# ;login:

THE MAGAZINE OF USENIX & SAGE
November 2001 • Volume 26 • Number 7

Special Focus
Issue: Security
Guest Editor: Rik Farrow

inside:

**INTRUSION DETECTION**
Survivability with a Twist
by Sven Dietrich

USENIX & SAGE
The Advanced Computing Systems Association &
The System Administrators Guild

# survivability with a twist

**by Sven Dietrich**

Sven Dietrich is a member of the technical staff at the Carnegie Mellon Software Engineering Institute in Pittsburgh, PA. When he does not snort packets, he conducts research in computer security and survivable network technology.

*spock@cert.org*

Glancing at the hexadecimal pattern on the screen, I was quite happy with my intrusion detection system: I had netted some Ramen worm exploit code and handed it over to the appropriate incident response team. It had caught itself in one of the flytraps that was listening on the border network. This was quite surprising in light of the stumbling blocks I had encountered in the preceding few weeks.

## The Mission

This catch should not have been that surprising. During the 18 months preceding this incident I had built up a network analysis and intrusion detection system (IDS) geared toward dealing with distributed denial of service (DDoS) attacks.[1] One of the difficulties I had encountered during this time was maintaining the set of systems to the point of self-sufficiency and self-repair, within reason at least, under common attack scenarios.

One of the fears of a security expert is the loss or absence of network forensics. "Network traces? Sorry, the network analysis systems are down/not available/not up yet." Network forensics provide clues to a security expert in the same way a detective would use clues to solve a mystery or a crime. Typical examples are packet flows collected at the router level, TCP-wrapper[2] logs at the host level, and intrusion detection system logs. Frequently, an intruder will try to disable monitoring systems prior to launching the real attack so that it cannot be reconstructed. The mission is to be able to reconstruct the incident, if not fully, at least partially.

## Departure from Basic Security Concepts

The computer security field assumes a binary model, where a system either resists the attack or is compromised. Many years of research have gone into the effort of building higher and higher walls that the intruders cannot overcome. This inevitably leads to an arms race with the attackers, as they attempt to undermine, breach, or otherwise transgress those walls.

Rather than try to compete in this game, the field of survivability takes a different spin. The concept of survivability, as currently defined, is the ability of a system to fulfill its mission in the presence of attacks, failures, and accidents.[3] In the survivability worldview, one accepts the compromise or failure of one or more components of the system, as long as the mission can be completed, even if a reduced or degraded mode is necessary. The field explores various techniques for building survivable systems, including techniques borrowed from dependability and fault-tolerance, among others. For more philosophical foundations on the definition of the term, please refer to the references.[4]

## Motivation

As I had investigated some initial incidents, I was often faced with the frequent absence of evidence. I wanted to be able to analyze most aspects of the attack, whether it was denial of service or not. Therefore, I needed a system that could operate, even partially, in the presence of a DDoS attack to gather as much information on the attack as possible. Even the local legitimate "security scans" that would periodically check for unpatched and insecure systems could rattle the hosts quite a bit.

## The Approach

Since my budget was small, my choices were somewhat limited. I decided to take baby steps and proceed empirically. Based on my first setup (a sensor piping data back to a data collection host over a secure shell connection), I identified operating systems and hosts stable enough for my purpose. However, I could not stop there. That summer was very hot and the local power company kept implementing rolling brownouts and blackouts. I wanted to look at all the components of this little system: what hosts would come back up and how would they recover after the "uninterruptible" power system lost power?

How robust was my setup really? I had found several uninterruptible power systems and distributed them among the hosts. As the system lived in a set of racks, colleagues would repeatedly trip over power and network cables as they were trying to access their own prototype systems. Of course this kept me on my toes. I had managed to build a set of redundant systems – replication of services, excellent! A geographic dispersion of the systems and their associated data files followed to minimize the impact of a local failure. My next step was to create a heterogeneous environment, so that one potential attack would not necessarily affect all of the systems involved, which was achieved by selecting a slightly different operating system for each host. (For those die-hard operating system aficionados who must know, the two operating systems were OpenBSD and NetBSD.)

These so-called in-band attacks would travel in the data stream that one is recording and would act on a listening tool itself . . .

## Slicing the Network Stream

Since a complete loss of disks, systems, or other components could not be excluded from consideration, I wanted to record the data in several ways, so as to "rebuild" events later. It was absolutely critical that I be able to return to any given point in the data sets so I could perform the aforementioned network forensics. Gathering packet headers using argus,[5] full-packet recording using tcpdump,[6] keyword detection across protocols and ports using ngrep,[7] and signature detection and anomaly detection using Snort[8] were some methods for the slicing. Effectively, one performs several types of data reduction while one drinks from the fire hose that is the network stream.

Some types of attack, such as a buffer overflow in tcpdump, could disable, compromise, or blind intrusion detection systems. These so-called in-band attacks would travel in the data stream that one is recording and would act on a listening tool itself or affect a susceptible operating system, causing it to hang or crash. By varying the way one looked at the data, the chances that at least a few monitoring tools would remain orthogonal to the problem were relatively high. Even that was not sufficient; I wanted to reconfigure my system in the event of loss, corruption, or compromise of one of my components.

## The Interconnection Network

In order for the different hosts to exchange status information, network stream data and logs, I built a redundant interconnection network, opaque to the outside observer. By performing queries, from the simple pinging of a host to a more complex one such as interrogating via a series of TCP/IP client-server messages whether a particular service was still running, the other hosts were given a view of the system as a whole. In the event of a non-response, the remainder of the system could then reconfigure by firing up appropriate replacement processes, such as a new packet logger or signature detector, without impacting the current role of the host. Each host had a primary role assigned to it, with a list of secondary and tertiary roles that would be assumed in case of the loss of a primary-role host. Several strategies were conceivable. One very simplistic one, noticing that a primary-role host had vanished and taking the role of that host if the local host was capable of doing so, seemed to work well enough. Another one, pushing out the newly learned information to the remaining known hosts, led to a collaborative decision.

Network time information, a source of accurate time which is important both for correlating events across the globe and coordinating between hosts, relied on two separate sources: a radio clock receiving the official time signal and a GPS-based clock, both on local networks. These timings were crucial for the analysis of the collected exploit and attack data. More details describing this system can be found in an upcoming paper.[9]

## The Twist

Sitting in deep thought in front of my monitor, I recalled my first day back after my vacation. Two of my components, computationally powerful yet non-critical as they were, had for unexplained reasons been disconnected from both the interconnection and the border networks. I had discounted it as a mistake I had made since I had put the system into survivable mode before going on my vacation. Actually, it turned out to be a confirmation of the security credo that insider threats account for a significant number of the problems. As the non-critical components were removed from the system, the monitoring system proceeded to reconfigure itself in order to compensate for that loss and make the mission survive.

What I had designed to work against the outsider threat ended up working against an insider threat, intentional or not, and it was put to the test in a perfect setting: I had gathered sufficient network data to understand the Ramen incident and suggest mitigation measures.

The mission had succeeded.

REFERENCES

1. Sven Dietrich, "Scalpel, Gauze, and Decompilers: Dissecting Denial of Service (DDoS)," *;login:* (November 2000), theme issue on security.

2. Wietse Venema, "TCP Wrapper – Network Monitoring, Access Control, and Booby Traps." 3rd USENIX Security Symposium, Baltimore, MD (September 1992), *http://www.porcupine.org/*.

3. R.J. Ellison, David Fisher, Rick Linger, Howard Lipson, Tom Longstaff, Nancy Mead, "Survivable Network Systems: An Emerging Discipline," Software Engineering Institute Technical Report No. CMU/SEI-97-TR-013 (November 1997).

4. CERT Survivability Research page, *http://www.cert.org/research/*; John C. Knight, Matthew C. Elder, "Fault Tolerant Distributed Information Systems," International Symposium on Software Reliability Engineering, Hong Kong (November 2001); Jonathan Millen, "Local Reconfiguration Policies," IEEE Symposium on Security and Privacy, Oakland, CA (May 1999).

5. argus: *ftp://ftp.andrew.cmu.edu/pub/argus/*.

6. tcpdump: *http://www.tcpdump.org/*.

7. ngrep: *http://sourceforge.net/projects/ngrep/*.

8. Marty Roesch, "Snort – Lightweight Intrusion Detection for Networks," USENIX LISA XIII (December 1999), *http://www.snort.org/*.

9. Sven Dietrich, "AMPLIFIDS – A Survivable Ensemble," in preparation.