Efficient Surveillance of Information Systems Online

Michel Dagenais
michel.dagenais@polymtl.ca
Dept. of Computer and Software Engineering, Ecole Polytechnique
P.O. Box 6079, Station Centre-Ville, Montreal, QC, Canada, H3C 3A7

Abdelwahab Hamou-Lhadj
abdelw@ece.concordia.ca
Dept. of Electrical and Computer Engineering, Concordia University
1455 de Maisonneuve West, Montreal, QC, Canada H3G 1M8

Mario Couture
Mario.Couture@drdc-rddc.gc.ca
Software Analysis and Robustness Group, Defence Research and Development Canada
Valcartier, 2459 Boul. Pie XI Nord, Québec, QC, Canada G3J 1X5

Dominique Toupin
dominique.toupin@ericsson.com
Process, Methods and Tools Group, Ericsson
8500 Decarie, Mont-Royal, Qc, Canada, H4P 2N2

Abstract

Cyber attacks are rapidly becoming a major threat to proper secure government and military operations. From deployed wireless systems subjected to jamming, denial of service attacks and intrusion attempts, to in-house computers connected to secure networks infiltrated by malicious software, numerous serious computer incidents are increasingly encountered. Low level system tracing, traditionally used for debugging, may be used for host based surveillance. We have developed a framework for detecting abnormal behavior, reacting to the threat, and monitoring the effectiveness of the response before escalation. The main advantage of the new system is the combination of low level tracing information with powerful abstraction and anomaly detection techniques. Our tracing mechanisms extract very detailed execution traces with minimal overhead, increasing the detection capability without affecting operation or alerting the attackers.

Introduction

In the recent past, several governmental organizations and high-tech industrial companies have reported serious computer security incidents, including military units and contractors. The problem is getting more acute as online information systems are deployed and used pervasively, and relied upon for critical tasks. Furthermore, all these systems are getting widely networked and often connected to the Internet. Even in a non Internet connected network, there is the risk of an inside attacker or a compromised host.

The typical recommended measures to secure a system are to use a firewall, install software updates regularly, minimize the number of ports and services made available to the Internet, and use anti-virus software as well as host and network based intrusion detection systems. These measures are relatively effective for the average computer, preventing well known attacks from succeeding.

Even though considerable progress was made in the past 10 years to build secure software [1], new vulnerabilities are still constantly being discovered and exploited. There is a lucrative black market for

such new *zero day* exploits, since they can succeed in penetrating computer systems protected by most intrusion detection systems. These zero day exploits are often reserved for especially attractive and well protected systems, for example in the army and at defence contractors.

Network surveillance has also progressed with newer systems capable of detecting problems beyond simply well known attacks. Nevertheless, several types of attacks create very little network traffic, infiltrating nodes quietly. It is therefore necessary to monitor the critical computer systems from the inside, with host based intrusion detection, in addition to network based intrusion detection.

Traditional System Tracing Offers Reusable Probing

Several different tools exist to record system execution details. Logging tools (e.g. syslog on Linux) are in common use and write to log files relatively rare but important events (e.g. user login, web page access...). More sophisticated tracing tools have been used for performance analysis and debugging. The SGI Performance Co-Pilot and the Intel Parallel Tools and Trace Analyzer are good examples of tracing frameworks used for parallel scientific programming [2].

Many operating systems come with a low level tracing infrastructure, for example Solaris with dtrace. The Linux kernel contains a few hundred static probes that cover system calls, interrupts, file accesses, scheduling, network activity and more. These static probes have been inserted by Linux kernel developers at important locations in the execution paths of the kernel. It is also possible to insert dynamic probes with kprobes, when one decides at execution time that tracing data is desired from a location where no static probe is present.

Several tools are available to trace these static and dynamic probes, including ftrace, perf, SystemTap [3] and LTTng [4]. LTTng is particularly interesting since it was optimized for low overhead tracing and scalability. It comes with a user-space tracing counterpart, the UST library. When tracing is used on a complex online system in production, it is important that the perturbation be minimal, such that the production system keeps functioning and that any problem remain visible while tracing. The cost of having non activated static probes is extremely small, a few tens of bytes of program memory and a delay so small that it can barely be measured. The cost of an active static probe with LTTng is about 200ns each time it is hit. This represents about 1% overhead on the execution time for a detailed execution trace with all the system calls and interrupts recorded. The cost of a dynamic probe only exists when it is inserted and amounts to approximately 2000ns each time it is hit.

Evolving from Debugging to Continuous Execution Surveillance

Interestingly, low overhead tracing tools, connected to static probes available at all important execution points throughout the kernel, are very useful for host based surveillance. The low overhead is important to minimize the performance impact of the surveillance, reducing the interference with the host services and making the surveillance less detectable. The detailed execution traces generated on several critical nodes then need to be retrieved by a trace analysis host. A simple network with a few critical nodes being traced is illustrated in Figure 1.

The tracing host is responsible for observing the behavior of the monitored hosts through the traces, analyze the traces to detect problems, decide on corrective measures, activate these measures and monitor their effect. This is implemented as a sequence of steps illustrated in Figure 2. The normal mode of operation is fully automated, since the required reaction time to block an infiltration may be much less than one second. However, an operator can monitor the process and manually activate or override corrective measures.

The first step is to synchronize the traces, i.e. computing the clock offset of each computer and

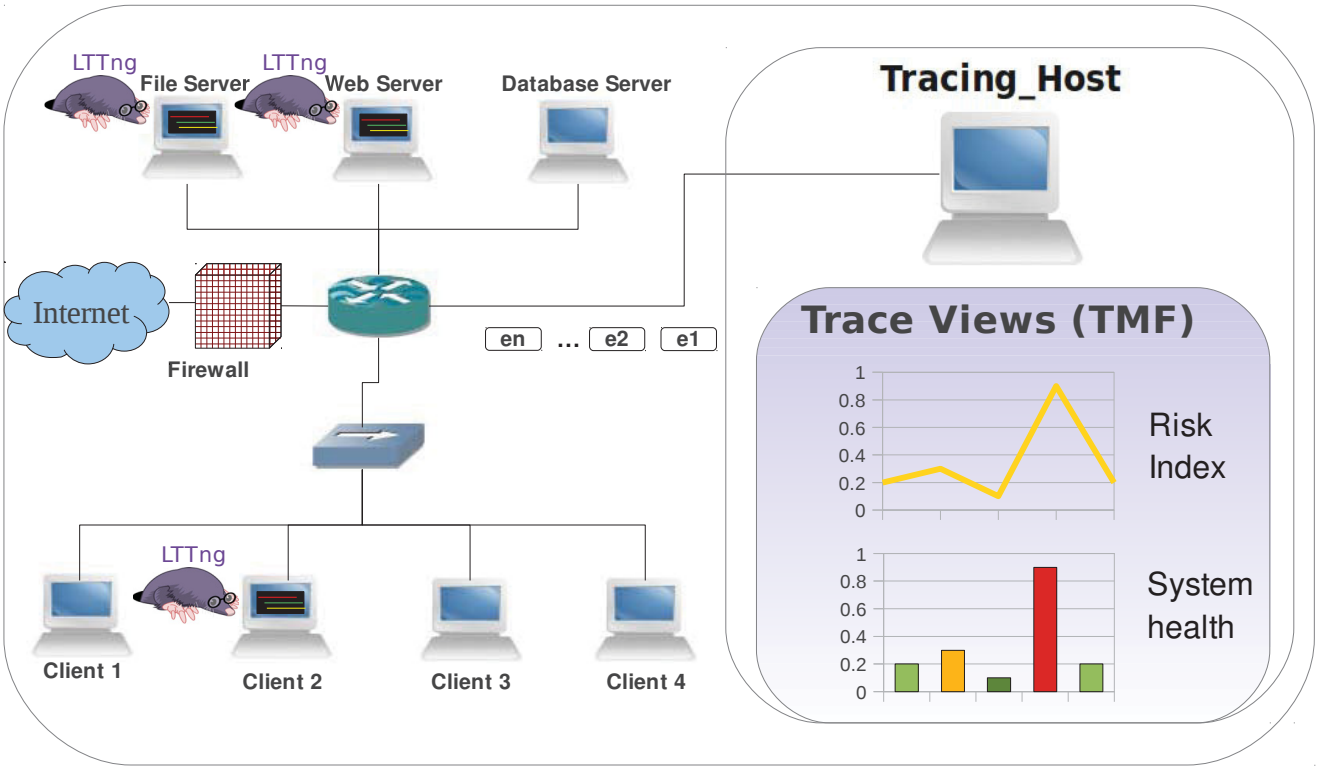

Figure 1: Example of a network with tracing enabled for surveillance.

obtaining a common reference time. Indeed, each computer has an independent clock which may have an offset of tens of milliseconds with respect to other computers. Traces synchronization is therefore extremely useful in order to analyze sequence of related events occurring on different computers [5].

In a second step, trace abstraction takes place, where events are categorized (e.g. *normal* service interrupt to update the system clock every millisecond) or grouped into higher level events (represent by a *sequential file reading* event the several *system calls* and *interrupts* events needed to open a file, read its content buffer by buffer, sending requests to the disk controller...). This abstraction step is useful in reducing the volume of events needed for the higher level analysis. The more detailed information remains available in case a problem is detected and a finer analysis is required.

The third step is automated fault identification. There, a number of patterns of abnormal behavior and known attacks are checked against the tracing data, using as much as possible the higher level events. An efficient Finite State Machine pattern matching procedure has been developed to simultaneously verify the tracing data against a large number of patterns. In parallel with the automated fault identification, instead of looking for patterns of anomalies, the trace correlation module compares the tracing data with normal traces, interpreting deviations as potentially abnormal behavior. The focus on abnormal behavior detection makes this system particularly effective against new, *zero day exploit*, cyber-attacks.

The decision module interprets the alerts created because of faults or abnormal behavior, and contains several components. The Alert Optimization component looks at combinations of alerts in order to refine the priority associated with each. For instance, one dropped network connection or one failed password attempt is a reasonably frequent and normal event (low priority alert) while numerous such events in close succession are a strong indication of serious problems (denial of service and brute force attacks) warranting a high priority.

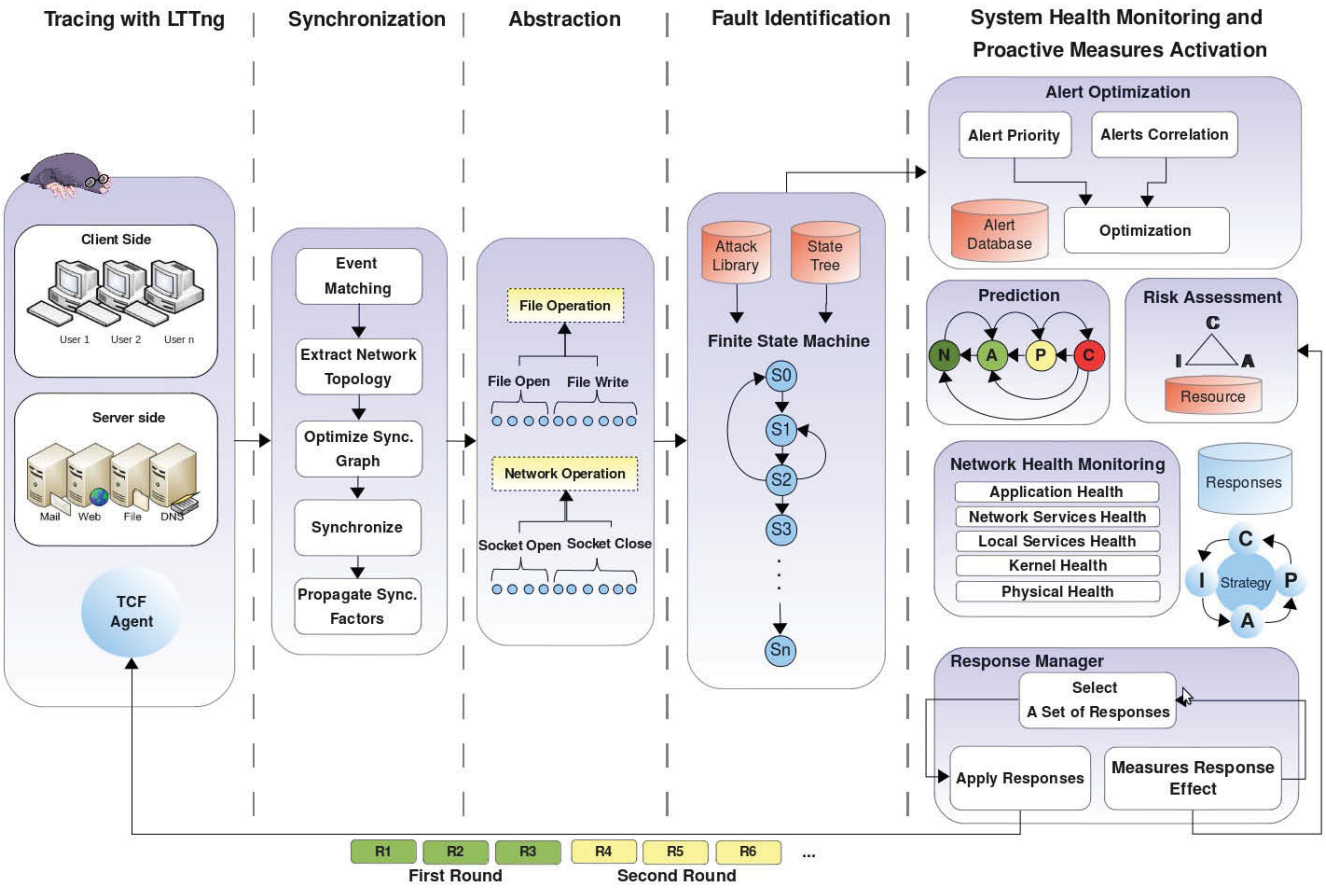

Figure 2: Trace Analysis Architecture.

The second component, Prediction, looks at the different alerts, either fully or partially matched patterns in the automated fault identification module. It attempts to predict the next likely step in a possibly ongoing attack, using Hidden Markov Models. The third component, Risk Assessment, looks at each predicted fault with the associated probability of occurrence and target damage consequence.

The fourth and last component, Prevention, selects a reaction strategy based on the risk assessment (probability and damages to the targets), the importance of each asset to the overall strategy (maintaining a critical service, preserving the confidentiality of the information on a computer), and the consequences and cost of the reaction measures associated with a strategy. Indeed, one computer may offer a critical service to soldiers in the field but not contain very sensitive information, in which case a breach of confidentiality is preferable to shutting down the service. Another computer, used by a scientist, may contain highly sensitive information and should be cut immediately from the network if an attack is under way that may result in a breach of confidentiality.

Once the prevention strategy is selected, the associated reactive measures are executed on the target computers. As these actions take place, the systems continue to be monitored and the risk assessment is reevaluated in order to decide to escalate with more reactive measures, to maintain the current measures or to start retracting some of these reactive measures.

In a typical setup for system surveillance, all the critical systems are traced with a moderately detailed kernel trace. Such a trace contains all the system calls generated by the applications as well as all the network communications performed. Applications running on the traced systems (including malware, or commands executed as a result of a successful attack and exploit of a vulnerability) are not aware of the tracing taking place at the kernel level. The tracing host, receiving and analyzing the traces of the

target systems, will typically not run any other service, it will abstract the trace data into higher level events, including alerts. This information will then be sent to a higher level node responsible for deciding if a problem is present and taking reactive measures if warranted.

Poly-Tracing a Successful Architecture

The Poly-Tracing project was funded by Defence Research and Development Canada (DRDC), NSERC and Ericsson Research and involves 5 faculty members, 2 research associates, 12 graduate students in 4 universities as well as 10 research and development engineers at DRDC and Ericsson. The objectives of the project for DRDC were to develop an open source Tracing and Monitoring Framework suitable to efficiently detect and react to cyber attacks. Ericsson placed more emphasis on Tracing and Monitoring, for debugging and monitoring the performance and functionality of distributed multi-core systems. In both cases, this framework is an infrastructure they need, not something to be marketed and sold, and therefore an Open Source model was selected. The Poly-Tracing tools thus form a free, open and reusable demonstration vehicle for this technology [6]. The architecture of the framework is illustrated in Figure 3.

Because of the open source nature of the project, several other organizations with similar infrastructure needs did collaborate and contributed expertise, source code, free patent licenses and internships to students. In a few cases, organizations even paid some of the developers for specific improvements and ports to other platforms. These contributions span more than 90 developers and 20 different organizations.

LTTng and UST were used and further developed within Poly-tracing as a low overhead tracing infrastructure. The Eclipse environment was used as programming platform for the Tracing and Monitoring Framework (TMF) developed as part of the project [7]. On the tracing host, in the TMF user interface, the operator can list the target computers which can be monitored. For each target computer, a list of programs, applications or kernel, is provided. For each program, the list of available static probes is provided. The operator may then select which probes to activate. Of course, in the typical case, a predefined default set of probes is activated.

TMF uses the Target Communication Framework (TCF) to interact with the LTTng and UST tracing tools on the target systems [8]. The interaction includes starting a trace session, activating the tracing probes and retrieving the tracing data as it is generated. If a problematic condition is encountered, for example suspicious activity on a target computer, it is possible to activate more tracing probes at any time. Moreover, if needed, a dynamic probe may be inserted at a new location in an already running program. A dynamic probe is specified in the source code using the debugger interface. This is similar to video surveillance systems where numerous fixed cameras are available and may be activated or not, with the possibility to temporarily add mobile cameras in new locations. The Target Communication Framework is also used to issue reactive measures when a problem is detected on a host.

Results and Discussion

The Poly-Tracing project is still ongoing but it has already proven very successful, with a follow-on project in preparation. LTTng has reached a nice level of maturity while still being extended and improved upon. It provides a low overhead tracing infrastructure that is flexible, probes are easily inserted and can be activated at any time. It has minimal impact on the traced system, yet produces all the information necessary to follow in great details the system execution and detect problems. Similarly, the Tracing and Monitoring Framework and the traces synchronization algorithms are reaching maturity. The detection and decision modules represent a considerable challenge and are being iteratively and incrementally refined. The interested readers are encouraged to evaluate, compare,

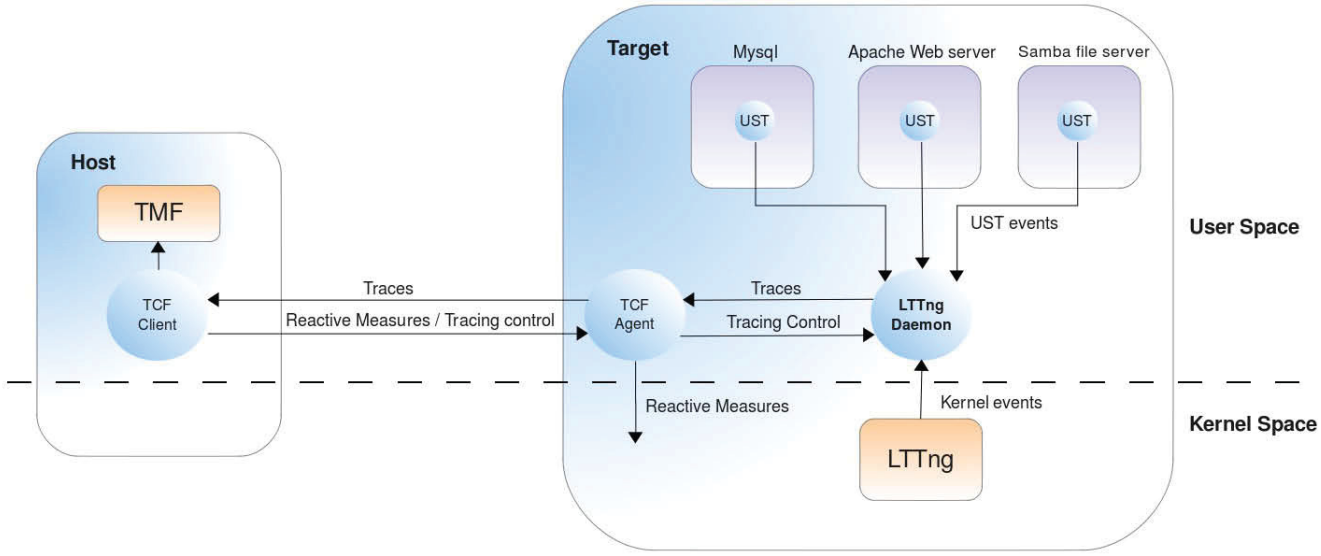

Figure 3: LTTng, TCF, TMF tools architecture.

comment on and contribute to this infrastructure. Preconfigured virtual machines are available to quickly and easily try these tools [9].

In retrospect, several aspects of the project organization have greatly contributed to this success. Firstly, building all the software infrastructure as open source significantly simplified the interactions between the project partners and outside collaborators (Linux kernel developers, Eclipse foundation, other companies with interest in tracing), obviating the need for complex agreements and maximizing the possibilities of outside contributions and synergy.

The collaboration of numerous organizations with different goals has been instrumental in obtaining a much better final product. For example, IBM was particularly interested in multi-core scalability, which resulted in the jointly developed new user-space RCU library, a prime reason for the excellent performance of the UST library. While Ericsson was already focused on multi-core scalability, other organizations had not yet felt this need but will benefit from this capability in the coming years. Another interesting example is Google which insisted on the compactness of traces, leading to several optimizations in this area, again benefiting all users.

A final important aspect of the project is the modularity of the approach. For instance, the kernel and user-space tracing modules produce the exact same trace format with self-contained metadata. Moreover, this format is being used as inspiration for the Multi-Core Association work on a Common Trace Format [10]. The work on the different tracers and the analysis tools can thus proceed in a more modular and independent way, and the infrastructure can easily interact and interoperate with other intrusion detection systems. In a similar manner, the Eclipse framework was selected as an implementation environment for the Tracing and Monitoring Framework (TMF). TMF is structured in such a way that several analysis modules can operate in parallel on a trace, sharing the events reading and dispatching module, the graphical user interface synchronized views (e.g. shared notion of current event, current time, current time interval), and the modeled state representing the traced systems (current table of processes, opened network connections...).

References

[1] R. Charpentier, M. Debbabi, A. Mourad, and M-A Laverdière. “Security Hardening of Open Source Software”, in the Open Source Business Resource Journal, June 2008, Open Journal Systems

Publishers.

[2] R. Asbury and M. Wrinn, “MPI tuning with Intel Trace Analyzer and Intel Trace Collector”, 2004 IEEE International Conference on Cluster Computing, September 2004, San Diego, California.

[3] V. Prasad, W. Cohen, F. Eigler, M. Hunt, J. Keniston and B. Chen, “Locating System Problems Using Dynamic Instrumentation”, Proceedings of the Linux Symposium, Ottawa, Canada, 2005.

[4] P.-M. Fournier, M. Desnoyers, M.R. Dagenais, “Combined Tracing of the Kernel and Applications With LTTng”, Proceedings of the Linux Symposium, Montreal, Canada, 2009.

[5] Benjamin Poirier, R. Roy, and M. R. Dagenais, “Accurate offline synchronization of distributed traces using kernel-level events,” SIGOPS Operating Systems Review, vol. 44, no. 3, pp. 75–87, 2010.

[6] Poly-Tracing Team, “Distributed Multi-Core Tracing Research Project”, http://dmct.dorsal.polymtl.ca/, viewed March 31, 2011.

[7] F. Chouinard et al, “Eclipse Tracing and Monitoring Framework”, http://www.eclipse.org/linuxtools/projectPages/lttng/, viewed March 31, 2011.

[8] F. Burton et al, “Eclipse Target Communication Framework”, http://www.eclipse.org/projects/project_summary.php?projectid=tools.cdt.tcf, viewed March 31, 2011.

[9] Poly-tracing Team, “Test Virtual Machines”, http://lttng.org/content/test-virtual-machines, viewed March 31, 2011.

[10] M. Desnoyers et al, “Common Trace Format”, http://www.efficios.com/ctf, viewed March 31, 2011.

Michel Dagenais is professor at Ecole Polytechnique in the Computer and Software Engineering Department. He is specialized in tracing and monitoring tools, performance analysis and distributed systems. Over the years he visited and collaborated with researchers at many of the largest industrial and governmental research centers. He has organized several international Tracing Mini-Summits and is a member of the Multi-Core Association Tracing Infrastructure Working Group, working on a Common Trace Format standard.

Abdelwahab Hamou-Lhadj is professor at Concordia University in the department of Electrical and Computer Engineering. He specializes in trace abstraction for the behavioral analysis of software systems, for both program understanding and security purposes.

Mario Couture received B.Sc. and M.Sc. degrees in Physics from Université du Québec à Rimouski, and M.Sc. degree in Electrical Engineering from Laval University. He joined Defence Research and Development Canada (DRDC Valcartier) in 2002 as a Defence Scientist in the Software Analysis and Robustness Group. His research interests are mainly oriented toward the study and design of leading edge mechanisms for refined on-line surveillance/protection of military information systems.

Dominique Toupin completed a M.Sc. in Software Engineering at Ecole Polytechnique de Montreal. He is in the Tools and Methods group at Ericsson and is directly involved in the quest for better tools to help face the new challenges brought by the design and operation of multi-core real-time online systems.