AFRL-IF-RS-TR-2001-21
Final Technical Report
March 2001

FOUNDATIONS AND SUPPORT FOR SURVIVABLE SYSTEMS

Cornell University

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. AO E297

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-2001-21 has been reviewed and is approved for publication.

APPROVED: John C. Faust

JOHN FAUST
Project Engineer

FOR THE DIRECTOR:

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

FOUNDATIONS AND SUPPORT FOR
SURVIVABLE SYSTEMS

Fred B. Schneider

Contractor: Cornell University
Contract Number: F30602-96-1-0317
Effective Date of Contract: 1 September 1996
Contract Expiration Date: 31 December 1999
Short Title of Work: Foundations and Support For Survivable Systems
Period of Work Covered: Sep 96 - Dec 99

Principal Investigator: Fred B. Schneider
Phone: (607) 255-7316
AFRL Project Engineer: John C. Faust
Phone: (315) 330-4544

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and was monitored by John C. Faust, AFRL/IFGB, 525 Brooks Road, Rome, NY.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE MARCH 2001	3. REPORT TYPE AND DATES COVERED Final Sep 96 - Dec 99

4. TITLE AND SUBTITLE
FOUNDATIONS AND SUPPORT FOR SURVIVABLE SYSTEMS

5. FUNDING NUMBERS
C - F30602-96-1-0317
PE - 62301E
PR - E017
TA - 01
WU - 04

6. AUTHOR(S)
Fred B. Schneider

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Cornell University
Department of Computer Science
4130 Upson Hall
Ithaca NY 14853

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Project Agency
3701 North Fairfax Drive
Arlington VA 22203-1714

Air Force Research Laboratory/IFGB
525 Brooks Road
Rome NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2001-21

11. SUPPLEMENTARY NOTES
Air Force Research Laboratory Project Engineer: John C. Faust/IFGB/(315) 330-4544

12a. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
Computing systems for managing critical infrastructures must tolerate failures and be resistant to attack. This report presents a summary of accomplishments of a project that has explored techniques for building such survivable critical-infrastructure systems. Mechanisms were developed for ensuring integrity of hosts that execute mobile code and for ensuring fault-tolerance of computations that are structured in terms of mobile code. Automated techniques for analyzing the fault-tolerance of distributed systems were also explored. Finally, a research program into security policy enforcement was initiated, by both characterizing what policies are enforceable and devising new object-code rewriting methods for security policy enforcement. A list of the publications produced by the project appears as the final section of this report.

14. SUBJECT TERMS
Survivable Systems, Agent Integrity, System Fault Tolerance, Enforceable Security Policies, Language-Based Security

15. NUMBER OF PAGES
24

16. PRICE CODE

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

Table of Contents

Final Report:

Foundations and Support for Survivable Systems

F30602–96–1–0317

Principal Investigator: Fred B. Schneider

Department of Computer Science
Cornell University
Ithaca, New York 14853

Summary of Accomplishments

Computing systems for managing critical infrastructures must tolerate failures and be resistant to attack. This project has explored techniques for building such survivable critical-infrastructure systems. Mechanisms were developed for ensuring integrity of hosts that execute mobile code and for ensuring fault-tolerance of computations that are structured in terms of mobile code. We also explored automated techniques for analyzing the fault-tolerance of distributed systems. And, finally, we initiated a research program into security policy enforcement, by both characterizing what policies are enforceable and devising new object-code rewriting methods for security policy enforcement.

A list of the publications produced by the project appears as the final section of this report. Included among those 22 publications are two books—a graduate level monograph on reasoning about concurrent programs and a now widely-cited National Research Council volume on information systems trustworthiness. Also, two patents in the area of fault-tolerance were granted to the principal investigator and his industrial collaborators.

Detailed Description of Technical Progress

Agent Integrity

Agents comprising an application must not only survive (possibly malicious) failures of the hosts they visit, but they must also be resilient to hostile actions by other hosts. Replication and voting enable an application to survive some failures of the hosts it visits. Hosts that are not visited by agents of the application, however, can masquerade and confound a replication scheme. Two classes of protocols to solve these *agent integrity* problems were developed as part of this project [1,9]. One class uses chained cryptographic certificates; the second class uses cryptographic signature-sharing. We were then able to unify these protocols by viewing them in terms of delegation. In each, the principals are sets of hosts (services) and authorization is transferred from one principal to another.

In some settings, hosts being visited by agents cannot be replicated, so the preceding protocols do not apply. This led us to investigate protocols for agent fault-tolerance without host replication.[1] With these *NAP protocols*, execution of an agent A on a host is monitored by agents (napping) on other hosts [20]. If the failure of A or of the host on which A executes is detected, then one of the napping agents performs a recovery action. This recovery action might involve retrying A, dispatching a different agent to some other host, or alerting the computation's initiator of a problem. NAP is not resilient to hostile host failures, but without using replication no scheme can be.

The difficult part of implementing NAP involves coordinating the napping agents. A protocol that tolerates multiple failures must have multiple agents napping, each monitoring execution. A coordination protocol is required to ensure that more than one napping agents does not detect and try to restart a failed agent. Our initial solutions to the coordination problem were complex enough that their correctness was suspect. This led us to show that the problem was actually an instance of the (fail-stop) reliable broadcast problem that we solved in 1983. And, by refining our 1983 protocol, we were able to support a broad class of strategies for how napping agents are disbursed in the network. This broader class of strategies allows our protocols also to work when the trajectory of an agent folds back on itself, visiting a host that is still running a napping agent.

[1]This work is joint with Dag Johansen at the University of Tromsoe (Norway) and Keith Marzullo at the Univ of California, San Diego.

Analysis of system fault-tolerance

Ad hoc reasoning about fault tolerance is unsatisfactory for large, critical-infrastructure systems. Only rigorous analysis with mechanized support can give the needed confidence; only a tool that is usable by system designers can have a real impact. Therefore, we continued our investigations (jointly with Scott Stoller) into a new verification framework that is specialized to fault tolerance [4,13]. The framework, which is based on a stream-processing model of computation, permits more natural specifications of fault-tolerance requirements than general-purpose formalisms and supports mechanized analysis of system fault-tolerance.

In stream-processing models, each component of a system is represented by an *input-output function* describing its behavior. For simplicity, processes are assumed to communicate only by messages transmitted along unbounded FIFO channels. Behaviors of a system can be determined from input-output functions describing the system's components by doing a fixed-point calculation. This provides a clean algorithmic basis for our analysis. Each input-output function encapsulates the implementation of a component, enabling a convenient separation of local and global analyses. Local analysis verifies independently for each component that the proposed input-output function faithfully represents its behavior. Global analysis, in the form of the fixed-point calculation, determines the system's behavior from the input-output functions.

The fixed-point calculation produces a graph, called a *message flow graph*, representing possible communication behaviors of the system. Each node of the graph corresponds to a component, and each edge is labeled with a description of the sequence of messages sent from the source node to the target node. Exact computation of all possible sequences of messages that might be sent is generally infeasible. So, to help make automated analysis feasible, our framework supports flexible and powerful approximations, or abstractions, as they are called in the literature on abstract interpretation. Traditionally, stream-processing models have not incorporated approximations. The approximations in our framework enable compact representation of the highly non-deterministic behavior characteristic of severe failures and also support abstraction from irrelevant aspects of a system's failure-free behavior. The latter reflects a separation of concerns that is crucial for making the fault-tolerance analysis tractable.

We use only conservative approximations, so the analysis never falsely implies that a system satisfies its fault-tolerance requirement. But approximations do introduce the possibility of false negatives: the analysis might not establish that a system satisfies its fault-tolerance requirement, even though it does.

A common approach to modeling failures is to treat them as events that occur non-deterministically during a computation, thereby making it difficult to separate the effects of failures from other aspects of the system's behavior and, consequently, to model the former more finely than the latter. In particular, one often wants to avoid case analysis corresponding to non-determinism in a system's failure-free behavior, while case analysis corresponding to different combinations of failures appears unavoidable in general in automated analysis of fault-tolerance. A *failure scenario* for a system is an assignment of component failures to a subset of the system's components. In our approach, each input-output function is parameterized by possible failures in the corresponding component; system behavior is analyzed separately for each failure scenario of interest.

In our framework, possible communications (in a given failure scenario) between two components are characterized by approximations of *values* (the data transmitted in messages), *multiplicities* (the number of times each value is sent), and *message orderings* (the order in which values are sent). Values and multiplicities are approximated using a form of abstract interpretation and a form of symbolic computation. Message orderings are approximated using partial (instead of total) orders.

Our analysis method was implemented in a prototype tool called CRAFT. And we have used CRAFT to analyze our protocols for agent integrity and the Oral Messages algorithm for Byzantine Agreement.

Enforceable Security Policies

A *security policy* defines executions that, for one reason or another, have been deemed unacceptable. To date, application-independent security policies—like mandatory and discretionary access control, information flow restrictions, and resource availability—have attracted most of the attention. But with the expanding role of computers in our infrastructure, specialized, application-dependent security policies are becoming increasingly important. For example, a system to support mobile code might prevent information leakage by enforcing a security policy that bars messages from being sent after files are read. To support electronic commerce, a security policy might prohibit executions in which a customer pays for a service but the seller does not provide that service.

Over the period of this grant, we developed a mathematical characterization of what security policies are enforceable [9]. First, we proved that enforcement mechanisms cannot exist for security policies that are not safety properties. Second, we developed a new class of enforcement mechanisms and proved that it is complete for the set of all enforceable security policies [22]. Our new class of mechanisms is based on *security automata*, automata that accept finite and infinite sequences.

A security automaton serves as an enforcement mechanism for some target system by monitoring and controlling the execution of that system. Each action or new state corresponding to a next step that the target system takes is sent to the security automaton and serves as the next symbol of that automaton's input. If the automaton cannot make a transition on an input symbol, then the target system is about to violate the security policy specified by the automation, and the target system is terminated.

We demonstrated the practicality of enforcing security policies expressed using security automata by constructing and evaluating tools to generate *inlined reference monitors* that implement security automata for both the Java Virtual Machine and Intel x86 machines. The first prototype (SASI) worked for programs written or compiled into Java virtual machine code (JVML) or Intel's x86 machine code; a second generation (PoET/PSLang) refined the approach for JVML. Specifically, given a security automaton SA that expresses a security policy and given a machine language program P, both SASI and PoET/PSLang add checks to P that are necessary in order to ensure that executing P is guaranteed not to violate the security policy defined by SA. In addition, using standard compiler analyses, our prototypes attempt to minimize the number of checks inserted.

Using SASI, we experimented with generalizations of two well known security policies: software fault isolation (SFI) and the Java Standard Security Manager. Our experiments confirmed that SASI generates code comparable with hand-coded, heavily optimized SFI tools for the x86, and in fact exceeds the performance of the hand-coded Java Standard Security Manager. Furthermore, security automaton specifications of the security policies have proven to be easy to write, understand, and modify. Using PoET/PSLang, we showed how to support the Java 2 "stack inspection" security policy without any support from the Java virtual machine. This, for example, allows Java 2 programs to be executed on previous generations of the Java run-time system; it also allows deployment of variations and refinements of the Java security policy.

Publications

(1) Y. Minsky, R. van Renesse, F.B. Schneider, and S.D. Stoller. Cryptographic support for fault-tolerant distributed computing. *Proc. of the Seventh ACM SIGOPS European Workshop "System Support for Worldwide Applications"* (Connemara, Ireland, Sept 1996), ACM, New York, 109–114.

(2) D. Johansen, R. van Renesse, and F.B. Schneider. Supporting broad internet access to TACOMA. *Proc. of the Seventh ACM SIGOPS European Workshop "System Support for Worldwide Applications"* (Connemara, Ireland, Sept 1996), ACM, New York, 55–58.

(3) F.B. Schneider. Notes on proof outline logic. *Deductive Program Design*, M. Broy, ed. ASI Vol. F152, Springer-Verlag, Heidelberg, 351–394.

(4) S.D. Stoller and F.B. Schneider. Automated Analysis of Fault-Tolerance in Distributed Systems. *Proc. First ACM SIGPLAN Workshop on Automated Analysis of Software*, Rance Cleaveland and Daniel Jackson, eds., (Paris, France, Jan. 1997) ACM, New York, 33–44.

(5) F.B. Schneider. *On Concurrent Programming.* Springer Verlag, NY, 1997, 473 pages.

(6) F.B. Schneider (ed.). Information Systems Trustworthiness—Interim Report. Computer Science and Telecommunications Board Commission on Physical Sciences, Mathematics, and Applications National Research Council. April 1997.

(7) D. Dolev, R. Reischuk, F.B. Schneider, and H.R. Strong. Report on Dagstuhl Seminar on Time Services, Schloss Dagstuhl, March 11–March 15 1996. *Real-Time Systems 12*, 3 (May 1997), 329–345.

(8) F.B. Schneider. Editorial: New Partnership with ACM. *Distributed Computing 10*, 2 (1997), 63.

(9) F.B. Schneider. Towards fault-tolerant and secure agentry. *Proc. 11th International Workshop WDAG '97* (Saarbrucken, Germany, Sept. 1997) Lecture Notes in Computer Science, Volume 1320, Springer-Verlag, Heidelberg, 1997, 1-14.

(10) D. Johansen, R. van Renesse, and F.B. Schneider. Operating system support for mobile agents. Republished in: *Readings in Agents*, Michael N. Huhns and Munindar P. Singh eds. Morgan Kaufman Publishers, San Francisco, California, 1997, 263-266.

(11) D. Gries and F.B. Schneider. Adding the everywhere operator to propositional logic. *Journal of Logic and Computation 8*, No. 1 (Feb. 1998).

(12) F.B. Schneider. On Concurrent Programming. Inside Risks 94, *Communications of the ACM 41*, No. 4 (April 1998), 128.

(13) S.D. Stoller and F.B. Schneider. Automated stream-based analysis of fault-tolerance. *Formal Techniques in Real-time and Fault-tolerant Systems*, Proc. 5th International Symposium FTRTRT'98 (Lyngby, Denmark, Sept. 1998), Lecture Notes in Computer Science, Vol. 1486, 113–122.

(14) F.B. Schneider. Towards trustworthy networked information systems. Inside Risks 101, *Communications of the ACM 41*, No. 11 (Nov. 1998), 144.

(15) F.B. Schneider. Improving networked information system trustworthiness: A research agenda. *Proceedings 21st National Information Systems Security Conference* (Arlington, Virginia, Oct. 1998), National Computer Security Center, 766.

(16) F.B. Schneider and S.M. Bellovin. Evolving telephone networks. Inside Risks 103, *Communications of the ACM 42*, No. 1 (Jan. 1999), 160.

(17) F.B. Schneider (editor). *Trust in Cyberspace.* National Academy Press, Dec. 1998, 331 pages.

(18) D. Johansen, R. van Renesse, and F.B. Schneider. Operating System Support for Mobile Agents. Republished in *Mobility: Processes, Computers, and Agents*, Dejan S. Milojicic, Frederick Douglis, and Richard

G. Wheeler (eds.), Addison Wesley and the ACM Press, April 1999, 557–563.

(19) D. Johansen, R. van Renesse, and F.B. Schneider. What Tacoma Taught Us. *Mobility: Processes, Computers, and Agents*, Dejan S. Milojicic, Frederick Douglis, and Richard G. Wheeler (eds.), Addison Wesley and the ACM Press, April 1999, 564–566.

(20) D. Johansen, K. Marzullo, F.B. Schneider, K. Jacobsen, and D. Zagorodnov. NAP: Practical Fault-tolerance for Itinerant Computations. *Proc. 19th IEEE International Conference on Distributed Computing Systems* (June 1999, Austin, Texas), IEEE, 180–189.

(21) F.B. Schneider, S. Bellovin, and A. Inouye. Building trustworthy systems: Lessons from the PTN and Internet. *IEEE Internet Computing*, 3, 5 (November-December 1999), 64-72.

(22) U. Erlingsson and F.B. Schneider. SASI enforcement of security policies: A retrospective. *Proceedings of the New Security Paradigm Workshop* (Caledon Hills, Ontario, Canada, September 22-24, 1999), Association for Computing Machinery, 1515 Broadway, NY, NY, 87–95.

Patents

(1) Transparent fault tolerant computer system. United States Patent 5,802,265, Sept. 1, 1998. Co-inventors: T.C. Bressoud, J.E. Ahern, K.P. Birman, R.C.B. Cooper, B.Glade, and J.D. Service.

(2) Transparent fault tolerant computer system. United States Patent 5,968,185, Oct. 19, 1999. Co-inventors: T. C. Bressoud, J. E. Ahern, K. P. Birman, R. C. B. Cooper, B. Glade, and J. D. Service.

DISTRIBUTION LIST

addresses	number of copies
JOHN C. FAUST AFRL/IFGB 525 BROOKS RD ROME, NY 13441-4505	10
CORNELL UNIVERSITY DEPARTMENT OF COMPUTER SICENCE 4130 UPSON HALL ITHACA, NY 14853	5
AFRL/IFOIL TECHNICAL LIBRARY 26 ELECTRONIC PKY ROME NY 13441-4514	1
ATTENTION: DTIC-OCC DEFENSE TECHNICAL INFO CENTER 8725 JOHN J. KINGMAN ROAD, STE 0944 FT. BELVOIR, VA 22060-6218	1
DEFENSE ADVANCED RESEARCH PROJECTS AGENCY 3701 NORTH FAIRFAX DRIVE ARLINGTON VA 22203-1714	1
ATTN: NAN PFRIMMER IIT RESEARCH INSTITUTE 201 MILL ST. ROME, NY 13440	1
AFIT ACADEMIC LIBRARY AFIT/LDR, 2950 P.STREET AREA B, BLDG 642 WRIGHT-PATTERSON AFB OH 45433-7765	1
AFRL/HESC-TDC 2698 G STREET, BLDG 190 WRIGHT-PATTERSON AFB OH 45433-7604	1

ATTN: SMDC IM PL 1
US ARMY SPACE & MISSILE DEF CMD
P.O. BOX 1500
HUNTSVILLE AL 35807-3801

COMMANDER, CODE 4TL000D 1
TECHNICAL LIBRARY, NAWC-WD
1 ADMINISTRATION CIRCLE
CHINA LAKE CA 93555-6100

CDR, US ARMY AVIATION & MISSILE CMD 2
REDSTONE SCIENTIFIC INFORMATION CTR
ATTN: AMSAM-RD-OB-R, (DOCUMENTS)
REDSTONE ARSENAL AL 35898-5000

REPORT LIBRARY 1
MS P364
LOS ALAMOS NATIONAL LABORATORY
LOS ALAMOS NM 87545

ATTN: D'BORAH HART 1
AVIATION BRANCH SVC 122.10
FOB10A, RM 931
800 INDEPENDENCE AVE, SW
WASHINGTON DC 20591

AFIWC/MSY 1
102 HALL BLVD, STE 315
SAN ANTONIO TX 78243-7016

ATTN: KAROLA M. YOURISON 1
SOFTWARE ENGINEERING INSTITUTE
4500 FIFTH AVENUE
PITTSBURGH PA 15213

USAF/AIR FORCE RESEARCH LABORATORY 1
AFRL/VSOSA(LIBRARY-BLDG 1103)
5 WRIGHT DRIVE
HANSCOM AFB MA 01731-3004

ATTN: EILEEN LADUKE/D460 1
MITRE CORPORATION
202 BURLINGTON RD
BEDFORD MA 01730

OUSD(P)/DTSA/DUTD 1
ATTN: PATRICK G. SULLIVAN, JR.
400 ARMY NAVY DRIVE
SUITE 300
ARLINGTON VA 22202