REPORT DOCUMENTATION PAGE

AFRL-SR-AR-TR-04-

0494

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE
	2-Sept.-2004	Final Report March 2000 – November 2003

4. TITLE AND SUBTITLE

CIPIAF for Information Assurance Institute

5. FUNDING NUMBERS

F49620-01-1-0312

6. AUTHOR(S)

Professor Fred B. Schneider

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Cornell University
4130 Upson Hall
Ithaca, NY 14853

8. PERFORMING ORGANIZATION REPORT NUMBER

39412

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

USAF, AFRL
AF Office of Scientific Research
801 N. Randolph Street, Room 732
Arlington, VA 22203

PIE

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

N/A

11. SUPPLEMENTARY NOTES

N/A

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for Public Release; distribution is Unlimited

12b. DISTRIBUTION CODE

N/A

13. ABSTRACT (Maximum 200 Words)

Three scientists were supported, each for one year, to learn about computer security. The research and educational accomplishments of each are summarized.

20041008 245

14. SUBJECT TERMS

Trustworthy systems, language-based security

15. NUMBER OF PAGES

11

16. PRICE CODE

N/A

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

CIPIAF for Information Assurance Institute

AFOSR Grant F49620-01-1-0312

Final Report

Fred B. Schneider
Computer Science Department
Cornell University
Ithaca, New York

1 Introduction

This CIPIAF award facilitated research efforts at the AFRL/Cornell Information Assurance Institute (IAI) while, at the same time, providing funds for three young researchers to acquire the background needed for doing research in computer system trustworthiness. We supported:

- **Michael W. Hicks**, already educated as a Computer Scientist when he arrived at IAI, used the CIPIAF award to become better educated about computer security. Hicks delayed starting as an assistant professor at University of Maryland by one year to work as a CIPIA fellow at IAI.
- **Michael A. Marsh** is a recent Ph.D. in physics. After taking undergraduate and masters-level courses during his first year at Cornell, Mike spent a year engaged in research on algorithms for implementing trustworthy distributed services. He has since accepted a position at the University of Maryland Institute for Advanced Computer Science.
- **Amal J. Ahmed**, already educated as a Computer Scientist when she started, used the CIPIAF award to become better educated about computer security. She recently accepted a position as a post-doctoral fellow at Harvard University, Division of Engineering and Applied Science, where she will spend the 2004-2005 academic year.

In the sections that follow, we detail the research accomplishments and publications for each of these young scientists.

2 Final Report: Michael W. Hicks

Period of support: 2001–2002

2.1 Description of Research Accomplishments

Cyclone. Cyclone[1] is a programming language based on C that strives to improve software security and reliability by eliminating large classes of vulnerabilities. In particular, the Cyclone language rules out programs that have buffer overflows, dangling pointers, format string attacks, and so on. High-level, type-safe languages, such as Java, Scheme, and ML also provide this assurance, but they lack the ability to control data representations and memory management that is C's hallmark. Furthermore, porting legacy C code to these languages and interfacing with legacy C libraries is a difficult and error-prone process. The goals of Cyclone are 1) to give programmers the same low-level control and performance of C while substantially improving the security and reliability of their code, and 2) to make it easy to port or interface with legacy C code.

Hicks was interested in Cyclone as a systems programming language; i.e. a language used to write operating systems, networking software, etc. To support systems programming, he explored richer interfaces for Cyclone API's that support usage invariants, modular programming, and he explored better support for low-level services, such as memory management and resource control. Hicks also worked with Greg Morrisett to add so-called *unique pointers* to Cyclone to allow `malloc/free`-style memory management while remaining provably safe and secure. Many aspects of this work improve on recent related work, particularly the ability to store unique pointers in non-unique data structures, and to "forget" unique pointers to be handled by a garbage collector.

MediaNet. MediaNet[2] is a network for delivering streaming media. The central technical challenge is one of availability: media streams must be delivered within resource-limited environments while using the network efficiently and meeting the demands of target applications. This project grew out of the DARPA-funded PCES project, which is concerned with delivering UAV reconnaissance data to military sources. In this context, we must also ensure the security of the data being delivered and the infrastructure that delivers it.

[1] Project homepage: `http://www.cs.cornell.edu/projects/cyclone/`.

[2] Project homepage: `http://www.cs.cornell.edu/people/mhicks/medianet.htm`.

MediaNet is constructed as an overlay network of *compute nodes*, which both forward and transform streaming data to meet user preferences. These preferences are described as data-flow graphs of computations, and these computations are scheduled on the compute nodes by an on-line global scheduler. Users can specify alternative, but lower-utility configurations for handling high load. The goal of the global scheduler is to maximize user utility while using the network efficiently under changing loads. Both the use of an on-line global scheduler and the use of user-driven, specification-based quality-of-service are novel contributions.

Hicks did the majority of the work for this project—including design, implementation (in Cyclone, to improve system reliability and security), and performance analysis—working closely with Robbert van Renesse, who first proposed the architecture and implemented a prototype of the global scheduler. Some of the results are summarized in a paper co-authored with van Renesse, where experiments show that MediaNet can deliver improved performance for users and the network relative to non-adaptive or locally-adaptive networks.

Hicks also worked with Mark Linderman of the Air Force's Rome Lab to understand how MediaNet might work as part of the Air Force's Joint Battlespace Infosphere (JBI) infrastructure. Many military applications will require streaming data (whether video/audio, weather data, target movement information, combat status, etc.) to be delivered in a timely and secure manner, under stressed conditions. While more work remains to be done, MediaNet should be able to deliver such data flexibly, and it maps well to the JBI architecture in many ways; for example, in-network computations resemble the JBI concept of a *fuselet*.

Dynamic Software Updatings. Hicks' doctoral dissertation focused on the problem of upgrading running software for a variety of reasons, such as to fix bugs, add new features, add monitoring support, etc. This ability is important for systems that provide non-stop service, such as financial transaction processing systems, air traffic control systems, network routers, etc. At the same time, the updating process should be robust and secure: access-control and automatic support must ensure that the updating mechanism cannot be used to subvert an updateable system.

Prior to arriving at IAI, Hicks had not performed any formal analysis to prove strong assurance properties for software updates. But once at Cornell, with Gavin Bierman, Peter Sewell, and Gareth Stoyle of the Cambridge University Computer Laboratory, he developed a formal system for modeling

dynamically updateable programs. A number of formal mechanisms were considered:

- As a foundation, they developed a calculus for *late binding*. This system is observationally equivalent to a call-by-value lambda calculus, but delays substituting a term for a variable until the "last possible moment." By being less eager, updates to higher-order functions can be captured in a sensible and elegant manner, which goes beyond prior approaches. Late binding is also a useful foundation for mobile code.
- Rather than make every bound variable potentially updateable, they designed a calculus in which the user designates whether a variable access should be fixed to a statically-known binding or whether it should look for the newest (dynamic) version. This calculus generalizes prior work in that it can, in a single calculus, express a variety of updating strategies proposed in the literature for different systems.
- Abstract types are useful in updateable programs: abstract values can be changed at any time as long as the abstract type implementation is also changed (and is not currently operating on the concrete representation). They generalized this idea to apply to named types that are not necessarily abstract by using a mechanism similar to existential types. This technique ought to provide both better assurance of correctness, and greater flexibility in expressing updates.

2.2 Educational Experiences

Hicks attended four Cornell Computer Science Graduate classes:

- CS 514: Distributed Systems (Fall 2001)
- CS 611: Programming Language Semantics (Fall 2001)
- CS 711: Special Topics in PL: Language-based security (Spring 2002)
- CS 513: Computer Security (Spring 2002)

He also lectured twice for Greg Morrisett's CS 314 Fall class in computer architecture, and regularly attended the Systems Lunch.

Publications

[1] Dan Grossman, Greg Morrisett, Trevor Jim, Michael Hicks, Yanling Wang, and James Cheney. Region-based memory management in Cyclone. *Proceedings of the ACM Conference on Programming Language Design and Implementation (PLDI)* (ACM, June 2002), 282–293.

[2] Michael Hicks, Angelos D. Keromytis, and Jonathan M. Smith. A secure PLAN. *IEEE Transactions on Systems, Man, and Cybernetics, Part C, 33* (3), Special Issue on Technologies Promoting Computational Intelligence, Openness and Programmability in Networks and Internet Services, Part I, August 2003.

[3] Michael Hicks, Jonathan T. Moore, and Scott Nettles. Compiling PLAN to SNAP. *Proceedings of the Third International Working Conference on Active Networks (IWAN)* Volume 2207 of Lecture Notes in Computer Science, Springer-Verlag (October 2001), 134–151.

[4] Michael Hicks, Jonathan T. Moore, David Wetherall, and Scott Nettles. Experiences with capsule-based active networking. *Proceedings of the DARPA Active Networks Conference and Exposition (DANCE)* (IEEE, May 2002), 16–24.

[5] Michael Hicks, Greg Morrisett, Dan Grossman, and Trevor Jim. Experience with safe manual memory management in Cyclone. *Proceedings of the International Symposium on Memory Management (ISMM)* October 2004.

[6] Trevor Jim, Greg Morrisett, Dan Grossman, Michael Hicks, James Cheney, and Yanling Wang. Cyclone: A safe dialect of C. *Proceedings of the USENIX Annual Technical Conference* (USENIX, June 2002), 275–288.

[7] Michael Hicks, Adithya Nagarajan, and Robbert van Renesse. User-specified adaptive scheduling in a streaming media network. *Proceedings of the IEEE Conference on Open Architectures (OPENARCH)* (April 2003), 87–96.

3 Final Report: Michael A. Marsh

Period of support: 2001–2003

3.1 Description of Research Accomplishments

Distributed Blinding Protocol. Distributed services often store secrets encrypted with a public key for which the system holds a private key that is split, using a secret sharing scheme, into multiple pieces. If such secrets must be transferred between two distributed services, then a protocol is required for converting a ciphertext encrypted with one public key into a ciphertext for the same plaintext encrypted with another public key. Developing such protocols was one of the subjects of Marsh's study, in collaboration with Lidong Zhou (Microsoft Research, Mountainview CA), Anna Redz (Royal Institute of Technology, Stockholm Sweden), and Fred B. Schneider.

Blind decryption is commonly used when the recipient of the secret is an individual. It involves the receiving individual selecting a random number (the *blinding factor*) which is then entangled with the encrypted secret so that the servers performing the decryption do not learn the secret value. In particular, decryption of a blinded encryption produces a plaintext that appears random but that the recipient can transform (by "dividing out" the blinding factor) into the original secret.

Under the auspices of this postdoc, Marsh and collaborators were able to extend blinding to the case where the recipient is another distributed service that must eventually hold the secret (encrypted with a different public key) and whose servers may not individually learn the secret. The new protocol requires constructing a pair of ciphertexts that encrypt the same random number. The random number must not be learnable by any individual server or small coalition of servers, and the ciphertexts must be provably plaintext-equivalent. The protocol accomplishes this for the ElGamal cryptosystem.

Marsh's specific contributions to this effort were in devising the initial protocol and later making modifications to eliminate particular weaknesses that were later discovered. Marsh also was the one who initially noticed the value of distributed blinding factors.

Cornell Data Exchange (CODEX). As an outgrowth of prior research at Cornell on a secure publish/subscribe system, Marsh determined that commercial off-the-shelf (COTS) systems could be used to disseminate encrypted data provided a secure mechanism existed for distributing the decryption key. This led to the development of CODEX, which is a key management and distribution service that employs distributed trust in order to

provide strong security guarantees about the confidentiality, integrity, and availability of the keys it stores.

CODEX uses Byzantine quorums to achieve availability and uses secret sharing to preserve confidentiality of certain cryptographic keys. Integrity of CODEX responses are guaranteed by using threshold cryptography to digitally sign those responses; a signature is produced only if enough CODEX servers have participated in processing the request. (The assumption is that an adversary is unable to compromise that many CODEX servers.)

CODEX supports a very flexible access control regime, which can be implemented with credential certificates. For example, the creator of a key managed by CODEX can specify the kinds of recipients permitted access to the key without knowing the identities of those recipients. Client keys are stored encrypted with the CODEX service public key, and blind decryption (see above) using the corresponding shared private key is employed to convey a client's key to an authorized recipient in a way that does not expose that key to any CODEX server.

Marsh implemented CODEX and has successfully tested it in both local and Internet environments. The source is approximately 42K lines of new C++ code, including comments. Some CODEX functionality is not yet implemented (notably the full generality of the access control mechanism) but the framework for these features are in place. A full description of CODEX and the experimental results was invited for publication in the inaugural issue of the new journal *IEEE Transactions on Dependable and Secure Computing.*

Byzantine Quorum Systems Software Package. For designs that use Byzantine quorum systems, a system developer must program message broadcasting, secure communications links, and coordination software to manage relations between hosts. Clearly, a software toolbox with this functionality would speed the implementation of new secure distributed system. Marsh thus invested time in designing and programming that toolbox.

Spurred by the need for these tools in constructing CODEX, work in earnest began in December 2002. The toolbox now includes twelve separate libraries. In addition to the general-purpose tools mentioned above, the toolbox also includes cryptosystem abstractions, secret sharing and threshold cryptography tools, and an event infrastructure for implementing single-threaded concurrency.

This set of libraries was successfully used to construct CODEX. While largely an independent effort, some of the implementation was influenced by

Lidong Zhou's COCA implementation.

3.2 Educational Experiences

Marsh attended various graduate-level courses while at Cornell, and he also regularly attended the weekly Systems Lunch seminar.

In Fall 2002, Marsh assisted Fred Schneider in teaching CS 513, System Security, believing that this would be a chance to learn a bit more about what's involved in taking a faculty position. In addition to answering student questions and grading projects, Marsh led the question-and-answer sessions and coordinated the grading policies and workload distribution between the other teaching assistants. Marsh also served as the official liaison between the students and teaching assistants with regard to grades and grading issues. He delivered two course lectures on the topic of secret sharing, the area of his research.

Marsh reviewed papers for the 16th IEEE Computer Security Foundations Workshop (CSFW) and the Workshop on Formal Aspects in Security & Trust (FAST). In addition, he attended the 2003 IEEE Symposium on Security and Privacy, which was held in Berkeley CA from May 11 to May 14.

Publications

[1] Michael A. Marsh, Lidong Zhou, Anna Redz, and Fred B. Schneider. Distributed Blinding for ElGamal and Its Application to Disseminating Secrets. Technical Report TR 2004-1920, Cornell Computer Science Department, January 2004.

[2] Michael A. Marsh and Fred B. Schneider. CODEX: A robust and secure secret distribution system. To appear, *IEEE Transactions on Dependable and Secure Computing 1*, No. 1.

4 Final Report: Amal J. Ahmed

Period of support: 2003–2004

4.1 Description of Research Accomplishments

L^3: A Linear Language with Locations. Advanced type systems have proven effective for enforcing safety and security properties in programming languages such as Java, Typed Assembly Language, and Cyclone. A central shortcoming in the type systems of all of these languages is that types can only capture invariants for mutable objects and not time-varying properties.

Type systems that support *strong updates* allow the type of a mutable object to change whenever the contents of the object is updated. That is, the type of a mutable location can change from program point to program point. Tracking strong updates is extremely useful when writing low-level systems code—for instance, to ensure that device drivers respect certain protocols—or to track security relevant properties in C code. Unfortunately, most typed imperative languages, including Java and ML, do not allow strong updates.

In joint work with Greg Morrisett and Matthew Fluet, Ahmed developed L^3, a linear language that supports strong updates. The essential idea is that for each memory cell there exists a unique (linear) capability which is required in order to access the cell. She and her collaborators also developed extensions needed to model shared (type-invariant) references, as in traditional imperative languages like ML and Java. Here, the capability to access a reference cell is unrestricted, but strong updates are disallowed. The extensions include primitives for *thawing*—regaining the ability to perform strong updates on shared references whose types are *frozen*—and for "re-freezing" a cell once the "frozen" type has been restored.

There has been a great deal of work on adapting some notion of linearity to real programming languages like Java, including ownership types and confinement types. L^3 could provide a convenient foundation for modeling many of these high-level mechanisms.

Dynamic Security Labels and Noninterference. Information flow policies provide the means to express strong security requirements for data confidentiality and integrity. Recent work on security-typed programming languages has shown that information flow can be analyzed statically, ensuring that programs will respect the restrictions placed on data. However, real computing systems have security policies that vary dynamically and that cannot be determined at the time of program analysis. For example, a file has associated access permissions that cannot be known with certainty until

it is opened. Although one security-typed programming language (JFlow and its successor Jif) included support for dynamic security labels, there had been no demonstration that a general mechanism for dynamic labels can securely control information flow.

Upon Ahmed's arrival at Cornell, Andrew Myers and Lantian Zheng were working on an expressive language in which information flow is securely controlled by a dependent type system, yet the security classes of data can vary dynamically. The goal was to show that any well-typed program in this language is provably secure because it satisfies *noninterference*, a strong end-to-end security property. When applied to confidentiality, for instance, noninterference ensures that confidential information cannot be released by the program.

Ahmed contributed to this project by developing a technique to prove noninterference even when the language supports (type-invariant) mutable references and cycles in the memory. The proof technique is based on constructing a model of types as *partial equivalence relations* (PERs) on pairs of expressions and their respective memory states. The model is based on the operational behavior of the language.

Myers and Zheng subsequently used another existing proof technique to establish noninterference for their language. However, Ahmed has recently started working with Greg Morrisett to try to use PER models to establish operational equivalence (and noninterference) in languages that are more expressive than that of Myers and Zheng; that is, languages for which existing proof techniques for noninterference do not suffice.

Foundational Proof-Carrying Code. Proof-carrying code (PCC) is a framework for mechanically verifying the safety of machine language programs. A program that is successfully verified by a PCC system is guaranteed to be safe to execute, but this safety guarantee is contingent upon the correctness of various trusted components, including a large set of low-level typing rules. Foundational PCC systems seek to minimize the size of the trusted computing base. Ahmed's prior work on Foundational PCC explored how to eliminate the need to trust complex, low-level type systems by providing machine-checkable proofs of type soundness for real machine languages.

Region-Based Memory Management and Foundational PCC. The Princeton Foundational PCC system lacks support for the reuse of memory. Informally, to add such support, one must be able to construct machine-

checkable soundness proofs for typed machine languages that permit the types of memory locations to change. Ahmed has been working with Andrew Appel and David Walker on how to add support for *region-based memory management* to the FPCC framework. A language with support for regions allows one to create a new region, allocate memory in some specified region, deallocate an entire region at once, etc. Thus far, they have shown how to construct type soundness proofs for a low-level lambda calculus with region primitives, without breaking the existing model used by the Foundational PCC system. The model of this region-calculus supports type-invariant mutable references in the style of ML and Java, as well as, the reuse of memory locations at different types.

4.2 Educational Experiences

Ahmed attended two graduate-level courses during the past year:

- CS 711: Advanced Programming Languages Seminar: Language-Based Security and Information Flow (Cornell University, Fall 2003)
- CS 255: Topics in Language-Based Security (Harvard University, Spring 2004)

She gave a lecture for Andrew Myers' CS 711 and also gave a guest lecture for Greg Morrisett's CS 255 on Foundational Proof-Carrying Code.

At Cornell, Ahmed regularly attended the weekly Languages and Compilers seminar, the Systems Lunch seminar, and the Programming Languages Discussion Group.

Publications

[1] Amal Ahmed, Matthew Fluet, and Greg Morrisett. $\mathbf{L}^3$: A linear language with locations. Technical Report, July 2004.

[2] Amal Ahmed, Limin Jia, and David Walker. Reasoning about hierarchical storage. *IEEE Symposium on Logic in Computer Science (LICS)* (Ottawa, Canada, June 2003), 33–44.

[3] Andrew C. Myers and Lantian Zheng. Dynamic security labels and noninterference. *Workshop on Formal Aspects in Security and Trust (FAST)*, (Toulouse, France, August 2004.)