

Project ifchk: Host Based Promiscuous Mode Detection and Handling

Joshua Birnbaum
Noorg, Inc.
engineer@noorg.org
www.noorg.org/ifchk

Richard L. Kline
Computer Science Department
Pace University
New York, NY USA
rkline@pace.edu

ABSTRACT

In the face of mounting threats to operating system and network security, proactivity is fast becoming a necessity as opposed to a retrofitted afterthought. If made a central component of site security policy, proactivity can go a long way towards thwarting unauthorized access and usage of computing resources. The timely application of operating system patches, knowledge of current security related vulnerabilities, frequent review of system log output along with the use of well chosen security tools are just some examples of how modern computing environments can withstand determined attacks. **ifchk** (interface check) is a security tool for network interface promiscuous mode detection, interface management and traffic trend analysis. Written in the C programming language and initially released under IRIX, an SVR4 based Unix implementation from Silicon Graphics, Inc. (SGI), ifchk is now in the final stages of a porting effort to Linux.

1. INTRODUCTION

With the number of organizations and individuals participating in the networked world growing, there is a heightened emphasis on maintaining the operational integrity of such a communications infrastructure. Within the context of system and network administration, such efforts are often a mix of identifying security threats and maintaining acceptable levels of operational performance. Sometimes, one is linked to the other. In other cases, it is not. ifchk is a tool that addresses both needs.

In this paper, we begin by elaborating on the motiva-

tions behind the creation of ifchk followed by a discussion concerning network interface promiscuous mode operation. Program functionality and an examination of program implementation issues then follow. This leads us into how the ifchk core is structured from an algorithmic point of view. We then review related work and conclude by discussing future directions of ifchk development.

2. MOTIVATION AND BACKGROUND

There were three main motivators behind the writing of the ifchk security tool. Firstly, the program author is a long time user of open source software and wanted to respond in kind by giving something back to the open source community. Secondly, the author is an experienced Unix system administrator who saw his entry into systems programming as a powerful method to further explore Unix and Linux operating system implementation. To this end, ifchk has paid off handsomely. This exploration was further enhanced by the porting effort, mentioned above, of ifchk from IRIX on SGI to Linux. Differences in OS implementation, as they relate to ifchk, allowed for the examination of different application programming interfaces (API's) on both systems. Thirdly, it is hoped that ifchk can serve as a learning aid for individuals who wish to engage themselves in systems programming. Given that the ifchk source code is freely available, it is hoped that such individuals will take advantage of the ability to review code and learn from it.

2.1 Promiscuous Mode Operation

Here, we begin our discussion of the theoretical underpinnings of the ifchk tool which will serve to create a contextual foundation for what is to come. What is discussed here is within the context of TCP/IP protocol stack (Figure 1) operations and how they relate to core ifchk functions.

Data link implementations (*e.g.*, ATM, Ethernet, etc), as part of their design, will typically encapsulate the data that they transmit within implementation specific units. These units of data bear implementation specific names. For ex-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 2004 Joshua Birnbaum and Richard L. Kline.


```
ec0: flags=d63<UP,BROADCAST,NOTRAILERS,RUNNING,PROMISC,FILTMULTI,MULTICAST>
      inet 192.0.2.2 netmask 0xffffffff broadcast 192.0.2.255
```

Figure 2: ifconfig output showing interface status and state.

1. ifchk will report on the state (normal, *down*, PROMISC, PROMISC [*]) of each interface attached to the system.
 - (a) The state normal refers to an interface that is up. It is reading from and writing data to the network and is not in promiscuous mode.
 - (b) The state *down* refers to an interface that is down. No communication of any kind is possible on this interface.
 - (c) The state PROMISC refers to an interface that is up. It is reading from and writing data to the network and IS in promiscuous mode.
 - (d) The state PROMISC [*] refers to an interface that has been shut down because ifchk was told, by the user invoking the program, to shut down any interfaces found in promiscuous mode. The interface then enters into the *down* state described above.
2. ifchk will shut down all interfaces running in promiscuous mode, if told to do so.
3. ifchk will report per-interface traffic metrics to help identify spikes in network traffic flow that may warrant further investigation. This is similar to output generated by the netstat command. netstat, like ifconfig, is standard on Unix and Linux systems and displays network status information such as the contents of the in-kernel routing table, integer counters describing both ingress and egress packet counts and per-protocol (TCP, UDP, ICMP, etc.) statistics.
4. ifchk logs everything that it finds via syslogd (the Unix/Linux system event logging daemon).

Console output from ifchk Linux (Figure 3) begins with a count of interfaces present on the system and then proceeds to describe the status of each.

```
interface(s): 6
  lo: normal
  eth0: PROMISC [*]
  bond0: *down*
  gre0: *down*
  tunl0: normal
  dummy0: *down*
```

Figure 3: ifchk output under Linux.

Console output from ifchk IRIX on a Silicon Graphics system with two interfaces (Figure 4) shows a dump of per-interface ingress/egress packet count information. Name refers to an interface name (e.g., ec0). Index refers to a kernel-assigned integer reference for that interface. Ipkts is a count of ingress packets. Opkts is a count of egress packets.

*** Network Interface Metrics ***			
Name	Index	Ipkts	Opkts
ec0	1	1479223	1843514
lo0	2	7112894	7112894

Figure 4: ifchk output under SGI IRIX.

The existence of the ifconfig and netstat programs with their respective abilities to reveal promiscuous mode activity (Figure 2) and display network status data raises a question. What is the use of ifchk reporting this information if ifconfig and netstat already do so? The answer to this question lies in trojaned binaries. Attackers will often attempt to conceal their presence on compromised systems so as to allow them to fulfill their objectives, whatever those objectives may be. This sometimes involves the erasure of legitimate ifconfig binaries that will report promiscuous interface activity if the attackers are carrying out any kind of network data reconnaissance. Such binaries are supplanted with compromised versions that will not report promiscuous activity even in the event that an interface is in promiscuous mode.

Netstat, with its ability to display network packet ingress/egress counts, can also be targeted for erasure and subsequent replacement with a trojaned version. A legitimate netstat binary will increment ingress and egress counters once for every one packet processed while trojaned binaries will not. Instead, trojaned binaries may increment once every arbitrary number of packets (e.g., 100, 1000, 10000). Recall that a promiscuous interface is not only processing data addressed to it, but a copy of the data addressed to all other systems on the network. This results in a much larger volume of network data to process and, as a result, elevated counts. Such a rise in packet count could alert system administrators to the presence of possible unauthorized activity prompting further investigation.

4. IFCHK IMPLEMENTATION

ifchk utilizes standard operating system services, as supplied by IRIX and Linux, in order to implement its functionality. Examples include ioctl() system calls, the BSD sockets

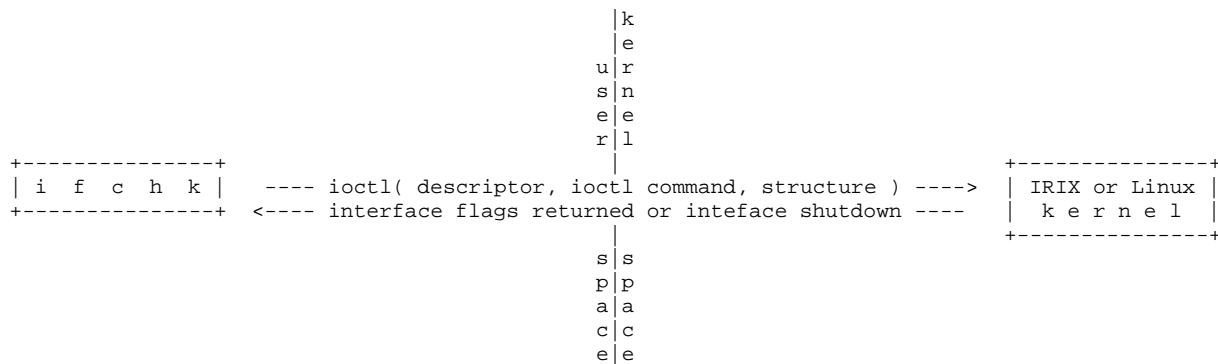


Figure 6: ioctl() command passing.

and uses the data structure that corresponds to that command. The command/data structure pairing is as follows:

RTM_GETLINK: gets information (flags, etc.) about a specific network interface; uses a structure of type `ifinfomsg`.

The `/usr/include/linux/rtnetlink.h` system header file defines the `ifinfomsg` structure. We then call `ioctl()` to disable the interface if it is in promiscuous mode (as detected by Netlink/Rtnetlink prior) and we are to shut it down.

At its core, `ifchk` uses a large for loop that iterates `n` times with `n` being equal to the number of interfaces present on the system as discovered, prior to loop entry, by `ioctl()` on IRIX or Netlink/Rtnetlink on Linux. For each iteration through the loop, we are testing the flags associated with the current interface under examination and printing its state (Figure 3). In addition to this, we shut that interface down, if applicable. Figure 7 illustrates the main loop.

```

for ( n = firstInterface; n != NULL; n++ )
{
    get interface status;

    if ( promiscuous && disable interface )
    {
        print interface status;
        disable interface;
    }

    else
    {
        print interface status;
    }
}
  
```

Figure 7: Pseudocode of main program loop.

At loop exit, we perform house cleaning chores such as memory deallocation and the closing of all open file descriptors in conformance with good programming practice.

5. RELATED WORK

Here, we discuss other work related to `ifchk`. CPM (Check Promiscuous Mode) and Sentinel are two such examples of tools that perform promiscuous mode detection.

CPM [1] is a host based promiscuous mode detector that, like `ifchk`, uses `ioctl()` system calls to print per-interface state information. Written by the Computer Emergency Response Team (CERT), a division of the Software Engineering Institute at Carnegie Mellon University in Pittsburgh, Pennsylvania, CPM reports the number of interfaces present on the system and then, for each interface, prints its name (e.g., `ec0`) and corresponding state. Possible interface states include `Normal` and `*** IN PROMISCUOUS MODE ***`. `ifchk` borrows from this output format as it provides a concise snapshot of interface operation at program runtime. This becomes increasingly important when `ifchk` is run on systems with many interfaces or interface aliases. Tests with `ifchk` Linux on systems with medium to large numbers of interfaces confirm the prudence of this approach.

Sentinel [7] takes a different approach to promiscuous mode detection in that it allows for the detection of promiscuous interfaces on remote systems. Sentinel allows users to perform several different tests to probe for promiscuous mode activity. Examples of the tests Sentinel performs include the following:

- With the DNS test, Sentinel initiates numerous fake TCP connections to nonexistent systems and monitors the network to see if any packet sniffers that might be running on target local systems attempt to resolve, via DNS queries, the IP addresses of those nonexistent systems. Sentinel will then sniff the DNS queries to check to see if it is a target system that is requesting name service resolution.
- With the Etherping test, Sentinel transmits an ICMP echo request message (ping) with a legitimate destina-

tion IP address but a fake destination ARP/MAC address to a target system. If the target is not sniffing the network, its network hardware will disregard the message. If, however, the target is sniffing network packets, the message is processed by its TCP/IP protocol stack implementation (as described in the section Promiscuous Mode Operation, above) and, as a result, a response is returned to the system on which Sentinel is running.

- The Arp test involves Sentinel transmitting an ARP request containing a bogus destination ARP address. Like Etherping above, a target not in promiscuous mode will disregard the request but a target running in promiscuous mode would process the request and send an ARP reply back to the system on which Sentinel is running.

ifchk differs from these two tools in that it disables interfaces that it finds in promiscuous mode, thus potentially isolating such systems from the network, and allowing for their subsequent analysis in an isolated environment. ifchk also differs from the above two implementations in that it reports per-interface ingress/egress packet counts. This is an aid in the detection of deviations in traffic volume that are at odds with established traffic volume trends.

6. CONCLUSIONS AND FUTURE IFCHK DEVELOPMENT

This paper presented ifchk, a method of network interface promiscuous mode detection and interface management. It is a tool that can be used by system and network administrators as an aid in securing systems and networks. It is written in C and made available for free at <http://www.noorg.org/ifchk>. A version for SGI IRIX is currently available, and a Linux version will be released shortly.

There are several functional additions planned for future ifchk releases with a view toward further developing the program into a serious production level security tool.

1. release a Linux ifchk port (now close to completion).
2. turn ifchk into a daemon process controlled via an external control utility. The signals API will be used to provide interprocess communication between the daemon and local control utility.
3. expand upon the above by allowing the local control utility to communicate with the ifchk daemons running on remote systems. The OpenSSL API will be used for encrypted communications between both endpoints thus ensuring data integrity.
4. Enhance the control utility to allow for concurrent communication with multiple ifchk daemon processes

running on different systems. The POSIX Threads API will be used to facilitate this asynchronous I/O capability.

REFERENCES

1. **Check Promiscuous Mode.** Open source software available from: <ftp://ftp.cerias.purdue.edu/pub/tools/unix/sysutils/cpm/>.
2. Dhandapani, Gowri, and Sundaresan, Anupama. *Netlink Sockets - Overview*. Available at <http://qos.ittc.ukans.edu/netlink/html/>.
3. **ethereal.** Open source software available from: <http://www.ethereal.com>.
4. Kuznetsov, Alexey. **iproute2.** Open source software available from: <ftp.inr.ac.ru/ip-routing>.
5. Oualline, Steven. *Practical C Programming, 3rd Edition*. O'Reilly Media Inc, Sebastopol, CA, 1997.
6. Reek, Kenneth A. *Pointers on C*. Addison-Wesley, Boston, MA, 1998.
7. **Sentinel.** Open source software available from: <http://www.packetfactory.net/projects/sentinel/>.
8. Stevens, W. Richard. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, Boston, MA, 1994.
9. Stevens, W. Richard. *Unix Network Programming, Volume 1, Sockets and XTI, 2nd Edition*. Addison-Wesley, Boston, MA, 1998.
10. **tcpdump.** Open source software available from: <http://www.tcpdump.org>.