Identifying and testing for insecure paths in cryptographic protocol implementations

K R Jayaram *
Department of Computer Sciences, Purdue University,
W Lafayette, IN 47906, USA
jayaram@purdue.edu

1. Introduction

Cryptographic protocols, which are also referred to as security protocols are used to process, store and transfer increasing volumes of information on our financial networks, health networks, and even our library systems, not to mention our conventional communication systems and our networked systems of personal and corporate computers. Users should be able to justifiably rely on their implementations to process, store, and communicate sensitive information securely.

Testing is indispensable even when a security protocol is formally verified because most formal verification techniques only guarantee the correctness of the design, under certain assumptions [8]. More importantly, no guarantees about the implementation are provided. A mathematical proof that an implementation of a security protocol conforms to its specifications is usually not feasible because it would require complicated formal semantics of the language in which it is written and the environment in which the protocol runs (the operating system and hardware). Information security is constantly endangered by errors in the contemporary protocol implementations. Examples of implementation errors leading to security vulnerabilities include: (i) Buffer Overflows and Race Conditions in Kerberos implementations which have resulted in an attacker gaining root access to the Key Distribution Server [7]; (ii) IPSec Vulnerabiltiy [6]; (iii) A man-in-the middle attack on the OpenSSL library forcing the usage of insecure SSL 2.0 protocol even if both the ends support SSL 3.0 [10].

"Security testing of a protocol" refers to the testing of an implementation of the protocol to show that it satisfies the stated security requirements in its specification i.e. to validate whether required security functionalities are implemented correctly. Security testing aims at identifying insecure paths. An Insecure path in a program is one that results from an implementation error and causes a security vulnerability. There can be other erroneous paths in a program that cause inconvenience. So every erroneous path is not an insecure path. While there has been a lot of research on the effectiveness of Model based testing (MBT) techniques for software, the efficacy of MBT techniques in security testing needs further research.

*The author would like to thank Prof. Aditya Mathur for his guidance in this project

2. Modeling Security Protocols

The first key issue that has to be addressed in the model-based testing of security protocols is the choice of the modeling formalism. The main criteria which in our opinion should guide the choice are: the formalism should have the necessary expressivity to capture all aspects of a security protocol, it should support model-based testing, and it should be easy to use.

A security protocol:-

1. involves concurrent principals (processes)- a protocol is essentially a way for 2 or more principals to communicate with each other.
2. Each principal may in turn have concurrent threads of computation.
3. Each principal is deterministic i.e. each thread of computation is deterministic.
4. Principals may have memory. For example, a security protocol may involve principal A sending principal B a random number r and expecting $f(r)$ in return.
5. Protocols may involve internal computation, e.g. computation of keys from master secrets and random numbers, data compression, etc.
6. Protocols may involve decision making and looping based on internal memory i.e. they may use internal variables to loop and branch.

Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06)
0-7695-2655-1/06 $20.00 © 2006 IEEE

Two visual formalisms that support requirements 1–3 above are statecharts and Finite State Machines (FSM). Even though FSMs do not support concurrency per se, one can use an FSM to model a single sequential thread and assume that all these FSMs are concurrent [3, 9]. This idea has also been applied to statecharts [2, 1]. However, FSMs do not support requirements 4, 5, and 6. Consequently, statecharts appear to be a natural choice to model security protocols.

3. Test Generation

Our approach to test generation is to use the Wp method [1]. One key issue is handling concurrency. In the Wp method, a concurrent state is flattened (product construction [1]) before constructing the transition cover set. This leads to state explosion. But some interleavings are infeasible because we have distinct concurrent principals (parts of the execution of one does not influence that of the other). We have developed a technique to eliminate such interleavings and consequently the corresponding states and transitions in the product (flattened) state. We plan to combine these techniques with Boundary-Value Analysis to identify malicious inputs and instantiate test traces derived from statecharts accordingly.

4. Adequacy Assessment

We plan to use adequacy assessment techniques based on control flow and data flow to evaluate the goodness of tests generated. Results from adequacy assessment are used in test enhancement i.e. identifying deficiencies in the tests generated w.r.t the assessment criteria and generating additional tests to overcome them. While test adequacy w.r.t control and data flow criteria does not imply program correctness, inadequacy usually implies that some aspects of a system have not been tested thoroughly.

But, to identify the efficacy of the tests generated in detecting security flaws, our approach is to use program mutation [5] and manual injection of malicious faults. We define *security mutants* as mutants which nullify specific security requirements or introduce insecure paths, which are essentially program vulnerabilities which lead to security flaws. We plan to generate security mutants by:-

1. Using existing mutant operators, but isolating those mutants that lead to security vulnerabilities. This is done using a tool like Proteum [4].

2. Manual injection of malicious faults.

We compute the mutation score relative to 'security' mutants only.

5. Conclusion

We propose to increase the reliability of cryptographic protocol implementations through testing and adequacy assessment. This provides us with some guarantees for the actual implementation. We note that this technique of test generation and enhancement via adequacy assessment complements formal verification. The formal model from which we generate tests can be verified, and we can generate test cases to ensure that the implementation corresponds to the model.

We are currently using the GNU implementation of the Transport Level Security (TLS) protocol (RFC 2246) to experiment with these techniques. We plan to experiment with other protocol implementations and compare statechart-based testing with random testing.

References

[1] K. Bogdanov. *Automated Testing of Harel's Statecharts*. PhD thesis, University of Sheffield, January 2000.

[2] S. Burton. *Automated Generation of High Integrity Test Suites from Graphical Specifications*. PhD thesis, University of York, Department of Computer Science, March 2002.

[3] T. S. Chow. Testing software design modelled by finite state machines. *IEEE Transactions on Software Engineering*, SE-4(3):178–187, May 1978.

[4] M. Delamaro, J. Maldonado, and A. Vincenzi. Proteum/IM 2.0: An integrated mutation testing environment. In *Proceedings of Mutation 2000: Mutation Testing in the Twentieth Century*, October 2000.

[5] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Hints on test data selection. *IEEE Computer*, 11(4):34–41, April 1978.

[6] NISCC IPSEC vulnerability advisory. `http://www.niscc.gov.uk/niscc/docs/al-20050509-00386.html?lang=en`.

[7] Security advisories for the Kerberos protocol. `http://web.mit.edu/kerberos/advisories/`.

[8] C. Meadows. Formal methods for cryptographic protocol analysis: Emerging issues and trends. *IEEE Journal on Selected Areas in Communications*, 21(1):44–54, January 2003.

[9] K. Sabnani and A. Dahbura. A Protocol Test Generation Procedure. *Computer Networks and ISDN Systems*, 15:285–297, 1988.

[10] OpenSSL vulnerability advisory. `http://www.openssl.org/news/secadv_20051011.txt`.