Positive Feedback and the Madness of Crowds

Hilarie Orman and Richard Schroeppel *
Department of Computer Science
University of Arizona

Abstract

This paper discusses the necessity of bringing security measures to very complex systems, and some ideas for how to do so. It has long been the hope that we would never reach such a situation — that security technology would eventually succeed in harnessing software technology and bringing it to heel. This has not happened, and it is now time to expand security technology to cope with the seemingly intractable problems of huge heterogeneous systems. The consideration of paradigms from other fields may be useful.

This paper hopes to motivate research into a fundamental and difficult problem that affects Internet security today: unpredicted or unintended dynamic behavior that can degrade or destroy the ability of the network to deliver messages. We will argue that the source of the problem is hidden variables that adversely affect composition of network systems at any service level: physical, link, network, transport, application, and even the end-user.

Introduction

The Internet is being knitted into our society so tightly and rapidly that it will soon become indistinguishable from society itself — life before the Internet will be unimaginable. This intermeshing of a complex system, human civilization, with a malleable and extensible technology, computers and communication, will not yield a simple, easily analyzable result. Consequently, our thinking about network security must take on a more comprehensive view, one that meshes technology with paradigms taken from social structures.

We cannot expect to develop an all-encompassing model of computer and network security in our society, because we are rapidly moving towards a world that is completely intertwined with and co-existent with its computer technology. Every system that connects to the Internet introduces security complications that we are unprepared to analyze; there is no foreseeable time when security expertise will catch up with reality.

*This work was supported by DARPA contract DABT63-94-C-0002.

Assuring some strong forms of security for very large networks is a pressing problem, one that has grown far away from the seminal ideas that engendered computer security thinking in the past. It is now necessary to deal with systems that are not only large, but ill-defined, dynamic, and increasingly linked to mechanisms that control our physical safety and security, even in non-computer areas.

Security thinking must encompass the totality of a system, attempting to go beyond what can be directly specified and formally analyzed. New factors, including physical phenomena, should be included when considering what constitutes a secure system.

In the past, approaches that limited design choices to analyzable structures were strongly recommended. Designs that limit and partition system capabilities have been viewed as total solutions because they factor the vulnerabilities into isolated areas. However, military and business needs constantly push at these boundaries, and the need to interconnect, and thus to complicate, is irresistible.

Two particular approaches are addressed in this document. One is the discovery of vulnerabilities in complex systems through constant self-examination, using heuristics oriented towards engineering paradigms. Another is the use of simulated software warfare to develop and hone the heuristics. The techniques for implementing the approaches cross the fields of formal security, intrusion detection, safe systems, practical computer security, artificial intelligence, mobile agents, and secure network protocols.

Availability must be considered as important as any other aspect of security if we are to have survivable, resilient networks. This brings networked system analysis up against difficult problems. Denial of service is rarely addressed by network security models. Formalization of it is problematical, and when applied to huge-scale network systems, it must address the real-world behavior of millions of independent entities - hosts, routers, users, services.

By regarding the network as a composition of dynamic systems, and by discovering the variables that control their macroscopic behavior, many classes of failure can be eliminated. If the vulnerabilities are not eliminated, it may be impossible to ever rely on the Internet as a global tool, and much of the promise of the Information Age may be lost.

Dynamic Behavior

This section concerns a particular class of system faults: positive feedback loops leading to resource unavailability. This is a generic problem with interconnected systems, one that can cause denial of service and the paralysis of security mechanisms. These faults should be considered part and parcel of secure system analysis, and they can often be anticipated by system modeling and monitoring.

Many computer system mechanisms that are relevant to security can be readily enumerated: the semantics and representation of authentication information, authentication protocols, access control and auditing, privacy mechanisms. Classical security is concerned with the correctness of the mechanisms. Intrusion detection takes a more dynamic view and looks for suspicious patterns, and database security addresses the problem of inference of the whole from its parts.

These all rely on a fairly static definition of a system — the mechanisms are static, the attributes that can be garnered by observation or logging are known in advance, the privileges and roles are defined uniformly for the domain of interest, which is limited to machines with particular IP addresses.

If we want to analyze a system for vulnerabilities that are related to dynamic conditions, or attributes that may affect the ability of the security mechanisms to operate (such as resource availability), we must create a model of that system, and we face the problem of finding parameters and mechanisms for the model.

Ideally, the system builders would identify all the parameters in advance. Realistically, they will specify much of what is important, but not all. The modeling task will involve incorporating new parameters constantly. For example, local network technology was stable for many years but is now changing rapidly, necessitating the incorporation of MTU, speed, and latency information as variables affecting subnets.

Because unknown components might be introduced via interconnection, system changes, or changes in user behavior, we need descriptions provided by the designers, but we don't know in advance what parameters are important. Further, the parameters of the model must change — we cannot model the entire system within itself [Eco]. We need to know if we can achieve a useful level of granularity for the modeling, if the model can be analyzed, how to trim away useless detail, and how to acquire new parameter sets.

When models are available, they can be used in conjunction with guards to discover failure modes, and to institute defensive measures. The model can predict possible failures, the guards can detect them, analysis can refine the guards, and a new model and specialized guards can be developed. We give two examples.

Example 1: A boot specification might say that a system will find some remote server, say for name/address resolution or for downloading boot code, without explicitly binding the server — it will be discovered via a broadcast protocol, for example. There might be several "deadly embrace" scenarios possible, but the LAN configuration will have appropriate server assignments so that this doesn't happen. Whenever the system configuration files change, however, the analysis that demonstrates this must be repeated. The assignment of the server roles for a LAN must be a discoverable attribute, if not an explicit property of the LAN.

As an alternative, the LAN configuration software must prevent the system administrator from introducing a deadly embrace either by specific server assignments or by configuring a client to use an algorithm that seeks a non-achievable set of servers.

A model of the systems and their rebooting behaviors can be used to validate a particular configuration, to prove that it has an achievable solution. An active guard might be generated to ensure that the reboot configuration files do not change and that at least one achievable server set remains in service.

Example 2: Electronic mail and newsgroup systems give users the privilege of storing material on remote systems and utilizing their network services for transmittal. These resources can be exhausted maliciously or inadvertently, resulting in denial of service for all users. Two common and similar abuses are the mailing list "mail loop" and the cross-posted "empty reply" news item. In each case, the original message is prefaced by headers and retransmitted to several sites. There is software for detecting and preventing this phenomenon, but it is specific to the particular services in question.

A generic model analysis could detect the possibility of the loops and the resource threat by simply "knowing" that a function for reinjecting messages exists anywhere in the message handling subsystem. The feedback loop

$$message = hdr \mid message$$

signals a potential resource exhaustion problem.

Once the possibility of the loop is identified, it is not necessary to change the software providing the service; one need only inject an active guard at the point of message delivery or storage. A generic guard which monitored network traffic for the simple case of replies could be easily generated.

When the guard sounded an alarm, human or machine assistant intervention could use attribute analysis to infer a pattern to use for specializing the guard to take an active role in preventing resource exhaustion. The specialization might lead to restricting traffic from the remote site initiating the reflecting messages, or it might temporarily impose a limit of three reflections before canceling the messages.

Designers must avoid inter-system behavior that leads to unbounded resource use, especially when it is exponential. On the other hand, stable systems do involve loops. A central challenge for designers is to identify and characterize the stable loops that define normal system behavior.

There are three basic types of control loops in dynamic systems. Their behaviors are described by the solutions to differential equations controlling them.

- Stable loops
- Unbounded positive growth
- Unstable growing oscillation

Being able to detect these when they arise as emergent behavior would be a valuable tool in network management. The key to the solution is to find the minimum variable set describing the loop, and to inject control mechanisms to eliminate or ameliorate the behavior.

It is customary to introduce negative feedback into a system to bound resource use. This often brings unwanted oscillatory behavior. One strategy to damp out oscillations is to introduce drag, but this approach is complicated in practice by the need for the *right amount* of negative feedback. The feedback must reflect system time constants, which frequently depend on external factors. Often there are unstable operating regimes, discovered only when the system is subjected to the real world.

Some Classic Loop Failures

Synchronization is another emergent behavior that can cause denial of service.

An inadvertent behavior in handling routing updates caused Internet outages for several minutes out of every update cycle. Sally Floyd at LBNL diagnosed this perplexing Internet behavior. She discovered that routers in the Internet backbone were synchronized by the I/O and computation burden of routing updates. The updates tended to occur at synchronized intervals, causing all routers to give priority to updates until all updates were processed. The synchronization was not predicted, making it difficult to diagnose.

A danger is that synchronization could be induced by malicious parties in order to exploit outdated information — key revocation is a time dependent security mechanism that could be thwarted in this way.

Whenever a process must lock a resource to use it, there is potential for inadvertent synchronization. If a process holds the resource too long due to an exceptional circumstance, several other processes may block. When the resource is finally released and the waiters are serviced, the waiters may become roughly synchronized. This can spoil performance of downstream subsystems that assume that work arrives at random intervals.

One way to prevent this synchronization is to introduce random delays into the admission policy for the locked resource, accepting some performance loss. A challenge is to detect synchronization at runtime, and add randomization only when needed. Of course, such a system is hard to analyze, and may be expected to develop different, more complex runtime pathologies — our version of the Heisenberg problem.

Deleterious loops often occur in restart sequences. Restarts are rarely tested thoroughly, and this can lead to severe problems when one system depends on another for its initial image. This illustrates the need for state analysis in the restart system when systems are composed — a degenerate loop leading to lack of progress is a pernicious failure and very difficult to detect before it occurs.

Exhaustive testing of all possible restart sequences of a complex system is usually impossible: There are too many combinations of partially available equipment, in too many possible intermediate stages of rebooting. Testing often requires that the system be out of service, and so must be limited to the most likely scenarios. When an unscheduled reboot is necessary, unpleasant surprises are frequent.

A classic reboot failure occurred in January 1990, when long distance telephone service on the East Coast was disrupted for hours. ATT was introducing new software into its telephone switches. The new software seemed solid, having been introduced gradually over a period of weeks. One difference in the new software was that rebooting a switch took longer. When the proportion of switches with the new software reached 25%, the reboot scheme became unstable. The problem surfaced when a switch went down for an unrelated cause. It began to execute automatic reboot, but an unanticipated interaction brought down the neighboring switch; the problem cascaded to bring down the entire network.

Recently the Domain Name System was revealed to have a lack-of-progress problem when a bug in some systems resulted in propagation of unsound information to all systems. The lack of integrity made correct translation of name to address impossible for many entries, and no automated detection or recovery was possible.

This is a degenerate loop, in which simple linear flow develops a parallel and incorrect flow. In this case correction cannot be done simply.

How to Combine without Loops?

Classic security models rely on non-interfering or partitioned systems. Unfortunately, the systems we need to analyze are *necessarily* interfering. The correct functioning of combined systems depends on stable behavior of shared parameters and the ability to damp out unbounded growth or oscillation. Restrictions on the semantics of combinations — such as the use of firewalls, packet filters, and a policy of mutual suspicion — are too limiting for complex systems.

We propose research into the study of the dynamic behavior of composed systems in order to protect their resources and to ensure that the composed system remains functional even under finely tuned attacks on its dynamic behavior.

A new form of system analysis is necessary to protect the security of the Internet, which is the largest composed system and arguably the largest possible composed system.

Hidden Variables

A central problem to be solved is the identification of variables that seem unimportant in the individual systems but which are crucial to the behavior of the composed system.

In the Internet routing example, the time to process an update and the average duration of the lock interval are hidden variables, that must be considered when composing systems.

The failure of the Ariane 5 rocket on its maiden launch illustrated the general principle that a minor variable in a subsystem can cause massive failure in a composed system. In this case, the problem was the overflow of a variable that could not overflow in Ariane 4; this dependency of a subsystem on a precondition was not checked in Ariane 5, largely because the subsystem was viewed as stable (i.e. error free) and it would be unnecessary and unwise to modify it. An additional factor was that the variable was related to a function that had no purpose in Ariane 5. Nonetheless, the variable overflowed, the subsystem and its backup failed simultaneously, and the main system sustained complete failure.

In 1990 a fire in Hinden, Illinois disrupted cross-country telephone service because a large number of the long distance cables passed through one building. Customers who thought they had purchased redundancy by acquiring separate telephone lines, often from multiple carriers, were surprised to find that there was little physical redundancy. The logically redundant lines were not truly redundant, because the underlying real-world implementation routed these lines through a single location.

The system models discussed earlier are valuable when analysts already have a fairly good idea of what the system is and how it works. Bringing the model to bear on real-world phenomena such as social behavior or physical infrastructure is an area that has not been seriously addressed. Yet we need to discover errors in ill-specified systems, particularly where the loops lead to denial of service attacks or compromise fault tolerance by eliminating redundancy.

And the Madness of Crowds

Human response times are part of the feedback process — our systems must avoid both resonance and inadvertent synchronization. Because human behavior is highly variable [MacKay], it can often explore regions of parameter space that were unanticipated by system designers.

An example from the 1970s: A newspaper had a computer system for processing want-ads. Operators accepted telephone ads and entered them into the computer system in real time. The central computer had just enough capacity to keep up with the data entry terminals. The operators discovered that they could cause a coffee break by all hitting "enter" simultaneously. The computer would be temporarily swamped by the processing to complete each ad, and the terminals would refuse data entry for a few minutes.

A more modern syndrome is the *Flash Flood* — a sudden focusing of usage patterns onto a small part of the Internet. Flash Floods are usually caused by human behavior [Coates], and we can expect this form of global attention focusing to be an increasingly common part of life on the Internet.

An example of this phenomenon occurred during the February 1996 chess match between Gary Kasparov and IBM's Deep Blue computer. The games were carried live on the Net. They attracted much more interest than anticipated, and any server having anything to do with the games was swamped with HTTP requests. IBM's WWW site for the match received millions of hits per day. The consequences in this case were merely annoyance and frustration, because no life-critical resource was in the path of the stampede.

In October 1987 the US stock market lost one third of its value in one day, largely due to a selling wave that was amplified by software for automated trading. This illustrates that combining complex systems is a dangerous activity — very simple failure modes can be introduced, even when one of the systems is considered to have stood the test of time (as one might argue that the modern stock market has).

This is not the classic problem of composing security policies; it is an issue of system stability more than correctness. It was not incorrect for all programs to sell at once, it was not incorrect for the market to honor the requests (insofar as it was capable). Rather, the system as a whole operated outside the expectations of the designers — the sellers created an unexpected positive feedback loop with themselves.

The implications obtained by extrapolating this behavior to a tightly connected global information system are astounding. Could we wake up one morning and find that information wars had been fought and won while we slept? Nations could disappear, whole economies might be eliminated.

Research Challenges

A pressing research challenge is to analyze specifications and code for potential resonances. Our specifications must explicitly include Time as a variable. For example, we must understand the interactions of timeouts in network communications protocols.

Another challenge is parameter discovery: Finding the important parameters that influence a system, while ignoring the unimportant ones. This can be particularly difficult when the parameter is not directly specified as a system variable, but is synthetic or emergent. Examples include *spare capacity* in all its guises.

Another challenge is runtime detection and recovery from looping behavior: such as how to break a loop safely, and detection of reboot or start-over failures such as the afflicted ATT switch. Damaging loops may be detected during design, test, or at runtime. Runtime detection is particularly attractive in conjunction with intrusion detection monitoring, and because it is independent of higher semantics, it has the potential to detect errors outside the model.

A theory of restart stability is urgently needed for designing today's distributed systems. This must extend down to the lowest levels of system restart.

Recovery from unmodeled failure (hardware, transient, or Ariane-like) is very hard to analyze, of course, but this is an area for research and formalization — how widely can we define safe and secure restart?

Security relevant system components, such as hierarchical and replicated public key servers or third party authenticators are of special concern and should motivate the beginnings of such research.

Infrastructure Representation

Because society is attempting to represent itself via digital descriptions to an ever increasing degree, this is an opportune time to begin research into incorporating physical engineering information into our computer security models. The expectation that the world of the future will be largely monitored and described electronically is not unreasonable; this is happening not by fiat but because it is the most convenient way to proceed.

The possibility that the world will represent its critical physical infrastructure for our inspection is double-edged. The aggregation of that information is essential for the modeling and vulnerability detection that we propose, but the information may also become available to inimical forces. The modeling effort and the active guards must be protected with the severe restrictions that characterize classical security efforts.

Social Phenomena

Social phenomena also fit into this modeling scenario, although humans are unlikely to provide blueprints of their behavior for online analysis any time soon. Nonetheless, the possibility of synchronized user agents, where agents are mobile software or other lightweight distributed processes, is a system failure mode of great concern.

In this case, the information aggregation is an even thornier problem, because we cannot expect Wall Street traders, for example, to indulge us by turning over their trading plans for public inspection. And in a world where several billion netizens are forming global communication cliques with high rapidity, we might not have time to detect and forestall destructive synchronized behavior.

For example, massive financial or social rearrangements could take place literally overnight, if a demagogue were to involve half the world's population in an apocalyptic teleconference. While we might not know what policy to follow in dealing with such scenarios, it is time to at least begin developing the technology for keeping pace with the brave new world that lurches toward us.

Warring Software

The active software guards of the previous section lend themselves to generalization as globally mobile security agents. These agents can detect vulnerable configurations, and they might also implement security countermeasures.

Security agents have the potential for extending the definition of "secure system" to a highly disparate collection of computing resources, insofar as they have any communications interconnection. They could allow the definition of a security policy for water supply management, for example. Water security software that "traveled" to computer sites dealing with water supply and evaluated security predicates dependent on inter-site cooperation — levels of impurities in supplies for adjacent neighborhoods — might be developed as a result of modeling efforts.

Of course, hostile forces are expected to engage in similar activities with less benevolent intentions. This leads to the scenario of constantly warring software: the security software agents vs. hostile software agents seeking to find and exploit vulnerabilities.

The dynamics of software wars will be difficult to predict. We will need to know how to build autonomous security agents, how they will interact with each other, and how to avoid having captured agents disclose sensitive information.

Environments for conducting this research safely are a prerequisite for experimentation, and the facilities will continue to be used to vet software agents before their release into the field.

The paradigms for such software may resemble espionage, but they are probably much closer in mechanism to biological immune systems and antigen recognition. The software warfare test environments are analogous to viral research laboratories.

Conclusion

As the networking world becomes more complicated, our security models must become more flexible, encompassing notions like partial failure and degraded service.

To cope with the complexity of the combined disparate elements that constitute large networks, those who profess expertise in security need to expand their analytic domain to include techniques from other areas: safety critical systems design, real-time systems, control theory, heuristic learning, engineering, physics, and psychology. Successfully melding selected elements from these fields into a new discipline is an exciting prospect.

References

Umberto Eco, "On the impossibility of drawing a map of the empire on a scale of 1 to 1", in *How to Travel with a Salmon*, 1994, Harcourt Brace and Company.

Charles Mackay, *Extraordinary Popular Delusions and the Madness of Crowds*, 1841. Reprinted by Harmony Books, 1980. A different edition is available from Project Gutenberg `<ftp://uiarchive.cso.uiuc.edu/pub/etext/gutenberg/etext96/ppdel10.txt>`

Robert M. Coates, "The Law", The New Yorker Magazine, 1947, reprinted in *The World of Mathematics*, J. R. Newman editor (1956), Simon & Shuster.