Is There An Alternative To Long-Term Secrets?

David S. H. Rosenthal

Mark I Seiden

Position Paper for

"Active Defense of Networks"

Santa Fe Institute 10/30-31/2002

Abstract

The conventional approach to eliminating vulnerabilities in distributed systems depends heavily on keeping secrets for the long term. This is a very difficult thing to do, especially in true peer-to-peer systems with no central control.

We describe some techniques that have been developed in the context of building a peer-to-peer system whose design requirements make keeping long-term secrets impractical. These techniques are based on very conservative assumptions and show some promise of hampering the activities of malicious peers. They include conducting "opinion polls" among large sets of replicas to establish trust in the consensus of the peers rather than in individual peers, the use of a reputation system based on computationally expensive operations, the use of uncertainty rather than secrecy, and constraining systems to run no faster than necessary.

Introduction

Computer systems in the real world, even those isolated from the Internet, may be subject to attacks by "bad guys". These bad guys range from "script kiddies" with no authorized access but only an Internet connection to disgruntled employees with authorized access. The traditional approach to hampering the activities of these bad guys involves keeping secrets for periods many orders of magnitude longer than typical operations of the system.

It is notoriously difficult to ensure in the long term that [DAV1996]:

- the secrets in question are known only to the authorized person(s),
- the secrets in question in fact remain known to those person(s),
- there is no way for the bad guy to successfully attack the system without knowing one of the secrets.

Systems that ignore these difficulties have been widely deployed, tested and found wanting. Other ways of hampering the bad guy's activities have not been thoroughly explored, perhaps because the ready availability of a kit of tools based on secrets that seems to prevent the bad guy from accessing or interfering with a system has bred a false sense of security [SCH2000].

We present a critique of the conventional approach to hampering the bad guy, and a set of techniques that show some initial promise of making life hard for the bad guy in some restricted application areas. Nothing we say should be interpreted as claiming that long-term secrets are not necessary - there is clearly a large class of applications for which they are essential. What we are suggesting is that there are applications for which long-term secrets are not essential. Using them in these applications has many disadvantages, and may in itself introduce unnecessary vulnerabilities. It appears possible that these techniques could help applications requiring secrets be more robust against the consequences of the secrets leaking.

The peer-to-peer system architecture has many attractive features, including some that enhance robustness. For example, a system with no central control is not vulnerable to technical or legal attacks on its control centers. On the other hand, systems with no central control make exercising authority and keeping secrets particularly difficult. The designers of such systems must expect the system to be compromised to some extent all the time. The system must aim to *co-exist* with the bad guys rather than *exclude* them.

Our techniques were developed in a peer-to-peer context and are particularly relevant there. They may be thought of as a way to build peer-to-peer systems with intrusion detection built-in, rather than layered on as an afterthought.

Hampering The Bad Guy Using Secrets

The canonical approach to securing a distributed system involves a hierarchy of secrets, typically the private halves of key pairs and the passwords that protect them:

- Typically, a small number of authorities with top-level secrets use them to

issue certificates testifying that an entity knowing a lower-level secret is who they claim to be. The assumption is made that each entity can and does prevent any other entity knowing the secret in question, and thus that knowledge of the secret is a proxy for identity.
- Access control lists are used to map these secret-knowing entities to operations they are, or are not allowed to perform in the context of the system. The assumption is that authorized entities will not abuse their access to the system.
- Entries in access control lists can be revoked, perhaps because some entity knowing the secret has been observed abusing the system. Certificates can be revoked after a time, or because it is suspected that the secret in question has leaked. The knowledge that the access control list entries and certificates are no longer valid will eventually percolate through the system to prevent the associated secret being used to perform the operations in question. The assumption is that this knowledge will percolate fast enough to prevent serious damage to the system ocurring while it does so.
- The public keys in these certificates can be also used to establish shared secrets (session keys) between entities, allowing efficient encrypted communication between authenticated entities. The assumption is that a shared secret, once established, is not leaked.

Hampering The Bad Guy Using Secrets Has Difficulties

In theory this approach is suspect, since its assumptions are unrealistic:

- The assumption that knowledge of the secret is a proxy for identity is definitely false for the bad guy, and is not credible in the long term even for the good guy. [STA2002] reports that "two-thirds of commuters at London's Victoria Station were willing to reveal their computer password in return for a ballpoint pen".
- The assumption that authorized access will not be abused is often false, for example in the case of a disgruntled employee. [STA2002] reports that "the bulk of computer-security-related crime remains internal".
- A bad guy who gains access to a typical system can inflict a great deal of damage in a short period of time, so the assumption that access can be revoked quickly enough to forestall damage is often false. [STA2002] reports that "Tom Cruise's character in *Minority Report* is still able to get into the police computer network while on the run because someone has neglected to revoke his access privileges".
- A bad guy who establishes a shared secret can easily share it with other bad guys or unsuspecting dupes, so the assumption that session keys are

private to the parties is risky.

In practice this approach suffers from a number of problems:

- It has proven very difficult to establish effective public key infrastructures (PKIs) of this kind [ELL1996,DAV1996].
- Over time, the secrets upon which systems of this kind are based almost inevitably leak. Nothing in the system itself detects that a secret has leaked, so that an extended period may elapse between the leak and its detection.
- Since the system cannot itself detect that a secret has leaked, another system (often called an intrusion detection system) is needed to monitor the first system in an attempt to detect the kind of anomalous activities that are thought to be the signature of unauthorized possession of a secret or attempts to evade the system's access controls. A skilled intruder will avoid these activities, for example by passive listening rather than active interference, and can remain undetected for arbitrary periods.
- Typically, the intrusion detection systems themselves have to be protected by secrets in the same way as the systems they are monitoring, leading to an infinite regress.
- The use of encryption to render communication between entities private in many cases denies useful knowledge not merely to the bad guy but also to the intrusion detection system, rendering it less effective.
- Once the leak is detected, the certificate testifying to the leaked secret and the corresponding access control list entries can be revoked. But it takes time for the revocation to take effect throughout the system, further extending the period during which the system is compromised.
- Once revocation has taken effect, all activities performed under the protection of the leaked secret from the (unknown) time of leakage to the (known) time when the revocation finally took effect everywhere in the system are suspect. The system must be carefully audited to determine what they were, and any bad activities corrected. The higher the level of the secret in question, the greater the impact of the leak. If one of the top-level secrets is discovered to have leaked, the entire system is suspect.
- If, as is often the case, more than one secret leaks but only one secret was detected as having leaked, the bad guy is in a position to listen in to the procedures used to recover from the leak, rendering his knowledge more complete and useful.
- Public systems often depend on a large number of certificate authorities, any of which can assert an identity for a client. This leads to vulnerabilities involving abuse or subversion of the certificate issuing process of the weakest link authority, or accepting certificates signed by bogus authorities.

- Private systems often use only a single certificate authority, leading to an inverse set of vulnerabilities. For example, the certificate authority must itself be protected by one or more secrets, which might leak and cause catastrophic failure of the entire internal system. Or they might be so well protected as to become inaccessible, for example through the dismissal of the only person knowing the secrets, resulting in an equally catastrophic failure.
- Elaborate precautions against having secrets leak are sometimes taken, including the use of tamper-resistant hardware to store them. But the hardware typically requires a password to unlock it, which in itself is a secret that can leak. The need to protect this password while in transit has created an industry devoted to such hardware as "PIN pads" for unlocking smart cards without the PIN (the smartcard password) leaving the protection of trustworthy hardware. This is another infinite regress - there can be no completely secure path from the user's brain to tamper-proof hardware.
- In their anxiety to prevent the secret leaking, systems often make it possible to use the secret without actually knowing it, which can be as damaging as leaking it. For example, a bad guy might penetrate a system, wait for a smart card attached to it to be unlocked, and then use the card to sign transactions which are substituted for or added to legitimate transactions.

In short, the fact that secrets leak is so inconvenient for the conventional approach that practical designs are based upon the hope that they will leak very rarely. This allows the design to assume that secrets remain so, and that leakage will be handled by exception. The difficulty of handling the exception has led to an entire industry devoted to reassuring users and administrators that the secrets are unlikely to leak.

Black & White vs. Shades of Gray

The conventional approach embodies a very black-and-white, binary view of the world. Entities are authoritative or not, guys are either bad or good, secrets are known or not known, systems are compromised or trustworthy.

A shades-of-gray, scalar view might be more realistic; one in which there was no central authority vulnerable to subversion or litigation, in which every component of the system suspected every other component to some extent, in which the probability of a secret being known increased rapidly over time, and in which the system must be expected to be compromised to some extent all the time. In other words, a system that attempts to coexist with the bad guy all the time, rather than one that works by excluding the bad guy completely for most of the time.

We do not claim to know how to build a system meeting these goals. Nevertheless, we believe that the LOCKSS (Lots Of Copies Keep Stuff Safe, http://lockss.stanford.edu) system under development at Stanford can provide some useful insights into the kind of techniques that might be used to do so [ROS2000].

An Overview of the LOCKSS System

The goal of the LOCKSS project is to preserve access to academic journals and other materials published on the web for the long term. It attempts to model the system that librarians have evolved over the millenia to preserve access to material published on paper:

- many institutions around the world acquire copies of the material,
- they use their local copy to provide access to their local readers,
- they preserve their copy to the best of their ability,
- if a copy is lost it may be replaced either from the original publisher or from another library via "inter-library loan".

The LOCKSS system operates in the same way:

- many libraries run persistent web-caches that collect material as it is published by crawling the publisher's web site,
- they use the caches to provide access to their local readers (by acting as web proxies) if the material is unavailable from the publisher,
- the caches cooperate to detect any damage to the preserved content,
- if damage is detected it can be repaired either from the original publisher or from another cache that remembers the requesting peer's prior possession of the material.

The goal of the system is to preserve not just the content as such, but also the services of having links to it and searches for it resolve and deliver the appropriate content. This requires a system that is extremely reliable as a whole. To do this we need a design that pays careful attention to failure modes and possible attacks. In particular, it cannot rely in fundamental ways on other services, such as DNS and PKI. There is no guarantee that these services can be preserved, or that the data they store will still be relevant.

To oversimplify a little, LOCKSS and other digital preservation systems provide the "null application". Nothing is supposed to change. This simplifies many of the difficult problems (such as synchronization and performance) in building dependable distributed systems, and allows easier exploration of unconventional approaches. However, it must be noted that even a successful implementation of the "null application" may not translate directly to more complex applications.

Overall, LOCKSS is designed to be a true peer-to-peer system that requires no central control or authority. Any such locus of control and authority would represent a single point of both technical and legal failure. Note that during testing we have some level of central control of the system to make the tests feasible.

The design of the LOCKSS peer-to-peer communication is based on a set of assumptions, which are intended to be as conservative as possible. They are discussed in more detail in [MIC2002]. Some assumptions relate to the network:

- Each peer has an IP address which serves as a weak form of peer identity.
- Each peer can unicast a message to another peer's IP address. If the addressed peer is on-line it will receive the message with some probability.
- Each peer can only receive messages that are either multicast, or unicast to their IP address (In effect, we assume that peers on the same subnet are in collusion).
- A unicast handshake to another IP address provides some confirmation of the weak IP address identity.

Some assumptions relate to the peers:

- Each data item is shared by some or all of the peers.
- The majority of the peers at any time are interested in other peers preserving the data items they share. They are loyal and follow the communication protocols correctly.
- Any peer may be off-line at any time.

Some assumptions, which are similar to those of Sit & Morris [SIT2002], relate to the "bad guys", or malicious peers:

- A malicious peer can forge messages, including both the contents and the IP source address. He can therefore spoof any peer as the originator or forwarder of a message.
- A malicious peer can send messages to any and all other peers.
- A malicious peer cannot receive messages unicast to loyal peers (i.e. the bad guy does not control the routers and switches in the Internet infrastructure).
- A malicious peer can know all the other malicious peers and conspire with them using communications that cannot be intercepted by the loyal peers.

Political Analogies

The basic ideas behind LOCKSS come from analogies with political systems. These are useful analogies because they are comparatively well-understood system which have evolved to resist subversion by the bad guy, and in which both trust and the lifetime of secrets are limited.

One key to democratic systems is that, subject to some constraints, the majority rules. Bad guys who succeed in becoming the majority automatically become good guys. In practice, all democratic systems coexist with various groups attempting to subvert them, but these groups are small.

Another key is that, because the electorate is very large, effective political communication is public. There is no way to identify accurately all those for whom a given message would be suitable, nor any way to be sure that those to whom the message was originally addressed will not forward it to others. Small groups of bad guys can communicate in secret, but once the conspiracy gets large enough to have a real effect much of its communication, especially that intended to recruit new members, will become public. Indeed, the advent of "sunshine", open meeting, and freedom of access to information laws in the more advanced democracies testifies to the importance for the system of public communication.

Public communication is the basis for the reputation system in politics. Political actors must ensure that their utterances are consistent, because they cannot be sure who will hear them and compare them with earlier utterances. Being caught making inconsistent statements damages a politician's reputation, and reduces the effectiveness of future statements.

The free press in a democratic system doesn't merely report the utterances of political actors and point out inconsistencies. It plays a more general role of monitoring the system for anomalous behavior and attracting attention to it once found, in other words acting as a intrusion detection system. The system is not intended to be invulnerable to all possible attacks, but rather to make attacks attract attention so that they can be thwarted either by the silent majority, or by extra-political activities such as the justice system.

An important attribute of a democratic system is inertia. Britain is often referred to as an elective dictatorship because its executive can change the law overnight, and there used to be no written constitution on the basis of which rapid changes to the law could be invalidated. Political and legislative changes that happen slowly, incrementally and with public debate are usually benign.

Designing With Political Analogies

These ideas lead to the LOCKSS system being founded on:

- voting
- public communication
- a reputation system
- intrusion detection inherent in the system
- considerable inertia

If, in a distributed system, each function is capable of being performed by a very large number of peers, we can arrange for the peers to vote on the result of the operations that make up the function. A bad guy would have to subvert the majority of the replicas to prevent the correct outcome. Given suficient diversity among the replicas, it would be hard to subvert enough of them without detection.

If communication between the peers is public, the votes are public and voting against the mainstream opinion becomes a reputation damaging activity. If votes are weighted by reputation, the bad guys have to be good replicas for a long time to build up a good enough reputation to mount an effective attempt at subversion.

Further, real democratic systems normally have elections that are closely-fought, but between a small number of parties none of which is trying to subvert the system. If we total the votes of the subversive and non-subversive parties, we find that landslide elections in favor of the non-subversive parties are the norm.

In a system voting on the result of operations we are in the same situation. In the absence of the bad guy we expect only a very few dissenting votes, caused by random failures. And the dissenting votes are very likely to disagree with each other. If the bad guys are active, we can expect a much higher proportion of dissenting votes, and a much higher probability of the dissenting votes agreeing with each other. Only if the bad guys subvert the overwhelming majority of the peers will the remaining good guys see what looks like a normal landslide, with the good guys on the losing side. Thus, in order to achieve their ends without detection the bad guys must subvert not just a majority, but an overwhelming majority of the peers. Elections can provide a system with intrusion detection as a built-in capability, not as an afterthought.

Opinion Polls and Sampling

In political systems actual elections are rare and expensive events. Opinion polls, in which a sample of the electorate is asked how they would vote if there were to be an election, have proven cheap enough to be frequent. Their accuracy has been validated by comparing them to actual elections. They depend for their accuracy on careful selection of an unbiassed sample of the electorate.

Similarly, distributed systems based on actual elections in which every replica is expected to take part have proven expensive. So expensive that their use is rare, and they never have more than a small number of replicas.

LOCKSS has much larger numbers of replicas - even in testing it has 50 or more. In production we hope for even larger numbers, but even 50 is too many to run the canonical byzantine fault tolerance protocol [LAM1978] efficiently. Instead LOCKSS samples the replicas, and has only a small sample vote in each poll.

If the sampling process can be designed to be unbiassed in the absence of the bad guy, and hard for the bad guy to bias, this can be a very successful strategy. Detailed discussion of sampling and polling techniques in peer-to-peer networks can be found in [MIC2002], but one concept is to have the peer calling the poll perform the sampling, and to prevent peers taking action except on the basis of polls they themselves called. Because the necessary poll size grows only very slowly when the number of replicas grows, this technique can scale to systems with enormous numbers of replicas. In effect, the use of sampled polls allows a peer to trust not other individual peers, but the peer's estimate of the consensus of the peers as a whole.

Public Communication and Reputation

LOCKSS uses a communication infrastructure with both real IP multicast and simulated multicast via unicast and application-level forwarding. It is designed with four goals:

- The peers should self-organize into an exponential graph (one in which each peer has about the same number of random links), because by doing so they can provide adequate connectivity while preventing the emergence of highly-connected peers [KEY2002]. Highly-connected peers would make the network vulnerable to attack [COH2001].
- Each peer should change its links in the exponential network frequently enough to prevent the bad guys mapping the network completely.
- Messages sent from a peer should be forwarded by other peers, subject to a hop-count, so as to reach an unpredictable subset of the other peers whose size increases with the initial hop-count.
- Messages should be unreliable datagrams and should not trigger predictable responses, to prevent the sender knowing whether the message was actually received by a target peer. This might require, for example, synthesizing the ICMP "Port Unreachable" response even

though the datagram was actually received.

The effect of this infrastructure is that messages between peers are public, in the sense that it is impossible for the sender to predict the set of receivers (unless the set includes only co-conspirators) [MIC2002]. Peers listen to this public communication and, based on the messages they hear, maintain their own private estimate of the reputation of the other peers whose identities they discover. These reputations are based on the weak IP address identity, reinforced by unicast handshakes. The reputation of a newly discovered identity is low.

If it were possible for a single bad guy to vote many times in a poll, the system would be vulnerable. It must therefore be computationally expensive to vote in a poll. At the same time, there must be a reward for making the effort to vote or no peer would bother to do so. The effect of voting on the peer's reputation at other peers is thus essential to making the system work.

Th epublic communication property means that peers other than those actually voting in the poll can monitor it, comparing the votes of other peers with the way they would have voted. This has two beneficial effects:

- the monitoring peers can update their estimates of the reputation of the voters,
- and if they see a contested election with the side they favor losing, they can decide to vote and sway the result.

These are powerful effects. The first spreads knowledge of the reputation of each replica widely and unpredictably, increasing the effectiveness of the reputation system. The second makes life tougher for the bad guy by mobilizing the silent majority to come to the polls and out-vote him.

Peers can use their estimates of the reputation of the voters to weight their votes. In LOCKSS we do this by insisting that, for a peer to take action as the result of being defeated in a poll, not merely must there be a quorum of votes with the majority disagreeing but also that the average reputation of the disagreeing votes must be above a threshold. This increases the importance to the bad guy of behaving as a good guy long enough to build up credibility.

We try to increase this effect further by providing hysteresis in the reputation system, making "good" actions increase reputation slowly and "bad" actions decrease it rapidly. This hysteresis and the initial low reputation of a new identity mean that an identity gradually becomes valuable and able to affect the system - throw-away identities cannot affect the system significantly and are thus not valuable.

Uncertainty

Note the common thread running through these ideas. The good guy is just observing the behavior of the system and responding to his observations. The bad guy is attempting to acquire knowledge about the system and use it to predict the system's behavior. Uncertainty, for example about the number and location of replicas, is not a hindrance to the good guy but is a real burden for the bad guy. By randomizing the behavior of the replicas we can make the task of predicting it very hard, raising the odds against the bad guy. By using an unreliable communication infrastructure that includes random routing we make it hard for the bad guys to focus communication on a victim peer (whose routing behavior they don't control) without detection by the others to whom the victim will forward the messages.

Sloth and Inertia

In the particular case of LOCKSS (and the general case of digital preservation) there is no reason to make the system run fast and many reasons to make it run slowly. Sloth, in the sense of making a system run no faster than absolutely necessary, has important advantages:

- A system that won't do anything quickly won't do what the bad guy wants quickly. Keeping the bad guy talking for a long time helps to detect and trace his activities and trigger "out of band" action to stop him. [WIL2002] and [SOM2000] are examples of slowing systems down once "bad" activities are detected but otherwise allowing work to proceed as fast as possible. If there is no need for the system to run fast there is no need to depend on unreliable detection of bad activities.
- A system that runs slowly and uses public communication can damage the reputation of a peer that exceeds the appropriate limits on the rate at which a peer can originate or forward messages. This does not directly limit bandwidth consumption, in that the rogue peer can send no matter what other peers think of it. But it can limit consumption indirectly, in that other peers will be reluctant to forward messages from senders with low reputations. More importantly, it does make many types of attack involving bad guys sending large numbers of messages unlikely to be effective in the long run.

Sloth in this context also implies inertia, in the sense that each (slow) step can effect only a small change in the state of a small set of peers. Making large changes involves many steps and necessarily takes a long time. This limits the rate at which any peer can change the system, and in particular bad guys can damage the system.

Loose Coupling and Autonomy

Physical systems that fail abruptly and without warning are regarded as too dangerous to deploy. Suspension bridges are not built with single stands of cable whose failure would be both abrupt and fatal. They are built with many, loosely coupled strands. The failure of a single strand is not fatal, but it is hard to ignore.

The key is to prevent failures propagating. In the suspension bridge cable example, cracks propagate through the material of a single strand but cannot propagate from one strand to another through the air gap. Many distributed systems have trust relationships between components along which failures can propagate, particularly when the mode of communication between components is client-server. In this model one component forces another to perform operations or change state on demand.

Peers in LOCKSS are loosely coupled and autonomous. Each is under separate administration. No peer can force another peer to do anything, each peer takes its own decisions based on the messages it receives and its estimates of the reputation of the other peers involved. No peer trusts another peer further than their assessment of its reputation warrants. The polling system allows peers to trust their own estimate of the consensus of the peer population as a whole, which is likely to be more reliable. Further, it allows peers to require an overwhelming majority of the peer population to agree before trusting the consenus, raising the bar for undetected subversion even more.

Related Work

KASTS [MAN2002a] has addressed the problem of preserving the ability to verify digital signatures for the long term. This is an analogous problem to LOCKSS, in that it requires a public, append-only database built from mutually suspicious components. KASTS' approach is more rigorous and less probabilistic than LOCKSS, but it still has some dependence on long-term secrets. It does include a reputation system of sorts, in which at each "poll" the peers come to consensus first about the set of peers they mututally trust, and then about the result of the poll amongst those peers.

Anthill [BAB2002] is a peer-to-peer infrastructure drawing on ideas from complex adaptive systems with strong biological analogies. It is based on mobile software agents "ants" travelling through a collection of peers each implementing a "nest" or context in which the ant can execute. The main similarity with LOCKSS is in the universal dependence on probabilistic peer behavior.

Cornelli et al. [COR2002] have implemented a reputation system for Gnutella. It shares some conceptual similarities with the LOCKSS reputation system but it is not integrated into a fault-tolerant environment.

Keyani et al [KEY2002] describe enhancements to Gnutella that detect an attack on the important highly-connected peers, and change the topology of the system in response to be more random and less vulnerable. They agree with us on the importance of exponential networks for resilience.

Williamson [WIL2002] and Somayaji & Forrest [SOM2000] show that slowing the operation of a subverted peer can be effective at limiting the rate at which damage can be inflicted in a distributed system.

Daswani & Garcia-Molina [DAS2002] demonstrate, using Gnutella as an example, how peers can use rate-limiters and similar "fairness" policies to limit the damage that a malicious peer that abuses the peer communication protocol can do to the overall service provided by a peer-to-peer network. In particular, they note the importance of peer operations that are computationally expensive and the policies used to assign resources to these operations.

Future Work

Based on experience with the LOCKSS prototype and simulations of alternative designs by Michalakis et al [MIC2002] we are re-implementing the system and hope to deploy it for extensive testing in early 2003 and production in 2004.

We hope to use the testbed provided by the deployed LOCKSS systems for further research in areas of difficulty including:

- Techniques for peer identity that might be better than the current (weak) identity provided by the peer's IP address.
- Simulated attacks.
- Ideas from Timeweave [MAN2002b] including the use of entanglement for public, tamper-resistant history.
- Applying the "opinion poll" technique to versioned information, including releases of the underlying software.

Conclusion

We have developed a set of techniques that show some promise of making it hard for the bad guy to interfere with the workings of a distributed implementation of the "null application" (digital preservation) but which don't require any secrets to be kept for extended periods. We have implemented most of them in a prototype that has run for over a year with about 60 peers in libraries scattered around the world. We are currently re-implementing an improved system which will be tested at larger scales and subjected to simulated attacks. We believe that these techniques can be used in combination with the conventional, secrets-based defenses to render more general distributed systems less vulnerable.

Acknowledgments

The LOCKSS project is supported by the National Science Foundation, the Andrew W. Mellon Foundation, Sun Microsystems Labs and Stanford Libraries. Special thanks are due to Dah-Ming Chiu of Sun Labs and Nikolaos Michalakis of MIT for simulations and discussions that led to many improvements in LOCKSS. Thanks are due to Jim Mitchell (Director of Sun Labs), Michael Keller (Librarian of Stanford) and Michael Lesk (ex-NSF) for their consistent support of the project.

Thanks are also due to the workshop participants, and to Mary Baker, Petros Maniatis, TJ Giuli and Mema Roussopoulos of Stanford CS for constructive criticism.

References

- [BAB2002] Babaoglu, O., Meling, H. & Montresor, A. "Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems". Proceedings of the 22th International Conference on Distributed Computing Systems (ICDCS '02), Vienna, Austria, July 2002. Also appears as Technical Report UBLCS-2001-09, University of Bologna, Italy.
- [COH2002] Cohen, R. et al. "Breakdown of the Internet Under Intentional Attack" Phys. Rev. Lett. 86, 3682 (2001).
- [COR2002] Cornelli, F. et al. "Implementing a Reputation-Aware Gnutella Servent". Proceedings of the International Workshop on Peer-to-Peer Computing, Networking 2002, Pisa, Italy, May 2002.
- [DAS2002] Daswani, N. & Garcia-Molina, H. "Query-Flood DoS Attacks in Gnutella" Stanford CS Preprint 2002-26 http://dbpubs.stanford.edu/pub/2002-26
- [DAV1996] Davis, D. "Compliance Defects in Public-Key Cryptography", Proc. 6th USENIX UNIX Security Symposium, San Jose, CA, July 1996.
- [ELL1996] Ellison, C. "Establishing Identity Without Certification Authorities", Proc. 6th USENIX UNIX Security Symposium, San Jose, CA, July 1996.
- [KEY2002] Keyani, P., Larson, B. & Senthil, M. "Peer Pressure: Distributed Recovery from Attacks in Peer-to-Peer Systems". Proceedings of the International Workshop on Peer-to-Peer Computing, Networking 2002, Pisa, Italy, May 2002.

- [LAM1978] Lamport, L. "The implementation of reliable distributed multiprocess systems" Computer Networks 2, 1978.
- [MAN2002a] Maniatis, P. & Baker, M. "Enabling the Archival Storage of Signed Documents." USENIX Conference on File and Storage Technologies (FAST), Monterey, CA, USA. January 2002.
- [MAN2002b] Maniatis, P. & Baker, M. "Secure History Preservation Through Timeline Entanglement." Proc. 11th USENIX Security Symposium, San Francisco, CA, USA. August 2002. (Related technical report arXiv:cs.DC/0202005.
- [MIC2002] Michalakis, N., Chiu, D-M & Rosenthal D. S. H. "Long Term Data Resilience using Opinion Polls", to appear in IPCCC 2003.
- [ROS2000] Rosenthal, D. S. H. & Reich, V. "Permanent Web Publishing", Freenix, San Diego CA, June 2000.
- [SCH2000] Schneier, B "Secrets and Lies", John Wiley & Sons, August 2000
- [SIT2002] Sit, E. and Morris, R. "Security Considerations for Peer-to-Peer Distributed Hash Tables", 1st Intl. Workshop on Peer-to-Peer Systems, 2002.
- [SOM2000] Somayaji, A & Forrest, S. "Automated Response Using System-Call Delays" Proc. 9th USENIX UNIX Security Symposium, Denver, CO, August 2000.
- [STA2002] Standage, T. "Securing the Cloud", *The Economist*, Vol. 365 No. 8269.
- [WIL2002] Williamson, M. "Throttling Viruses: Restricting propagation to defeat malicious mobile code", H-P Labs Tech Report HPL-2002-172 June 2002.