [11] similaire pour illustrer nos travaux consacrés à ce type d'attaques [12] sur les *chipsets* intégrant davantage de moyens de protection. Finalement, des scénarios d'attaques plus complets ont été développés par A. Triulzi. La modification du *firmware* d'une carte réseau lui a permis de transférer les trames respectant un certain motif dans la mémoire interne de la carte graphique, où a été implanté un serveur *shell* sécurisé [13]. Il a ensuite étendu ses travaux en montrant que cet accès distant peut être utilisé pour charger, depuis la carte graphique, un nouveau *firmware*, téléchargé depuis l'Internet, dans une autre carte réseau [14]. Ce dernier lui a permis d'altérer le comportement de la carte réseau de façon à ce que certaines trames soient directement transférées vers une autre carte réseau, contournant ainsi tout filtrage de paquets mis en œuvre au sein du système d'exploitation.

IV. Contremesures matérielles

Il est possible de s'aider de technologies matérielles, telles que des *Input/Output Memory Management Units* (I/O MMU), pour se protéger de ces différentes attaques DMA. Une I/O MMU est un ensemble de composants matériels intégrés aux *chipsets* récents qui permettent de virtualiser la mémoire principale pour les contrôleurs d'E/S et de filtrer, entre autres, leurs accès à celle-ci. Ces composants sont généralement placés dans le *northbridge*, en coupure entre le bloc formé par les contrôleurs d'E/S et la mémoire principale. Du fait de leur agencement dans l'architecture matérielle, ces composants voient transiter toutes les requêtes d'accès à la mémoire principale depuis les contrôleurs d'E/S et protègent ainsi la mémoire principale de tout accès frauduleux. En revanche, ces composants sont moins efficaces contre les attaques entre contrôleurs d'E/S, en particulier ceux internes au *southbridge*. Il est probable, par exemple, que la preuve de concept développée par A. Triulzi [14] continue à fonctionner malgré l'utilisation d'une I/O MMU. Nous avons d'ailleurs montré qu'une I/O MMU présente généralement certaines limites qui peuvent être exploitées par des attaquants [15]. Pour répondre à ces problématiques de sécurité, le PCI-SIG, consortium en charge de la spécification du bus d'E/S PCI Express, a défini un ensemble d'extensions regroupées sous le nom d'*Access Control Services* (ACS). Les composants implémentant ces extensions peuvent être configurés pour mettre en œuvre différents types de contrôle d'accès, en particulier pour transférer toute communication entre contrôleurs d'E/S au *northbridge* afin d'être validée par une I/O MMU par exemple, ou tout simplement pour les bloquer. Il est important de noter que l'activation de ces services peut avoir un impact significatif sur les performances. J. Rutkowska [16] a proposé une technique permettant de protéger des régions quelconques de la mémoire principale contre d'éventuelles attaques DMA pour des systèmes dépourvus d'I/O MMU. Son idée consiste à reconfigurer le *chipset* de façon à ce que le *northbridge* redirige automatiquement tous les accès depuis les contrôleurs d'E/S aux régions de mémoire que l'on souhaite protéger vers un autre emplacement. Une tentative d'accès à ces régions de mémoire protégées depuis un contrôleur d'E/S peut provoquer, par exemple, un blocage complet du système en redirigeant l'accès vers un emplacement invalide. Il est cependant difficile de mettre en œuvre cette technique pour bloquer les attaques entre contrôleurs d'E/S.

V. Conclusion

Dans ce papier, nous avons effectué un rapide état de l'art des attaques par E/S ciblant les systèmes informatiques conçus autour de l'architecture Intel x86. Notre étude a permis de dégager deux familles d'attaques par E/S : (1) celles qui ciblent la mémoire principale et (2) celles qui ciblent les éléments d'autres contrôleurs d'E/S. Nous avons également présenté quelques technologies matérielles qui permettent de s'en protéger et nous avons évoqué certaines de leurs limites. Comme suite à nos travaux, nous envisageons de caractériser plus finement les attaques par E/S afin d'en déduire une classification plus exhaustive et de proposer quelques contremesures génériques pour chaque classe identifiée.

Références

[1] B. Carrier and J. Grand, "A Hardware-based Memory Acquisition Procedure for Digital Investigations," *Digital Investigation Journal*, vol. 1, no. 1, pp. 50–60, Feb. 2004.

[2] J. Nick L. Petroni, T. Fraser, J. Molina, and W. A. Arbaugh, "Copilot - a Coprocessor-based Kernel Runtime Integrity Monitor," in *Proceedings of the 13th USENIX Security Symposium*, 9-13 Aug. 2004.

[3] L. Duflot, "Contribution à la sécurité des systèmes d'exploitation et des microprocesseurs," Thèse de Doctorat, Université de Paris XI, Oct. 2007. [Online]. Available : http://www.ssi.gouv.fr/archive/fr/sciences/fichiers/lti/these-duflot.pdf

[4] M. Dornseif, "0wned by an iPod - hacking by Firewire," in *PacSec/core04*, 11-12 Nov. 2004.

[5] M. Becher, M. Dornseif, and C. N. Klein, "FireWire - all your memory are belong to us," in *CanSecWest/core05*, 4-5 May 2005.

[6] A. Boileau, "Hit by a Bus : Physical Access Attacks with FireWire," in *RUXCON 2006*, Oct. 2006.

[7] D. R. Piegdon, "Hacking in Physically Addressable Memory," in *Seminar of Advanced Exploitation Techniques, WS 2006/2007*, 12 Apr. 2007.

[8] D. Aumaitre, "A Little Journey Inside Windows Memory," *Journal in Computer Virology*, vol. 5, no. 2, pp. 105–117, 2009.

[9] L. Duflot, Y.-A. Perez, G. Valadon, and O. Levillain, "Can you still trust your Network Card ?" in *CanSecWest/core10*, 24-26 Mar. 2010.

[10] G. Delugré, "Closer to metal : reverse-engineering the Broadcom NetExtreme's firmware," in *Hack.lu*, Luxembourg, 27-29 Oct. 2010.

[11] F. Lone Sang, V. Nicomette, and Y. Deswarte, "Démonstration d'une attaque DMA *peer-to-peer* sur une carte graphique," Jan. 2011. [Online]. Available : http://homepages.laas.fr/nicomett/Videos/

[12] F. Lone Sang, V. Nicomette, Y. Deswarte, and L. Duflot, "Attaques DMA *peer-to-peer* et contremesures," in *Actes du 9ème Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC)*, Jun. 2011, (à paraître : http ://www.sstic.org/2011/actes/).

[13] A. Triulzi, "Project Maux Mk.II - « I 0wn the NIC, Now I want a Shell ! »," in *PacSec/core08*, 12-13 Nov. 2008.

[14] ——, "The Jedi Packet Trick takes over the Deathstar (or : « Taking NIC Backdoors to the Next Level »)," in *CanSecWest/core10*, Mar. 2010.

[15] F. Lone Sang, E. Lacombe, V. Nicomette, and Y. Deswarte, "Exploiting an I/OMMU Vulnerability," in *Proceedings of the 5th IEEE Intl. Conf. on Malicious and Unwanted Software (MALWARE)*, Oct. 2010, pp. 7–14.

[16] J. Rutkowska, "Beyond The CPU : Defeating Hardware Based RAM Acquisition," in *BlackHat DC*, 28 Feb. 2007.

Attaques physiques à haut niveau pour le test de la sécurité des cartes à puce

P. Berthomé*, K. Heydemann†, X. Kauffmann-Tourkestansky‡, J.-F. Lalande*
*Centre-Val de Loire Université, Ensi de Bourges, LIFO, 18000 Bourges
†Université Pierre et Marie Curie, LIP6, 75252 Paris
‡Oberthur Technologies, Ensi de Bourges, LIFO, 92726 Nanterre
Emails : pascal.berthome@ensi-bourges.fr, karine.heydemann@lip6.fr,
x.kauffmann@oberthur.com, jean-francois.lalande@ensi-bourges.fr

***Résumé*—Dans cet article, nous proposons de décrire les hypothèses d'attaques physiques contre les cartes à puce afin de modéliser ces attaques à haut niveau. Cette modélisation cherche à représenter l'attaque au niveau du langage C par l'injection d'un morceau de code qui simule ses effets. L'intérêt du modèle est qu'il permet de simuler les attaques possibles à un niveau où le programmeur peut comprendre les effets sur le code qu'il développe. Cependant, le nombre d'attaques possibles est très grand ce qui empêche la réalisation exhaustive de tous les tests. Les résultats expérimentaux montrent comment identifier par simulation les attaques par saut qui aboutissent. Enfin, nous présentons nos perspectives de travaux qui concernent la vérification statique de ces codes attaqués.**

I. Introduction

Les attaques contre les cartes à puce peuvent être physiques [1]. En surveillant l'activité d'une carte à puce, des impulsions lasers peuvent réussir à perturber l'exécution du programme qui s'y déroule. Les conséquences sont difficilement contrôlables mais, à force de tentatives, peuvent conduire l'attaquant à réussir une perturbation qui lui donne un avantage par rapport aux sécurités implémentées dans la carte.

Contrecarrer ces attaques est un problème proche de l'étude du domaine de la tolérance aux fautes. [6] donne une comparaison très précise de ces deux problèmes. La différence majeure réside dans le fait qu'une faute est une erreur aléatoire alors qu'une attaque est une faute semi-contrôlée qui cherche à perturber un point précis du programme. De plus, les logiciels tolérants aux fautes cherchent souvent à être robustes à ces fautes. Pour la carte, la détection de l'attaque ou le crash du programme sont préférables à la continuation de l'exécution.

Dans le processus de développement du code pour carte à puce, représenté sur la figure 1, l'étude des attaques peut se faire en attaquant la carte physiquement, par exemple en étudiant la consommation énergétique ou le champ magnétique de celle-ci [4], [5]. Le matériel dévolu à ce type d'étude est coûteux et le processus long. Il est aussi possible de réaliser une campagne de tests dans un environnement simulé, par exemple en perturbant le binaire obtenu après compilation. Si une telle étude est plus aisée à mettre en pratique, l'étude des attaques physiques n'en est pas moins difficile.

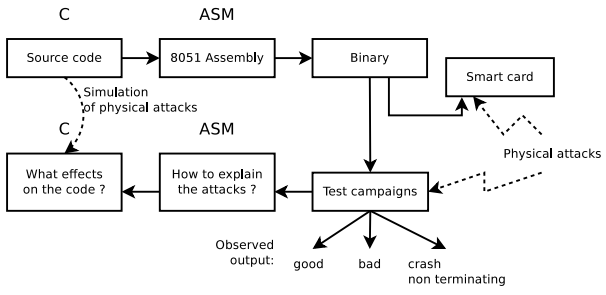

FIGURE 1. Méthodologie proposée

Comme représenté sur la figure 1, l'évaluation d'une attaque peut être classée dans trois catégories. La catégorie «good» est une exécution où l'attaque ne fait pas dévier le comportement du programme, par exemple le contenu de la sortie attendue. Dans le cas de la carte à puce, on peut considérer les trames de transaction avec le terminal. La catégorie «bad» correspond aux exécutions où l'attaque permet d'obtenir une plus-value pour l'attaquant. Cette plus-value peut être la fuite d'une information exploitable dans les trames de transaction ou un comportement déviant du comportement attendu. Dans ce cas, l'attaquant est susceptible de raffiner son attaque et d'arriver à produire une attaque encore plus dangereuse. Une dernière catégorie regroupe les exécutions où l'attaque produit un crash ou bloque le programme : ces deux comportements sont «bons» d'un point de vue de la sécurité.

L'étude d'une attaque donnant un résultat classé «bad» est un problème difficile puisqu'il s'agit de comprendre les effets d'une perturbation du code exécutable, par exemple la modification d'une opération arithmétique ou le saut d'un point à un autre du programme, comme présenté en Section II-A. Il est encore plus difficile, mais pas impossible, de revenir au code assembleur puis au code source C originel pour comprendre ce que l'attaque physique a perturbé au niveau des variables ou du contrôle de flot. Pour contourner cette difficulté, nous proposons de simuler les attaques directement au niveau du code C afin de s'affranchir des étapes passant par le code assembleur.

II. Méthodologie

Cette section présente un exemple d'attaque et les grands principes de la méthodologie de simulation proposée.

A. Exemple d'attaque

On considère le code représentant la vérification du PIN :

```
CLR A
MOV R5,A
MOV R4,A
MOV R7,#0D2H
MOV R6,#04H
LCALL _check_pin
MOV  pin_status,R7
RET
```

```
pin_status = check_pin(1234,0000);
return pin_status;
```

La dernière instruction MOV pourrait être attaquée au travers de son second argument, comme montré ci-après. Les effets se propageraient alors jusqu'au RET qui renverrait un mauvais résultat. Du point de vue du C, on peut simuler cette attaque comme dans le listing de droite.

```
CLR A
MOV R5,A
MOV R4,A
MOV R7,#0D2H
MOV R6,#04H
LCALL _check_pin
MOV   pin_status ,#OK
RET
```

```
pin_status = check_pin(1234,0000);
pin_status = OK;
return pin_status;
```

D'autres types d'attaques sont possibles, comme la perturbation du code opération d'une instruction (MOV ⇒ NOP), le remplacement d'un code opération par un JUMP, l'écrasement de la valeur d'un registre.

B. Injection d'attaques

L'injection d'attaques simulées au niveau du code source C permet de couvrir un sous-ensemble des attaques possibles au niveau de l'exécutable en imitant les conséquences que l'attaque réelle produit. Les points d'attaque et la nature de l'attaque font exploser le nombre d'attaques à simuler. Par exemple, si l'on considère une fonction à n instructions C et une attaque perturbant l'une des k variables ayant chacune une prise de valeur dans un domaine D, on doit simuler $n.k.|D|$ attaques du type *inst_1 ; attaque ; inst_2 ; ... inst_n ;*. Ceci étant, nous montrons dans [2] comment réduire les points d'attaques aux points pertinents en éliminant les points d'attaques inutiles c'est-à-dire sans effet. Par exemple *... x = attaque; x = 5; ...* est un exemple trivial d'attaque inutile.

Le deuxième type d'attaques intéressantes à couvrir sont les attaques par saut forcé qui se simulent aisément au niveau C en utilisant des *goto*. Il devient raisonnable de simuler l'ensemble des attaques possibles sur une fonction de taille n puisqu'il suffit de tester n^2 attaques possibles.

C. Exploitation du modèle d'attaque

La simulation des attaques physiques au niveau C permet d'obtenir des nouveaux codes C que l'on peut exploiter à l'aide de méthodologies de test ou de vérification. L'élaboration de batteries de tests permettent de s'assurer que le code fonctionnel s'exécute correctement et tolère l'attaque en l'ignorant (cas «good») ou en provoquant un crash (cas «crash»). Le code sécuritaire peut aussi être mis à l'épreuve et se révéler capable de réagir à l'attaque. Dans le cas contraire, il s'agit d'une attaque qui fait dévier le comportement attendu sans qu'une réponse sécuritaire ne se déclenche (cas «bad»).

La Figure 2 montre l'exploitation de la méthodologie sur un cas concret : il s'agit de l'étude du code de bzip2 issu des benchmarks SPEC 2006 au travers de l'attaque exhaustive par injection de sauts dans la fonction *BZ2_compressBlock*. Les axes donnent le numéro des lignes source et destination de l'attaque par saut qui est injectée pour chaque test. On repère aisément la séparation en diagonale entre les attaques devant être interrompues (kill) car provoquant des boucles infinies, et les attaques classées «good» c'est-à-dire ne perturbant pas la sortie de l'exécution de bzip2. Quelques attaques particulières réussissent (cas «bad»), l'exécution se termine et est assortie d'un résultat de compression perturbé. Ce sont précisément

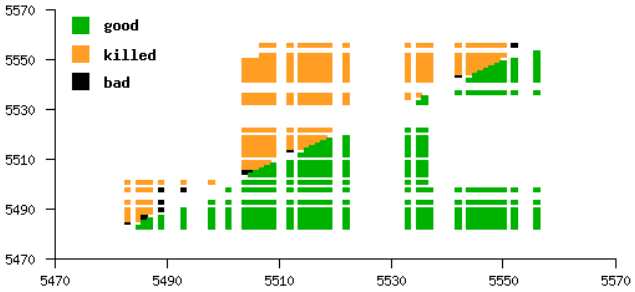

FIGURE 2. Profil de BZ2_compressBlock : classement de l'attaque en fonction de la ligne source et de la ligne destination du saut de l'attaque

ces attaques qu'il convient ensuite d'investiguer pour pouvoir implémenter d'éventuelles contre mesures sécuritaires.

III. CONCLUSION ET PERSPECTIVES

Ce papier présente la modélisation d'attaques physiques au niveau du code source C afin de simuler les effets d'attaques réelles contre le code embarqué dans des cartes à puce. Comme le montre les résultats expérimentaux, la modélisation des attaques à haut niveau simulant leur effet permet de déterminer les attaques qui réussissent.

Par la suite, nous envisageons d'investiguer les possibilités offertes par les outils de vérification statique, tels que Frama-C [3] qui pourront permettre de prouver formellement si une propriété du code source, auparavant garantie sur le code fonctionnel, devient violée pour le code sur lequel l'attaque simulée est injectée. La difficulté majeure de ce type d'approche réside dans l'ampleur du code à analyser et dans la modélisation des propriétés à vérifier.

RÉFÉRENCES

[1] R. Anderson and M. Kuhn. Tamper resistance-a cautionary note. In *Proceedings of the second USENIX Workshop on Electronic Commerce*, volume 2, pages 1 – 11, 1996.

[2] Pascal Berthomé, Karine Heydemann, Xavier Kauffmann Tourkestansky, and Jean-François Lalande. Attack model for verification of interval security properties for smart card C codes. In *5th ACM SIGPLAN Workshop on Programming Languages and Analysis for Security PLAS '10*, ISBN :978-1-60558-827-8, pages 1–12, Toronto Canada, 2010. ACM.

[3] Géraud Canet, Pascal Cuoq, and Benjamin Monate. A Value Analysis for C Programs. *2009 Ninth IEEE International Working Conference on Source Code Analysis and Manipulation*, pages 123–124, 2009.

[4] T.S. Messerges, E.A. Dabbish, and R.H. Sloan. Investigations of Power Analysis Attacks on Smartcards. *Power*, 1999.

[5] Jean-Jacques Quisquater and David Samyde. ElectroMagnetic Analysis (EMA) : Measures and Counter-measures for Smart Cards. In Isabelle Attali and Thomas Jensen, editors, *Smart Card Programming and Security*, volume 2140 of *Lecture Notes in Computer Science*, pages 200–210. Springer Berlin / Heidelberg, 2001.

[6] Philippe Teuwen. How to Make Smartcards Resistant to Hackers' Lightsabers ? In J. Guajardo and B. Preneel and A.-R. Sadeghi and P. Tuyls, editor, *Foundations for Forgery-Resilient Cryptographic Hardware*, pages 1–8, Dagstuhl, 2010.

Jean-François Lalande est Maître de conférences à l'ENSI de Bourges dans le Laboratoire d'Informatique Fondamentale d'Orléans (LIFO) depuis 2005. Il s'intéresse à la sécurité des systèmes d'exploitation et des programmes embarqués. Docteur en informatique de l'Université de Nice, il a travaillé sur le dimensionnement de réseaux de télécommunication dans l'équipe Mascotte (INRIA/I3S/CNRS/UNS).

Modeling and Detecting Intrusions in ad hoc Network Routing Protocols

Mouhannad Alattar
LIFC Laboratory
University of Franche-Comté, France
Email: firstName.lastName@univ-fcomte.fr

Francoise Sailhan
CEDRIC Laboratory
CNAM-Paris, France
Email: firstName.lastName@cnam.fr

Julien Bourgeois
LIFC Laboratory
University of Franche-Comté, France
Email: firstName.lastName@univ-fcomte.fr

***Abstract*—Ad hoc networks mostly operate over open and unprotected environments and are as such vulnerable to a wide range of attacks. Preventive techniques e.g., firewall and encryption, are no longer sufficient and should be coupled with advanced intrusion detection. We propose a distributed intrusion detection system that analyses activity logs so as to generate the rules which are used to detect intrusion. In order to deal with the distributed nature of an *ad hoc* network, the proposed system correlates information found in the multiple traces provided by surrounding devices. Performance is further evaluated, in terms of e.g., intruder detection rate and false positive.**

I. Introduction

Securing *ad hoc* networks is not a trivial task because these networks rely on open radio-based medium of communication. In addition, the lack of centralized points e.g., switches and routers, complicates the deployment of preventive strategies. Thus, traditional ways of securing networks e.g., firewall, should be enriched with reactive mechanisms including intrusion detection system. As a first step upon detecting intrusions, we first categorize the attacks threatening routing protocols; their central role, i.e., determining multi-hops paths among the devices, designates these latter as the favorite target of attackers. Then, we propose a Distributed Intrusion Detection System (DIDS), which analyses the activity logs so as to discover a sequence of events that characterizes an intrusion attempt. Then, logs are correlated so that the DIDS correctly identifies more intrusions and reduces the number of false positive. We further exemplify and experimentally evaluate the performance of our DIDS focusing on a specific attacks, the so-called spoofing link attack, which aims at undermining the Optimized Link State Routing (OLSR) protocol [4].

The remainder of this paper is organized as follows. We first present the attacks on *ad hoc* routing protocols (§II). Then, we envisage the development of the proposed DIDS (§III) and its evaluation (§IV). Finally, we conclude this article presenting future research directions (§V).

II. Attacks on OLSR

In *ad hoc* network, routing protocols constitute one of the favourite target of intruders; the reason is threefold. First, no security countermeasure is specified as a part of the RFCs proposed by IETF working group[1]. Second, the absence of the centralized infrastructure complicates the deployment of preventive measures e.g., firewalls. Third, any device may operate as router, which facilitates the manipulation of multi-hops messages as well as the compromising of the routing functionality. Attacks targeting *ad hoc* routing protocol fall into two main categories, passive (i.e., observing the traffic) *versus* active (i.e., unauthorized change is attempted). Active attacks are further sub-classified according to the undertaken action on the routing messages [1]:

- *Drop attacks* consist in dropping one (or further) routing message(s).
- *Modify and forward attacks* modify received routing message(s) before forwarding it.
- *Forge reply attacks* aim at sending false response(s) to routing message(s).
- *Active forge attacks* proactively generate novel routing message(s).

In this paper, we focus on a particular active forge attack corresponding to a spoofing link attack. We exemplify this attack using the OLSR protocol. In a nutshell, OLSR aims at maintaining a constantly updated view of the network topology on each device. One fundamental of OLSR is the notion of multipoint relay (MPR): a subset of 1-hop neighbors that covers all the 2-hops neighboors and forwards the control traffic in the entire network. In practice, a node recognizes the 1-hop neighboors through periodic heartbeat messages, termed Hello messages. A MPR declares the nodes that selected itself to act as MPR via *Topology Control* (TC) message, which is intended to be diffused in the entire network. Thanks to TC messages, any device computes the shortest path (in term of number of hops) to any destination, such path being represented as a sequence of MPRs. Overall, OLSR is subject to a variety of attacks, including spoofing link attack, that majoritly target MPRs ; these latter constituing an attractive post for launching further attacks (e.g., deleting messages). In practice, one possible strategy consists in corrupting the MPR selection.

A. Spoofing link attack

An attacker aspiring being selected as a MPR may corrupt the MPR selection (equation 1). Towards this goal, the intruder advertises a falsified local topology: I sends a Hello message ($V \xleftarrow{Hello(NS'_I)} I$) to a victim V so as to declare a neighboor set NS'_I differing from the real set NS_I. The difference results from inserting[2] (i) a non-existing node N ($N \notin \mathcal{N}$ with $\mathcal{N}$ defining the set of nodes composing the

[1] http://www.ietf.org/dyn/wg/charter/manet-charter.html

[2] A falsification of the local topology may also consists in suppressing existing neighboor(s) in the advertised neighbooring set. Nevertheless, rather than facilitating the selection of the intruder as MPR, this alteration reduces the connectivity of the intruder perceived by other and hence mortgages the probability of being selected as MPR.

network) and/or (ii) an existing, but non-neighbooring, node E ($E \in NS'_I \cap \mathcal{N} \ni: E \notin NS_I$). Thus, the connectivity of I is increased $Card(NS'_I \backslash NS'_I \cap NS_I) > 0$. Recall that the set of MPRs is selected so that all the 2-hop neighboors are covered, I is hence selected as a MPRs of V. This affirmation is verified as long as no other intruder I' (respectively legitimate node M) advertises the same neighboor N (respectively E) .

$$\begin{array}{c} I \xleftarrow{Hello(NS_V)} V, V \xleftarrow{Hello(NS'_I)} I, \\ (\exists N \in NS'_I \ni: N \notin \mathcal{N} \cap NS_I) \vee (\exists E \in NS'_I \cap \mathcal{N} \ni: E \notin NS_I) \\ \Downarrow \qquad\qquad\qquad\qquad \Downarrow \\ Card(NS'_I \backslash NS'_I \cap NS_I) > 0, \qquad Card((NS'_I \backslash [NS'_I \cap NS_I]) \cap \mathcal{N}) > 0, \\ \exists I' \in \mathcal{I} \ni: I' \in NS_V \wedge N \in NS_{I'} \qquad \exists M \in \mathcal{N} \backslash \mathcal{I} \ni: M \in NS_V \wedge E \in NS_M \\ \Downarrow \qquad\qquad\qquad\qquad \Downarrow \\ I \vee I' \in MPR_V, \qquad\qquad I \vee M \in MPR_V, \\ \Downarrow \\ I \in \mathcal{I}. \end{array} \quad (1)$$

Such an attack can be detected relying on an intrusion detection system.

III. Intrusion Detection

We proposed a DIDS that traces and detects the source of a network-based intrusion. This DIDS includes a host-based tracing mechanism that keeps track of the OLSR activities (routing logs) and analyses these latter to detect evidences. Each device further cooperates with one another so as to correlate evidences and match it against predefined intrusion signatures. In spoofing link attack, recall that the intruder increases artificially its connectivity so as to be chosen as MPR, we define three evidences that render a MPR suspect:

- a MPR relays packets in an abnormal way e.g., packets are dropped and do not reach their destination,
- a MPR is replaced suspiciously by a new MPR,
- a MPR is the only one covering one node.

If one of the above evidence is discovered, advanced correlation and investigation is performed: messages are exchanged with the 2-hop neighboors that are covered by the suspicious MPR.

IV. Performance evaluation

In order to evaluate our DIDS, we couple a network simulator (NS3)[3] [3] with LinuX Containers virtual machines (LXC)[4] [2]. NS3 simulates a MANET (and hence owns an implementation of the OLSR protocol) while each simulated devices owns a LXC container embeding a DIDSs. While offering the capability of monitoring the memory consumption of each node in the virtual machine, this platform permits to easily experiment a MANET (herein 25 mobile nodes including 5 intruders are simulated). Performance is further evaluated in terms of intruder detection rate, resource consumption and false positive (Figure 1). The detection rate rises up to 95% with a node's speed equal to 10 m/s. Increasing the speed leads to a decrease of the detection rate that reaches 55% with a node speed of 20 m/s (i.e., 180 km/h). This decrease results from the difficulty to obtain investigation correspondences when the devices mobility rises. In counterpart, the number of false positive observed is limited (ranging between 0 up to 3) and has almost no relation with the mobility. The network overhead resulting from the message generated by the DIDS is 10 times smaller than the overhead attributed to the OLSR protocol. Furthermore, our system causes an increment in the container used memory ranging from 16 to 52 MB, meaning that such a DIDS can be deployed on resource-constrained devices.

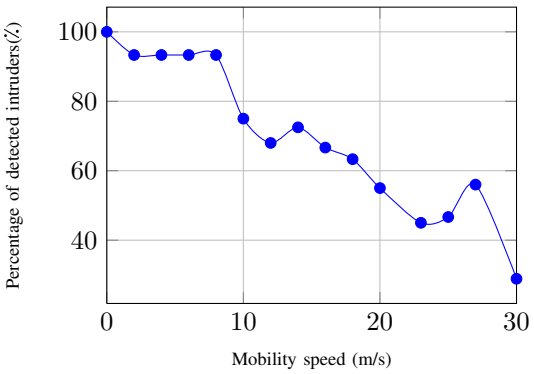

Fig. 1. Detection rate given a varying mobility.

Overall, our system has an accepted (respectively high) intruder detection rate in high (respectively moderate) mobility network.

V. Conclusion and Future work

In this paper, we present a distributed intrusion detection system that is focusing on the network intrusions targeting the routing protocol in MANET. The proposed DIDS reffers to a host-based detection system that matches logs against intrusion signatures. It requires no prior knowledge about the *ad hoc* network. Therefore, our system has an advantage over anomaly-based IDSs, which search for deviations from a normal expected behavior (and hence is generated from training data) so as to detect the intrusions. In counterpart, our system cannot detect a not-defined intrusion. DIDS performance is investigated against spoofing link attack. The detection rate reaches up to (95%) with a mobility speed more than the running average speed of a human being. The number of false positive is limited. Even though the amount of logs to be analyzed increases exponentially with network size, resource consumption is still suitable for the resource-constrained devices.

References

[1] N. Peng and S. Kun, *How to misuse aodv:a case study of insider attacks against mobile ad-hoc routing protocols*. Elsevier Science Publishers B. V., 2005.

[2] Sukadev Bhattiprolu and al., *Virtual servers and checkpoint/restart in mainstream linux*. SIGOPS Oper. Syst. Rev., 2008.

[3] George F. Riley and Thomas R. Henderson, *The ns-3 network simulator*, In Modeling and Tools for Network Simulation. Springer Berlin Heidelberg, 2010.

[4] T. Clausen and P. Jacqueti, *Optimized link state routing protocol (olsr)*. IETF experimental RFC 3626, october 2003.

Mouhannad Alattar is currently a PhD student at the LIFC laboratory in University of Franche-Comté. His current research is centered around securing *ad hoc* networks. He received the M.S. degree in computer networks from the University of Franche-Comté in 2009. Before, he received Bachelor degree in Computer Science and Software Engineering from Al-Baath University, Homs, Syria, in 2006. Mouhannad also worked as a developer and system analyzer in Jabco company for Web and software solutions from 2005 to 2008, Syria.

[3] http://www.nsnam.org

[4] http://lxc.sourceforge.net

Towards a System Architecture for Resilient Computing

Miruna STOICESCU* Jean-Charles FABRE Matthieu ROY
CNRS ; LAAS ; 7 avenue du colonel Roche, F-31077 Toulouse Cedex 4, France
Université de Toulouse ; UPS, INSA, INP, ISAE ; UT1, UTM, LAAS ; F-31077 Toulouse Cedex 4, France

***Abstract*—Nowadays, systems are not only becoming increasingly complex and heterogeneous but are also opened towards their environment by means of context-awareness. Furthermore, in the vast majority of cases, their specifications periodically evolve, leading to new versions. As resilient systems are expected to continuously provide trustworthy services, they must cope with changes coming from the environment or from the specifications and reconfigure in order to adapt to them. We propose a framework for designing and developing such systems.**

***Keywords*—reconfigurable, adaptive, pervasive, fault-tolerant**

I. PROBLEM STATEMENT

The evolution of systems is ineluctable and leads to the notion of *resilient computing*[3]. Among the evolution scenarios that can occur during the operational life of a fault-tolerant application, we focus on those that may have an impact on the fault tolerance mechanisms:

- one or several changes in the application assumptions which influence the choice of the fault tolerance mechanism,
- changes in the requirements in terms of fault tolerance,
- changes in the system configuration (e.g. number of processors, memory resources, network bandwidth).

The essence of a *resilient application* lies in the fact that when faced with any of these changes or even with a combination of them, it must adapt itself while ensuring the observance of the safety criteria.

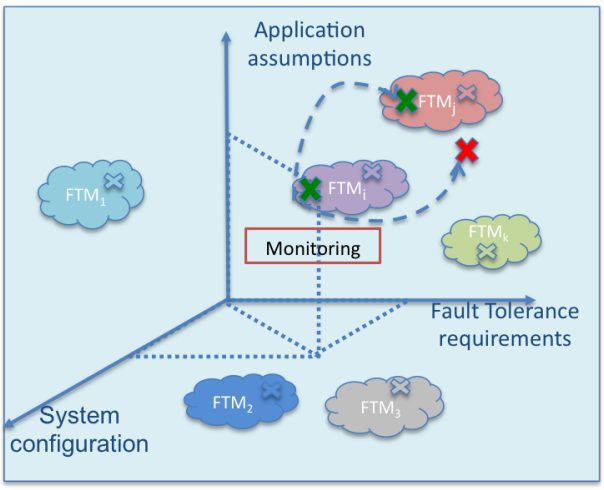

Fig. 1. Context-aware fault tolerance computing

This work has been supported by the french national agency (ANR) under contract ANR-BLAN-SIMI10-LS-100618-6-01.

∗ Corresponding author: miruna.stoicescu@laas.fr

We view the evolution of a system during its operational life as a trajectory in a space characterized by the parameters which can cause the aforementioned evolution. The three evolution scenarios can be aggregated into a frame of reference for this space, the three axes being: the application assumptions (e.g., whether it is deterministic or not), the fault tolerance requirements (e.g., the fault model) and the current system configuration (e.g., the resources it is using). We claim that a system's evolution can be represented in this vector space as shown in Figure 1. The combination of these three types of information indicates the most convenient fault tolerance mechanism to use, if any exists.

Given this frame of reference, we consider that each fault tolerance mechanism covers a certain region of space as shown in Figure 1 . Obviously, the union of all our fault tolerance mechanisms does not cover the entire space. A monitor keeps track of values in terms of the three axes and allows placing the state of the system for a given configuration in a certain region of this space. A change observed by the monitor triggers a transition towards a new region which might be covered by another fault tolerance mechanism or might be empty if there is currently no solution. In this case, we can still give some indications as to what must be adjusted in order to reach a region for which a fault tolerance mechanism exists.

Once our view of the problem stated, a certain number of issues arise, such as: how do we associate fault tolerance mechanisms with the combination of the three measures? how do we describe these fault tolerance mechanisms in a way which allows us to change them dynamically? how do we actually perform the transition from one mechanism to another? The following section presents our view on these matters.

II. OUR APPROACH

Separation of concerns is a generally accepted idea for introducing fault tolerance mechanisms in an application in a flexible way which allows subsequent modification and reuse. According to this principle, software architectures consist of two layers where the base provides the required functionalities and the top contains the fault tolerance mechanism(s). As we target the adaptation of fault tolerance mechanisms, we must manage the dynamics of the top layer, which can have two causes:

- The application layer remains unchanged but the fault tolerance mechanism must be modified either because the fault model changes or because an event in the environment, such as resource availability, makes it unsuitable.
- Changes in the top layer are indirectly triggered by modifications in the application layer which make the

fault tolerance mechanism unsuitable. In this case both layers execute a transition to a new state.

In order to achieve the adaptation of the fault tolerance mechanisms in both scenarios, we build on the representation from the previous section and refine it in three steps.

A. Description of the frame of reference

The three parameters labeling our axes are actually multidimensional vectors but for the sake of visibility we have chosen an elementary representation.

- The fault tolerance requirements are part of the non-functional specifications of an application. Our main focus is on the fault model, i.e., the types of faults which must be tolerated by the application. We base our fault model classification on known types [1], e.g., physical faults, design faults.
- The application assumptions regroup the characteristics of an application which have an impact on the choice of the fault tolerance mechanism but are not the same as the functional specifications of the application. These characteristics include: whether the application is stateless or stateful (i.e., if in case of failure we must rebuild its state for it to keep running), whether its state is accessible or not and whether the application is deterministic or not.
- The system configuration contains information such as the number of processors available, the available bandwidth, the memory resources.

B. Fault Tolerance Mechanisms

In order to place the mechanisms in the previously described frame of reference, we must first be able to classify them using the given criteria. In our approach, there are three steps in the selection process for finding the most adequate fault tolerance mechanism for an application: first, the fault model is identified, then the application assumptions and finally the current system configuration. The same process will be applied for classifying them.

C. System evolution

The state of a fault-tolerant application represents a point in the space given by our frame of reference. The application being fault-tolerant, this point must be in a region covered by a certain mechanism. A change (in terms of one of the three axes) is equivalent to a new state of the application, therefore a new point. As our purpose is to guarantee resilience when facing change, the application must always be "accompanied" by an adequate fault tolerance mechanism on its evolutionary trajectory. Therefore, we must either provide a fault tolerance mechanism encompassing a region which contains the new point or, should we fail to do so, "guide" the application towards a new region where a mechanism can be provided. We must design and build our fault tolerance mechanisms in view of such transitions and adaptations. The "distance" between two mechanisms as represented in our frame of reference should be equivalent to the difference between their implementations. If the new mechanism is close to the old one, we should be able to generate it through minor adjustments. The transition between distant mechanisms will most likely demand more complex modifications or even complete replacement.

The planned development process for achieving the smooth and safe transition between fault tolerance mechanisms consists of the following elements:

- a complete classification of the fault tolerance mechanisms we intend to use
- a method for describing and storing configurations corresponding to the fault tolerance mechanisms
- a set of tools allowing us to develop applications which are easily reconfigurable at runtime
- one or several algorithms for performing the transition between mechanisms

III. Conclusion and perspectives

Our work lies at the intersection of three research domains, namely Fault Tolerant computing, Autonomic Computing [2] and Ubiquitous Systems. The fault tolerance mechanisms we are considering are well-established solutions for certain use-cases/scenarios. This has led us to the concept of design patterns for fault tolerance, by drawing a parallel with the design patterns from software engineering. Each fault tolerance design pattern has an associated architecture. In order to operate a transition between fault tolerance mechanisms, we must be able to manipulate the configuration through its description file. A design principle that is commonly accepted in the area of reconfigurable systems is the use of component-based technologies (CBSE) for developing the management framework. We will use component-based middleware as a basic layer for our adaptive architecture. Finally, a decision needs to be made concerning the granularity of the reconfiguration. We can imagine two approaches for replacing a fault tolerance mechanism with another one:

- an atomic approach, where the old mechanism is completely replaced by the new one
- a differential approach, where we operate at a finer level by adding/removing the components corresponding to the difference between the two mechanisms

The transitions between mechanisms will most likely be described through state-machines. The algorithms behind the transitions could lead to design patterns for system evolution.

References

[1] A. Avizienis, J-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secur. Comput.*, 1:11–33, January 2004.

[2] J-O. Kephart and D-M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[3] J-C. Laprie. From dependability to resilience. In *International Conference on Dependable Systems and Networks (DSN 2008), Anchorage, AK, USA, pp. G8-G9*, volume 8, 2008.

Miruna Stoicescu is a 1st year PhD student at LAAS–CNRS. Her research interests include reconfigurable systems and ubiquitous computing. She received the Research Master's degree in Computer Science and Telecommunications, Multimedia stream, from Institut National Polytechnique de Toulouse, France in 2010, the Certified Engineer degree from Ecole Nationale Supérieure d'Electronique, d'Electrotechnique, d'Informatique, de Télécommunications et d'Hydraulique de Toulouse, France in 2010 and the Certified Engineer degree from the Polytechnic University of Bucharest, Romania in 2009.

On Developing Dependable Positioning

Christophe Pitrey
CEDRIC Laboratory
CNAM, France
Email: christophe.pitrey@cnam.fr

Francoise Sailhan
CEDRIC Laboratory
CNAM, France
Email: francoise.sailhan@cnam.fr

***Abstract*—Developing indoor positioning remains challenging, especially when we consider an emergency and disaster scenario wherein positioning infrastructures fail. In this case the objective consist in providing to first rescuers (firefighters) robust and reliable positioning under harsh conditions. Toward this goal, we present a navigation system, then, we survey the class of faults that threaten this system along with possible ways of handling it.**

I. POSITION AND NAVIGATION

The proposed positioning system makes use of active and contact-less Radio-Frequency IDentification (RFID): firefighters are equipped with RFID readers while RFID tags are disseminated in the building so as to serve as anchors. The pair-wise distance separating the reader to an anchor, is estimated based-on the Received Signal Strength (RSS)[1][2]. The resulting distance is used to multilaterate the position of the reader. Given that the signal may be perturbated (§1), successive RSS measurements are sanitized, i.e., filtered: infrequent values and outliers are removed so as to ensure that the largest quorum of consistent values is selected. Then, based on the measurements provided by m tags, the position (x_p, y_p) is approximated by multilateration. Multi-laterating consists in determining the intersection of m circles; each circle $\mathscr{C}i$ being centered on an anchor and characterized by a radius corresponding to the measured RSS:

$$(x_p - x_i)^2 + (y_p - y_i)^2 = RSS_i^2 \quad (1)$$

with (x_i, y_i) defining the anchor's coordinates. Multilateration can be seen as resolving a least-squared problem, i.e., finding the

$$\min_{(x_p,y_p)} \sum_{i=1}^{m} (x_p - x_i)^2 + (y_p - y_i)^2 = RSS_i^2 \quad (2)$$

Nevertheless, as illustred in Figure 1, the uncertainty resulting from RSS measurements implies that Equation 2 is not solved on a single point given by (x_p, y_p). Solving such a problem involves an increased complexity which is simplified using a two-steps approximated geometric process. First, rather than relying on the m tags that are in transmission range, the 3 nearest tags are selected. This choice is motivated by the fact that the RSS measurement accuracy decreases as the distance increases (logarithmic precision). Then, a central estimated point Pe is obtained by the intersection of two lines, each one passing through two points of intersection between two circles. Futhermore, 3 intersections points $(I1, I2, I3)$ nearest the point Pe are retained. Each point corresponding to a triangular vertices (Figure 2). Second, the central position P is determinated by the intersection of the medians of the triangle given by the 3 vertices. The resulting coordinates are further transmitted to one another so as to be displayed. For this purpose, a multi-players (multi-firefighters) game-enabled displaying platform (Game Maker [1]) is utilized. Overall, such a navigation system is subject to a variety of threats ranging from signal distortion (§2), trilateration (§3), transmission and distribution (§4), each of those requiring a specific handling.

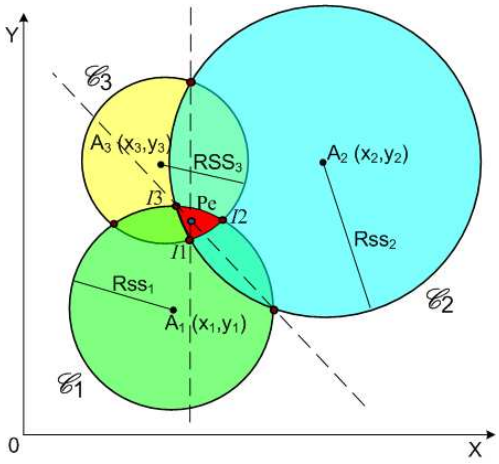

Fig. 1. *Trilateration.*

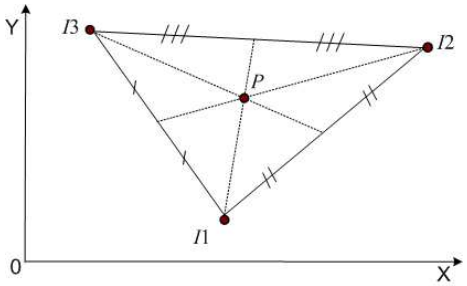

Fig. 2. *Central position of a triangle.*

II. RADIO WAVE

The signal (and hence the RSS measurements) are impacted by a variety of distortions. More particularly, an antenna does not radiate uniformly and the propagation of the signal differs depending on the angles of emission and arrival of the signal. In addition, this signal is subject to any interference created by any electrical equipment operating on, or around, the spectrum (recall that the operational RFID spectrum is unlicensed and hence not protected). Moreover, radio waves suffer from propagation, reflection and absorption together

[1] http://www.yoyogames.com/gamemaker

mainly due to the presence of obstacles and reverberating surfaces in the building frame e.g., reinforced concrete, metal... Note that human are constituted of 80% of water and hence constitutes a strong reflector to radio signal. These phenomena impact the strength and quality of the received signal which in turn affects the RSS measurement of one another positioning system. Additional disaster-specific parameters including, variable atmospheric conditions (e.g., thermal, brightness) and presence of hydrometeors (e.g., vapor, rain drop) affect the signal characteristic. Overall, the aforementioned faults can be classified as static or time-varying. The former mainly results from (i) the antenna irregularities and (ii) the physical arrangement of the obstacles. To tackle this issue, external and omni-directional antenna should be preferred and fingerprinting should be applied. This consists in recording the RSS at each location. Then, RSS is measured so that the best-matching RSS that were previously recorded, is identified and its related location deducted. During a disaster, time-varying faults are mainly unpredictable and depend on the operational environment. This advocates for providing sensor-enabled, context-aware and self-healing positioning.

III. TRILATERATION

A multi-lateration-enabled 2D-positioning of a firefighter requires to place anchors in a way which ensures the appropriate coverage of the geographical area and the significant redundancy. In practice, at least m non-collinear anchors should be in transmission range with a firefighter. Such a staggered placement necessitates careful network planning, i.e., covering, identifying, positioning anchors and recording the related information. Here, a manual acquisition may lead to a multi-lateration failure unless pre-checking (e.g., research of duplicate records) and control mechanism (consisting in e.g., verifying that the recorded tags' characteristics are actually consistent with the observed positions) are performed. Despite these mechanisms, full coverage cannot be assumed during a disaster: anchors are expected to fail as the sinister evolves. In addition, power-supply failures are observed (battery-powered tags and readers may still fail in spite of a low-energy consuming and an automatic alert triggering in case of low-remaining energy).

IV. TRANSMISSION & DISTRIBUTION

Positioning information is intended to be shared with any stakeholder. However such information may be either corrupted or lost at the source that generates it, during its transit, or at the receiver (§2). So, the use of a shared frequency band implies that other emitters interfere. This necessitates either providing pseudo-random spectral frequency hopping or listening before talking process. It may also happen transmission errors, packet de-sequencing, disconnections lead to message/packet loss and disconnections resulting from a node failure [3]. To resist to these former, one should use reliable protocol, e.g., Transmission Control Protocol (TCP) and to verify data integrity, and re-establish connections. The system should be tolerant of faults and it must correct automatically these faults. It should be always available and operational due to program or memory corruption and then prevent the spread of this fault to the other elements. It will also be attentive that each user uses the same recorded tags and play the same Game Maker versions to avoid an erroneous or different display. All these fault-tolerant mechanisms should hence be provided as part of the positioning system.

V. CONCLUSION

We presented an indoor RFID-based positioning system, easy to use, involving a low financial cost. This system displays the position of a firefighter based on trilateration calculations (RSS measurements of anchors in range). We have implemented a prototype and during our testing some approximations on positioning were revealed. In spite of all efforts realized, guarantee the survivability of such a positioning system still remains particularly challenging. Indeed our system is subject to various faults: The hardware design, the radio interference, the building frame and atmospheric evolution during a disaster will cause a variety of threats, which influence trilateration and disturb the exact display position of a firefighter. The transmission of information between the participants should be reliable, resistant and tolerant of faults to prevent disconnection and to guarantee the system availability. We must therefore create a policy test of our system that verifies the coverage and placement of each anchor. These tests check also human error input and the duplicated records. So, in order to maintain the confidence of firefighters to the positioning system, it's of prime necessity to display the measurement uncertainty, degree of accuracy/precision (precision circle drawn around the firefighter) and degree of confidence. It is vital to know the robustness of our system because here the lives of firefighters are at risk. Hence, we plan to study the dependability of our indoor positioning system, by developing an injection fault solution. Through this tool we will be able to realize the scenarios for modeling errors or interferences and to check their impact on the functioning of the system. The advantage of this system is that it is replicable as many times as you want each scenario to test the robustness of our localization solution.

ACKNOWLEDGMENT

The authors would like to thank Stephane Rovedakis for its helpful discussion and review. This work has been supported by the french national agency (ANR) under contract ANR-BLAN-SIMI10-LS-100618-6-01.

REFERENCES

[1] Pitrey C., *SURPAT: SÛReté du PATrimoine*, mémoire d'ingénieur, CNAM, 2010.

[2] Sailhan F., Astic I., Michel F., Pitrey C., Uy M., Gressier-Soudan E., Gerbaud P., Forgeot H.. *Conception d'une solution de localisation et de surveillance, à base de RFIDs actifs, défits et perspectives*, INFORSID GEDSIP Workshop, 2009.

[3] Avizienis A., Laprie J-C., Randell B., Landweh C. E., *Basic Concepts and Taxonomy of Dependable and Secure Computing*, IEEE Trans. Dependable Sec. Comput., 2004.

Christophe PITREY received his engineer degrees from the *Conservatoire National des Arts et Métiers* (CNAM, Paris, France) in 2010. He is currently a PhD student at CEDRIC Laboratory of CNAM. His PhD topic is focused on faults injection in Wireless Sensor Networks. Christophe has worked about 10 years at GEMALTO in particular on the industrialization of electronic payment terminals (production monitoring, testability, security software and hardware of TPE). Prior to this appointment, he was a technician at STENTOFON COMMUNICATION responsible for installation, repair and advice on radio paging systems.

Index of Authors