[1] S. P. Chung and A. K. Mok. Collaborative intrusion prevention. In *16th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pages 395-400, June 2007. [bib | DOI]

Intrusion prevention systems (IPSs) have long been proposed as a defense against attacks that propagate too fast for any manual response to be useful. In an important class of IPSs, the host-based IPSs, honeypots are used to collect information about attacks. The collected information will then be analyzed to generate countermeasures against the observed attack. Unfortunately, these IPSs can be rendered useless by techniques that allow the honeypots in a network to be identified ([1, 9]). In particular, attacks can be designed to avoid targeting the identified honeypots. As a result, the IPSs will have no information about the attacks, and thus no countermeasure will ever be generated. The use of honeypots is also creating other practical issues which limit the usefulness/feasibility of many host-based IPSs. We propose to solve these problems by duplicating the detection and analysis capability on every protected system; i.e., turning every host into a honeypot. In this paper, we will first lay out the necessary features of any scheme for such large scale collaboration in intrusion prevention, then we will present a framework called collaborative intrusion prevention (CIP) for realizing our idea of turning every host into a honeypot.

Keywords: groupware, security of datacollaborative intrusion prevention system, honeypots

[2] J. B. Grizzard. *Towards Self-Healing Systems: Re-establishing Trust in Compromised Systems*. PhD thesis, Department of Electrical and Computer Engineering, Georgia Institute of Technology, March 2006. [bib | http]

Computer systems are subject to a range of attacks that can compromise their intended operations. Conventional wisdom states that once a system has been compromised, the only way to recover is to format and reinstall. In this work, we present methods to automatically recover or self-heal from a compromise. We term the system an intrusion recovery system. The design consists of a layered architecture in which the production system and intrusion recovery system run in separate isolated virtual machines. The intrusion recovery system monitors the integrity of the production system and repairs state if a compromise is detected. A method is introduced to track the dynamic control flow graph of the production system guest kernel. A prototype of the system was built and tested against a suite of rootkit attacks. The system was able to recover from all attacks at a cost of about a 30% performance penalty.

Keywords: virtual machine, end-user security, intrusion detection, intrusion recovery, rootkit

[3] S. Jain, F. Shafique, V. Djeric, and A. Goel. Application-level isolation and recovery with solitude. *SIGOPS Oper. Syst. Rev.*, 42(4):95-107, May 2008. [bib | DOI]

When computer systems are compromised by an attack, it is difficult to determine the precise extent of the damage caused by the attack because the state changes made by an attacker and those made by regular users can be closely intertwined. This problem occurs due to implicit sharing in operating systems, and it can be especially severe for persistent state. In particular, the file system provides a single namespace that when compromised can have cascading effects on the entire system, making intrusion analysis and recovery a time-consuming and error-prone process.

In this paper, we present Solitude, an application-level isolation and recovery system that is designed to both limit the effects of attacks and simplify the post-intrusion recovery process. Solitude uses a copy-on-write filesystem to provide a transparent, restricted privilege isolation environment for running untrusted applications, and it uses an explicit file sharing mechanism across the isolation environments that limits attack propagation without compromising functionality. Solitude provides two modes of recovery. If a sandboxed application proves to be untrustworthy, a course-grained recovery method allows easily removing the footprint of the software. However, if a user mistakenly moves malicious files to the trusted environment via explicit file sharing, then Solitude uses data dependency tracking to allow fine-grained recovery.

[4] R. Janakiraman, M. Waldvogel, and Q. Zhang. Indra: a peer-to-peer approach to network intrusion detection and prevention. In *Twelfth IEEE International Workshops on Enabling*

Technologies: Infrastructure for Collaborative Enterprises (WET ICE), pages 226-231, June 2003. [bib | DOI]

While the spread of the Internet has made the network ubiquitous, it has also rendered networked systems vulnerable to malicious attacks orchestrated from anywhere. These attacks or intrusions typically start with attackers infiltrating a network through a vulnerable host and then launching further attacks on the local network or Intranet. Attackers rely on increasingly sophisticated techniques like using distributed attack sources and obfuscating their network addresses. On the other hand, software that guards against them remains rooted in traditional centralized techniques, presenting an easily-targeted single point of failure. Scalable, distributed network intrusion prevention techniques are sorely needed.

We propose Indra - a distributed scheme based on sharing information between trusted peers in a network to guard the network as a whole against intrusion attempts. We present initial ideas for running Indra over a peer-to-peer infrastructure to distribute up-to-date rumors, facts, and trust information in a scalable manner.

Keywords: Internet, client-server systems, intranets, safety systems, security of data information sharing, intranet, intrusion detection and rapid action, local network, network intrusion detection, network prevention, peer-to-peer infrastructure

[5] M. E. Locasto, S. Sidiroglou, and A. D. Keromytis. Software self-healing using collaborative application communities. In *NDSS*, 2006. [bib | .pdf]

Software monocultures are usually considered dangerous because their size and uniformity represent the potential for costly and widespread damage. The emerging concept of collaborative security provides the opportunity to re-examine the utility of software monoculture by exploiting the homogeneity and scale that typically define large software monocultures. Monoculture can be leveraged to improve an application's overall security and reliability. We introduce and explore the concept of *Application Communities*: collections of large numbers of independent instances of the same application. Members of an application community share the burden of monitoring for flaws and attacks, and notify the rest of the community when such are detected. Appropriate mitigation mechanisms are then deployed against the newly discovered fault. We explore the concept of an application community and determine its feasibility through analytical modeling and a prototype implementation focusing on software faults and vulnerabilities.

Specifically, we identify a set of parameters that define application communities and explore the tradeoffs between the minimal size of an application community, the marginal overhead imposed on each member, and the speed with which new faults are detected and isolated. We demonstrate the feasibility of the scheme using Selective Transactional EMulation (STEM) as both the monitoring and remediation mechanism for low-level software faults, and provide some preliminary experimental results using the Apache web server as the protected application. Our experiments show that ACs are practical and feasible for current applications: an AC of 15,000 members can collaboratively monitor Apache for new faults and immunize all members against them with only a 6% performance degradation for each member.

[6] D. J. Malan and M. D. Smith. Host-based detection of worms through peer-to-peer cooperation. In *Proceedings of the 2005 ACM Workshop on Rapid Malcode (WORM)*, pages 72-80, November 2005. [bib | DOI]

We propose a host-based, runtime defense against worms that achieves negligible risk of false positives through peer-to-peer cooperation. We view correlation among otherwise independent peers' behavior as anomalous behavior, indication of a fast-spreading worm. We detect correlation by exploiting worms' *temporal consistency*, similarity (low temporal variance) in worms' invocations of system calls. We evaluate our ideas on Windows XP with Service Pack 2 using traces of nine variants of worms and twenty-five non-worms, including ten commercial applications and fifteen processes native to the platform. We find that two peers, upon exchanging snapshots of their internal behavior, defined with frequency distributions of system calls, can decide that they are, more likely than not, executing a worm between 76% and 97% of the time. More importantly, we find that the probability that peers might err, judging a non-worm a worm, is negligible.

[7] D. Schnackengerg, H. Holliday, R. Smith, K. Djahandari, and D. Sterne. Cooperative intrusion traceback and response architecture (CITRA). In *Proceedings of the DARPA Information Survivability Conference & Exposition II (DISCEX)*, volume 1, pages 56-68, 2001. [bib | DOI]

The Cooperative Intrusion Traceback and Response Architecture (CITRA) was originally developed as an infrastructure for integrating network-based intrusion detection systems, firewalls, and routers to trace attacks back to their true source and block the attacks close to that source. Prototype implementations of CITRA have proven useful for integrating other security mechanisms in support of automated response to both intrusions and other changes in security status of a system. This paper provides an overview of CITRA policy mechanisms and how CITRA integrates diverse security technologies to improve system defense.

Keywords: computer network management, security of data, software architecture, supervisory programs, system monitoringCITRA, CITRA policy mechanisms, Cooperative Intrusion Traceback and Response Architecture, firewalls, network-based intrusion detection systems, routers, security technologies, system defense

[8] V. Shmatikov and M.-H. Wang. Security against probe-response attacks in collaborative intrusion detection. In *Proceedings of the Workshop on Large Scale Attack Defense (LSAD)*, pages 129-136, 2007. [bib | DOI]

Probe-response attacks are a new threat for collaborative intrusion detection systems. A probe is an attack which is deliberately crafted so that its target detects and reports it with a recognizable “fingerprint” in the report. The attacker then uses the collaborative infrastructure to learn the detector's location and defensive capabilities from this report.

We analyze the fundamental tradeoff between the ability of a collaborative network to detect epidemic threats and security of individual participants against probe-response attacks. We then design and evaluate a collaborative detection system which provides protection against probe-response attacks. Unlike previously proposed collaborative detection networks, our system supports alert sharing while limiting exposure of members' identities.

[9] G. Wells, A. Chalmers, and P. Clayton. Linda implementations in java for concurrent systems. *Concurrency and Computation: Practice and Experience*, 16(10):1005-1022, August 2004. [bib | DOI | http]

This paper surveys a number of the implementations of Linda that are available in Java. It provides some discussion of their strengths and weaknesses, and presents the results from benchmarking experiments using a network of commodity workstations. Some extensions to the original Linda programming model are also presented and discussed, together with examples of their application to parallel processing problems.
