Malicious ICMP Tunneling: Defense Against the Vulnerability

Abhishek Singh, Ola Nordström, Chenghuai Lu, Andre L M dos Santos

Georgia Tech. Information Security Center (GTISC)
Center for Experimental Research in Computer Systems (CERCS)
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
{abhi,nalo,lulu,andre}@cc.gatech.edu

Abstract. This paper presents a systematic solution to the problem of using ICMP tunneling for covert channel. ICMP is not multiplexed via port numbers and the data part of the ICMP packet provides considerable bandwidth for malicious covert channels. These factors make it an integral part of many malicious software like remote access and denial of service attack tools. These tools use ICMP to establish covert communication channels. In this paper a stateless model is proposed to prevent ICMP tunneling. A Linux kernel module was implemented to demonstrate the proposed stateless solution. The module enforces a fixed payload policy for ICMP packets and virtually eliminates ICMP tunneling which arises due to the data carrying capability of ICMP. The performance impact on end hosts and routers due to the stateless monitoring model is described.

1 Introduction

The Internet and the World Wide Web (WWW) have had a phenomenal growth during the past few years, interconnecting average users and average user to expert users who are not always good neighbors. Many Internet attacks, both simulated and real, have been described by the security community and have appeared in mainstream media. Two factors that have contributed to widespread Internet attacks are the lack of security as an initial design consideration for the Internet and the average user's inadequate knowledge of threats faced every time their computer is connected to the Internet. Although the attacks vary on their form, most of them have a common goal of leaving a back door open for a future communication with a victim machine.

Internet communication is based, in addition to the Internet Protocol (IP), on three basic protocols: Transmission Control Protocol (TCP), User Datagram Protocol (UDP), and Internet Control Message Protocol (ICMP). Firewalls, depending on the services required by their internal networks, totally block or partially filter Internet packets using one or more of these protocols. An attacker has to decide which protocol or protocols they will use to communicate with a backdoor installed on a host that has been compromised. The attacker's objective is to make the traffic generated by the backdoor appear as much as possible to be normal traffic so it is not blocked by the firewall. TCP and UDP, the most widely used protocols by application servers, are the ones that can be abused with the greatest chance of allowing the traffic to pass through. TCP and UDP packets can carry information either manipulating unused parts of the packet or making the payload look legitimate. For example, various header fields like ACK flags and port number can be used to establish covert communication using TCP or UDP [10]. Knowing about the possibility of this kind of attacks, firewalls like IP Filter can prevent covert channels that make use of TCP/UDP header fields for its communication. IP Filter [9] uses stateful packet filtering. The state engine not only inspects the presence of ACK flags in TCP packets but also includes sequence numbers and window sizes in its decision to block or to allow packets. However, IP Filter does not check the content of ICMP packets and hence fails to prevent covert channels that can arise due to misuse of the payload of ICMP packets. Therefore, although TCP and UDP continue to be a subject for studies in vulnerabilities, ICMP also provides several means for stealth traffic.

ICMP tunneling was first reported in the 1997 [8] [7]. Initial versions of ICMP tunneling enabled an attacker to execute remote commands and steal information from a compromised machine. Although ICMP tunneling has been used for user-user and user - machine communication, its most damaging usage has been for coordination of distributed denial of service attacks. In early February 2000, a distributed denial of service attack was launched against Yahoo, Amazon, eBay and other popular Internet sites. The attacks on Yahoo, eBay, Amazon.com and E*Trade resulted in a loss of approximately $1.2 billion, according to The Yankee Group, a Boston consulting firm. It is reported in [11] that almost all of the tools used on the distributed denial of service (DDOS) attacks on Yahoo, Amazon, eBay, and E*Trade internet sites, have used ICMP for covert communications between the DDOS clients and the attacker's handler program. Some of the most widely known distributed denial of service attack tools like Tribe Flood Net2K [1] and Stacheldraht rely on ICMP tunneling to establish communication channels between the compromised machines and the hacker's machine. Since ICMP tunneling is very simple to deploy and can cause a significant amount of damage it has been classified as a high risk security threat by Internet Security Services [4] and SANS [12].

The rest of the paper is organized as follows. Section 2 presents the solutions that are currently being used to prevent ICMP tunneling. Section 3 presents experimental result of a modified application using ICMP tunneling. Section 4 discusses the proposed solution and its performance impact on routers and on end hosts. Finally, in section 5 conclusion and future work are presented.

2 ICMP Tunneling Vulnerability

Most types of ICMP packets like echo_request/echo_reply (commonly used for ping) have the capability of carrying data in its payload. This data carrying capability of ICMP can be used to establish covert channels. Various malicious applications like "Loki" [8] [7] use the data carrying capability of ICMP to establish covert channels. The use of ICMP for covert communication presents one big advantage over the use of TCP or UDP. ICMP packets use fewer parameters than TCP or UDP. For example, ICMP does not use port numbers. Port number gives an additional parameter for firewalls to filter suspicious traffic. The first 64 bits of the original IP datagram's data are used by the host to match ICMP error messages to the appropriate application level process. The simplicity and lack of parameters used in ICMP have made it popular for hackers designing tools that require covert communication.

Many solutions have been proposed to prevent ICMP tunneling. Some of the proposed solutions currently being used to prevent ICMP tunneling are discussed in the following paragraphs.

Disable all ICMP traffic. Disabling all ICMP traffic prevents covert communication using ICMP packets. However, ICMP messages are required to check the status of a network and communicate IP packet errors. Therefore, disabling all ICMP messages prevents covert communication, but also prevents users on one side of the network protected by the firewall to check the status of machines on the other side or receive valid ICMP packets related to transient network problems. This limitation is not acceptable in some environments and is not a general solution.

Disable part of the ICMP traffic allowed by a firewall. For example, disable incoming echo_request, while allowing outgoing echo_request. If naively implemented, policies like this will still allow covert communication, limiting only which host needs to start a communication. In addition, outgoing ICMP packets could be used to establish a unidirectional channel to send compromised information. A modified server can periodically send ICMP packets containing sensitive information. For example, an e- commerce web server could switch to ICMP mode of communication for 10 second after every 30 minutes. During these 10 seconds it could send to a receiving machine, credit card information that has been captured during the previous 30 minutes.

Limit the size of ICMP packets. Large ICMP packet can be seen as suspicious by an IDS system that could inspect the ICMP packet and raise an alarm. However, since there are legitimate uses for large ICMP packets it is difficult to determine if a large ICMP packet is malicious. For example, large echo_request packets are used to check if a network is able to carry large packets. Differentiating legal from illegal large packets is even more difficult if covert communication is encrypted. An IDS needs to be able to determine if a packet is encrypted or not. Distinguishing encrypted from non-encrypted packet still remains an open interesting research problem.

Preserve the state of ICMP packet to check for covert channel. Some firewalls like Raptor use state preservation technique to prevent ICMP tunneling. A daemon called pingd runs as an application process on the firewall. If the firewall is the target of ping then pingd responds to the client normally. If the firewall is not the target of ping then pingd will construct a new echo request with a new sequence number, new time to live, and a new payload (with new checksum). When the reply is received it is ensured that the data is the same as what had been sent, and the sequence number and responders IP address are valid and as expected. After a successful check the firewall transmits the echo_reply to the original client. Although state preserving technique can easily prevent ICMP tunneling, it is a computing intensive process. Therefore, currently application level personal firewalls do not use state preserving techniques. Application level personal firewalls [8] running on personal computers only monitor inbound and outbound Internet traffic and alerts the user when an application is attempting access their personal computer or their machine is trying to access something on the Internet. As per the configuration rule, personal firewalls can either block ICMP or allow it irrespective of the fact that the ICMP is being used for tunneling. Protecting personal machines from ICMP tunneling is very important since there are a large number of personal machines connected to high speed links which in turn can be used to launch DDOS attacks.

Although some of the solutions presented above can be acceptable, ideally the prevention of ICMP misuse due to their data carrying capabilities should be able to provide the following.

- It should enable users to use ICMP messages for administrative purposes freely.
- It should allow large size of ICMP so users can find out if the network can carry large size of data packets.
- It should be able to prevent personal machines that are not behind powerful state preserving firewalls from being used as DDOS slaves.

3 ICMP Tunneling: Case Study

An application that uses ICMP tunneling was implemented and studied to better understand the ICMP tunneling efficiency and capabilities. The application, a remote access tool, is described in this section.

Remote Access Tools allows a user to access data and control a remote computer. Back Orifice (BO2K) [2] was used to test ICMP tunneling due to the easy availability of its source code. The communication infrastructure of BO2K was moved to ICMP. An ICMP echo_request contained the remote command issued by a client of BO2K and an echo_reply contained the information from the machine that was running BO2K server. Strong authentication and encryption was also implemented to evaluate the impact of these features. The Bellovin and Merritt [2] key exchange known for its strong authentication and establishment of session keys was implemented in the BO2K communication protocol. The session key generated by the Bellovin and Merritt protocol was then used to encrypt the data using 3DES. The server and the client of the modified BO2K were installed on a pair of personal computers. The machines were connected via the Internet and were using the personal firewall as their application level firewall.

3.1 Results

Since ICMP uses raw sockets for its communication, root (administrator) privileges are required. For the experimentation purposes it was assumed that the root privileges could be obtained. More details about getting the root permission can be found in [3]. However, in most of the commonly used operating systems like Windows ME/98 root permission is not required. In addition, it was also assumed that the firewall is customized to allow ICMP packets.

The personal firewall used in the machines was set up such that it raise an alarm in the form of a pop up window each and every time it sees an incoming or an outgoing data packet. For an incoming connection, the pop up window displays the IP address of the machine initiating the connection and the protocol, which is being used for the connection. For an outgoing connection, the pop up window shows the IP address of the destination machine along with the protocol used for the communication. A server and a client BO2K were installed on machines running the Windows ME operating system. Snapshots of the connection initiation steps and the real time interaction steps are omitted due to limited space. However as per our observation the ICMP mode of communication provides false information to the firewall. Even though ICMP packets are being sent by the application, the firewall infers them to be sent by the operating system. The real time interactions enabled by the modified application shows that ICMP tunneling is highly efficient and can be used in many malicious applications. Even though ICMP packets are being used by the application, the personal firewall infers these ICMP packets as the control messages issued by the operating system.

4 Proposed Solution

The solution to prevent ICMP tunneling should:

– Enable administrators to use ICMP messages freely.
– Enable large size of ICMP to be used.
– Work for every machine connected to the Internet.

Instead of the expensive and centralized state preserving model used by industrial firewalls, a simple stateless model is proposed to prevent tunneling. The stateless model caters to the above mentioned requirements. The proposed solution, which requires a common agreement upon the allowable payload of ICMP messages, should be implemented in the ICMP protocol implementation of the kernel. It should be enforced either when an ICMP packet is going up the network stack or when it is going down the stack. The solution uses the algorithm shown in Figure 6. This algorithm first scans the content of the payload of the

ICMP packet against a predefined set. If any malicious content is found in the payload an alarm is raised. The algorithm then zeros out the entire data carrying field irrespective of the ICMP type.

Linux was chosen for the implementation of the proposed solution because of its freely available source code. The Netfilter framework was used to extend existing functionality within the kernel. Netfilter [15] is a set of hooks inside the Linux 2.4.x kernel's network stack which allows kernel modules to register callback functions, called when a network packet traverses one of the predefined hooks. The handlers will execute as if they were part of the packet processing pipeline directly. Netfilter allows a module to register the IP_POST_ROUTING hook that is called after a packet has been through the routing table and right before it is delivered to the outgoing interface (typically an Ethernet device). The ICMP monitor (icmp_mon) is registered with the IP_POST_ROUTING handler, and thus allowing icmp_mon to process all locally outbound packets as well as forward packets since all packets go through the post routing hooks. The icmp_mon module at runtime calls icmp_mon_erase and icmp_mon_scan to perform a combination of actions.

- icmp_mon_erase; Zeros out unused portions of ICMP messages.
- icmp _mon _scan; Scans packets for predefined strings

The icmp_mon_scan module raises an alarm when it detects some suspicious content in the data portion of the packet. The packet is then forwarded to the icmp_mon_erase module. Irrespective of the fact that it is carrying some malicious strings the data field of ICMP is filled with zeros. Thus the proposed solution can stop even encrypted traffic. Since the packet is analyzed when it is outbound, the proposed solution will work in the case of packets that are locally generated or are forwarded on behalf of another machine. Hence this patch can work in machines that are acting as a gateway, router, sensor or as a host.

```
ICMP_MON (ICMP packet)
Begin
    input : ICMP Packet
    for every ICMP packet
    Begin
        icmp_mon_scan (data portion of ICMP packet)
        If the data field matches with the signature
            Raise an alarm.
    End
    icmp_mon_erase{data portion of the ICMP packet}
    Begin
        Fill in the data portion with zeros.
    End
End
```

The module can be compiled directly into a monolithic kernel. The filter hooks are registered when the modules init routines are called at the boot time if it is compiled into the kernel. The proposed module cannot be bypassed by user space applications since it does not interact with any. Iptables is the user space command that is used to modify Linux firewalling rules. Iptables is typically used to add and remove filtering rules pertaining to regular TCP/IP packets. Because icmp_mon is not part of iptables, the only way to disable the scanning is to remove the module (using rmmod), which is not possible in this scenario since the module is statically compiled into a monolithic kernel. When using a monolithic kernel the machine would have to be rebooted into a different kernel to disable the icmp_mon from operating. This is not trivial to do on a compromised host machine. In addition, if the proposed patch is running on a gateway or at a sensor that is monitoring network traffic then stronger restrictions and event logs can make this option even more difficult to attain.

The proposed solution will make it impossible for an adversary to setup an ICMP covert communication channel on a compromised machine since packets are scanned and erased by the kernel. The stateless model implemented in the kernel benefits from the fact that there is no way to turn off this functionality by simply terminating or modifying an application, which could easily be done on a compromised machine if packets were queued to a user-space intrusion detection application. The proposed stateless model of scanning and erasing the data fields can be implemented in the kernel used by hosts, sensors, gateways and routers so as to completely eliminate ICMP tunneling.

4.1 Results of the Proposed Solution

The icmp_mon scanning and erasing times were tested on ICMP ECHO request packets. The testing machine was a Dual Pentium III 450 MHz machine with 512 MB of RAM. Internally all network packets are time stamped upon arrival in the Linux kernel. The time from arrival up to when the icmp_mon routines are referred as the "Kernel Time", or specifically the time the packet has been traversing inside the kernels packet processing. The total time spent inside the icmp_mon routine, ignoring the overhead caused by the netfilter hooks will be referred to as the "erase/scan time" depending on the operation performed.

The first test was implemented in the kernel of a host machine. In the second test the proposed module was implemented in the kernel of a machine acting as a router. For each of these tests the time to process packets of fixed size in the kernel was compared with the scan and erase time. The tests on host machine and router were again repeated with the packets of variable sizes. All the tests were performed using the standard Unix "ping" command; it was typically run as follows "ping -c Count -s Size -f host", were the Count and Size were either fixed or varied depending on the test. The "-f" option was used to flood ping, the icmp_mon module would sum the processing times for a given number of packets (corresponding to the Count given to the ping command) then the mean time was calculated after the given number of packets had passed through. The gettimeofday() call was used to get the time, on the Intel platform the granularity available is microseconds.

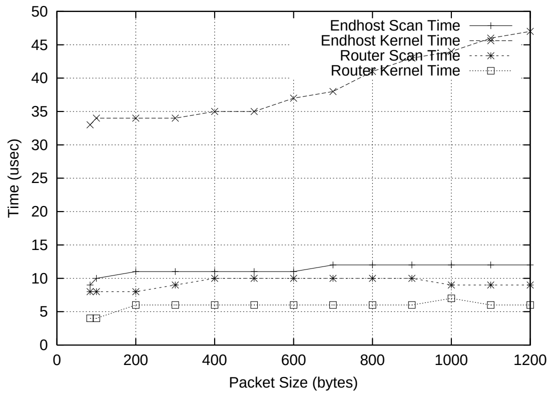

Fig. 1. Scanning vs. Kernel Time for Variable Size Packets

The graph shown in figure 1 presents the performance cost associated with the string scanning routines of the icmp_mon. The size of the packet was increased from 100 bytes to 1200 bytes and time to process 1000 packets of each size was measured and plotted in the graph shown in figure 1. This test was performed both for the router and for the end host. The simple words searched for were "passwd", "root", "tmp" ,"etc", "ls" , and "dir". They were never present in the ping packets thus the scans never raised any alarms. As per the graph for the router the scan time is more than the time to process packet. The machine is simply receiving the packets in one interface and sending them out another, which makes the kernel time incredibly small. In the case of the end host the time to process a packet increases with the packet size, however the processing time is small as compared to the kernel time. The scan time shows a marginal increase, however it remains constant when the size of packet is increases from 700 bytes to 1200 bytes.

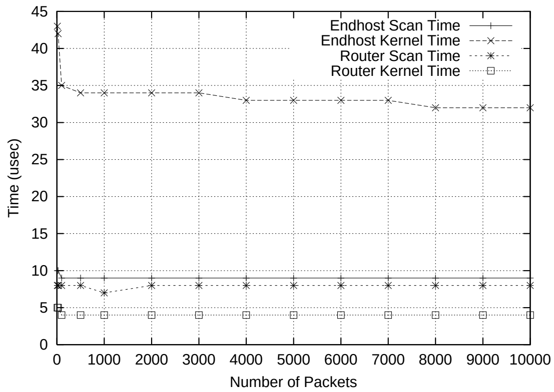

Fig. 2. Scanning vs. Kernel Time for Fixed Size Packets

The graph shown in figure 2 shows the time taken for scanning the ICMP payloads of fixed size of 84 bytes. For this test the number of packets was increased from 10 to 10000 and the scan time was plotted. A 2 microsecond increase was observered when the number of packets increased from 1000 to 10000. As the number of back to back packets increased (due to the ping flood) the kernel time per packet decrease. The kernel time goes down because the flood of packets allows the kernel routines to grab multiple packets back to back without returning to a different routine (thus loosing time due to context switching). When the machine is acting as a router the scan time is again more than the time to process a packet by the kernel. The scan time shows a marginal increase as the number of packet increases.

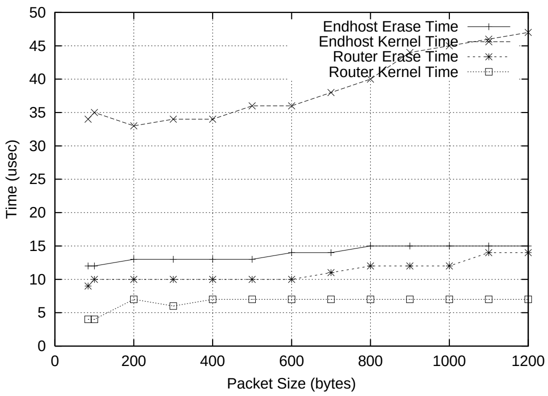

Fig. 3. Erase Time vs. Kernel Time for Variable Size Packets

The result showing in figure 3 gives the average time spent inside the kernel and the icmp_mon _erase routines when the patch is running on a router and on and end host. This test was performed by keeping the number of ICMP packet to 1000 packets for each size and increasing the size of packets from 84 bytes to 1200 bytes. As the packet size increases, the time to process packet inside the kernel for the end host increases. This increase is almost linear. Size has no effect on the processing time for the end routers. The time is contrasted against and increasing packet size, starting with 84 byte IP packets (54 unused/padded bytes inside the PING), the default "ping" packet size. The time spent increases marginally up to about 700 byte packets. As the size of the packet increases erase time increases. Erase time increases by around 5 microsecond in routers and around 3 microseconds for end host kernel when the size of packet is increased from 84 bytes to 1200 bytes.

The result shown in figure 4 shows the performance result of the erase time contrasted with the fixed packet size. As the number of back to back packets increased (due to the ping flood) the kernel time per packet shows a slight decrease but the erase time shows a very low variance. The decrease in the

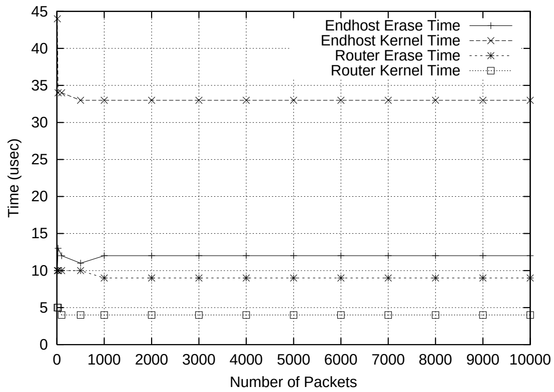

Fig. 4. Erasing Time vs. Kernel Time for Fixed Size Packets

kernel time is due to the loss in context switching. The increase in the time to erase 10000 packets as compared to 1000 packets is very small and can be considered constant. From these tests following conclusions can be made.

- In the case of routers the packets get the initial timestamp after it has been fully received by the driver. Then the kernel simply looks at the ip dst field and adds the packet buffer to the output queue, which is a simple pointer assignment. Thus the total kernel processing time is marginal, extremely small, a few micro seconds as compared to the scan and erase time. This means that the overhead for scanning/erasing is much higher in comparison.
- In the case of end hosts the scan and erase time is very small as compared to the total time to process the packet.
- For fixed size packets the scan and erase time remains constant or shows a very low variance as the number of packets increases.
- As the packet size increases from 84 bytes to 1200 bytes the scan and erase time increases. The worst case increase was observed to be 5 microseconds.

5 Conclusion and Future Work

ICMP tunneling can be used in an efficient way by malicious software. To prevent ICMP tunneling kernel modifications were proposed to enforce a fixed predefined payload policy for ICMP packets. If the proposed solution becomes an integral part of kernels that runs as host, gateways, and DMZ routers then it will be impossible to establish ICMP tunnels. Another way to remove ICMP tunneling could be to simply truncate the data field of ICMP. However truncation of the data field will require amendments in the RFC [6] [5] that supports data field for ICMP. Scanning and erasing of the ICMP data field is compliant with RFC and prevents ICMP tunneling irrespective of the type of firewall used.

The results show that simply marking out unused and potentially dangerous portions of ICMP packets is a straightforward task and requires little overhead on a modest system. Simple string scans are also not costly and can be done to test for unencrypted covert communication. This is highly recommended for the end hosts where it offers minimal overhead on the system. For routers it can be expensive. However, weighted against the potential security risks the marginal overhead can be worth the security benefits in some cases. Moreover if the proposed solution became an integral part of kernels in operating systems like Solaris, Windows, and Linux, which runs on host machines; then the routers will not have to examine all the ICMP packets that it comes across. The router could in this case adopt some sort of probabilistic scheme to check some ICMP packets and allow the other packets to pass through. This would reduce the load on the routers.

This work presents some of our initial steps to prevent ICMP tunneling. In this work covert channel due to the data carrying capability of echo_request/echo_reply was considered. Various others fields in TCP, UDP, IPv4, IPv6 and ICMP can be the potential candidate for the establishment of covert channel. Ongoing work explores the fields in every protocol and proposes the use of either stateless or stateful model for the removal of covert channel.

References

[1] CERT Advisory. Denial of service attack tools. http://www.cert.org/advisories/CA-1999-17.html.

[2] Backorifice SDK Documents. http://bo2k.sourceforge.net/indexnews.html.

[3] Root Exploit and Dos in the Linux Kernel. http://linux.oreillynet.com/pub/a/linux/2001/10/22/insecurities.html.

[4] ISS. Loki icmp tunneling back door. http://www.iss.net/securitycenter/static/1452.php.

[5] Postel J. Internet control mesage protocol - darpa internet program protocol specification. *RFC 792*, September 1981.

[6] Postel J. Internet protocol - darpa internet program protocol specification. *RFC 791*, September 1981.

[7] Phrack. Loki 2(the implementation). http://www.phrack.com/show.php?p=51&a=6.

[8] Phrack. Project loki. http://www.phrack.com/show.php?p=49&a=6.

[9] Guido Van Rooji. Real stateful tcp packet filtering. In *10th USENIX Secutrity Symposium*, August 2001.

[10] Craig H. Rowland. Covert channels in the tcp/ip protocol suite. http://www.firstmonday.dk/issues/issue25/rowland.

[11] Sans. Icmp attacks illustrated. http://www.sans.org/rr/threats/ICMP_attacks.php.

[12] Sans. Intrusion detection faqs. http://www.sans.org/resources/dfaq/icmp_misuses.php.