Using embedded sensors for detecting network attacks*

Florian Kerschbaum Eugene H. Spafford Diego Zamboni

Center for Education and Research in Information Assurance and Security
1315 Recitation Building
Purdue University
West Lafayette, IN 47907-1315
{kerschf,spaf,zamboni}@cerias.purdue.edu

September 25, 2000

Abstract

Embedded sensors for intrusion detection consist of code added to the operating system and the programs of the hosts where monitoring will take place. The sensors check for specific conditions that indicate an attack is taking place, or an intrusion has occurred. Embedded sensors have advantages over other data collection techniques (usually implemented as separate processes) in terms of reduced host impact, resistance to attack, efficiency and effectiveness of detection. We describe the use of embedded sensors in general, and their application to the detection of specific network-based attacks. The sensors were implemented in the OpenBSD operating system, and our tests show a 100% success rate in the detection of the attacks for which sensors were instrumented. We discuss the sensors implemented and the results obtained, as well as current and future work in the area.

1 Introduction

The field of intrusion detection has received increasing attention in recent years. One reason for this is the explosive growth of the Internet and the large number of networked systems that exist in all types of organizations. The increase in the number of networked machines has lead to an increase in unauthorized activity [6], not only from external attackers, but also from internal attackers, such as disgruntled employees and people abusing their privileges for personal gain [27].

To detect and counteract unauthorized activity, it is desirable for network and system administrators to monitor activity in their networks. Because even a single host can generate several megabytes of logging and audit data in a single day, it is desirable to have tools that can automatically monitor computer systems and detect when unauthorized activity is taking place. These tools are commonly known as intrusion detection systems.

In the last few years a number of intrusion detection systems have been developed, both in the commercial and academic sectors. These systems have used different approaches to detecting unauthorized activity, and have given us some insight into the problems that still have to be solved before we can have intrusion detection systems that

*Portions of this work were supported by sponsors of CERIAS. This paper was published in Proceedings of the First ACM Workshop on Intrusion Detection Systems, November 2000, Athens, Greece.

are useful and reliable in production settings for detecting a wide range of intrusions.

Most of the existing intrusion detection systems have used central data analysis engines [e.g. 10, 17] or per-host data collection and analysis components [e.g. 15, 26]. Even systems designed using software agents [e.g. 2, 3] have in practice implemented agents as separate processes. All of these approaches are subject to the following problems:

- They continuously use additional resources in the system they are monitoring, even when there are no intrusions occurring.
- Because the components of the intrusion detection system are implemented as separate processes, they are subject to tampering. An intruder can potentially disable or modify the programs running on a system, rendering the intrusion detection system useless or unreliable.

In this document we show the use of embedded sensors to detect a specific set of network-based attacks. These sensors are built into the code of the operating system. This makes them use additional resources only when they are executed or triggered, and be more resistant to tampering.

1.1 Intrusion Detection

Intrusion detection is defined as "the problem of identifying individuals who are using a computer system without authorization (i.e., 'crackers') and those who have legitimate access to the system but are abusing their privileges (i.e., the 'insider threat')" [21]. We add to this definition the identification of *attempts* to use a computer system without authorization or to abuse existing privileges. Our working definition matches the one given by Heady et al. [14], where an intrusion is defined as "any set of actions that attempt to compromise the integrity, confidentiality, or availability of a resource," disregarding the success or failure of those actions.

The definition of the word *intrusion* in a common dictionary [19] does not include the concept of an insider abusing his or her privileges to perform unauthorized actions, or attempting to do so. A more accurate phrase to use is *intrusion and insider abuse detection*. In this document we use the term *intrusion* to mean both intrusion and insider abuse.

An intrusion detection system is a computer system (possibly a combination of software and hardware) that attempts to perform intrusion detection, as defined above. Most intrusion detection systems try to perform their task in real time [21], but there are also intrusion detection systems that do not operate in real time, either due to the nature of the analysis they perform [e.g. 16] or because they are geared for forensic analysis [11, 36].

Intrusion detection systems are usually classified as host-based or network-based [21]. Host-based systems base their decisions on information obtained from a single host (usually audit trails), while network-based systems obtain data by monitoring the traffic in the network to which the hosts are connected.

The definition of an intrusion detection system does not include preventing the intrusion from occurring, only detecting it and reporting it to an operator. There are some intrusion detection systems (for example, NetRanger [8]) that try to react when they detect an unauthorized action. This reaction usually includes trying to contain or stop the damage, for example, by terminating a network connection.

1.2 Detecting network attacks

In the last few years, the number of remote attacks using the TCP/IP protocol suite [9] has grown in number. This has been caused in large part by the known design flaws in the TCP/IP protocol suite [4], and by the explosive growth of systems using TCP/IP that are connected to the Internet. A large number of network attacks have been made public, including source code for exploiting them.

Some of these attacks exploit faults in the networking code of operating systems, while others exploit basic vulnerabilities of the TCP/IP protocol suite.

Many network attacks are denial-of-service attacks, which cause disruption of the services provided by the attacked host. Others allow the attacker to gain unauthorized access to resources of services offered by the attacked host, and others allow the attacker to gain information about the host that can be used to mount further attacks.

This document focuses on an effort to provide real-time detection of a set of common network attacks. Other intrusion detection systems have implemented detection of these attacks, but most of them have implemented the detectors as independent processes that gather and interpret packets flowing on the network. This approach imposes an extra load on the system where the intrusion detection system is running (because it has to process all the packets that pass through the network), introduces an extra delay in the detection of the attacks (because the packet data has to be passed from the networking layers of the operating system to a user-level process for interpretation) and is subject to attacks as described by Ptacek and Newsham [28].

We describe a different approach to the implementation of the detectors: inserting them in the networking code of the attacked host. Small detectors, called *embedded sensors*, for specific attack signatures are placed within the Unix kernel code that processes network packets. This takes advantage of the processing that the kernel already does, therefore reducing system load. Also, because the sensors are placed in the code segment determined to be responsible for the flaw, they can reliably indicate an attack, reducing the number of both false positives and false negatives. Finally, because the sensors become part of the kernel code, they are very difficult to disable or tamper with.

2 Embedded sensors for intrusion detection

We define an *embedded sensor* as a piece of software (conceivably aided by some hardware components) that monitors a specific variable, activity or condition of a host. Because the sensor monitors the system directly and not through an audit trail or through packets on a network, we say that it performs *direct monitoring*, and because the sensor is part of the program or system it monitors, we say that it is an *internal sensor* [33].

Embedded sensors are built by modifying the source code of the program that will be monitored. Sensors should be added to the code at the point where a security problem can be detected in the most efficient way by using the data available at that moment. If implemented correctly, the sensor will be able to determine whether an attack is taking place by performing very simple checks.

2.1 How sensors work

Figure 1 shows an example of a very simple sensor. The code on the left is potentially vulnerable to a buffer-overflow attack [1] because the value of the HOME environment variable is being copied to a buffer without checking its length. On the right, lines 2–6 have been added, and constitute a sensor. This sensor computes the length of the HOME environment variable, and if it is longer than the buffer into which it will be copied it generates an alert. This example assumes that the function `log_alert` has been defined elsewhere. Of course, the string "buffer overflow" is provided only as an example—in a real sensor, a more descriptive message would be provided.

The example in Figure 1, as per our definition of sensor, does not try to prevent the overflow from happening, it only tries to note its occurrence. Potentially, sensors could try to stop the intrusions they detect. For example, our sample sensor could cut the HOME environment variable to 255 characters to ensure that it will fit in the allocated

```
1  char buf[256];
2  strcpy(buf, getenv("HOME"));
```

Code before inserting sensor

```
1  char buf[256];
2  {
3    if (strlen(getenv("HOME"))>255) {
4      log_alert("buffer overflow");
5    }
6  }
7  strcpy(buf, getenv("HOME"));
```

Code after inserting sensor

Figure 1: Example of vulnerable code before and after inserting an embedded sensor.

buffer. However, the effects of sensors modifying data or altering the program flow are much harder to analyze. For this reason, our current work has focused on using embedded sensors only for detection.

This example gives an idea of how embedded sensors work in general: they look at the information available in the program to determine if an intrusion has occurred, or if an attack is taking place. If such a condition is found, an alert is generated.

2.2 What is good about embedded sensors

Using embedded sensors for data collection in an intrusion detection system has the following advantages over using external sensors (implemented as separate programs):

- They can obtain data at its source, or at the place where it is more convenient to obtain. Data does not have to traverse through an external program interface for the sensor to get it, because the sensor can read it directly off the program's data structures. This reduces the delay between the generation of the data and when the intrusion detection system can make use of it.

- Data is never stored on an external medium before the sensor obtains them, therefore the possibility of an intruder modifying the data to hide its tracks (for example, an intruder modifying a log file) is practically eliminated.

- Embedded sensors are part of the program they monitor, therefore they cannot be disabled (as it is possible with an external sensor, which can be killed or disabled) and they are very difficult to modify to produce incorrect results.

- Embedded sensors can analyze the data (at least partially) at the moment they are acquired, therefore reducing impact on the host due to data traversing between different components (and possibly between different contexts in the operating system, such as user and kernel contexts in a Unix system).

- Embedded sensors can look for very specific conditions that signal attacks, instead of reporting generic data for analysis. This means that the amount of data that needs to be reported, collected and analyzed by higher-level analysis engines is much smaller, reducing both network traffic and processing load.

- Embedded sensors are only executed when the task they perform is required (this is, when the section of code they are a part of is executed) and they are not executed as separate processes or threads, but as part of the monitored program. For this reason, they do not cause a continuous CPU usage overhead on the host. This makes it possible to incorporate a much larger number of sensors on a single host than it would be possible on a system where all the sensors are implemented externally [34].

- Embedded sensors can look for attempts to exploit a vulnerability, independently of

whether the vulnerability actually exists in the host where the sensor is running. For this reason, embedded sensors can detect attempts at intrusions that have already been fixed, or even that are specific to a different platform or operating system (for example, a sensor in a Unix system could detect attacks specific to Windows NT).

2.3 What is bad about embedded sensors

Embedded sensors have the following disadvantages with respect to external sensors:

- Their implementation requires having access to the source code of the operating system and its programs.
- They are more difficult to implement, because they require modifications to the source code of the operating system or its programs.
- They have to be implemented in the same language as the program in which they are being incorporated.
- They have the potential of causing much more damage to the system if they contain errors. A piece of code embedded in the kernel of a Unix system can bring the whole system down if it acts incorrectly.
- Improperly implemented sensors (for example, one that repeatedly executes a costly operation) can have detrimental effects on the performance of the system. Because they can exist at very low levels of the operating systems, sensors also have the possibility of disrupting operations that depend on precise timing.
- They are difficult to port to different operating systems.

3 Embedded sensors for detecting network attacks

We have implemented a number of embedded sensors for detecting common network attacks. In this section we describe this implementation and the results obtained.

3.1 Implementation platform

The sensors have been implemented in the OpenBSD 2.6 operating system [18, 24], one of the three main variations of the free operating systems based on the BSD distribution. This operating system was chosen for the following reasons:

- The source code is available, which makes it easy to instrument the sensors both in the kernel and in the system programs. Extensive documentation is available [18, 35] about the internals of the kernel and its networking code.
- The source tree is centrally managed and distributed as a single directory tree. This makes it more manageable than Linux, for example, where source code for different components of the operating system are distributed as separate packages. The source tree of OpenBSD closely mimics the layout of the system itself, making it easy to locate the code for different programs and subsystems.
- The OpenBSD project is known for its attention to security, and is the first widely used operating system that has gone through an extensive line-by-line security audit process. Most of the security problems for which sensors will be implemented have already been fixed in OpenBSD. By looking at the security patches and at the change log for each file it is easier to locate the portions of code where the problems existed. This helps in determining where the sensors for each intrusion have to be placed. In fact, in many cases

the code that fixed the problem could be easily identified, and the only thing that needed to be done was adding the code for producing a notification. Finally, because the problems themselves no longer exist, it is easier to try attacks against the instrumented system without worrying about the adverse effects they could have on the system.

- It runs on multiple architectures, including low-end Intel and Sparc systems, which makes it easier to set up a network of instrumented systems for experimentation.

3.2 Sources of information

The sensors implemented were mapped to entries in the CVE (Common Vulnerabilities and Exposures) database [7, 20]. The CVE is not a taxonomy or a classification scheme, and is used only as a fairly complete and recognized list of known vulnerabilities and attacks. Linking each sensor to a CVE entry facilitates discussion and reference, and ensures that no repeated sensors are implemented. For the work reported in this document, version 20000118 of the database was used.

Information about the attacks themselves, including exploits, was gathered from the common sources on the Internet [5, 25, 29, 32, 37].

As a special case, sensors for different variants of port scanning [13] were implemented. Port scans are not considered attacks by themselves, but are commonly a prelude to an attack, therefore useful to detect. Port scans do not have CVE numbers.

3.3 Logging mechanisms

The sensors as reported in this document were concerned more with correctness than with the mechanism used for logging their results. For this reason, these sensors use simple `printf` statements to produce reports. Within the OpenBSD kernel, these statements result in the messages being printed to the system console and stored in the system message buffer (also accessible through the standard **dmesg** command).

However, work is also underway to provide appropriate logging mechanisms for embedded sensors. We are working on determining which logging techniques are more appropriate for the kinds of information generated by the sensors and for the ways in which that information is used. As of this writing, a first implementation of a sensor-specific logging mechanism has been finished, and the sensors are being modified to use that mechanism instead of the standard `printf` call.

3.4 Sensors implemented

In this section we describe the attacks for which sensors have been implemented, and comment on some specific issues about each one of them. For each attack, its CVE number is mentioned, and in most cases, the section of code in which the sensor was implemented is shown (in many cases the code has been reformatted for space). Thus, we can see that most sensors are very short and simple, yet they provide very advanced detection capabilities (as shown in Section 3.5).

The added or modified lines have been highlighted in each code section. All the sensors have been wrapped in `#ifdef` directives and in an `if` clause, so that they can be disabled both at compile time and at run time. We are in the process of integrating the run-time control variables to the kernel parameters mechanism available in OpenBSD, through which some kernel parameters can be modified at run time without the need to recompile the kernel. The ability to disable the sensors at runtime may not be completely desirable, as it offers the possibility for an attacker to disable the sensors if he manages to obtain sufficient privileges in the system. For the purposes of testing, however, the capability of enabling and disabling sensors at runtime is necessary.

3.4.1 Land (CVE-1999-0016)

This attack consists of a TCP SYN packet sent to an open port on the attacked machine with the source IP address and port set to the same as the destination. On a vulnerable machine, this puts the kernel into an infinite loop sending ACK packets to itself in a interrupt level that prevents all user processes from running, effectively locking the machine.

OpenBSD 2.6 filters those packets shortly after they have entered the system while the socket is still in state LISTEN. The sensor has been placed there, right before the packet is dropped.

```
case TCPS_LISTEN: {
...
  if (ti->ti_dst.s_addr ==
      ti->ti_src.s_addr) {
     /* ESP */
#ifdef ESP_LAND
     if (esp.sensors.land)
        printf("LAND attack\n");
#endif
        goto drop;
     }
... }
```

3.4.2 Teardrop (CVE-1999-0052)

This attack consists of two overlapping IP fragments. The first is a large fragment, while the second fragment is short and "fits" within the first fragment (it starts at a later offset but ends before the first fragment). The IP reassembly routine in vulnerable systems gets confused by the conflicting offsets and tries to allocate memory for the new data using a negative value, which crashes the system. There are multiple derivatives of this attack that exploit various fixes, devices and environments. One of those will be discussed in the next section.

In OpenBSD 2.6 a special check for this was put in the reassembly code. Again the sensor has been placed there.

```
i = p->ipqe_ip->ip_off +
    p->ipqe_ip->ip_len -
    ipqe->ipqe_ip->ip_off;
if (i > 0) {
  if (i >= ipqe->ipqe_ip->ip_len) {
     /* ESP */
#ifdef ESP_TEARDROP
     if (esp.sensors.teardrop) {
        printf("TEARDROP attack\n");
        ...
     }
#endif
    goto dropfrag;
  }
```

3.4.3 PIX DoS (CVE-1999-0157)

This exploits the same bug as Teardrop, but it was developed specifically for the Cisco PIX firewall, which does not filter fragments with offset not equal to zero. A special sensor for this has been placed at the dots in the previous code segment.

```
if (p->ipqe_ip->ip_off > 0)
  printf("PIX flavored\n");
```

3.4.4 TCP RST DoS (CVE-1999-0053)

This attack sends a forged TCP RST packet and disconnects an established TCP connection, whose port numbers the attacker knows or guesses [12]. Vulnerable TCP stacks do not check the sequence number of the RST packet and reset a valid connection. The sensor is placed in the check for the attack that already exists in the code for RST packet handling.

```
if (tiflags & TH_RST) {
  if(ti->ti_seq!=tp->last_ack_sent){
    /* ESP */
#ifdef ESP_RSTDOS
    if (esp.sensors.rstdos)
       printf("TCP RST DOS attack\n");
#endif
    goto drop;
  }
  ... }
```

3.4.5 Ping of Death (CVE-1999-0128)

The ping of death is a fragmented ICMP echo request whose reassembled length is longer than the maximum length of an IP packet. This is a comparatively old vulnerability, therefore a detailed description is omitted here. OpenBSD 2.6 checks for this vulnerability and the sensor is placed there.

```
if ((next+(ip->ip_hl<<2))
     > IP_MAXPACKET) {
  /* ESP */
#ifdef ESP_PINGOFDEATH
  if (esp.sensors.pingofdeath &&
      ip->ip_p == IPPROTO_ICMP)
    printf("PING'O'DEATH attack\n");
#endif
  ... }
```

3.4.6 NetBSD TCP Race Condition (CVE-1999-0396)

This attack stalls a blocking server by deleting a socket after it is seen by the server, but before it is accepted [22]. The attacker completes the three-way handshake and then immediately disconnects. The advisory [22] mentions using RST packets to remove the socket, but FIN packets could also be used, as in a fast port scanner like **nmap** [13]. This attack has to be completed between the awakening of the server after the handshake and before it calls the `accept` function on the new socket. An attacker would most likely need more than one attempt to achieve this. It would also be targeted to one specific TCP server port. If both conditions are true, the sensor that also detects port scans will classify this as a race attack. FIN and RST packets with the initial sequence number plus one in the established state are reported to the port scan detector, which is discussed in a Section 3.4.15.

3.4.7 Linux Blind Spoofing (CVE-1999-0414)

Some versions of the Linux TCP implementation are vulnerable to spoofing of TCP connections without completing the three-way handshake [23]. The attacker does not need to guess the initial sequence number and can spoof connections from addresses whose connections remain "invisible." The sequence of packets for the attack is: one SYN packet, then possibly multiple packets, all without ACK flag, and finally one FIN packet without the ACK flag, or one packet with FIN and all the payload data. Although the connection remains in the SYN_RECEIVED state, the FIN packet will cause the delivery of queued data to the socket and its application. OpenBSD 2.6 ignores the data of packets without the ACK flag. The sensor raises an alarm on FIN packets without ACK flag in the state SYN_RECEIVED.

```
case TCPS_SYN_RECEIVED:
  if (tiflags & TH_ACK) {
    ...
  } else {
    /* ESP */
#ifdef ESP_BLINDSPOOF
    if (esp.sensors.blindspoof) {
      if (tiflags & TH_FIN)
        printf("BLIND SPOOF attack\n");
    }
#endif
  }
```

3.4.8 Win Nuke (CVE-1999-0153)

This attack sends out-of-band data to TCP port 139 (**netbios-ssn**). Most vulnerable systems (although this is a Windows-specific attack, other products, such as SCO OpenServer, have shown to be vulnerable as well) crash. This attack can only occur if the TCP three-way handshake has been completed successfully. Therefore, the attack requires a server listening on that port. In UNIX systems this server could be SAMBA [30].

In our model, it would have been preferable to place the sensor in the code of the SAMBA server, because this attack is specific to that service. For the purposes of this study, a sensor has been placed in the kernel.

This case exemplifies one of the big advantages of embedded sensors: although the vulnerability

does not occur in OpenBSD, a sensor can be implemented to detect the attack.

```
/* TCP input routine */
...
/* ESP */
#ifdef ESP_WINNUKE
if (esp.sensors.winnuke) {
  if ((ntohs(th->th_dport) == 139)
      && (th->th_urp))
    printf("WINNUKE attack\n");
}
#endif
```

3.4.9 Echo-Chargen Connections (CVE-1999-0103)

This attack spoofs packets between two attacked machines on the **echo** or **chargen** port, respectively, using UDP. Those services are provided by the **inetd** program. Therefore, the sensors are best placed in that program, where the error occurs. The code for these sensors is longer (with respect to the other sensors shown—but only 25 lines were added to the `inetd.c` file) and not shown here.

3.4.10 TCP Sequence Number Prediction (CVE-1999-0077)

If the initial TCP sequence number is not chosen randomly, an attacker might guess it and complete the three-way handshake with a spoofed IP address. Vulnerable implementations use a predictable algorithm for sequence number generation. OpenBSD 2.6 does not. The sensor works by detecting out-of-sequence packets with the ACK flag on sockets in state SYN_RECEIVED and SYN+ACK packets in state SYN_SENT. We show the code from SYN_RECEIVED. The SYN_SENT state is very similar.

```
if (tiflags & TH_ACK) {
 ...
 if(SEQ_LEQ(th->th_ack,tp->snd_una)||
    SEQ_GT(th->th_ack,tp->snd_max)){
   /* ESP */
#ifdef ESP_TCPSEQNR
   if (esp.sensors.tcpseqnr)
     printf("TCP SEQNOPRED attack\n");
#endif
   goto dropwithreset;
} ... }
```

3.4.11 TCP SYN Flood (CVE-1999-0116)

The target is flooded with forged TCP SYN packets that fill the connection table or even main memory and cause a DoS on new connections or overload the host, respectively [31].

OpenBSD 2.6 deals with this attack in a special subroutine. If the number of half-open connections reaches a certain limit, every time a new connection is received, one of the pending connections in state SYN_RECEIVED is dropped. It first tries to drop a connection to the same port. The sensor is placed at the entry of this subroutine. The code of the sensor is similar to other sensors shown and is omitted here.

3.4.12 ICMP Unreachable messages (CVE-1999-0214)

This attack sends a forged ICMP error packet of type unreachable, which has the effect of resetting an established connection on vulnerable systems. This attack relies on a combination of flaws in the design of IP and is therefore hard to prevent and detect. ICMP packets may be caused by the remote host or from any gateway along the path. They might be generated for several reasons, even during an established connection, e.g. route changes. For protocols using UDP there is no clear distinction between a cleverly forged and a real ICMP packet. For TCP it is uncommon that during an established session errors occur. OpenBSD 2.6 ignores those messages and the sensor is placed there.

3.4.13 ICMP Redirect messages (CVE-1999-0265)

The CVE list indicates that some TCP/IP implementations, especially in embedded controllers,

are vulnerable to malformed ICMP redirect messages. The documentation is confusing about those malformed packets and the general design vulnerability of ICMP redirects. Technical details about the malformed packet are not published. Our sensor therefore tries to detect malicious, well-formed redirect messages.

An ICMP redirect message that did not originate from the current gateway, or that redirects to a gateway on a different network, is suspicious. In OpenBSD 2.6 such messages fail to change the routing tables, and the corresponding code is where the sensor is placed. This method of detection has the unfortunate effect that an attacker from the local network may still be able to perform the attack. For this reason, and under the reasoning that ICMP redirect messages are a rare occurrence, we have also implemented a sensor that emits an alert on every ICMP redirect message received. This sensor can be turned off if necessary. The sensors were placed in the ICMP layer.

```
/* ESP */
#ifdef ESP_ICMPREDIRECT
if (esp.sensors.redirectsens)
  printf("REDIRECT message\n");
if (esp.sensors.redirecterr) {
  struct rtentry *rt;
  rt = rtalloc1(sintosa(&icmpsrc),0);
  if(rt &&
     ifa_ifwithnet(sintosa(&icmpdst))
        != rt->rt_ifa)
    printf("REDIRECT attack\n");
  else
    if (rt &&
        satosin(rt->rt_gateway)
         ->sin_addr.s_addr
           != ip->ip_src.s_addr)
      printf("REDIRECT attack\n");
}
#endif
```

3.4.14 Smurf / Fraggle (CVE-1999-0513)

In Smurf the attacked host is flooded with ICMP echo reply packets that are generated by forged ICMP echo request packets sent to an IP broadcast addresses, appearing to come from the victim host. All the hosts in the address range reply, flooding the target. One packet from the real attacker can cause a flood of packets to the target. Therefore this attack involves two parties: the reflector addressed by the broadcast packet and the target, the spoofed source address.

OpenBSD 2.6 is configurable to ignore IP broadcast echo requests, but does not in the default configuration, because echo requests to broadcast addresses are a valuable tool for network management. Nevertheless one sensor detects those packets and raises an alarm that this machine has been used as a reflector for Smurf.

```
case ICMP_ECHO:
  if (!icmpbmcastecho &&
      (m->m_flags&(M_MCAST|M_BCAST))
       != 0) {
    /* ESP */
#ifdef ESP_SMURF
    if (esp.sensors.smurf_reflect)
      printf("SMURF as reflector\n");
#endif
 ... }
```

The echo replies arriving at the flooded host are not structurally distinguishable at the ICMP layer from other echo replies. ICMP echo replies do not contain a field for multiplexing the packet to a listening socket (like a port number in UDP). However, the ping program (which is the most frequent cause for the generation of echo requests) sets the identification field in the ICMP packet to its process id (pid) and listens on a raw socket for all ICMP traffic, reporting replies if the address, sequence number and id field match. Our sensor uses the same method on all echo replies, by comparing the ICMP id to the pid of all raw ICMP sockets (this is done in the `esp_smurf()` function, not shown). If no match is found, the packet is discarded at the ICMP layer and a counter for unrequested echo replies is increased. If a match is found, it has to be delivered to the appropriate raw sockets and processed there.

```
case ICMP_ECHOREPLY:
  /* ESP */
```

```
#ifdef ESP_SMURF
  if (esp.sensors.smurf) {
    if (esp_smurf(ip, icp))
      goto freeit;
    goto raw;
  }
#endif
```

To make this method work, the process ID has to be stored when ICMP raw sockets are created. This shows another advantage of using embedded sensors: additional information can be made available when necessary for the purposes of detection.

```
case PRU_ATTACH:
...
/* ESP */
#ifdef ESP_SMURF
   if (esp.sensors.smurf &&
       ((long) nam) == IPPROTO_ICMP)
      so->so_pgid = curproc->p_pid;
#endif
```

The alarm of Smurf packets is rate limited. A legitimate use of **ping** will probably be interrupted when there are still echo reply packets in the network to be delivered to the host. Those packets should not raise an alarm, although they do match the signature. A network layer timer that runs for three seconds examines the counter and raises an alarm only if it exceeds a threshold.

A derivative of the Smurf attack is the Fraggle attack, which is based on the same principle, but uses UDP instead of ICMP packets. The detection of this attack is much harder because there are many legitimate UDP services based on IP broadcasts (e.g. BOOTP, RIP and NetBIOS). We have implemented sensors for this attack, but they are still unreliable and in development.

3.4.15 Port Scanning

Port Scanning is a probing technique that sends packets to a large number of ports trying to identify the services running on that machine [13]. There are several different packets that can be used for this. A short description of them and the state of the socket when those packets arrive follows. Those packets are then reported to the port scan detector, which accumulates them in a reporting routine and prints the summarized alarm of the scan in a timer-based alarm routine. Additionally to these special packets, all TCP packets that cannot be delivered to a socket are reported.

Full scan: The attacker completes the three-way handshake and then disconnects or reports a connection error. For the target host these are either SYN packets with no socket or FIN/RST packets in the established state with the initial sequence number plus one.

Half or SYN Scan: The attacker only sends a forged SYN packet and decides on the reply (RST or SYN+ACK). If the port is open an RST is sent after the SYN+ACK. For the target host, these are again SYN packets with no socket or RST packets received in state SYN_SENT.

FIN / XMAS / NULL Scans: These are packets with special flag combinations— FIN: only FIN flag; XMAS: FIN+PUSH+URG; NULL: no flags. These packets often get dropped on listening ports without an RST packet reply, while closed ports do reply. OpenBSD 2.6 is vulnerable to this. These packets are seen by the target host in ports that are in the LISTEN state.

3.4.16 Detecting port scans

Although UDP-based port scans are also possible, we have only implemented detection for TCP port scanning. The following variations of the techniques previously mentioned exist:

- FTP bounce scanning does full TCP scans and hides the attacker's IP address from the target.
- Fragmentation scanning may bypass some packet filters on the path.

- ID scanning uses Half Scanning and hides the attacker's IP address. Reserved flag bits may be set in the TCP header to bypass some port scan detectors.

All these appear to the attacked host as one of the scanning techniques mentioned before.

The reporting procedure (`esp_portscan()`) logs all reported packets into a data structure where port scans are accumulated, indexed by source address, to support detection of simultaneous port scans from multiple sources. It keeps a table of all IP addresses that recently sent port scans. A chained hash table is used to access the table field. The length of the chain is fixed to limit the processing overhead. This technique has been used also in **scanlogd** [13]. Furthermore, all attackers are also entered at the tail of a doubly linked list sorted by recent access. This may require setting five list pointers.

The alarming procedure runs every three seconds and processes the sorted linked list of attackers for entries beyond a threshold (ten seconds), if there are any. It then sends an alarm about the techniques used, the attacker's IP address and a summary of the ports probed. To distinguish full TCP scans from the NetBSD Race attack, multiple connects to the same port, are reported as race attacks, while a single probe or multiple probes to different ports, are reported as a port scan.

The advantages of this type of port scan detection are a low overhead, as not many packets have to be processed by the reporting routine, and the list to check for the alarming routine is sorted, requiring minimal overhead if no alarms are generated. It also provides a high degree of accuracy that can reliably detect even a scan for a single port, which was an attack observed frequently in the test environment that is connected to the Internet.

3.5 Testing the sensors

A test suite of exploit programs was assembled to test the sensors. The exploit programs were acquired preferably from the same sources that published the vulnerabilities, if they made them available. Those exploits supposedly work against vulnerable systems. They also provide more technical details than some descriptions. However, for the following attacks exploits were not available: TCP RST DoS, TCP Sequence Number Prediction, ICMP Unreachable and Redirect Attacks, NetBSD Race and Linux Blind Spoof. For these attacks, we wrote our own exploits according to the descriptions. Some publicly available exploits worked incorrectly, e.g. Land, due to an incorrect TCP header checksum, many Teardrop attacks, due to ordering errors, and some Smurf attacks, due to an incorrect socket for broadcast packets. If no working substitute was found, we wrote our own exploits. For port scanning **nmap** was used. A successful Fraggle detection was not possible due to its inherent difficulties, and the Echo-Chargen sensor does not report correctly, because of I/O redirection in **inetd** (this will be corrected once we start using a sensor-specific reporting mechanism). The test suite was run supervised from a remote machine on the same local area network (LAN) and all attacks were detected reliably.

An independent tester ran the same set of attacks. First the attacks were run over the campus network, with different network technologies and possibly even filtering in between. The results were that only a very small number of attacks arrived at the target. This experience shows that most attacks are of rather low quality (as also the experiences above indicate) and are very fragile to the network environment. The packet log shows that all received attacks were detected. The test was repeated from a machine on the same LAN and the results match those of the supervised test.

In the testing period the host reported some attacks not generated as a controlled experiment, notably port scans. To verify their correctness they were compared to the packet log and all could be verified as real attacks.

4 Conclusions and future work

We have shown an application of the use of embedded sensors for intrusion detection, in particular to the detection of common network attacks and port scans.

The excellent detection rate is very encouraging and shows that this approach to intrusion detection is very promising. The simplicity of most detectors suggests that the impact on the host is low, but it should be verified with benchmarks. We implemented sensors for 15 different network attacks by adding only 73 lines of code to the OpenBSD kernel, 25 lines to the **inetd** program, and two support files with a total of 354 lines of code. Most sensors are no longer than 4 lines of code, and some sensors are for platforms other than OpenBSD (like WinNuke, which is a Windows-specific attack), which shows that a single host instrumented with internal sensors can detect attacks for different architectures and operating systems.

The sensors have been the simplest in the cases where the kernel itself already checks for the attacks. This has been mostly the case for recent vulnerabilities that depend on implementation flaws: Land, Teardrop, TCP RST DoS and ICMP Unreachable DoS. Some sensors required additional checks at specific places to raise an alarm: Ping of Death, NetBSD race and Linux Blind Spoof.

There are attacks that are based in the design of the protocols: SYN flood, Port Scanning, ICMP redirects and ICMP unreachable. ICMP redirects and ICMP unreachable attacks are undetectable if they are done cleverly. Port Scanning is undetectable if no connections attempts to closed ports are made, which contradicts its intention of gathering information about the target. SYN floods are prevented in OpenBSD 2.6 by resource limitation and active disconnects of pending connections.

For the scope of this study, mainly the kernel has been investigated. But as the WinNuke and Echo-Chargen cases show, there are network attacks for which sensors are better placed outside the kernel. Also some sensors require more computation, namely Smurf and Port Scanning. If rate information is helpful or necessary, sensors split into two parts: a reporting function that gathers data and an alarming function that checks for attack thresholds and raises the alarms.

Fraggle attacks require additional work. Sensors in a set of possibly vulnerable applications combined with a rate limitation should detect them. In future work sensors could be placed into all states of TCP, not only in the initial states of a connection, reporting all unexpected flag combinations. This might be able to detect yet unexploited vulnerabilities. A UDP port scanner was planned, but not implemented, and a larger list of vulnerabilities should enhance the results achieved.

The reporting mechanism used in this study was the one already available in the system. As the Echo-Chargen sensor shows, this is not always convenient, and special reporting mechanisms are necessary. We are currently researching the needs and problems related to these reporting mechanisms, taking into account the specific needs and behavior of the sensors. An initial implementation has been done, and the existing sensors are being modified to use it. One of the problems to be solved is that sensors may exist in both kernel- and user-space processes. To avoid unnecessary context switches, hooks into the reporting mechanism should exist in both spaces. However, all the results need to be eventually coalesced for analysis. We have not determined yet the best way of handling this data flow.

Work is also underway in the implementation of sensors for other types of attacks and intrusions, not only network-based.

As more sensors are implemented and more information is collected, we expect to gain insight into the types of information needed to detect different intrusions and attacks. Also, if embedded sensors show to be a viable approach, maybe operating system vendors will start including them in their distributions, allowing intrusion detection systems to have access to much more useful data about system behavior.

5 Acknowledgments

We would like to thank Christopher Telfer for performing the unsupervised tests.

6 Author biographies

Florian Kerschbaum is a Ph.D. student at Purdue University where he is working for CERIAS. He obtained his M.S. in Computer Science from Purdue University. His research interests are Intrusion Detection and Security Algorithms.

Eugene H. Spafford is a professor of Computer Sciences at Purdue University, the university's Information Systems Security Officer, and is Director of the Center for Education Research Information Assurance and Security. CERIAS is a campus-wide multi-disciplinary Center, with a broadly-focused mission to explore issues related to protecting information and information resources. Spaf has written extensively about information security, software engineering, and professional ethics. He has published over 100 articles and reports on his research, has written or contributed to over a dozen books, and he serves on the editorial boards of most major infosec-related journals.

Dr. Spafford is a Fellow of the ACM, Fellow of the AAAS, senior member of the IEEE, and is a charter recipient of the Computer Society's Golden Core award. Among other activities, he is chair of the ACM's U.S. Public Policy Committee, a member of the Board of Directors of the Computing Research Association , and is a member of the US Air Force Scientific Advisory Board. He regularly serves as a consultant on information security and computer crime to law firms, major corporations, U.S. government agencies, and state and national law enforcement agencies around the world.

Diego Zamboni is a Ph.D. student at Purdue University, where he is working in CERIAS in Intrusion Detection research. He obtained his M.S. in Computer Science from Purdue University. Previously he obtained his bachelor's degree in Computer Engineering from the National Autonomous University of Mexico, where he was in charge of the security for the Unix machines at the Supercomputing Department. He also established the University's Computer Security Area, one of the first Computer Security Incident Response Teams in Mexico.

More information about the authors can be found at `http://www.cerias.purdue.edu/personnel.php3`.

References

[1] Aleph One. Smashing the stack for fun and profit. *Phrack Magazine*, 7(49):File 14, 1996. Appeared also on Bugtraq (`http://www.securityfocus.com/archive/1/5667`).

[2] Jai Sundar Balasubramaniyan, Jose Omar Garcia-Fernandez, David Isacoff, Eugene Spafford, and Diego Zamboni. An architecture for intrusion detection using autonomous agents. In *Proceedings of the Fourteenth Annual Computer Security Applications Conference*, pages 13–24. IEEE Computer Society, December 1998.

[3] Bruce Barnett and Dai N. Vu. Vulnerability assessment and intrusion detection with dynamic software agents. In *Proceedings of the Software Technology Conference*, April 1997.

[4] S.M. Bellovin. Security problems in the TCP/IP protocol suite. *Computer Communication Review*, 19(2):32–48, April 1989.

[5] BugTraq. Mailing list archive. Web page at `http://www.securityfocus.com/`, 1999–2000.

[6] CERT Coordination Center. CERT/CC statistics. Web page at

http://www.cert.org/stats/cert_stats.html, 1999.

[7] Steven Christey, Mann, and Hill. Development of a common vulnerability enumeration. Workshop RAID99, September 1999.

[8] Cisco Systems. Cisco secure IDS. Web page at http://www.wheelgroup.com/univercd/cc/td/doc/pcat/nerg.htm, January 2001.

[9] Douglas Comer and David Stevens. *Internetworking with TCP/IP: Principles, Protocols and Architecture*. Prentice-Hall, Englewood Cliffs, NJ 07632, USA, 1991. ISBN 0-13-468505-9.

[10] Mark Crosbie, Bryn Dole, Todd Ellis, Ivan Krsul, and Eugene Spafford. IDIOT—users guide. CSD-TR 96-050, COAST Laboratory, Purdue University, 1398 Computer Science Building, West Lafayette, IN 47907-1398, September 1996. URL http://www.cerias.purdue.edu/techreports/public/96-04.ps.

[11] Dan Farmer and Wietse Venema. Computer forensics analysis class handouts. Web page at http://www.fish.com/forensics/, August 1999. Accessed in May 2000.

[12] FreeBSD Security Officer. TCP RST denial of sevice. FreeBSD Security Advisory SA-98:07, FreeBSD, Inc., October 1998. URL ftp://ftp.FreeBSD.org/pub/FreeBSD/CERT/advisories/FreeBSD-SA-98:07.rst.asc.

[13] Fyodor (fyodor@dhp.com). The art of port scanning. Internet http://www.insecure.org/nmap/nmap_doc.html, September 1997.

[14] R. Heady, G. Luger, A. Maccabe, and M. Servilla. The Architecture of a Network Level Intrusion Detection System. Technical report, University of New Mexico, Department of Computer Science, August 1990.

[15] L. Heberlein, G. Dias, K. Levitt, B. Mukherjee, J. Wood, and D. Wolber. A Network Security Monitor. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, May 1990.

[16] Gene H. Kim and Eugene H. Spafford. The design and implementation of Tripwire: A file system integrity checker. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, pages 18–29, Fairfax, Virginia, November 1994. ACM Press.

[17] T. F. Lunt, A. Tamaru, F. Gilham, R. Jagannathan, P. G. Neumann, H. S. Javitz, A. Valdes, and T. D. Garvey. A Real-Time Intrusion Detection Expert System (IDES) – Final Technical Report. Technical report, SRI Computer Science Laboratory, SRI International, Menlo Park, CA, February 1992.

[18] Marshall Kirk McKusick, Keith Bostic, Michael J. Karels, and John S. Quarterman. *The Design and Implementation of the 4.4BSD Operating System*. Addison-Wesley, Reading, MA, USA, 1996. ISBN 0-201-54979-4.

[19] Merriam-Webster. “intrusion”. Merriam-Webster OnLine: WWWebster Dictionary. http://www.m-w.com/dictionary, 1998. Accessed on May 16, 1998.

[20] MITRE. Common vulnerabilities and exposures. Web page at http://cve.mitre.org, 1999–2000.

[21] Biswanath Mukherjee, Todd L. Heberlein, and Karl N. Levitt. Network intrusion detection. *IEEE Network*, 8(3):26–41, May/June 1994.

[22] NetBSD Security Officer. select(2)/accept(2) race condition in TCP servers. NetBSD Security Advisory 1999-01, The NetBSD Foundation, 1999. URL `ftp://ftp.NetBSD.ORG/pub/NetBSD/misc/security/advisories/NetBSD-SA1999-001.txt.asc`.

[23] Network Associates, Inc. Linux blind TCP spoofing. CIAC Bulletin J-035, Computer Incident Advisory Capability, March 1999. URL `http://www.ciac.org/ciac/bulletins/j-035.shtml`.

[24] OpenBSD. Web page at `http://www.openbsd.org/`, 1999–2000.

[25] Packet Storm. Web page at `http://packetstorm.securify.com`, 2000.

[26] Phillip A. Porras and Peter G. Neumann. EMERALD: Event monitoring enabling responses to anomalous live disturbances. In *Proceedings of the 20th National Information Systems Security Conference*, pages 353–365. National Institute of Standards and Technology, 1997.

[27] Richard Power. 1999 CSI/FBI computer crime and security survey. *Computer Security Journal*, Volume XV(2), 1999.

[28] Thomas H. Ptacek and Timothy N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, Secure Networks, Inc., January 1998.

[29] RootShell. Web page at `http://www.rootshell.com`, 2000.

[30] Samba. Web pages at `http://www.samba.org`, 2000.

[31] Christoph L. Schuba, Ivan V. Krsul, Markus G. Kuhn, Eugene H. Spafford, Aurobindo Sundaram, and Diego Zamboni. Analysis of a denial of service attack on TCP. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 208–223. IEEE Computer Society, IEEE Computer Society Press, May 1997.

[32] SecurityFocus. Web page at `http://www.securityfocus.com/`, 1999–2000.

[33] Eugene Spafford and Diego Zamboni. Data collection mechanisms for intrusion detection systems. CERIAS Technical Report 2000-08, CERIAS, Purdue University, 1315 Recitation Building, West Lafayette, IN, June 2000.

[34] Eugene H. Spafford and Diego Zamboni. Intrusion detection using autonomous agents. *Computer Networks*, 34(4):547–570, October 2000.

[35] W. Richard Stevens. *TCP/IP Illustrated*, volume Volume 2—The Implementation of *Professional Computing Series*. Addison-Wesley, 1994.

[36] Kymie M. C. Tan, David Thompson, and A. B. Ruighaver. Intrusion detection systems and a view to its forensic applications. Technical report, Department of Computer Science, University of Melbourne, Parkville 3052, Australia, year of publication unkown. URL `http://www.securityfocus.com/data/library/idsforensics.ps`.

[37] X-Force. Web page at `http://xforce.iss.net`, 2000.