Configurable Middleware-Level Intrusion Detection for Embedded Systems*

Eivind Næss[1], Deborah A. Frincke[2], A. David McKinnon[3], David E. Bakken[1]
[1]*Washington State University, Pullman, Washington USA*
{enaess,bakken}@eecs.wsu.edu
[2]*Pacific Northwest National Laboratory, Richland, Washington USA*
deborah.frincke@pnl.gov
[3]*Atomic ORBS, Richland, Washington USA*
mckinnon@acm.org

Abstract

Embedded systems have become integral parts of a diverse range of systems. Unfortunately, research on embedded system security, in general, and intrusion detection, in particular, has not kept pace. Embedded systems are, by nature, application specific and therefore frameworks for developing application-specific intrusion detection systems for distributed embedded systems must be researched, designed, and implemented. In this paper, we present a configurable middleware-based intrusion detection framework. First, we present a system model and a concrete implementation of a highly configurable intrusion detection framework that is integrated into MicroQoSCORBA, a highly configurable middleware framework developed for embedded systems. Our framework autogenerates application-specific intrusion detection systems by exploiting application-specific information accessible to the middleware framework (e.g., object interfaces and method signatures). Next, a set of configurable intrusion detection mechanisms suitable for embedded systems is presented. A performance evaluation of these mechanisms, is presented at the end of the paper.

1. Introduction

Embedded systems have become ubiquitous. New devices and systems have emerged, such as cellular phones, PDAs, and wireless networks. Older technologies also reap the benefits of embedded processing, for example a typical automobile now includes two-dozen microprocessors [31]. Over 98% of all microprocessors are now deployed in embedded systems [29]. Unfortunately, security research targeting resource-constrained distributed embedded systems has not kept pace.

An Intrusion Detection System (IDS) detects intrusions, normally defined as compromises of a system's confidentiality, integrity, or availability properties. An IDS may also respond to such events. The very nature of distributed embedded systems (e.g., low physical security, limited resources, limited computational power) makes them inherently more vulnerable to intrusions [7], [20], [21]. Numerous intrusions into Supervisory Control and Data Acquisition (SCADA) systems have been documented [4], [7], [10]. This vulnerability is a significant security threat and cyber-terrorism concern because SCADA systems often control critical infrastructures (e.g., gas pipelines, electrical power grids).

We propose integration of IDS in embedded systems via middleware, based on two observations. First, a middleware framework has access to all of the communication streams between the distributed objects within the distributed system. Therefore, coding and performance efficiencies can be achieved because the IDS-aware middleware can directly send message signatures and parameter values to the IDS. Otherwise, the IDS has the additional burden of unmarshalling (decoding) the raw network packet before it has access to this message information—a burden that many resource-constrained embedded systems can not bear. Second, a middleware framework has access to the structure of the application logic of the distributed system, as manifested in the interface definitions for each of the distributed objects. Thus, an application-specific IDS can be generated with a minimal amount of effort by the middleware's IDS-aware code generation tools. For example, an application developer could specify that, for

* This research was supported in part by two Cisco University Research Program donations and Grant NSF-CISE EHS-0209211 from the National Science Foundation's Embedded and Hybrid Systems program.

performance reasons, only a subset of the application's messages and objects be instrumented with IDS sensors. Alternately, the developer could direct the middleware layer to choose which IDS sensors and policies to use based upon message parameter types. Switching from one IDS configuration to another requires only a reconfiguration of the middleware layer, rather than a costly rewrite of application layer code.

To date, only limited research on intrusion detection tailored to the needs of resource-constrained embedded systems has been performed. Likewise, little research into supporting intrusion detection at the middleware level has been performed. To effectively meet the security and resource constraints of distributed embedded systems, however, a better understanding of the integration of intrusion detection and middleware frameworks is needed. In this paper we report our key contributions:

- A model for middleware-based, application-level intrusion detection targeting small resource constrained networked embedded devices
- A highly configurable set of mechanisms and responses for middleware-based intrusion detection, which are suitable for resource constrained embedded devices
- A set of reusable application-based security policies based on the primitives provided by this intrusion detection framework
- An experimental evaluation of our middleware-based intrusion detection framework.

The remainder of this paper is organized as follows: first, related work is presented; then a model for embedded middleware-level intrusion detection is presented; followed by descriptions of a framework that implements this model; next, its performance evaluation is presented; and then the paper ends with a few concluding remarks.

2. Related work

The Embedded Sensor Project (ESP) at Purdue University developed an IDS framework using internal sensors and embedded detectors [27], [32], and proved their feasibility in both network- and host-based environments [32]. We implemented additional primitives and have also integrated MIDES, our intrusion detection framework, into a middleware framework.

CylantSecure [6] is a behavioral-based intrusion detection and prevention system that integrates internal sensors into the kernel of an operating system, after the initial design of the operating system. MIDES, in contrast, integrates its procedural based sensors into a distributed application during its initial (middleware) design and development cycle.

Internal sensors have proven useful for detecting anomalous deviations in system call usage by an application, for instance by Forrest et al. [9]. One implementation, process homeostasis (pH) [26], has been embedded into the kernel of an operating system and is capable of automated responses such as time-delay embedding and aborting system calls. Unlike pH, MIDES can observe the execution pattern of an application directly because it is integrated into the application's middleware layer, and therefore has access all of the application's high-level method invocations.

There are few application-based IDSs to date. Application intrusion detection is described by Sielken [24], but only on a theoretical level. Almgren and Lindqvist describe an application-based approach for data collection and intrusion detection integrated into the Apache web-server [1]. Their approach targets a particular type of application, which is intended for the high-end computing market. In contrast, MIDES supports a wide range of applications via its integration into a middleware framework.

Several commercial vendors have implemented network-based intrusion detection systems (e.g., Cisco [5] and Arbor Networks [22]). Unfortunately, these network-based systems can not easily access application-specific data whereas MIDES's middleware-based approach can.

Applications that Participate in their Own Defense (APOD), built on the Quality Objects (QuO) framework [2], integrates external tools such as Snort [25] and IPTables [11] to provide intrusion detection functionality. This approach is interesting, but is not feasible for small, resource-constrained embedded systems because of the external dependencies.

Previous research on middleware-based intrusion detection systems has been conducted by Stillerman, et al. [28] and Marrakchi, et al. [14]. Both devised anomaly-based intrusion detection systems that measure a client's deviation from a previously established baseline. EMIDS is a richer framework that supports multiple application-specific policies. Furthermore, MIDES integrates its intrusion detection mechanisms into the CORBA ORB, stub, and skeleton code, whereas these other frameworks only use CORBA interceptors to extract data from the ORB. Thus, MIDES has a significant advantage in creating application-specific IDS.

Other CORBA middleware frameworks exist that address the problem of intrusion detection by using Byzantine fault tolerance mechanisms [12], [13], [23]. Intrusion detection and fault tolerance is a useful combination; however, these intrusion tolerant systems are too resource intensive for deployment on the resource-constrained embedded systems targeted by MIDES.

3. A Model for Embedded Middleware-Level Intrusion Detection

A model for an embedded middleware-level intrusion detection system (EMIDS) is presented in the following subsections. We begin with the system model for EMIDS, and then data collection, application-level policies, and IDS responses are discussed. An in-depth discussion of EMIDS is in [19].

3.1. System Model

EMIDS is a model for an application-based intrusion detection system, based on anomaly and misuse detection, which is integrated into a middleware layer. This model allows for reasoning about how middleware-level information such as distributed object APIs and the communication streams between distributed objects may be used within an IDS. Furthermore, the development of IDS-aware middleware tools (e.g., code generators, IDL compilers) allows for the efficient embedding of IDS sensors and detectors into an application's middleware layer. This is a significant advantage that EMIDS has over traditional IDSs, which are often limited to gathering data from low-level system and network calls.

Data collection within the EMIDS model is performed by sensors embedded into the application or middleware framework, as illustrated in Figure 1. The data is processed by an IDS-kernel, which:

1. Provide a high level of configurability, which can be extended to support dynamic reconfiguration of the security policies or responses.
2. Support multiple types of sensors and responses through a generalized interface.

EMIDS sensors are integrated into the application or middleware layer, as illustrated in Figure 1. The different sensor types are discussed in depth in the next section. Based on the outcome of the data analysis or IDS-kernel configuration, the appropriate response will be triggered. Responses can be used to target the middleware connections or the application itself.

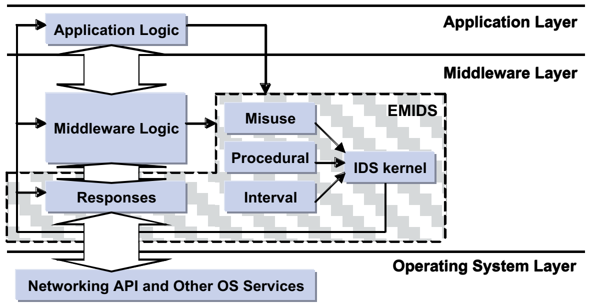

Figure 1. Architectural diagram of EMIDS model

3.2. Sensors and Detectors

Sensors and detectors provide essential data input into an IDS. EMIDS supports interval-, procedural-, and misuse- based intrusion detection sensors and detectors.

Interval-based sensors are external IDS sensors that monitor and detect either parameter values or method invocation frequencies that lie outside of an interval of acceptable values. This sensor type is well suited for embedded systems that operate in a predictable or steady state manner (e.g., temperature controllers, fixed interval control loops).

Procedural-based sensors, as used in the EMIDS model, are internal IDS sensors that collect data based on the execution pattern of the component that is being monitored. IDS-enabled middleware tools can automatically embed these sensors into either the entry or exit points of an application's functions.

Misuse-based detectors are embedded at locations in an application's source code that contain known vulnerabilities; or that can naturally detect misuse [27]. Embedded detectors are powerful, but lack the generality of the interval- or procedural-based sensors, which impacts the ease with which they may be automatically embedded by IDS-enabled middleware tools.

3.3. Application-Based Security Policies

The EMIDS model is an application-based IDS that provides high level application-based security policies. An application can be modeled by its execution patterns and have thresholds set which determine when anomalous behavior is observed [3]. These models are built over a period of time, with help from automated instrumentation embedded within the middleware layer. A model is then stored into a profile that describes the acceptable behavior of the application. Some policies may be described by a single, predetermined threshold value and therefore are not dependant upon a stored profile.

Interval-based sensors are used to detect variations in parameter values or method invocation frequencies. These values can then be compared against known, acceptable values (e.g., a minimum or maximum acceptable temperature value). Some example interval-based policies are minimum, maximum, or average value comparisons.

Procedural-based sensor policies require the use of a stored profile of acceptable behavior against which the currently observed behavior is compared with. The result of this analysis yields a value that is compared against a threshold value that defines acceptable behavior.

3.4. Responses

Responses to intrusion attempts can be embedded into either the middleware or application layer when using the EMIDS model. The middleware can be configured to incorporate responses that log events, delay invocations between the host and an adversary, ban the IP-address or terminate the connection of an intruder. Response policies should in general not alter the execution path of the application because this may lead to an unpredictable outcome [32].

EMIDS integrates an IDS with a middleware framework which allows the IDS responses to be narrowly targeted to specific clients that exhibit anomalous behavior. In contrast, a traditional host-based IDS can not distinguish between all of the clients executing on a remote host.

Generating false positive intrusion events is a concern of all IDS, including EMIDS. A false positive may occur due to improper or insufficient training of a procedural-based sensor; or setting inaccurate threshold values for the interval-based sensors. Some IDS response policies will have a stronger negative impact on non-anomalous behavior than other responses.

4. A Configurable Framework Providing Middleware-Level Intrusion Detection

We designed and implemented a highly configurable middleware-based IDS framework based on the previously presented EMIDS model, named *Middleware-based Intrusion Detection for Embedded System (MIDES)*. MIDES was implemented within *MicroQoSCORBA*, a middleware framework designed for resource-constrained embedded systems [15], [16], [17]. The following subsections present an overview of MIDES; its application-based policies; and its responses.

4.1. Overview of MIDES

MicroQoSCORBA is a CORBA-based middleware framework that possesses a high degree of fine-grained configurability. Configurability is achieved using the MicroQoSCORBA's Interface Definition Language (IDL) compiler and code generation tools, which generate CORBA stub and skeleton code and CORBA libraries. These are tuned to an application's functional and non-functional constraints and the hardware that it is executing on [15], [17]. For example, MicroQoSCORBA can remove support for unused parameter or data types [15]. MicroQoSCORBA's fault tolerance [8] and security [17] subsystems can be fine-tuned based upon an application's non-functional constraints. MIDES is integrated into the MicroQoSCORBA framework as a highly configurable and independent module.

Integrating MIDES into the MicroQoSCORBA framework entailed extending the framework's IDL compiler and other tools so that they became IDS-enabled. With these additions, the IDL compiler now generates two additional files that individually configure the respective client or server IDS implementation. These files are used to set up the relationship between MIDES's data, policy, profile, and response implementations.

A MIDES data object is a general representation of an output from an internal sensor. As illustrated in Figure 2, a data object is resolved by the IDS kernel to a policy object. A policy object resolves to appropriate responses and profiles. Profiles are implemented so they can be reused by different policies referenced by the same data point. Responses are configured globally where their functionality can be fitted for the specific purpose of the application. This allows responses to be instantiated only once. A misuse-based detector provides a data object that uniquely identifies the configured response(s).

All sensors and detectors described in Section 3.2 are supported by MIDES. Interval-based sensors are automatically injected into stub and skeleton code by the IDS-aware IDL compiler. Various configuration settings direct these sensor's data collection requirements (e.g., data rates, parameter types). Application-based security policies provided by MIDES can be based on either static or dynamically computed values. Procedural-based sensors are embedded into the application at the beginning and possible exits of each function. These sensors monitor an application to detect deviations from a previously determined profile. MIDES computes these execution-based profiles with a simple sliding windows algorithm as in [9], which runs in a linear time bound, O(n). The sliding window profile is kept in a sparse tree that only stores a limited number of combinations. This algorithm was chosen because of its simplicity, rather than its exponential size bound, which is too resource intensive for small embedded systems.

Two embedded detectors are implemented in MicroQoSCORBA to illustrate the use of configurable detectors. These detectors limit the number of

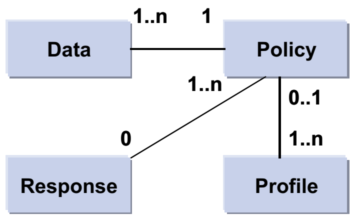

Figure 2. The relationships between a MIDES data point, policy, profile and response

connections made by a client to a server as an indication of misuse and resource starvation. Their logic is further discussed in the following section.

4.2. Application-Based Security Policies in MIDES

Several reusable application-based security policies are implemented in MIDES, as shown in Table 1. The *Maximum* and *Minimum Value* policies can reveal application semantic errors such as invalid temperature ranges. *Delta Value* is a policy that detects unexpected variations in values over a short amount of time. A policy such as *Maximum Average* is used to check the maximum distance from a moving average for each measurement, which overcomes some of the shortcomings of the minimum and maximum policies. The *Cumulative Distribution Function* (CDF) implementation will identify rare values, given normally distributed data values.

Two embedded detectors have been integrated into the MicroQoSCORBA middleware framework for use by MIDES. The two application-based policies are:

1. A server is allowed to accept a configured maximum number of concurrently running connection from a particular client
2. A server must prevent a client from making repeated connections attempts that fully exhaust the server's resources within a specified time frame.

4.3. Transparent Support for Application-Based Responses

Several intrusion detection responses have been implemented in MIDES (Table 2). The individual responses to specific internal sensors are omitted.

Table 1. Application based policies currently provided by MIDES

Application-Based Security Policies in MIDES					
Policy	Interval based	Procedural based	Profile required	Average Stddev	Description
Maximum Value	Yes	Yes	No	None	Maximum allowed value
Minimum Value	Yes	Yes	No	None	Minimum allowed value
Delta Value	Yes	No	Yes	None	Difference in value over time - $\Delta(V/t)$
Maximum average	Yes	Yes	Yes	Average	Maximum distance from moving average
CDF	Yes	No	Yes	Both	Cumulative Distribution Function

Legend: Column two and three indicates whether or not the application-based security policy can be configured for the internal sensor specified by the respective column title. Column four indicates the policy's need for profiling; and at last in column five, an indication if the profile depends on the computed statistical average or standard deviation.

Time-delays can be configured to slow down a connection or application object for a limited amount of time. IP-ban bans an IP-address either permanently or for a given interval. Audit stores a pre-defined number of audit records in memory and can write events to a file. The terminate connection response immediately terminates a connection. MIDES can also be configured to block an invocation based on the output of an interval-based sensor.

MIDES, like pH [26], can slow down parts of an application or network connection by embedding a time delay in the code for a procedural-based sensor. Audit and Time-delay is the only responses provided by MIDES, which are able to target the application alone. Responses targeting the communication link between two hosts include audit, time-delay, connection termination, and banning the remote host's IP-address. MIDES's responses can check a dynamically updated list of trusted sources before activation.

There is a risk related to the possibility that a false positive response may induce a self-inflicted denial of service. This risk must be considered by an application developer before deploying a response.

5. Performance Evaluation of MIDES

An evaluation of our implemented MIDES framework was conducted in order to determine the feasibility of integrating intrusion detection into a middleware framework designed and implemented for resource-constrained embedded systems. MIDES was developed in Java and implemented as a configurable and subsettable module of the MicroQoSCORBA middleware framework [17], [18], [19]. The MIDES framework could also be ported to C++ and integrated into the C++ version MicroQoSCORBA.

5.1. Experimental Setup Configuration

A performance evaluation was conducted on two different platforms. The first platform consisted of two desktop PCs (2.4 GHz Intel® Pentium® CPUs with

Table 2. General responses that are supported by MIDES

Application-Based Responses in MIDES			
Response	Risk	Target	Description
Audit	L	A / C	Generate an audit record
Time-delay	M	A / C	Delay the connection for a period of time
Terminate Connection	H	C	Terminate the connection to remote host
IP-Ban	H	C	Ban the IP-address of a remote host

Legend: The risk in case of a false positive are categorized as Low, Medium, or High (L/M/H). The target of the response can either be Application or Connection based, (A/C).

1 GB of memory) running Slackware 9.1 (2.6 kernel) on a 100 Mbps network. The evaluation code was compiled and executed with Sun's Java 2 Software Development Kit (version 1.4.2_04). The second platform consisted of two Dallas Semiconductor TINI boards [30]. These TINI boards are powered by a 40 MHz 8-bit DS80C390 CPU with 512 Kbytes of memory and connected with a 10 Mbps network. Sun's Java SDK was used to create Java 1.1.8 compatible class files that were converted into a compressed archive suitable for execution on the TINI boards with the TINIConvertor tool (version 1.02e).

The performance evaluation, by design, consisted of a very simple application based upon the CORBA IDL in Source Listing 1. MicroQoSCORBA was configured to use CORBA IIOP version 1.2. Details regarding the training of the MIDES anomaly detection algorithms are given in [19]. Additionally, no intrusions were attempted during this evaluation so that we could determine the best-case performance overheads associated with integrating the MIDES framework into MicroQoSCORBA.

Listing 1. Test Application Interface description

```
module timing {
   interface foo {
      long bar (in long arg1);
   };
};
```

5.2. Evaluation Results

Tables 3, 4 show the application file sizes and performance overheads of various MIDES configurations. These tables support setting tradeoffs between desired intrusion detection support and performance (or resource usage) constraints of the application's embedded hardware.

The application sizes resulting from embedding different internal sensors and an IDS-kernel into an MIDES application are reported in Table 3. The interval-based sensor includes a maximum policy and an audit response, so it is smaller than the procedural-based sensor, which incorporates additional data structures and an algorithm. The misuse-based detector does not require any policies or profile overheads and therefore required less space. As mentioned, the TINI files are compressed and therefore smaller than the Linux class file sizes; also, none of the values in Table 3 include the size of Java's runtime libraries.

Introducing application-based security policies increases the minimum memory overhead by 3.2 KB to 3.7 KB for the client and server, respectively, on the Linux based PCs. For the TINI platform, these increases are 8.6 KB on the client side and 8.8 KB on the server side.

The end-to-end latencies measured by invoking `foo.bar(…)` repeatedly for various MIDES policy configurations are reported in Table 4. Previous

Table 3. Application sizes listed by type of sensors integrated

Sensors/Detectors	Application sizes (bytes)							
	Linux				Tini			
	Client	%	Server	%	Client	%	Server	%
Baseline	63735	-	61250	-	23869	-	20599	-
Interval	83060	30.32%	80584	31.57%	29792	24.81%	26527	28.78%
Procedural (SW)	89286	40.09%	86854	41.80%	31711	32.85%	28393	37.84%
Misuse	87416	37.16%	78020	27.38%	33137	38.83%	29822	44.77%

Table 4. End-to-end latencies listed by type of policies configured

Policies	End-to-End Latencies (ms)							
	Linux				Tini			
	Client	%	Server	%	Client	%	Server	%
Baseline (no ids)	0.117	-	0.117	-	254.44	-	254.44	-
Interval-based								
Maximum	0.122	4.27%	0.122	4.10%	290.48	14.16%	295.76	16.24%
Minimum	0.123	4.96%	0.122	3.93%	290.57	14.20%	295.41	16.10%
	0.123	5.38%	0.124	6.32%	307.43	20.82%	314.32	23.53%
Maximum of Average	0.124	5.73%	0.124	5.81%	310.54	22.05%	317.64	24.84%
CDF	0.124	6.32%	0.125	6.84%	426.88	67.77%	445.82	75.21%
Frequency-based	0.123	5.13%	0.122	4.27%	314.62	23.65%	317.60	24.82%
Procedural-based								
Sliding Windows	n/a	-	0.130	11.11%	n/a	-	555.83	118.45%
PST	n/a	-	0.125	6.84%	n/a	-	297.00	16.73%

research has shown that about 5% of the Linux and 25% of the TINI invocations are slowed down by Java garbage collection and other platform specific overhead [17]. To more accurately compare the best-case end-to-end latencies between these platforms, these "slow" invocations have been filtered out of the results reported in Table 4, using the algorithm in [17].

The Maximum of Average and CDF policies compare the current value with a trace of 30 previous values. There is a significant difference between MIDES policies that do not require a profile and those policies that do (e.g., Max. of Average and CDF).

Comparing the least and the most resource demanding policies on the client side to the baseline configuration results in latency increases of 2.5% to 6.9% on the Linux based PCs. On the TINI boards these overheads increased by 14.2% to 89.6%. Using an interval sensor results in a few extra system-calls and object creations and therefore adds only a minor increase to the end-to-end latencies.

The procedural-based detection mechanism is configured with a trace length of 30 values and a window size of 5. This increases the overhead of performing the invocations by 17.4% to 17.8% for the Linux client and server, respectively. The overhead is larger on the TINI boards, ranging from 41% to 48% for the client and server, respectively.

Only time-delay and IP-ban require a direct implementation in the middleware layer. The code for IP-ban is executed once per connection and the code for time-delay is executed for each invocation. Only the time-delay response introduces any significant end-to-end latency delay. This response's overhead ranged from 3.2% to 5.1% on Linux and 38.2% to 46.8% on TINI, as compared to the baseline performance.

6. Conclusions and Future Work

This paper presents EMIDS, a model for application-based intrusion detection integrated into a middleware framework designed for embedded systems. We have implemented a highly configurable middleware-based intrusion detection system, MIDES, based upon our EMIDS model. A set of mechanisms and reusable application-based security policies and responses, suitable for embedded systems, was provided. Finally, the experimental evaluation of MIDES and its mechanisms illustrate the feasibility of middleware-based intrusion detection for resource-constrained embedded systems.

This paper also discussed how implementing responses in traditional, OS-based differs from implementing them into an application-based IDS. We described some advantages of integrating intrusion detection into the middleware layer. In particular, this allows for the automated embedding of application-specific intrusion detection sensors, a non-trivial task for other IDS frameworks.

Future work for EMIDS includes experiments conducted in a real-world application's development, deployment, and execution cycle to show the feasibility of the approach presented. Additionally, research could be performed to see if application-based policies can be dynamically configured and adaptable in a continuously changing environment.

7. Acknowledgments

The authors would like to thank Tarana R. Damania and the other members of the MicroQoS-CORBA team, past and present, who have helped develop MicroQoSCORBA. We would also like to thank Jennifer R. Jackson, Anthony A. Kempka, and Bryan H. McMillan for reviewing this paper. A final thank you goes to the anonymous reviewers for their feedback.

8. References

[1] M. Almgren, and U. Lindqvist. Application-Integrated Data Collection for Security Monitoring. In *Proceeding of Recent Advances in Intrusion Detection* (RAID), LNCS, pp. 22-26, Davis, CA, October 2001, Springer.

[2] M. Atighetchi, P. P. Pal, C. Jones, P. Rubel, R. E. Schantz, J. P. Loyall, and J. A. Zinky. Building Auto-Adaptive Distributed Applications: The QuO-APOD Experience. The 3rd International Workshop on Distributed Auto-adaptive and Reconfigurable Systems, in conjunction with the 23rd International Conference on Distributed Computing Systems, May 19-22, 2003, Providence, Rhode Island, USA.

[3] R. G. Bace. *Intrusion Detection*. Macmillan Technical Publishing, 201 West 103rd Street, Indianapolis, IN 46290.

[4] A. S. Brown. "SCADA vs. the hackers – Can freebie software and a can of Pringles bring down the US power grid?" December 2002, Mechanical Engineering Magazine.

[5] Cisco Self Defending-Network, Online, 2004. See website http://www.cisco.com.

[6] CylantSecure. Online, 2004, See website http://www.cylant.com.

[7] R. F. Dacey. United States General Accounting Office Critical Infrastructure Protection, Challenges in Securing Control Systems, See http://www.gao.gov/cgi-bin/getrpt?GAO-04-140T.

[8] K. E. Dorow. Flexible Fault Tolerance in Configurable Middleware for Embedded Systems. In *Proceedings of 27th International Computer Software and Applications Conference* (COMPSAC 2003): Design and Assessment of Trustworthy Software-

Based Systems, pp. 563-569, 3-6 November 2003, Dallas, Texas, USA, IEEE Computer Press, 2003.

[9] S. Forrest, S. Hofmeyr, A. Somayaji, and T. Longstaff. “A sense of self for Unix processes,” In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pp. 120–128, IEEE Computer Press, 1996.

[10] B. Gellman. Cyber-Attacks by Al Qaeda Feared. Washington Post Thurs., June 27, 2002; Page A01.

[11] IPTables. Online, 2004, See website http://www.netfilter.org.

[12] K. P. Kihlstrom, P. Narasimhan. ”The Starfish System: Providing Intrusion Detection and Intrusion Tolerance for Middleware Systems,” IEEE Workshop on Object-oriented Realtime Dependable Systems, Guadalajara, Mexico, January 2003.

[13] D. Malkhi, M. Reiter, AT&T Labs. “Unreliable Intrusion Detection in Distributed Computations” 10th Computer Security Foundations Workshop (CSFW '97), June 10 - 12, 1997, Rockport, Massachusetts. USA.

[14] Z. Marrakchi, L. Mè, B. Vivinis and B. Morin. “Flexible Intrusion Detection Using Variable-Length Behavior Modeling in Distributed Environment: Application to CORBA Objects.” In *Proceedings of 3rd Recent Advances in Intrusion Detection Workshop* (RAID 2000) Toulouse, France, October 2-4, 2000, (Eds. H. Debar, L. Mè, and F. Wu), LNCS 1907, pp. 130-144, Springer-Verlag, 2000.

[15] A. D. McKinnon. *Supporting Fine-grained Configurability with Multiple Quality of Service Properties in Middleware for Embedded Systems*. Doctoral Dissertation, School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA, December 2003.

[16] A. D. McKinnon, K. E. Dorow, T. R. Damania, O. Haugan, W. E. Lawrence, D. E. Bakken, J. C. Shovic, “A configurable middleware framework with multiple quality of service properties for small embedded systems,” In *Proceedings of the 2nd IEEE International Symposium on Network and Computing Applications* (NCA2003), IEEE Computer Society, 2003, pp. 197–204.

[17] A. D. McKinnon, D. E. Bakken, and J. C. Shovic, “A configurable security subsystem in a middleware framework for embedded systems,” *Computer Networks*, vol. 46, iss. 6, pp. 771-795, 20 December 2004.

[18] MicroQoSCORBA, Online, 2004, See website http://microqoscorba.net

[19] E. Næss. *Configurable Middleware-level Intrusion Detection Support for Embedded Systems*. MS Thesis, School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA, August 2004.

[20] P. Oman, E. Schweitzer, and D. Frincke. “Concerns About Intrusions into Remotely Accessible Substation Controllers and SCADA Systems,” 27th Annual Western Protective Relay Conference, Paper #4, (October 23-26, Spokane, WA), 2000.

[21] P. Oman, E. Schweitzer, and J. Roberts. Safeguarding IEDs, “Substations, and SCADA Systems Against Electronic Intrusions,” *Proceedings of the 2001 Western Power Delivery Automation Conference*, Paper No. 1, (April 9-12, Spokane, WA), 2001.

[22] Peakflow SP. network intrusion detection system, Online, 2004. See website http://www.arbor.com.

[23] D. Sames, B. Matt, B. Niebuhr, G. Tally, B. Whitmore, D. E. Bakken. “Developing a Heterogeneous Intrusion Tolerant CORBA System.” In *Proceedings of International Conference on Dependable Systems and Networks* (DSN'02), June 23 - 26, 2002, Washington, D.C., USA.

[24] R. S. Sielken. “Application intrusion detection,” Technical Report CS-99-17, Department of Computer Science, Univ. of Virginia, June 1999.

[25] Snort. Online, 2004, See website www.snort.org.

[26] A. Somayaji and S. Forrest. “Automated response using system-call delays.” In *Proceedings of the 9th USENIX Security Symposium*, August 2000. URL http://cs.unm.edu/~forrest/publications/uss-2000.ps.

[27] E. Spafford, D. Zamboni. “Data collection mechanisms for intrusion detection systems.” CERIAS Technical Report 2000-08, CERIAS, Purdue University, 1315 Recitation Building,West Lafayette, IN, June 2000.

[28] M. Stillerman, C. Marceau, and M. Stillman. “Intrusion Detection for Distributed Applications,” *Communications of the ACM*, Vol. 42, No. 7, pp. 62-69, July, 1999.

[29] D. Tennenhouse. “Embedding the Internet: Proactive Computing,” *Comm. of the ACM*, May, 2000.

[30] TINI. Online, 2004. See website http://www.ibutton.com/TINI.

[31] J. Turley. *The Essential Guide to Semiconductors*. Prentice Hall, 2003, Professional Technical Reference, Upper Saddle River, NJ 07458, www.phptr.com.

[32] D. Zamboni. *Using Internal Sensors for Computer Intrusion Detection*. PhD thesis, Purdue Univ., West Lafayette, IN, August 2001, CERIAS TR 2001-42.