High Assurance for Interactive Applications in Ad Hoc Networks

David Naumann* and Susanne Wetzel

Stevens Institute of Technology,
Department of Computer Science,
Castle Point on Hudson,
Hoboken, NJ 07030, USA
{naumann,wetzel}@cs.stevens-tech.edu

Abstract. Advances in wireless networking and small, low-power storage and processing technologies are bringing ubiquitous computing closer to reality. We are nowhere near the vision articulated by Weiser in which computers disappear into the fabric of everyday life. But we do have cellular phones, smart cards, personal digital assistants (PDA) with wireless web connections, and limited deployments of many other devices, like positioning sensors and theft alarm systems in automobiles and cameras in traffic lights. In the future, extremely large numbers of devices will be deployed for myriad purposes. There are tremendous technical challenges to building scalable infrastructures for decentralized and ad hoc networking with adequate performance to meet application objectives. Just as great are the challenges in eliminating the many weaknesses of current technologies which make them vulnerable to attacks with potentially catastrophic consequences in economic terms and for national security.
This paper surveys a project that addresses the problem as to how to meet both security and performance goals in applications using ad hoc short-range (wireless) networking and extensible software architecture bringing together two lines of security research, cryptographic protocol design and static program analysis.

Keywords: Ad hoc networks, static program analysis, security, privacy, ubiquitous computing.

1 Introduction

In ubiquitous computing applications, the need to minimize size, weight, and cost leads to severe constraints on computing power, communication power, and storage capacity. Wireless connections and mobility result in highly dynamic network topology. The absence of physical connectivity cannot be used as a means to control access. These characteristics open up security vulnerabilities not present in wired networks and conventional software systems [21]. Ubiquitous computing poses new challenges, exacerbates old problems, and upsets the

* Supported by NSF grant CCR-0208984.

balance for many system design tradeoffs. On the positive side, resource-limited systems are amenable to means of assurance such as load-time static analyses that would be impractical for large software systems.

Consider a monitoring system used for schoolchildren on a field trip. On their way out of the bus, each child grabs a disposable wristband, "swiping" it on a reader and saying their name into the teacher's PDA before running off. The unobtrusive wristbands track position and have a panic button; the PDA shows the position of each child and sounds an alert if they stray too far. The system has intrinsic security requirements. For example, the link between a particular wristband's ID and the name of the child should be authenticated so that if a panic signal is received, it is associated with the right child. Some attacks are unavoidable: a person attempting to abduct a child clinging desperately to the wristband could disrupt the signal from the device. But an attacker should not be able to substitute another wristband signaling "OK" and surreptitiously toss it into another child's backpack for cover.

More subtle requirements involve privacy. Permanent records should not be kept of children's behavior in detail beyond culturally accepted norms. The teacher might well retain a trace of the day's activities as evidence in case of trouble, but accumulation of large amounts of seemingly innocuous information poses risks well known in totalitarian societies.

Consider building a system like this using current technology. Bluetooth (`www.bluetooth.org`) could be used for low-cost short range wireless links. The teacher's unit would likely be a generic PDA providing a common software platform used by a number of applications including *mobile code* installed at the teacher's discretion, like an interactive tour guide downloaded upon entrance to a museum. Other applications could manipulate personal financial information or sensitive information about one of the children: the name and type of car driven by an uncle who is to pick them up after school.

Next-generation applications of all kinds, including ad hoc networking, are being developed on the basis of extensible object-oriented *middleware* [14, 13, 8], in order to achieve higher productivity, shorter development cycles, interoperability, and end-to-end control of system resources. But the flexibility of object-oriented (OO) programs comes at a cost: there are many avenues for interference among applications. Security goals for one application may be compromised by exploits involving seemingly unrelated applications which may have no intrinsic security requirements of their own. Encapsulation to prevent accidental and malicious interference is an old problem [16]. However, old solutions such as hardware and OS support for separation of address spaces are of limited use in resource-poor devices and in the presence of OO interfaces making extensive use of shared objects and callbacks [22].

If a particular application fails to meet security requirements with high assurance, there may be broad consequences. The economic and social benefits of widespread adoption will be lost if failures lead to unfavorable news reports about the underlying technology. Perhaps even worse, there is a real possibility of widespread adoption and deployment of systems with serious flaws. There

are grave risks to safety and reliability, privacy, and quality of life. Even more than for desktop and enterprise systems, it is important to prevent mistakes before wide deployment; it could be very costly or impossible to patch or disable low-cost devices embedded in consumer goods, transportation matériel, etc.

This paper surveys a project that addresses the problem as to how to meet both security and performance goals in applications using ad hoc short-range (wireless) networking and extensible software architecture. In broad outline, we address the questions

(a) what are the security vulnerabilities in state-of-the-art short-range ad hoc networking protocols?
(b) what improvements and new protocols can better achieve security and performance goals?
(c) what middleware architectures and access control mechanisms can best achieve extensibility and development productivity without compromising security and reliability?
(d) what static analyses can be used for high assurance of implementations involving mobile code and cross-layer optimizations?

These questions bring together two lines of security research, cryptography and static program analysis, focusing on already active. This project focuses on integrating the research in end-to-end case studies.

2 Research Environment

The current research is based on a prototype system called Code Blue which is a human-centric application that uses short-range wireless networking for body-worn sensors. The Code Blue system lets dancers manipulate user-configurable light and sound effects via wearable sensors that detect motion, body temperature, proximity, etc. Each Sensor Module transmits via a Bluetooth wireless link to a central Access Point linked to a controller. This "Midi Engine" plays music and controls lights using a Midi data stream originating in a data file but processed in response to sensor inputs. Unlike audio formats that record sampled sound, Midi consists of a sequence of packets (called *events*) which serve as instructions, typically for notes played by sound synthesis equipment. Midi files are very compact. In Code Blue, a sensor input stream controls an "Algorithm" manipulating packets on a particular channel. Bending one's arm could cause a bandpass filter to sweep across a drum track. Clapping hands could cause a spotlight to move and a phrase to be played on synthesized violin.

In the future, prototypes for similar sensor-based applications such as sports training and physical therapy will be developed that reuse much of the Code Blue hardware and software.

3 Research

3.1 Wireless Security

Security and privacy in ubiquitous computing in general and ad hoc networks in particular are of great concern and pose great challenges. Recent work has shown that currently deployed technologies such as Bluetooth and 802.11 have serious shortcomings in these regards (see [15, 11, 6, 10]).

Conducting research to find solutions to the above mentioned security problems with the goal to impact ongoing standardization efforts is one of the main thrusts of our work. In particular, the sensor network based on Bluetooth technology serves as a test bed to implement the attacks described in [15].

In an initial step of the research effort, the sensor network based on Bluetooth technology serves as a test bed to implement the known attacks. This proof of practicability is of great importance since the seriousness of the vulnerabilities of the Bluetooth security concept continue to be vastly underestimated within the greater Bluetooth community. In a later stage of the project, we will use the test bed to implement and analyze newly developed security protocol solutions.

Our research on security protocol design focuses on two main aspects. The first one explores ways to introduce improved security mechanisms in higher layers of the Bluetooth protocol stack. It has been proven to be extremely difficult to implement changes in the lower protocol stack (possibly involving changes in the Bluetooth chip itself). The reasons are manifold and range from company interests to performance discussions and compatibility issues. The fact that certain devices implement only a limited protocol stack disallows for the obvious solution to this problem which defers security mechanisms to the application layer. Secondly, our research addresses the trade-off between security and performance. Obviously, increased security comes at a cost to performance. Since not all applications will require security and privacy mechanisms, the security concept should allow for dynamic tailoring of security mechanisms to application specific needs without introducing back doors that could be exploited in more restrictive and sensitive contexts.

Additional research interests include roaming between different technologies (bearers) and providing privacy in multi-hop networking.

3.2 Encapsulation and Static Analysis

Many systems fall short of meeting security goals due to flaws in implementation. Software flaws such as buffer overflows make it possible for an attacker to insert malicious code which may then exploit other flaws [1, 23]. A remarkable portion of the flaws that are exploited in current systems are simple failures of memory safety, e.g. using integer arithmetic to forge a pointer to private kernel code and then treating that integer as a jump target. No matter the strength of a cryptographic protocol, flaws in its implementation may allow it to be circumvented or subverted, e.g. by malicious code obtaining secret keys. Such flaws are failures of *encapsulation* or enforcement of abstraction boundaries —such as protocol

layers or the public interface to a cryptographic "black box". The problem is a fundamental one addressed by long established means such as complete separation of process address spaces with hardware support or dynamic monitoring of execution. In ubiquitous computing applications, hardware support may be absent and resources too limited to afford the substantial performance penalties of dynamic monitoring.

High level languages such as Java check source code encapsulation boundaries statically, through strong type checking and data abstraction constructs. Source-level checking is needed in order for developers to prevent flaws, but there is also a need to check object code at load time, especially mobile code from untrusted sources. For bytecodes, such checking has already come into common practice with Java and the .NET Common Language Runtime. For native machine code, and for simpler and more efficient VMs, several groups are developing means to augment low-level code with sufficient information about its memory use or other behavior to make to make efficient load-time checks possible [17, 19]. In principle, this approach can be used for any security policy, but so far it has been developed only for very basic memory safety properties.

Confidentiality is at the core of many security policies, for both applications and middleware (e.g. session keys are secret). Secure information flow is not simply access control: through covert channels such as timing and control flow, information can be leaked from one process to another without a direct data path in memory [9, 20]. Access controls help control "information release but do not control information propogation" [18].

For simple program constructs, it has been shown that efficient static checks, in the form of typing rules, can ensure secure information flow (e.g. [25]). In OO programs, unbridled use of inheritance, dynamic binding, and shared mutable data (within address spaces) can violate abstraction boundaries. The Java access control system was rendered insecure because a leaked reference exposed access to an internal data structure, making it possible to forge crytographic authentication [24]. Banerjee and Naumann were able to adapt ideas from the OO confinement literature [24, 7] to give a simulation result for a Java-like language [3] and to give an efficient static analysis for confinement [2] that applies to common OO design patterns [2]. On this basis they were able to extend earlier work on static checking of secure information flow [25] to a large fragment of Java that includes unrestricted callbacks and aliased mutable objects [4].

Although there has been extensive research on static checking of information flow, it has seen little practical application, in part because the conservativity of the analyses has made them impractically restrictive [20] and in part because for some policies noninterference is too strong a property. It is access control that is widely used in practice. The so-called *stack inspection* mechanism deployed in Java [12] and in the .NET Common Language Runtime is designed to support fine-grained access policies for OO mobile code. Banerjee and Naumann have recently given a novel analysis generalizing the typing systems of [25, 4] to track access-control permissions and enforce confidentiality [5]. The rules account for system calls that check callers' permissions and can be used by trusted programs

for manipulation of high security information and also by untrusted programs for manipulation of low information. Policy designates both trust (authorization of different permissions for different programs) and confidentiality levels.

4 Conclusion

In our project, realistic case studies facilitate assessment of existing proposals and tools, our own and others', and bring to light specific problems that are amenable to high-assurance static checking based on rigorous theory. Here is an example case study, one which brings together research in encapsulation with research in protocols. A dance group installs a new processing algorithm to be driven by a particular sensor in Code Blue. Installation means adding Java code to the Midi Engine and also loading a small C program onto the Sensor Module (it improves performance by dropping sensor data that is not of interest to the processing algorithm). The C code, of course, runs in the same address space as the Bluetooth stack including crypto protocols. Suppose one of those protocols has a Trojan horse that leaks a secret key (possibly through a covert channel rather than explicit data flow) to the sensor program. That program sends bits of the key as sensor data to the processing algorithm at the Midi Engine. It might pass the key on in Midi packets. It could also send it, via covert channel, to some unrelated application that happens to be running on the machine.

References

1. Ross Anderson. *Security Engineering.* Wiley, 2001.
2. Anindya Banerjee and David A. Naumann. Ownership confinement ensures representation independence for object-oriented programs. Journal version of [3], submitted., 2002.
3. Anindya Banerjee and David A. Naumann. Representation independence, confinement and access control. In *POPL*, pages 166–177, 2002.
4. Anindya Banerjee and David A. Naumann. Secure information flow and pointer confinement in a Java-like language. In *15th IEEE Computer Security Foundations Workshop*, pages 253–270, 2002.
5. Anindya Banerjee and David A. Naumann. Using access control for secure information flow in a Java-like +language. In *IEEE Computer Security Foundations Workshop*, 2003. To appear.
6. Nikita Borisov, Ian Goldberg, and David Wagner. Intercepting mobile communications: The insecurity of 802.11. `http://www.isaac.cs.berkeley.edu/isaac/wep-faq.html`, 2001.
7. David G. Clarke, James Noble, and John M. Potter. Simple ownership types for object containment. In Jørgen Lindskov Knudsen, editor, *ECOOP 2001 - Object Oriented Programming*, 2001.
8. Munir Cochinwala. Using objects for next generation communication services. In Elisa Bertino, editor, *ECOOP 2000: European Conference on Object-Oriented Programming*, pages 388–393, 2000.
9. D. Denning and P. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 20(7):504–513, 1977.

10. Scott Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the key scheduling algorithm of RC4. In *Selected Areas in Cryptography*, volume 2259 of *LNCS*, 2001.
11. Scott R. Fluhrer and Stefan Lucks. Analysis of the e_0 encryption system. In *Selected Areas in Cryptography*, 2001.
12. Li Gong. *Inside Java 2 Platform Security*. Addison-Wesley, 1999.
13. Ravi Jain and Farooq Anjum. Java call control. In T. Jepsen, editor, *Java in Telecommunications: Solutions for Next Generation Networks*. Wiley, 2001.
14. Ravi Jain, Farooq Anjum, Paulo Missier, and Subramanya Shastry. Java call control, coordination, and transactions. *IEEE Communications Magazine*, January 2000.
15. Markus Jakobsson and Susanne Wetzel. Security weaknesses in bluetooth. In D. Naccache, editor, *Progress in Cryptology: Cryptographers' Track at RSA Conference*, volume 2020 of *Lecture Notes in Computer Science*, pages 176–191, 2001.
16. Butler Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10):613–615, October 1973.
17. Greg Morrisett, Karl Crary, Neal Glew, and David Walker. From system F to typed assembly language. *ACM Transactions on Programming Languages and Systems*, 21(3):528–569, 1999.
18. Andrew C. Myers and Barbara Liskov. Protecting privacy using the decentralized label model. *Transactions on Software Engineering and Methodology*, 9(4):410–442, 2000.
19. George C. Necula and Peter Lee. Safe kernel extensions without run-time checking. In *Operating System Design and Implementation*, pages 229–243, 1996.
20. A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE J. Selected Areas in Communications*, 21(1):5–19, January 2003.
21. Frank Stajano. *Security for Ubiquitous Computing*. John Wiley and Sons, February 2002.
22. Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. ACM Press and Addison-Wesley, New York, NY, 1998.
23. John Viega and Gary McGraw. *Building Secure Software*. Addison-Wesley, 2002.
24. Jan Vitek and Boris Bokowski. Confined types in Java. *Software Practice and Experience*, 31(6):507–532, 2001.
25. Dennis Volpano and Geoffrey Smith. A type-based approach to program security. In *Proceedings of TAPSOFT'97*, number 1214 in Lecture Notes in Computer Science, pages 607–621. Springer-Verlag, 1997.