REPORT DOCUMENTATION PAGE

AFRL-SR-AR-TR-06-0049

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE February 26, 2006	3. REPORT TYPE Final Report July 1, 2003 - Nov 30, 2005

4. TITLE AND SUBTITLE Trust in Security-Policy Enforcement Mechanisms Final Report	5. FUNDING NUMBERS F49620-03-1-0156
6. AUTHOR(S) Schneider, Fred B. Morrisett, Greg	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Computer Science Department Cornell University Ithaca, NY 14853	8. PERFORMING ORGANIZATION REPORT NUMBER 42769
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFOSR Suite 325, Room 3112 875 Randolph Street Arlington, VA 22203-1768 NM	10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for Public Release; distribution is Unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 Words)*

This project investigated language-based approaches for enforcing security policies and proactive approaches for implementing trustworthy distributed services.

One avenue of language-based work produced Cyclone, a type-safe variant of C. The Cyclone language retains the familiar syntax and semantics of C code, but provides the strong security guarantees of modern languages such as Java.

A second avenue of language-based work explored a general class of policy enforcement mechanism based on in-line reference monitors (IRM), which insert checks and actions in an application to ensure the resulting code will respect the policy when executed. The class of policies enforceable through IRMs was shown not to correspond to any class of the Kleene hierarchy. In addition, a certified IRM rewriter framework was developed for Microsoft .NET code. It produces explicit evidence so an independent proof checker can determine that rewritten code respects a desired security policy.

Finally, proactive obfuscation was investigated as a basis for achieving independence of server replicas comprising a service. This resulted in new agreement protocols to handle servers that periodically have their storage purged and reloaded (to eliminate undetectably compromised code and data). It also produced a semantic characterization of how obfuscation compares with strong typing, finding that the two are comparable, a surprising result.

14. SUBJECT TERMS Security, trustworthy systems, language-based security, inlined reference monitor, certifying compilation			15. NUMBER OF PAGES 11 / 16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT Unlimited

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

Trust in Security-Policy Enforcement Mechanisms

AFOSR Grant F49620-03-1-0156

Final Report

1 July 2003 – 30 November 2005

Fred B. Schneider
Computer Science Department
Cornell University
Ithaca, New York
(607) 255-9221 (phone)
fbs@cs.cornell.edu

Greg Morrisett
Computer Science Department
Harvard University
Cambridge, Massachusetts
(617) 495-9526 (phone)
greg@eecs.harvard.edu

Objectives

A key part of building trustworthy software systems is having some basis to believe that the system will behave as expected in the presence of attacks and/or failures, as well as design and programming errors. And any basis for having trust in a computing system will ultimately be grounded in human comprehension, for only if we can understand and predict how an artifact behaves can we trust it. Humans are limited in what they can understand, though. So research to facilitate creating trustworthy systems that are large and complex must somehow provide a means to amplify trust in artifacts that humans can understand. We need the means to leverage components we do trust and the means to transfer trust from one part of a system to another.

The objective of this project was to investigate two particularly promising avenues for trust amplification,. Both avenues built on unexplained but nevertheless empirically observed asymmetries—one between proof generation and proof checking, the other between public key encryption and public key decryption:

- Taking language-based security the next step, by further developing the Cyclone type-safe variant of C and by marrying certifying compilation with program analyzers and IRM rewriters as a means of making language-based enforcement technologies more trustworthy.

- Combining proactive secret-sharing and threshold cryptography in connection with quorums for replication-management as a means of implementing trustworthy services in settings satisfying only weak assumptions.

Accomplishments

Cyclone. Implementation errors, such as buffer overruns, memory leaks, and integer overflows account for an alarming number of successful attacks. To address these concerns, the project developed a type-safe variant of C called Cyclone. The Cyclone language retains the familiar syntax and semantics of C code, so that legacy programs can be easily ported to an environment with the same strong security guarantees of modern languages such as Java.

The original version of Cyclone used a combination of static type-checking and run-time tests to detect and prevent implementation errors. To minimize run-time overhead—and to decrease the probability of run-time failure—this project extended the compiler with support for *extended static checking* (ESC).

The ESC component of the Cyclone compiler was first used to eliminate run-time checks that could not be eliminated through the static type system, including checks on array subscripts, pointer arithmetic, etc. This was accomplished by first constructing a *verification condition*—a logical assertion whose validity ensures that the run-time check will never fail. The verification conditions for each check were then fed to a custom theorem prover. When the theorem prover could successfully prove the verification condition, the corresponding check was eliminated. When the prover could not discharge the verification condition, a warning was issued to the programmer.

The primary challenges in this effort were (i) making the verification condition generation scalable and precise and (ii) finding fast but sound heuristics for the theorem prover. For instance, previously proposed algorithms could generate verification conditions exponentially larger than the original source code. The project developed a new algorithm that ensures the conditions are quadratic in the size of the source in the worst case and linear in practice.

Once these hurdles were overcome, the compiler was able to automatically eliminate 96% of the run-time checks in the roughly 80,000 lines of Cyclone code that make up the standard libraries and compiler. Furthermore, the additional analyses and transformations only added 2% overhead to compile times.

Inlined Reference Monitors. In earlier work, PI Schneider developed a model of enforceable security policies and showed that, in principle, all such policies could be realized using some form of a *reference monitor*. A reference monitor observes system execution and blocks actions that would violate a desired security policy. Today, most reference monitors are implemented by an operating system using hardware-enforced primitives. However, the class of policies that can be directly enforced using hardware mechanisms is severely limited because the machine has no knowledge of application-level semantics, only limited opportunity to intervene, and few provisions for extending policies to cover new situations.

To overcome these limitations, a more general class of enforcement mechanisms based on *inlined reference monitors* (IRMs) was studied under the auspices of this funding. An IRM tool takes a policy and rewrites untrusted code, inserting checks and actions that ensure the resulting code will respect the policy when executed. Inlining the monitor's code allows it to have complete access to the internals of the application, and allows it to tailor the policy to the application and its abstractions. Furthermore, the IRM approach makes it possible to insert corrective actions instead of simply halting the application.

Building on Schneider's earlier work, the project developed a more refined characterization of what policies can be enforced using inlined reference monitors. Specifically, the PIs developed a model based on standard Turing machines, adapted Schneider's criteria for enforceable security policies, and introduced computability requirements. Static analysis and classical reference monitors were also integrated into the model. This allowed comparing the relative power of the various enforcement mechanisms and relating the power of these mechanisms to standard computability results. For instance, it was relatively easy to show that the class of policies precisely supported by static analysis could also be supported by both classical and inlined reference monitors. In addition, introducing a computability requirement on reference monitors was found to be necessary, but not sufficient, for precise characterization of the class of policies they can actually realize. And a new property, called "benevolence" was identified; it provides a more accurate upper bound on the power of reference monitors.

The most surprising and important results obtained concern the general framework of IRMs. The class of policies originally characterized by Schneider was shown not to include all policies enforceable through IRMs (and vice versa). Indeed, the class of policies enforceable through IRMs was shown not to correspond to any class of the Kleene hierarchy. This is a surprising and an important result, because it shows that inlined reference monitors are truly a powerful security enforcement technique.

In addition to a theoretical study of IRMs, a number of practical issues were studied too. For instance, to achieve acceptable performance, an IRM rewriter needs to perform a number of sophisticated optimizations so that checks and IRM state updates are inserted only where needed. Furthermore, an IRM rewriter must be careful to ensure that the integrity of the reference monitor cannot be violated by untrusted code. In practice, this means IRM rewriters are relatively large tools (similar to compilers) that undoubtably contain bugs, and thus should not be trusted.

To address this concern, the project combined the ideas behind proof-carrying code (PCC) with IRM rewriting and developed a *certified* IRM rewriter framework for Microsoft .NET code. In this framework, the rewriter produces explicit evidence that enables an independent proof checker to determine that the rewritten code respects the desired security policy. The proof checker is relatively small compared to rewriters and is therefore likely to be more trustworthy.

The evidence that the rewriter provides to the checker is in the form of extended typing annotations. A model of the .NET intermediate language was constructed along with a proof that, given proper evidence that the program type-checks (under the extended type system), any execution sequence of the code will respect the policy.

Implementing Trustworthy Services. To be *trustworthy*, a service must tolerate failures and attacks. The means to build fault-tolerant services has been at hand for some time—replication of servers on independent hosts. Scaling that up to handle attacks was a third research focus of this grant. In particular, although server failures are often observed to be independent, independence in the presence of attacks is typically not seen. An adversary that disrupts one replica will probably be able to exploit the same vulnerability at other replicas and disrupt them as well—having $2f + 1$ replicas might tolerate up to f faulty hosts, whereas all hosts (hence all replicas) would succumb to a single attack.

Server diversity thus is central to implementing trustworthy services with replicated servers. Such diversity can be introduceed automatically during compilation, loading, or in the run-time environment by using an *obfuscator*, which transforms a *base* program into a *morph* according to some semantics-preserving transformations. Different subsets of these transformations yield different morphs, with many of the transformations themselves being non-deterministic. An *obfuscation key* input to the obfuscator determines the exact transformations used for computing a specific morph, and two different obfuscation keys are likely to produce unpredicatably different morphs of the same base program.

Use of an obfuscation key not known by attackers defends against attacks that work by exploiting detailed knowledge of base server code. But over time, details of a morph's code could become known to attackers, so simply running different morphs on each server is not sufficient. This is just the well known failing of the oft-maligned "security by obscurity". There is, however, a promising defense available—proactive execution of obfuscation, whereby each server periodically selects a fresh (secret) obfuscation key, computes a new morph, and executes that morph for its next window of vulnerability.

Two classes of questions had to be addressed in getting proactive obfuscation to work; that drove the research. First, were questions about protocols to ensure that morphs would be terminated and restarted in a way that resisted compromise by attackers. The needs here were not being addressed by extant agreement protocols, because morphs have their storage purged and reloaded (to eliminate undetectably compromised code and data) at the start of each window of vulnerability. Not only were such *amnesia-insensitive* agreement protocols developed as part of this project, but a firewall that employs proactive obfuscation was designed and prototyped. Having the prototype allowed different protocols to be explored and ensured that protocol design was grounded in the realities of an actual system context.

The second class of problems concerned foundations of the approach:

What classes of attacks would be blunted by proactive obfuscation? Specific correspondences between classes of transformations and classes of attacks they defend against was known at the start of this project. But although knowing this correspondence is important when designing a set of defenses for a given threat model, knowing the specific correspondences is not the same as knowing the overall power of mechanically-generated diversity as a defense. This project explored that latter, broader, issue, by

- giving a semantics for proving results about the defensive power of obfuscation;
- giving a precise characterization of attacks, applicable to viewing diversity as a defense;
- developing the thesis that mechanically-generated diversity is comparable to type systems, and deriving an admittedly unusual type system equivalent to obfuscation in the presence of finitely many keys;
- exhibiting, for a C-like language and non-trivial obfuscator, an increasingly tighter sequence of type systems for soundly approximating obfuscation under arbitrary finite sets of keys. Surprisingly, the more accurate type systems are based on information flow. No type system corresponds exactly to the obfuscator under arbitrary finite sets of keys, and therefore approximations are the best that can be achieved.

Personnel Supported

Faculty: Fred B. Schneider and Greg Morrisett.

Postdoctoral Researchers: Robbert Van Renesse.

Graduate Students: James Cheney, Sigmund Cherem, Matthew Fluet, Kevin Hamlen, Ruijie Wang, and Yanling Wang.

Undergraduate Students: Gregory Roth.

Publications:

1. Daniel J. Grossman. Type-Safe Multithreading in Cyclone. *ACM Workshop on Types in Language Design and Implementation* (New Orleans, LA, January 2003).
2. Yaron Minsky and Fred B. Schneider. Tolerating malicious gossip. *Distributed Computing 16*, 1 (Feb 2003), 49–68.
3. James Cheney and Christian Urban. System description: Alpha-Prolog, a fresh approach to logic programming modulo alpha-equivalence. *Workshop on Unification* (Valencia, Spain, May, 2003).
4. Frederick Smith, Dan Grossman, Greg Morrisett, Luke Hornof, and Trevor Jim. Compiling for template-based run-time code generation. *Journal of Functional Programming*, 13(3):677-708, May 2003.
5. Daniel J. Grossman. *Safe Programming at the C Level of Abstraction.* Ph.D. Thesis, Cornell University, August 2003.
6. Fred B. Schneider. Least Privilege and More. *IEEE Security and Privacy*, Volume 1, Number 3 (September/October 2003), 55–59. Also appears in *Computer Systems: Theory, Technology, and Applications*, (A. Herbert and K. Jones, eds). Springer-Verlag, New York, 253–258.
7. Dag Johansen, Robbert van Renesse, and Fred B. Schneider. WAIF: Web of Asynchronous Information Filters. *Future Directions in Distributed Computing*, Lecture Notes in Computer Science, Volume 2585 (Schiper, Shvartsman, Weatherspoon, and Zhao, eds.) Springer-Verlag, 2003, 81–86.
8. Fred B. Schneider. Least privilege and more. *Computer Systems: Papers for Roger Needham*, Andrew Herbert and Karen Sparck Jones, eds. Microsoft Research, 2003, 209–213.
9. Matthew Fluet and Daniel Wang. Implementation and Performance Evaluation of a Safe Runtime System in Cyclone. *Proceedings of the SPACE 2004 Workshop*, (Venice, Italy, January 2004).
10. Fred B. Schneider. The Next Digital Divide. Editorial. *IEEE Security and Privacy* 2, 1 (January/February 2004), 5.

11. William Josephson, Emin Gun Sirer, and Fred B. Schneider. Peer-to-peer authentication with a distributed single sign-on service. *Proceedings Third International Workshop on Peer-to-Peer Systems* (IPTPS'04, San Diego, CA, February 2004), ACM.

12. James Cheney. The Complexity of Equivariant Unification. *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP 2004)*, (Turku, Finland, July 2004).

13. M. J. Gabbay and J. Cheney. A Proof Theory for Nominal Logic. *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science (LICS 2004)*, (Turku, Finland, July 2004), 139–148.

14. James Cheney. Nominal Logic Programming. Ph.D. Thesis, Cornell University (August 2004).

15. Matthew Fluet and Greg Morrisett. Monadic regions. In *Proceedings of the ACM International Conference on Functional Programming* (ICFP'04), (Park City, Utah, September 2004), 103–114.

16. Fred B. Schneider. Time out for station identification. Editorial. *IEEE Security and Privacy 2*, 1 (September/October 2004), 5.

17. Michael Hicks, Greg Morrisett, Dan Grossman, and Trevor Jim. Experience with safe manual memory-management in Cyclone. In *Proceedings of the ACM International Symposium on Memory Management*, (ISMM'04), (Vancouver, British Columbia, October 2004), 73–84.

18. Robbert van Renesse and Fred B. Schneider. Chain replication for supporting high throughput and availability. *Sixth Symposium on Operating Systems Design and Implementation* (OSDI '04), (San Francisco, California, December 2004), USENIX Association, 2004, 91–104.

19. Dan Grossman, Michael Hicks, Trevor Jim, and Greg Morrisett. Cyclone: A type-safe dialect of C. In *C/C++ User's Journal*, 23(1):6–13, January 2005.

20. Greg Morrisett, Matthew Fluet, and Amal Ahmed. L^3: A linear language with locations. *Seventh International Conference on Typed Lambda Calculi and Applications* (TLCA'05), (Nara, Japan, April 2005), 293–307.

21. Scott D. Stoller and Fred B. Schneider. Automated analysis of fault-tolerance in distributed systems. *Formal Methods in System Design 28*, 2 (March 2005), 183–196.

22. Fred B. Schneider. It depends on what you pay. Editorial. *IEEE Security and Privacy 3*, 3 (May/June 2005), 5.

23. Michael R. Clarkson, Andrew C. Myers, and Fred B. Schneider. Belief in Information Flow. *Proceedings 18th IEEE Computer Security Foundations Workshop* (Aix-en-Provence, France, June 20-22, 2005), 31–45.

24. Lidong Zhou and Fred B. Schneider. APSS: Proactive secret sharing in asynchronous systems. *ACM Transactions on Information and System Security 8*, 3 (August 2005), 1–28.

25. Amal Ahmed, Matthew Fluet, and Greg Morrisett. A step-indexed model of substructural state. In *Proceedings of the ACM International Conference on Functional Programming* (ICFP'05), (Tallinn, Estonia, September 2005), 78–91.

26. Lidong Zhou and Fred B. Schneider. Implementing trustworthy services using replicated state machines. *IEEE Security and Privacy*, Volume 3, Number 5 (September/October 2005), 34–43.

27. Matthew Fluet and Greg Morrisett. Monadic regions. *Journal of Functional Programming*, to appear.

Interactions/Transitions

- Schneider is the director of the AFRL/Cornell Information Assurance Institute.

- For 2003, Schneider served as Chief Scientist of the Griffiss Institute, New York State Cybersecurity consortium, and served on the Board of Directors for the following 2 years.

- Schneider served as a member of the following industrial advisory boards: CIGITAL Technical Advisory Board; Fast Search and Transfer Technical Advisory Board; IBM Corporation Autonomic Computing Advisory

Board; Intel Microprocessor Research Lab Advisory Board; Microsoft's Trustworthy Computing Academic Advisory Board; Packet General Networks Technical Advisory Board.

- Schneider served on the following other advisory committees: UK Dependability Interdisciplinary Research Collaboration (DIRC), Steering Committee; ACM Advisory Committee on Security and Privacy (ACSP); National Research Council Computer Science and Telecommunications Board; CSTB study Committee on Improving Cybersecurity in the U.S; NSF/CISE Advisory Committee.

- Schneider served on the editorial boards for: *Distributed Computing*, *Information Processing Letters*, *High Integrity Systems*, *Annals of Software Engineering*, *ACM Computing Surveys*. He is also co-managing Editor for Springer-Verlag's Texts and Monographs in Computer Science and Associate Editor-in-Chief for *IEEE Security and Privacy* magazine.

- Schneider is Chief Scientist of the NSF TRUST Science and Technology Center, which includes U.C. Berkeley, Carnegie-Mellon University, Cornell University, Stanford University, and Vanderbilt University.

- Morrisett serves on Microsoft's Trustworthy Computing Academic Advisory Board and the Fortify Technical Advisory Board.

- Morrisett serves on the editorial boards for: *Journal of Functional Programming* and *ACM Transactions on Programming Languages and Systems.* He also serves on the Advisory Committee for the Semantics, Applications, and Implementations of Program Generation (SAIG) conference; and the ACM Conference on Types in Language Design and Implementation.

- Morrisett served as the program chair for the 2003 ACM Symposium on Principles of Programming.

DoD Interactions and Technology Transitions

- As a consultant to DARPA/IPTO, Schneider chaired the independent evaluation team for the OASIS Dem/Val prototype project. This project funded the design of a JBI system intended to tolerate a class

A Red Team attack for 12 hours. Schneider also served on the independent evaluation team for the DARPA/IPTO Self-Regenerative Systems program.

- Schneider served twice on AFRL search committees for Senior Scientists in Information Assurance Technology.
- Morrisett and Schneider each briefed the Infosec Research Council's "Research Hard Problems" study; Schneider also served as a reviewer for the final resport.
- Morrisett worked on the development of Microsoft's tools for automatically finding security flaws in production code, based on his experience with Cyclone. He and Schneider also worked with student Kevin Hamlen and Microsoft researchers on the implementation of the .NET rewriting tool for inline reference monitors.
- AT&T research worked with Morrisett to develop the Cyclone language, compiler, and tools. In addition, researchers at the University of Maryland, the University of Utah, Princeton, the University of Washington, the University of Pennsylvania, and Cornell are all using Cyclone to develop research prototypes.

Honors and Awards: 7/2003 – 11/2005

F.B. Schneider:

- Doctor of Science (*honoris causa*), University of NewCastle-upon-Tyne (2003).