**AFRL-IF-RS-TR-2002-229**
**Final Technical Report**
**September 2002**

# SECURITY AGILITY FOR DYNAMIC EXECUTION ENVIRONMENTS

**Trusted Information Systems**

**Sponsored by**
**Defense Advanced Research Projects Agency**
**DARPA Order No. F267**

*APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED*

**The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.**

**AIR FORCE RESEARCH LABORATORY**
**INFORMATION DIRECTORATE**
**ROME RESEARCH SITE**
**ROME, NEW YORK**

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2002-229 has been reviewed and is approved for publication

APPROVED: Robert J. Vaeth

ROBERT J. VAETH
Project Engineer

FOR THE DIRECTOR:

WARREN H. DEBANY, Technical Advisor
Information Grid Division
Information Directorate

| REPORT DOCUMENTATION PAGE | | | | *Form Approved*<br>*OMB No. 074-0188* |
|---|---|---|---|---|

| | | |
|---|---|---|
| **1. AGENCY USE ONLY (Leave blank)** | **2. REPORT DATE**<br>SEPTEMBER 2002 | **3. REPORT TYPE AND DATES COVERED**<br>Final Jul 97 – Sep 00 |

| | |
|---|---|
| **4. TITLE AND SUBTITLE**<br>SECURITY AGILITY FOR DYNAMIC EXECUTION ENVIRONMENTS | **5. FUNDING NUMBERS**<br>C - F30602-97-C-0225<br>PE - 62301E<br>PR - F267<br>TA - 71<br>WU - 02 |
| **6. AUTHOR(S)**<br>Tim Fraser, Mike Petkac, and Lee Badger | |
| **7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**<br>Trusted Information Systems<br>NAI Labs, Network Associate<br>3060 Washington Road<br>Glenwood Maryland 21748 | **8. PERFORMING ORGANIZATION REPORT NUMBER** |
| **9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)**<br>Defense Advanced Research Project Agency AFRL/IFGB<br>3701 North Fairfax Drive 525 Brooks Road<br>Arlington Virginia 22203-1714 Rome New York 13441-4505 | **10. SPONSORING / MONITORING AGENCY REPORT NUMBER**<br>AFRL-IF-RS-TR-2002-229 |

**11. SUPPLEMENTARY NOTES**

AFRL Project Engineer: Robert J. Vaeth/IFGB/(315) 330-2182/ Robert.Vaeth@rl.af.mil

| | |
|---|---|
| **12a. DISTRIBUTION / AVAILABILITY STATEMENT**<br>APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED | **12b. DISTRIBUTION CODE** |

**13. ABSTRACT** ***(Maximum 200 Words)***

The Security Agility for Dynamic Execution Environments project developed practical solutions to problems faced by traditional applications in environments governed by dynamically reconfigurable security policies, In such environments, applications that are unaware of the security policy's dynamic nature may crash or misbehave when confronted with security policy changes that revoke their resources. They may fail to recover when subsequent security policy changes restore their access to resources. They may fail to abort activities that are rendered illegal by security changes made while the activities are already in progress. The project's primary result was the development of a software toolkit for retrofitting existing dynamically linked applications with new "agile" mechanisms to avoid or compensate for these failures. With the help of the software toolkit, existing UNIX applications can be retrofitted with new functionality that allows them to operate effectively in new environments governed by dynamically reconfigurable security policies, even in cases where the application's source code is not available.

| | |
|---|---|
| **14. SUBJECT TERMS**<br>Dynamic Execution Environments, Security, Policy | **15. NUMBER OF PAGES**<br>19 |
| | **16. PRICE CODE** |

| | | | |
|---|---|---|---|
| **17. SECURITY CLASSIFICATION OF REPORT**<br>UNCLASSIFIED | **18. SECURITY CLASSIFICATION OF THIS PAGE**<br>UNCLASSIFIED | **19. SECURITY CLASSIFICATION OF ABSTRACT**<br>UNCLASSIFIED | **20. LIMITATION OF ABSTRACT**<br>UL |

# Contents

# 1. Summary

The Security Agility for Dynamic Execution Environments project (hereafter referred to as the Agility project) developed practical solutions to problems faced by traditional applications in environments governed by dynamically reconfigurable security policies. Dynamic coalitions require distributed systems that support dynamically reconfigurable security policies. Only security policies that can be reconfigured during system runtime can address the complex events that are common in these environments, such as shifts in alliances, changes in personnel, intrusion alerts, and process migrations due to hardware losses.

Unfortunately, the act of dynamically reconfiguring the global security policy of a distributed system may have undesirable side-effects: Applications that are unaware of the security policy's dynamic nature may crash or misbehave when confronted with security policy changes that revoke their resources. They may fail to recover when subsequent security policy changes restore their access to resources. They may fail to abort activities that are rendered illegal by security changes made while the activities are already in progress. To ensure correct system operation, applications must be aware of the dynamic nature of the security policy that governs their environment, and they must be able to adapt to security policy changes during runtime with little or no manual assistance. Through the development of a series of increasingly sophisticated software toolkits, the Agility project explored practical solutions to retrofit this awareness and adaptation, or "agility", into existing applications.

The primary result of the Agility project was the final version of its software toolkit. However, the development of the software toolkit brought several conceptual results to light, as well. These conceptual results can be summarized as follows:

Applications use resources and services to perform useful work. In environments where a security policy determines which resources and services are available for use, changes to this security policy can affect an application's ability to perform useful work. The Agility project showed that there are at least two broad classes of application failures due to security policy change: first, applications may misbehave when resources they had previously acquired are unexpectedly revoked by a security policy revision, and second, server applications that are unaware that security policy changes may occur during their runtime may improperly grandfather access rights, allowing their clients to illegally continue performing previously legal operations that have been made illegal by a revised security policy.

By successfully implementing and demonstrating the effectiveness of its software toolkits in a laboratory testbed, the Agility project provided experimental evidence that, once an existing application is known to fail in response to particular kinds of security policy changes, it is possible to compensate for or avoid these failures by augmenting the application with additional functionality. Furthermore, the Agility project demonstrated that the same techniques used to

compensate or avoid these failures can also be used to add new application-level security policy enforcement mechanisms.

In theory, there are at least five generic ways in which an application can positively respond to an unexpected resource revocation. These responses include polling or blocking until a future policy change cancels the revocation, proceeding despite the lack of resources, terminating, or adapting to the new environment by attempting to find alternate resources. However, the Agility project's initial toolkit development efforts showed that in practice, it is difficult to provide a toolkit that implements all of these generic responses in a manner that suits all imaginable applications in all imaginable situations. Nonetheless, the initial version of the software toolkit suited a sufficient number of applications in a sufficient number of situations to have practical value.

In an effort to address the failures of applications not satisfied by the initial software toolkit, later versions of the software toolkit focused on providing a more general infrastructure to better support solutions tailor-made to individual applications. The later software toolkits also used interposition as a means of adding functionality to dynamically linked applications without modifying their program text, and without access to their source code.

In retrospect, it seems clear that the problem investigated by the Agility project is not specific to the realm of security research. Similar problems can be found in environments that do not enforce security policies, but do admit untimely resource revocations (as in distributed computing environments with unreliable communications) or unexpected errors that require the abortion of in-progress processing (as in database management systems). It can be said that the application failures witnessed and addressed by the Agility project were not caused by some problematic aspect peculiar to systems that enforce complex security policies. Instead, the failures were the natural result of running applications designed to operate in a traditional UNIX environment in a new environment where many assumptions considered safe in a UNIX world do not hold.

However, given the significant role traditional UNIX applications play in the infrastructure of many modern distributed computing environments (and in the Internet in particular), there is considerable practical value in the ability to run familiar tried-and-true UNIX applications in new hardened environments with dynamically reconfigurable security policy enforcement functionality. With its software toolkit, the Agility project shows that it is possible to adapt applications designed to operate in a UNIX environment to operation in a new environment governed by a dynamically reconfigurable security policy. Furthermore, since the software toolkit enables retrofitting even in cases where the application's source code is not available, there are cases where the effort required to retrofit an application using the software toolkit is significantly less than the effort required to attempt a port or complete reimplementation.

# 2. Introduction

The Security Agility for Dynamic Execution Environments project developed practical solutions to problems faced by traditional applications in environments governed by dynamically reconfigurable security policies. Dynamic coalitions require distributed systems that support dynamically reconfigurable security policies. Only security policies that can be reconfigured during system runtime can address the complex events that are common in these environments, such as shifts in alliances, changes in personnel, intrusion alerts, and process migrations due to hardware losses.

Unfortunately, the act of dynamically reconfiguring the global security policy of a distributed system may have undesirable side-effects: Applications that are unaware of the security policy's dynamic nature may crash or misbehave when confronted with security policy changes that revoke their resources. They may fail to recover when subsequent security policy changes restore their access to resources. They may fail to abort activities that are rendered illegal by security changes made while the activities are already in progress. To ensure correct system operation, applications must be aware of the dynamic nature of the security policy that governs their environment, and they must be able to adapt to security policy changes during runtime with little or no manual assistance. The Security Agility for Dynamic Execution Environments project developed solutions to enable this awareness and adaptation, or "agility" in applications.

The project's exploration of this problem was structured around the development of a freely available practical software toolkit for engineering agility in applications. Development occurred on platforms ranging from the BSD/OS 2.1 Domain and Type Enforcement prototype, [1] to Windows NT, FreeBSD 3.2, and Linux 2.2.

This is the final report for the Security Agility for Dynamic Execution Environments project. This report attempts to summarize its results in a concise manner. Three previous reports exist which present the results of the project's three phases in much greater detail ([2], [3], and [4]). This report is intended for researchers and administrators who wish to be informed of the significant findings of the project without having to become aware of all of the implementation details.

The remainder of this report is divided into three sections. First, Section 3 summarizes the methods used to undertake the project. This is followed by Section 4, which summarizes its results. Finally, Section 5 presents our conclusions.

# 3. Methods, Assumptions, and Procedures

The ultimate goal of the Security Agility project was to develop a practical and general solution to the problems faced by applications constrained by dynamically configurable mandatory access control policies, in the form of a software toolkit. Due to this focus on producing useful software, the project's methodology emphasized the implementation, evaluation, and incremental improvement of an increasingly sophisticated and general series of software prototypes. This section summarizes the goals that motivated development during each of the project's three intermediate phases and the methods and tools used to meet these goals.

## 3.1. Phase One

The primary goals of the first phase of the project were, first, to find and document examples of application failures caused by runtime security policy changes, and second, to develop an initial collection of software techniques to compensate for these failures. During Phase One, a relatively large number of experimenters worked independently, in parallel, using the following method to discover and develop remedies for application failures:

Each experimenter employed a commodity PC workstation running a new version of the BSD/OS 2.1-based Domain and Type Enforcement prototype (hereafter called DTE) [1]. This new version of DTE was enhanced by the experimenters to perform mediation on file descriptor usage operations such as read and write. Older versions of DTE performed mediation only on file descriptor creation operations, such as open. This additional mediation allowed the new version of DTE to revoke resources in a manner more consistent with the Security Agility project's target environment, Quorum.

Because of its BSD/OS base, DTE provided many applications critical to a distributed UNIX environment, including remote login, auditing, FTP and HTTP services (to name a few). The experimenters divided responsibility for the applications among themselves. Each experimenter began by examining the source code of their critical applications, looking for evidence that they might fail when faced with a security policy change. Each of the experimenters then concentrated their efforts on whichever of their critical applications (or group of related critical applications) they judged most likely to fail.

In addition to these critical applications, DTE provided a mechanism for enforcing mandatory access control policies that could be dynamically reconfigured during runtime. This dynamically reconfigurable enforcement mechanism provided an essential part of the dynamic security policy environment that was targeted by the Security Agility project, allowing experimenters to conduct their initial exploration via interaction with a real operating system rather than a simulation or a formal model. This availability of a real operating system obviated much of the need for the experimenters to make assumptions about how systems in the project's target environment might operate.

Each experimenter proceeded by observing their chosen critical application running under a variety of security policies. Experimenters used their knowledge of distributed UNIX environments and DTE security policies to create security policy changes designed specifically to cause failures in their chosen critical applications. Since many of the experimenters were former DTE developers, their detailed understanding of DTE security policies made this ad-hoc approach to discovering application failures effective.

Once they had discovered a security policy change that caused one or more failures of sufficient severity to prevent their chosen application from providing useful service, each experimenter proceeded to modify their application, implementing a solution to compensate for the failure. Each experimenter was encouraged to implement their solution without regard for generality or commonality between applications, in an effort to generate the most diverse range of solutions possible. The experimenters accomplished all of their software development using the tools and environment provided with the BSD/OS 2.1 system.

Once the experimenters had implemented their individual solutions, they worked together to produce a combined demonstration of their software. In order to create a demonstration environment that more closely resembled the Agility project's target environment, the experimenters expended some effort into implementing application-level DTE-like policy enforcement functionality for the Apache HTTP server. Quorum was envisioned by its designers as a distributed system made up of heterogeneous nodes, some capable of enforcing mandatory access control policies, and some not. Consequently, the demonstration environment included both DTE and non-DTE BSD/OS nodes. By itself, the non-DTE BSD/OS node was of limited use in a demonstration dependent on changes in security policy configuration. However, the DTE-like functionality allowed the apache HTTP server on the non-DTE BSD/OS node to enforce a dynamic mandatory access control policy. Using this functionality, the demonstration could include HTTP client failures due to changes in the BSD/OS node's HTTP server's mandatory access control policy, making the BSD/OS node a useful part of the demonstration.

Further details concerning the methods used to complete phase one of the project can be found in the Security Agility for Dynamic Execution Environments Initial Prototype Evaluation Report [2].

## 3.2. Phase Two

The primary goals of the second phase of the project were, first, to identify and model whatever commonality existed among the application failures observed in the Phase One, and second, to abstract the solutions implemented in phase one away from the specific details of their applications, and integrate them into a general toolkit. Implicit in the first goal was the project's hypothesis that there was some sort of common cause behind the application failures - some sort of common architectural deficiency that made it difficult for applications to operate in environments governed by dynamic security policies.

Because its goals were less exploratory and more analytical, the second phase proceeded with fewer experimenters than the first. While some experimenters analyzed the failures and solutions from phase one in hope of finding commonality, others undertook the engineering task of separating the solutions from their applications, generalizing them, and integrating them into a reusable toolkit. The toolkit took the form of several libraries implementing solution functionality, coupled with a modified C library that allowed the solution functionality to be added to an existing application.

In Phase Two, the experimenters also expended considerable effort in an attempt to move the experiment from the BSD/OS DTE platform to the Windows NT platform in order to better suit the needs of the customer. Since DTE functionality was not available in the Windows NT operating system, the application-level DTE-like functionality developed in phase one for the Apache HTTP server became the primary enforcer of mandatory access control policies on Windows NT platform. In this effort, the experimenters made use of Windows NT version 4 and Microsoft's Visual C++ environment and tools.

At the conclusion of Phase Two, the experimenters produced a demonstration of their integrated toolkit based on both BSD/OS-DTE and Windows NT platforms. Further details concerning the methods used to complete Phase Two of the project can be found in the Security Agility for Dynamic Execution Environments Security Toolkit Evaluation Report [3].

## 3.3. Phase Three

Experience gained in the first two phases of the project influenced the goals in the third and final phases, adjusting them somewhat from what was envisioned during the project's earliest days. The primary goal of the third phase of the project was to enhance the second phase's toolkit for use in a distributed environment. In addition, the unexpected emphasis on the Windows NT platform in phase two introduced a new secondary goal: to enable the use of the toolkit on applications for which no source was available. Because the project's planners assumed that this phase would require only incremental improvement of the existing software toolkit, this phase of the project proceeded with the smallest number of experimenters (at some points, only one).

Meeting the second goal (interoperability with closed-source applications) took the greatest effort in Phase Two. After observing the success of other projects which used interposition techniques to enhance existing closed-source programs [5], the experimenters decided to replace their existing mechanism for adding toolkit functionality to applications (the modified libc) with a new interposition-based mechanism. The experimenters implemented the new mechanism on Windows NT first, since its dynamically linked COFF binary format provided good support for interposition techniques.

Unfortunately, the aging BSD/OS-DTE platform did not support a proper dynamically linked binary format. Consequently, the experimenters were forced to port their BSD/OS toolkit to the closely related FreeBSD 3.2 platform before implementing a suitable interposition scheme using FreeBSD's dynamically linked ELF binary format. The experimenters expended further porting effort when the customer expressed a preference for the Linux platform. Fortunately, like FreeBSD, Linux also supported the ELF binary format. The experimenters used the kernels and tools included in the RedHat Linux 6.0 and 6.1 distributions to accomplish this task.

Further details concerning the methods used to complete phase three of the project can be found in the Security Agility for Dynamic Execution Environments Distributed Security Toolkit Evaluation Report [4].

# 4. Results and Discussion

This section summarizes the major results of the Security Agility for Dynamic Execution Environments project. The main result of the project was the production of a general software toolkit designed to address the problems faced by applications running in environments with dynamically reconfigurable security policies. Secondary results included a classification of policy-change-related application failure modes, a series of increasingly sophisticated intermediate toolkit prototypes, insight into what kinds of functionality a toolkit should provide (complete mechanisms vs. general infrastructure), and techniques for augmenting applications with new mechanisms to enforce security policies. Each result is discussed below; specific findings are described in offset paragraphs.

## 4.1. Classification of Failures due to Security Policy Change

> *Applications use resources and services to perform useful work. In environments where a security policy determines which resources and services are available for use, changes to this security policy can affect an application's ability to perform useful work.*

This assertion was the genesis of the Security Agility for Dynamic Execution Environments project (hereafter referred to as the "Agility project"). It was first expressed by the developers of the Domain and Type Enforcement Firewalls project [6] who observed that critical infrastructure applications sometimes failed during experiments that involved changing the security policy during runtime.

The Agility project's first activity identified and documented many examples of these failures in applications used to support distributed computing. Failures included a variety of improper application behaviors, ranging from hangs and crashes (usually due to the revocation of resources) to non-compliance with security policy revisions (usually due to improperly grandfathered access rights). The most significant applications examined were *accton*, *crond*, *httpd* (Apache), *libc* (those parts dealing with DNS), *lpr/lpd*, *named*, *portmap*, *rlogin-gw* (part of the TIS Firewall Toolkit), *rlogind*, *rshd*, and *syslogd*. From this group, the experimenters chose *httpd*, *rlogin-gw*, *rlogind*, *rshd*, and *syslogd* as the best representatives of the whole, and demonstrated their runtime failures in the Phase One demonstration [2].

> *There are at least two broad classes of application failures due to security policy change: first, applications may misbehave when resources they had previously acquired are unexpectedly revoked by a security policy revision, and second, server applications that*

> *are unaware that security policy changes may occur during their runtime may improperly grandfather access rights, allowing their clients to illegally continue performing previously legal operations that have been made illegal by a revised security policy.*

On the BSD/OS-DTE prototype system, the *syslogd* application (the UNIX logging daemon) was particularly vulnerable to security policy changes that revoked its access to its output files. The *syslogd* application was capable of writing log messages to several output files - this capability could be used to place different categories of log messages into different files. Experimenters observed that once the *syslogd* application discovered that its ability to write log messages to a given output file had been revoked by a security policy change, it would never write to that output file again, even if a subsequent security policy change rescinded the revocation.

Furthermore, the combined system of the *rlogin-gw* firewall proxy and the *rlogind* and *rshd* applications was unable to properly pause or terminate in-progress user login sessions when security policy changes revoked the users' remote access rights. Once a valid user was successfully authenticated, experimenters observed that the *rlogin-gw* firewall proxy, *rlogind*, and *rshd* would allow the user's session to continue, even if the user was subsequently rendered invalid by a mid-session security policy change.

A more detailed classification of failures related to security policy change can be found in Section 2 of the Security Agility for Dynamic Execution Environments Initial Prototype Evaluation Report [2]. In addition, further discussion of the phase one demonstration, including the failure modes of *rlogin-gw* and *syslogd*, can be found in Section 3 of that document.

> *After a security policy change, the known security properties of formally verified ("trusted") applications might be insufficient to support the revised policy. (Conjecture.)*

As discussed in Section 2 the Evaluation Report cited above, the formal verification of a given "trusted" application may have relied on the application's operating environment to provide some useful properties. For example, the application's original verifiers may not have bothered to verify that the application cannot write to a given file, because they knew (assumed) that the underlying system's security policy enforcement mechanism would prevent such writes. If the security policy of the underlying system is revised to no longer prevent these writes, parts of the application's formal argument may be invalidated. Although this argument seems logical, a lack of formally verified applications prevented the Agility project from verifying it experimentally. Consequently, it must be viewed as conjecture.

## 4.2. Development of Agile Behaviors to Compensate for Failures

> *Once an existing application is known to fail in response to particular kinds of security policy changes, it is possible to compensate for or avoid these failures by augmenting the application with additional functionality.*

The task of augmenting an application with additional functionality to overcome failures caused by security policy change was called "making the application agile" in Agility project jargon. Making an application agile generally involved adding two kinds of functionality: 1) functionality that made the application aware of the potential for security policy change and allow it to detect changes; and 2) functionality to make the application react to policy changes in a constructive manner. The Agility project demonstrated the effectiveness of this technique experimentally by making several applications agile, including *rlogin-gw*, *rlogind*, *rshd*, and *syslogd*.

For the Phase One demonstration, the experimenters added new functionality to the *syslogd* application by directly modifying its source code. First, they added a mechanism that allowed a policy management component external to the *syslogd* application to notify running instances of *syslogd* via a software interrupt (signal) whenever policy changes occurred. Then, they added a mechanism that enabled the *syslogd* application to react to these notifications by attempting to reopen output files that it had lost due to previous revocations. In combination, these two new mechanisms allowed *syslogd* to recover its full functionality after a temporary period without access to some or all of its output files.

For the Phase One demonstration, experimenters also added new functionality to the combined system of the *rlogin-gw* firewall proxy and the *rlogind* and *rshd* applications. As with the syslogd application, the experimenters added a mechanism that allowed an external policy management component to notify a *rlogin-gw* of a security policy change via a software interrupt. They also added a mechanism that caused *rlogin-gw* to terminate any in-progress session belonging to a user whose remote access rights had been revoked by a security policy change. In the Phase Two demonstration, this mechanism could also be dynamically configured to suspend, rather than terminate in-progress sessions.

> *The same techniques used to make applications agile can also be used to add new application-level security policy enforcement mechanisms.*

This result was important in the context of the Agility project for two reasons. First, the solutions developed by the Agility project were intended for use in a heterogeneous distributed computing environment similar to Quorum, in which application might migrate from place to place over time. The experimenters imagined that some applications might depend on servers to provide certain security guarantees by enforcing a particular security policy. They reasoned that such an application might find itself migrated to a new location where the servers do not have sufficient security policy enforcement mechanisms to provide the security guarantees it needs. In these cases, the experimenters felt that it would be a valuable capability to be able to augment the deficient servers at the new location with whatever additional security policy enforcement mechanisms were required to support the migrated application.

Second, the ability to add new security policy enforcement mechanisms to servers made it possible to experiment with changes in security policies enforced by server applications, rather than operating systems. This capability allowed the experimenters to observe the effects of security policy change on applications that ran solely on operating systems with minimal security policy enforcement mechanisms (in comparison to DTE), such as Windows NT. For the Phase One demonstration, the experimenters added a runtime reconfigurable DTE-based security policy enforcement mechanism to the *httpd* (Apache) server application. For later demonstrations, the experimenters demonstrated the applicability of the DTE-based security policy enforcement to non-server applications, including *chdir*, *cp*, *ls*, *su*, and the *tcsh* and *bash* shells.

## 4.3. Abstraction of General Agile Behaviors in Toolkit Form

> *In theory, there are at least five generic ways in which an application can positively respond to an unexpected resource revocation. In practice, it is difficult to provide a toolkit that implements all of these generic responses in a manner that suits all imaginable applications in all imaginable situations. Nonetheless, such a toolkit can be made to suit a sufficient number of applications in a sufficient number of situations to have practical value.*

The Agility project identified five generic ways in which an application can positively respond to an unexpected resource revocation. Upon discovering that its access to a needed resource has been revoked by a security policy change, an application may:

> **poll:** Repeatedly test for renewed access to the revoked resource until the revocation is rescinded.
>
> **suspend:** Suspend processing (sleep) until the occurrence of some external event, such as another security policy change, or an administrative signal to continue.
>
> **proceed:** Proceed without accessing the revoked resource, possibly storing whatever data is required to perform the accesses later on, once access to the revoked resource is renewed.
>
> **terminate:** Halt further processing.
>
> **adapt:** Attempt to find an alternate resource that is accessible according to the revised policy.

The experimenters implemented these generic responses in the Security Agility Toolkit. During the course of this development, they observed that some of these behaviors were not applicable to certain applications in certain situations:

> **poll/suspend:** It may not be feasible for servers to sleep or wait for any length of time in a polling loop. They may have to handle further requests from clients in a timely fashion, and these requests may require the successful completion of all previous requests before proceeding.

**proceed:** Proceeding may not be an option for applications whose subsequent processing requires the application to first properly access the revoked resource.

**terminate:** Although termination is easily implemented, and prevents the application from performing further bizarre behaviors, it also prevents the application from performing any further useful work.

**adapt:** With applications not designed to handle the loss of resources, alternate resources are not always available.

## 4.4. Emphasis on Application-specific Behavior Infrastructure

*A toolkit that provides infrastructure to support the development of application-specific agile behaviors can bring agile behaviors to applications not satisfied by the earlier generic-behavior toolkit.*

Once the difficulty of implementing universally applicable generic behaviors became clear, the experimenters changed the focus of the improvements they made to succeeding versions of the Security Agility Toolkit. Instead of attempting to provide a collection of complete, universally-applicable generic behaviors, later versions of the toolkit focused on providing the infrastructure required to support: 1) the creation of new agile mechanisms tailored to the needs of specific applications; and 2) the integration of these new agile mechanisms with existing applications.

## 4.5. Interposition Enables Agility with Closed-source Applications

*Interposition provides a means of adding functionality to dynamically linked applications without modifying their program text, and without access to their source code.*

The later versions of the Security Agility Toolkit used interposition at the library/linker interface to integrate new mechanisms with existing applications. The new mechanisms themselves were implemented as dynamically loadable shared libraries. The ability of the interposition technique to enable the augmentation of existing applications without the use or modification of application source code has been demonstrated by independent efforts [5]. This result was confirmed by the final version of the Security Agility Toolkit. A detailed description of the final enhancements to the toolkit can be found in Section 4 of the Distributed Security Agility Toolkit Evaluation Report [4].

## 4.6. Integration with Intrusion Detection

*Agile functionality would be useful in environments where intrusion detection systems trigger security policy changes, particularly in cases where applications must activate and deactivate supplementary security policy enforcement mechanisms as the overall system's security posture changes. (Conjecture.)*

As described above, the experiments completed during the Agility project show that the functionality provided by the Security Agility Toolkit can: 1) help applications cope with the adverse effects of security policy change; and 2) allow applications to activate and deactivate their own supplementary security policy enforcement mechanisms during runtime. It seems reasonable to claim that this functionality would be useful regardless of whether the security policy changes were triggered administratively or automatically by an intrusion detection system [7]. However, in all the experiments undertaken during the Agility project, the security policy changes were always triggered manually by the experimenters, not automatically by an intrusion detection system. Consequently, assertions about the effectiveness of the Security Agility Toolkit when coupled with intrusion detection must be treated as conjecture.

## 4.7. Insight into the Security Agility Problem

> *The problem investigated by the Agility project is not specific to the realm of security research. The problem is not peculiar to systems that enforce complex security policies. Similar problems can be found in environments that do not enforce security policies.*

The problem the Agility project investigated is an instance of the general problem of making applications function properly in environments where access to resources may be intermittent and the rules governing what operations are desirable at a given moment are not under the complete control of the application performing those operations. Similar problems can be found in databases and fault-tolerant systems.

Applications exist which are fault-tolerant by design. Even in traditional UNIX environments, DNS lookup applications are capable of querying a list of alternate servers should their primary server be rendered unreachable by circumstances beyond their control. Similarly, database applications are capable of backing out of in-progress transactions when they find that intermediate steps have unexpectedly failed. These applications demonstrate solutions to instances of the same general problem addressed by the Agility project. Their solutions differ from the ones explored in the Agility project only in that they are not specifically targeted at failures caused by security policy changes.

The problem investigated by the Agility project is not a problem raised by the development of Quorum-like systems as much as it is a problem raised by the use of UNIX applications in a new environment for which they were not designed. The existence of working databases and fault-tolerant applications suggests that, if the designers of the UNIX applications examined in the Agility project were allowed to adjust their designs and implementations for (that is, port their applications to) a Quorum-like environment, their applications would handle security policy changes properly and would not require retrofitting.

Nevertheless, the toolkits and techniques developed by the Agility project have practical value because they enable system-builders to retrofit and extend familiar existing UNIX applications with less effort than would be required to port or reimplement them.

# 5. Conclusions

The Security Agility for Dynamic Execution Environments project (hereafter referred to as the Agility project) was an investigation of the problems that occur when applications designed for static execution environments (such as traditional UNIX) are run on systems where runtime security policy reconfiguration may change an application's execution environment at any time.

Most applications designed to operate in a UNIX environment fall into this category. In traditional UNIX environments, once an application acquires a local resource, its access to that resource is generally never revoked. Furthermore, changes in user authorization policies made while a user login session is in progress generally do not take effect until the user's next session. Consequently, application designed to operate in a UNIX environment often do not include functionality to cope with runtime changes in security policy, resource revocations, or resource reinstatements. When faced with these unfamiliar events, these applications often fail.

The Agility project investigated the effects of running applications designed to operate in a UNIX environment in a test environment similar to the one envisioned for Quorum. In this test environment, a global mandatory security policy governs each application's access to resources. Applications must expect revisions to this mandatory security policy during their runtime revisions which revoke their access to previously acquired resources, reinstate their access to previously revoked resources, or require them to modify their handling of requests already in progress. The Agility project found that most application designed to operate in a UNIX environment did not expect these revisions, and failed as a result.

In the test environment, the Agility project observed two broad classes of failures: 1) those involving a failure to cope with the loss or reinstatement of a revoked resource; and 2) those involving a failure to apply new security policy rules to service sessions already in progress. The Agility project demonstrated that these failures could be avoided, or at least compensated for, by retrofitting application with additional mechanisms, and that it was possible to construct software toolkits to make this retrofitting easier. The Agility project produced a series of two toolkits: the first provided a number of fully-implemented generic mechanisms intended to be suitable for all applications; the second provided these generic mechanisms plus the general infrastructure required to build new mechanisms tailored to specific applications. The second toolkit was also designed to be fully effective on operating systems without support for DTE.

Both toolkits demonstrated practical value. The ready-made solutions provided by the first toolkit provided quick fixes for the failures exhibited by many applications. However, not unsurprisingly, the solutions were sufficiently generic to handle all imaginable applications in all imaginable situations. This deficiency was addressed by the infrastructure for building application-specific mechanisms provided by the second toolkit.

The results of the Agility project show that it is possible to adapt applications designed to operate in a UNIX environment to operation in a Quorum-like environment. Furthermore, based on the evidence of the applications we have retrofitted and demonstrated in our test environment, we assert that the use of the Security Agility Toolkit makes adapting UNIX applications easier than porting or reimplimenting them, at least for applications as complex as BSD/OS *syslogd* or Apache *httpd*.

# References

[1] L. Badger, D. Sterne, D. Sherman, and K. Walker. "A Domain and Type Enforcement UNIX Prototype," In *USENIX Computing Systems* Vol. 9, No. 1, Winter 1996.

[2] K. Oostendorp, T. Fraser, M. Petkac, J. Grillo, B. Uecker, and L. Badger. *Security Agility for Dynamic Execution Environments Initial Prototype Evaluation Report*. Technical Report 0741, TIS Labs, June 1998.

[3] M. Petkac, E. Cantori, W. Morrison, L. Badger, *Security Agility for Dynamic Execution Environments Security Agility Toolkit Evaluation Report*. Technical Report 0765, NAI Labs, June 1999.

[4] M. Petkac and L. Badger, *Security Agility for Dynamic Execution Environments Distributed Security Agility Toolkit Evaluation Report*. Technical Report 00-018, NAI Labs, September 2000.

[5] T. Fraser, L. Badger, and M. Feldman. "Hardening COTS Software with Generic Software Wrappers." In Proceedings of the 1999 IEEE Symposium of Security and Privacy, Oakland, CA, May 1999, p. 2.

[6] Karen A. Oostendorp, Lee Badger, Christopher D. Vance, Wayne G. Morrison, Michael J. Petkac, David L. Sherman, and Daniel F. Sterne. "Domain and Type Enforcement Firewalls*,"* In *Proceedings of the 13th Computer Security Applications Conference*, San Diego, California, December 1997.

[7] M. Petkac and L. Badger. "Security Agility in Response to Intrusion Detection", To appear in *Proceedings of the 16th Annual Computer Security Applications Conference*, New Orleans, Louisiana, December 2000.