

A Specialization Toolkit to Increase the Diversity in Operating Systems

Calton Pu,¹ Andrew Black, Crispin Cowan, Jonathan Walpole

Dept. of Computer Science and Engineering
Oregon Graduate Institute
P.O. BOX 91000, Portland, OR 97291-1000, USA
Phone: +1-503-690-1214, FAX: +1-503-690-1553
email: calton@cse.ogi.edu

Charles Consel

Dept. of Computer Science
University of Rennes/IRISA
Rennes, France
email: consel@irisa.fr

Abstract

Virus and worm attacks that exploit system implementation details can be countered with a diversified set of implementations. Furthermore, immune systems show that attacks from previously unknown organisms require effective dynamic response. In the Synthetix project, we have been developing a specialization toolkit to improve the performance of operating system kernels. The toolkit helps programmers generate and manage diverse specialized implementations of software modules. The Tempo-C specializer tool generates different versions for both compile-time and run-time specialization. We are now adapting the toolkit to improve operating system survivability against implementations attacks.

Index terms: specialization, information survivability, operating systems, software diversity.

¹If the paper is accepted, Calton Pu will give the presentation at the workshop.

¹Effort sponsored by the Defense Advanced Research Projects Agency and Rome Laboratory, Air Force Material Command, USAF, under agreement number F30602-96-1-0331. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

¹The views and conclusions contained in this document are those of the authors and should not be

1 Motivation and Significance

The construction of the National Information Infrastructure in the United States and Global Information Infrastructure (GII) in the world is moving forward. With the recent terrorist attacks such as the Oklahoma City bombing, the regional wars around the world, and still unresolved incidents such as the TWA flight 800, the concern for the integrity of the GII is shared by the governments, businesses, and the general public. The potential threat of information warfare and information terrorism is real and our preparation insufficient, as demonstrated several years ago by the Morris Internet Worm, which was fortunately harmless other than a massive denial of service. The United States government has recognized the importance of this problem with a recent Broad Agency Announcement [11] by the Defense Advanced Research Projects Agency (DARPA), calling for proposals to perform research specifically aimed at creating the technology for an area called *Information Survivability*, to improve the resilience of GII against attacks.

A significant portion of the information survivability program is inspired by biological models in general and immunity models in particular. Although steady progress is being made in the information security area, traditional security techniques are focused on keeping intruders out, with little help after the penetration has happened. Since most of the network researchers agree on the inevitability of penetration and initial damage, new techniques must be developed to react to new threats quickly and effectively. Of the ideas presented so far, immunity-based models seem to offer the most promising approach to increase GII system resiliency against attacks. Concretely, we focus on implementation attacks, where the attacker (e.g., virus or worm) exploits specific representation of code or data in the system. We call these attacks *security faults*.

An immunity-based approach to improve the survivability of the system against security faults is to increase the variety in the representation of components in the system. The general approach starts with detailed modeling of each kind of attack such as viruses and worms, followed by the development of monitoring techniques, and then response and recovery strategies against the attacks. Each of the steps is a significant research challenge by itself. Probably the modeling and monitoring steps benefit the most from previous work on security. However, dynamic and effective response and recovery methods to counteract network attacks represent areas in need of research. In particular, we need fast, flexible, and low overhead mechanisms to support the incorporation and activation of a variety of response strategies against attacks.

In the Synthetix project [3, 12, 5], we have been developing a toolkit for systematic *specialization* of operating system (OS) kernels. The first purpose of the specialization toolkit

interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency, Rome Laboratory, or the U.S. Government.

is to help kernel programmers to improve system performance, but it is now being extended to support survivability needs. Our focus on dynamic adaptive mechanisms complements other research projects on modeling, monitoring, and response strategies. Synthetix specialization of OS kernels uses an explicit specification of *invariants* that allow dynamic generation of code at run-time to improve OS performance while preserving modularity. All the specialized cases preserve the same OS kernel functional interface semantics. Reusing these performance-oriented invariants and introducing *artificial* invariants where necessary, we can generate OS modules as complex and varied as we specify them to be, while maintaining the OS functionality.

Using the Synthetix toolkit has several advantages. First, explicit specification of invariants allows the system to *guard* their validity statically at compile time and dynamically at run-time. The invariants also facilitate the integration of available program verification and theorem proving technology. Furthermore, the variations introduced by artificial invariants are easy to verify since they are completely local. Second, specialization tools handle invariants as general predicates, thus combining invariants referring to performance and security fault resistance uniformly. Thus we can add survivability without sacrificing performance entirely. Third, specialization is inherently adaptive due to run-time code generation. Therefore, during the peace time, the system can run at high efficiency (with relatively low variety), and switch to high variety when under attack. The ability to adapt continuously is essential when facing attackers with adaptive or learning capability.

2 Project Overview

Information survivability, particularly of large software systems, depends on their ability to tolerate security faults. With an immunity-based approach, our first goal is to increase the inherent variety of OS code, to create resilience against malicious attacks (e.g., viruses and worms), and to contain the damage after the attacks have partially succeeded. Our technique, based on specialization [4, 6, 7, 13, 14] is particularly useful due to its ability to cooperate with verification techniques through explicit definitions of quasi-invariants, and to cooperate with security wrappers through meta-interfaces.

Concretely, we are concerned with operating system survivability under two kinds of attacks. The first kind of attack is machine code dependent, for example, a virus or worm taking advantage of raw binary representation of programs and data, either in memory or on disk. We call this kind of attacks *hardwired*, since they only work for the exact intended configuration. The second kind of attack is a more subtle one, from programs that execute legal kernel calls but somehow performing functions outside the original intentions. For example, the worm program written by R.T. Morris, Jr. [8] uses the debugging feature of sendmail. We call the second kind of attack *contextual*, since they take advantage of

legitimate functionality outside of their intended context. When an attack succeeds, we call the penetration a *security fault* in analogy to other kinds of faults in the system.

Although we have been developing specialization primarily for performance, it is applicable to other important system properties such as reliability. There are no restrictions on the specification of quasi-invariants about their purpose. For example, quasi-invariants make explicit the assumptions made by the specialized code, thus facilitating code verification. In applying specialization to improve operating system survivability, we plan to codify security-related quasi-invariants and create mechanisms to guard these security faults. The key advantage of our approach is that the same tools developed for specialization can be and should be used towards the preservation and enhancement of all important system properties such as performance, reliability, and survivability.

On the systems building side, we plan to extend the ongoing research on *specialization* in the Synthetix project to respond to security faults. Although specialization was originally intended for simplifying execution paths for performance gains, it is relatively straightforward to extend the technique to complicate execution paths for survivability gains. Against hardwired attacks, for example, we plan to extend the specialization toolkit (see Section 3) to introduce detours in the execution path during the specialization process. These detours are harmless regarding correct functionality of operating system code (other than slightly longer response times), but they dynamically change the executable representation in fundamental ways, thus stopping attacks hardwired against specific executables. Against contextual attacks, we plan to extend the specialization toolkit to support easy additions and reductions in functionality of specialized code. Attempts to use missing functionality are caught by the toolkit and analyzed. If a call is considered legitimate and accepted, it is added dynamically to the system and completed. Otherwise, the execution is rejected and the attack rebuffed.

In contrast to traditional security and fault tolerance work, our approach is intended for both tolerance of and *resistance* to security faults. Fault tolerance usually means building redundancy into the system to handle situations where predictable faults occur. Our goal is to combine specialization with system monitoring research and provide dynamic adaptation to resist detected attacks. By introducing randomization into dynamic adaptation, we hope to increase operating system survivability by increasing the resistance to security faults as they occur.

3 Specialization Toolkit for Survivability

The key idea of specialization is to take advantage of certain assumptions (invariants and quasi-invariants) in the system to simplify the execution path. For example, in a Unix `read` kernel call, it is not necessary to lock the file and some system data structures (e.g., the

`inode`) if the file is under exclusive access by one process. Specialized code is a clear win if the assumptions hold for a significant amount of time. However, occasionally the quasi-invariants are invalidated; in the Unix file example, when another process opens the same file, a *guard* should detect the violation of exclusive access quasi-invariant and replace the specialized code with a generic kernel call that contains file access synchronization code.

To support methodical specialization, we have been developing a toolkit that helps the kernel programmer in the tedious but important task of making specialized modules correct. For example, we have developed a specializer (called Tempo-C) [2, 1] for dynamic partial evaluation of C code, to simplify program optimization. We are also implementing a guard checker, a program that detects all the locations where quasi-invariants may be invalidated (assuming that the kernel code does not contain arbitrary pointer arithmetic, which may touch any memory location at run-time). Even though the toolkit has been designed for applying specialization to improve performance and modularity, it supports the generic replacement of modules under circumstances that can be verified to be correct. That is, a module is replaced if and only if the quasi-invariants of the replacement are verified to hold.

The toolkit design is based on the concept of *specialization class*, which represents all the specialized versions of a module. Specialization classes contain the invariants, the guards for quasi-invariants, and actions to be taken when guards are triggered by the invalidation of a quasi-invariant. Specialization tools operate both on specialization classes (using type information) and on specialization objects (using instance information).

The variety of specialization objects is a direct function of the number of invariants its specialization class contains. Seldom used code, for example, may not have been specialized since there is little performance gain. For all modules compiled under Tempo-C, however, we can complicate its physical representation by adding artificial invariants. By artificial invariants we mean predicates that do not alter the logical execution path under the normal interface, but that change the physical layout of data structures and code location. In other words, we can safely introduce some no-ops as part of the specialization process and the program will still work.

Traditionally, program verification is done statically, before the program is executed. Since specialization involves run-time code generation, verification of specialized programs is a serious research question. Our idea is to verify entire specialization classes, which contain all the instances. If we succeed in convincingly verifying specialization classes and our toolkit, then code generated dynamically (in a disciplined way) may become more trusted than before, with its attendant advantages. Since specialized code is created and used only when its invariants are guaranteed to be true, we believe the verification of specialization classes is feasible. In addition, type systems have been used in program verification and we intend to build on that line of work.

In Synthetix we have extended partial evaluation by applying to a dynamic environment. Instead of partially evaluating programs only on immutable invariants, we also apply partial evaluation using quasi-invariants. The correctness of thus specialized programs is maintained by guards that watch over the invalidation of quasi-invariants. The key idea is to separate the events that may affect program correctness from the optimization of programs. Analogously, we plan to separate the events that trigger dynamic code generation from the verification of programs themselves. This way, we expect to extend static verification of programs to the verification of programs generated at run-time, by watching over the quasi-invariants.

Another technique we have used successfully in both Synthetix and extended transaction research is Open Implementation [10, 9], which advocates the separation of a module's functional interface from its *meta-interface*, which contains attributes and implementation details that affect the other properties of the module, such as performance and fault tolerance. We plan to apply it generally in the interface design of Synthetix research, for example, separating out the security interface and specification of survivability requirements into a security fault tolerance microlanguage. This approach allows us to preserve legacy functional interfaces while developing our own meta-interface to communicate with the newly specialized portion (the wrapper) of the system.

An integral part of our research is an experimental evaluation to test and demonstrate the effectiveness of the extended specialization toolkit in a significant operating system component. For example, we have demonstrated the performance gains of specialization in the HP-UX Unix File System [13]. Suppose we choose file system code as a test vehicle, the extended specialization toolkit is expected to increase the survivability of the component. Verification, analysis, and evaluation are discussed in sections below.

4 Dynamic Adaptation

By extending the specialization toolkit we are creating a mechanism to increase the variety of OS kernel code. The policy of when and how to control the code variability is an equally important question. Our research is not intended to settle the policy question. Instead, we plan to build the support for several practical policies, including operator control, independently randomized (each component of the system chooses a degree of variety randomly), and feedback-based (higher variety when intrusion is detected). For example, to survive an attacker with operator privileges and learning ability would require at least a combination of feedback and randomized strategies.

As long as specialized modules maintain reasonable performance, higher variety is a safe choice. In addition, we can define a notion of distance between two variants in terms of how different their physical representations are. By introducing a larger number of artificial

invariants, we can make specialized modules that are further apart. Assuming that modules that are more distant from each other will be less likely to succumb to the same attack, and also assuming that we can propagate dynamic specialization faster than the spread of the attacking virus/worm, then a sufficiently large system using specialization can be made tolerant against increasing levels of security faults by augmenting its variety in quantity and in quality (distance).

As mentioned in the previous section, we plan to adopt existing monitoring tools for security. From specialization point of view, we want to integrate with as many monitoring tools as possible. Therefore, our primary concern is defining a generic interface between dynamic adaptation specialization and monitoring. The generic interface will facilitate the integration of different monitoring tools since we only have to build mediator and translators between the imported tools and the generic interface.

Our research on dynamic adaptation will be driven by a practical demonstration scenario. We plan to build specialized operating system components as part of the experimental evaluation. As monitoring tools detect the attacks, adaptation actions are triggered, and new code is put in place. Typical adaptation action is to replug the current version with another version of equivalent functionality, but different survival properties. We will consider two alternatives here.

The new version may add complexity to make the specialized code more difficult to penetrate. This is the most likely appropriate response to hardwired attacks. Many of the hardwired attacks are tied closely to particular implementations, and changing the implementation will increase system resistance to such security faults. The additional complexity may be another “natural” version with different quasi-invariants, or an “artificial” version with artificially introduced quasi-invariants.

Alternatively, the new version may reduce complexity and simplify the code. This may be an appropriate response to contextual attacks. Many of the contextual attacks depend on using features and facilities that were not intended for a particular situation, thus creating an unforeseen loophole. The sendmail debugging facility is an obvious example. There are many similar features that are convenient to leave in, but that could lead to security weaknesses. During the “peace time”, there is no harm in leaving these features in. It may be important to take them out quickly during “war time”. This is particularly the case for the millions of computers that will not have system administrators to recompile programs or to turn obscure system features on and off.

So far we have considered only conventional attacks that have limited or no adaptation ability. Once the specialization toolkit becomes available, we must consider the possibility of its use in the construction of attacking vehicles such as virus and worm programs. In the biological world, we already know that adaptive microorganisms such as malaria and AIDS

virus are among the most difficult to control due to their fast mutation rates. Analogously, current technology to identify virus signatures may need to be improved to take into account specialization techniques used by the attackers.

Although our primary goal is to develop adaptive countermeasures against hardwired and contextual attacks, we must also investigate the possibility of applying the specialization techniques to reduce their own effectiveness as a countermeasure. There are obvious approaches to preserve the integrity and identifiability of *our* specialization toolkit with encrypted signatures, for example. But for the long term, we will also investigate the inherent limitations (and potential) of specialization as applied to attacking software on the one side and operating system survivability on the other side.

5 Experimental Evaluation

An integral part of this research is to validate our results, both the tools and the demonstration systems, since claiming survivability without assurance has little meaning. The subcontract address the issue of verification directly, both in the static verification of specialized code and in the development of verification tools for the dynamic assurance of run-time generated code. However, the variety of foreseeable attacks greatly exceed our resources for complete verification. We will use experimental observation, evaluation, and statistical techniques to help us validate the additional survivability provided by our techniques and tools.

We plan to test our tools in a systematic way in a limited environment. Instead of simultaneous variety in space, we plan to create variety over a period of time. After generating a variant, we subject it to attack and verify the integrity of the system, and then move on to the next variant. With statistical sampling techniques we may be able to establish confidence in our experimental evaluation of the system within a reasonable budget and time schedule.

We plan to evaluate our system systematically. The first step is to use the Synthetix methodology to define the quasi-invariants and then use the toolkit to create the specialized code and guards. The second step is to verify statically the specialized code that it does what it is supposed to do. The third step is to design simulated attack scenarios and implement attack generators to attempt penetration. The fourth step is to observe the responses of the specialized code, evaluate the effectiveness of the attack and defense, and use this information to improve our technique and tools.

For concreteness, let us assume that we use the extended specialization toolkit to increase the survivability of the Unix File System component of an operating system. We first specify the security and survivability quasi-invariants and use the extended toolkit to create

specialized code and write the guards. After the code and guards have been statically verified, we design simulated attacks. We may make simplifying assumptions to test specific aspects of security fault resistance. For example, we may assume the attacker has obtained root privileges and is attacking the file system from that vantage point. The interesting part of the experiment is to observe the responses of specialized code under the attacks.

For static specialization, our evaluation consists of generating different attack scenarios and running them through the many different flavors of specialized code. The task is to tally the statistics on the survivability of the variants of specialized code under the different attack scenarios. Our hypothesis is that as the complexity and variety of specialized code increases, an increasing percentage of specialized code should survive (be immune to) a specific attack.

One step further is the evaluation of dynamic adaptation. After combining a monitoring system with the static specialization evaluation scenario, we add the dynamic adaptation part of our toolkit. The experiment consists of attack detection (either through a real monitoring system or simulated monitoring) and starting the dynamic adaptation. As the attacks intensify, our response is to increase the variety of specialized code through runtime code generation and replugging. The result is an evaluation of the effectiveness of the dynamic adaptation mechanism in the increase of variety and the observed survival rate in response to specific attacks. Note that the variety of code can be increased by both increasing and decreasing the complexity of specialized code.

After we have evaluated survivability independently, the next goal is to show that survivability and security can be combined with good performance. The experiment is to study the performance and survivability rates of the variety of specialized code and observe the different trade-offs. Statically, we expect the survivability to increase with complexity and therefore decreased performance. The additional complexity and survival overhead, however, may still be competitive as compared to a generic algorithm in a normal implementation. The interesting part of this study is the evaluation of dynamic adaptation. We expect the strategy to decrease code complexity (if applicable) to improve both the survivability and performance of the system.

The extended specialization toolkit obviously builds on existing Synthetix efforts. Although the Synthetix tools have been designed for performance and modularity, we believe their extension to tolerate security faults can be done in a systematic way. There are several parts of current tools that will help security, including dependency analysis, predicate verification, as well as pointer analysis and aliasing. The main new research question is the addition of a new dimension to specialization: security fault resistance in combination performance. Our challenge is to show that the specialization idea can accommodate and harmonize multiple dimensions and the extended toolkit can support both and resolve the trade-offs between them.

6 Conclusion

We have described our plans for increasing operating system (OS) resistance against security faults, e.g., virus and worm attacks. Our work builds on the existing Synthetix project [3, 12, 5] research on specialization. The basic idea is to increase the representational variety of OS components in a systematic way, by using the Synthetix specialization toolkit. The same way biological systems need variety to resist bacterial and virus attacks, we see the need to increase the variety of software systems, even if all varieties provide the same functions.

Our research for increasing the variety of software systems goes beyond building a single prototype to experiment with the usefulness of the idea. We are building a concrete toolkit to support the systematic construction of specialized components in the OS, which will provide OS designers many choices, for example, in the trade-off between higher resistance against security faults and performance. The toolkit supports both compile-time specialization and run-time specialization, allowing the specialized OS to adapt dynamically to resist sudden attacks.

This research complements other projects working on immunity-based approaches to increase the variety of components and systems. Our toolkit is a mechanism that facilitates the construction of software systems with static and dynamic variation in representation while supporting the same functionality. When used in combination with techniques to monitor attacks and to respond to attacks, we hope to increase the survivability of software systems in the GII of the near future.

References

- [1] C. Consel, L. Hornof, F. Noel, and E.N. Volanschi. A uniform approach to compile-time and run-time specialization. In *Proceedings of the 1996 Dagstuhl Workshop on Partial Evaluation*, Germany, February 1996.
- [2] C. Consel and F. Noel. A general approach for run-time specialization and its application to *c*. In *Proceedings of the 1996 ACM Symposium on Principles of Programming Languages*, Florida, January 1996.
- [3] C. Consel, C. Pu, and J. Walpole. Incremental specialization: The key to high performance, modularity and portability in operating systems. In *Proceedings of ACM Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, Copenhagen, June 1993.
- [4] Charles Consel, Calton Pu, and Jonathan Walpole. Incremental Specialization: The Key to High Performance, Modularity and Portability in Operating Systems. In *Proceedings*

- of ACM Symposium on Partial Evaluation and Semantics-Based Program Manipulation (PEPM'93)*, Copenhagen, Denmark, June 1993.
- [5] C. Cowan, T. Autrey, C. Krasic, C. Pu, and J. Walpole. Fast concurrent dynamic linking for an adaptive operating system. In *Proceedings of the International Conference on Configurable Distributed Systems*, Maryland, May 1996.
 - [6] Crispin Cowan, Tito Autrey, Charles Krasic, Calton Pu, and Jonathan Walpole. Fast Concurrent Dynamic Linking for an Adaptive Operating System. In *International Conference on Configurable Distributed Systems (ICCDs'96)*, Annapolis, MD, May 1996.
 - [7] Crispin Cowan, Calton Pu, and Jonathan Walpole. Specialization Objects: A Reflective Interface for Specialization. Report CSE-95-024, Dept. of Computer Science and Engineering, Oregon Graduate Institute, Portland, OR, December 1995.
 - [8] Mark W. Eichin and Jon A. Rochlis. With Microscope and Tweezers: An Analysis of the Internet Virus of November 1988. In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, September 1990.
 - [9] Gregor Kiczales. Towards a new model of abstraction in software engineering. In *Proc. of the IMSA'92 Workshop on Reflection and Meta-level Architectures*, 1992. See <http://www.xerox.com/PARC/spl/eca/oi.html> for updates.
 - [10] Gregor Kiczales, Jim des Rivières, and Daniel G. Bobrow. *The Art of the Metaobject Protocol*. MIT Press, 1991.
 - [11] DARPA Program Managers. Survivability of large scale information systems. *Commerce Business Daily*, DARPA BAA(96-40), August 28th 1996. Technical Areas Cited in BAA: Public Health and Immune Systems; Adaptive Architectures for Survivable Systems; Techniques For Creating Variability and Diversity.
 - [12] C. Pu, T. Autrey, A. Black, C. Consel, C. Cowan, J. Inouye, L. Kethana, J. Walpole, and K. Zhang. Optimistic incremental specialization: Streamlining a commercial operating system. In *Proceedings of the Fifteenth Symposium on Operating Systems Principles*, Colorado, December 1995.
 - [13] Calton Pu, Tito Autrey, Andrew Black, Charles Consel, Crispin Cowan, Jon Inouye, Lakshmi Kethana, Jonathan Walpole, and Ke Zhang. Optimistic Incremental Specialization: Streamlining a Commercial Operating System. In *Symposium on Operating Systems Principles (SOSP)*, Copper Mountain, Colorado, December 1995.
 - [14] Jonathan Walpole, Crispin Cowan, Andrew Black, Jon Inouye, Calton Pu, and Shanwei Cen. Customizable Operating Systems. Report CSE-95-023, Dept. of Computer Science and Engineering, Oregon Graduate Institute, Portland, OR, December 1995. Submitted for review.