Honeyd Detection via Packet Fragmentation

Jon Oberheide and Manish Karir
Networking Research and Development
Merit Network Inc.
1000 Oakbrook Drive
Ann Arbor, MI 48104
{jonojono,mkarir}@merit.edu

Abstract

In this paper we describe a serious flaw in a popular honeypot software suite that allows an attacker to easily identify the presence and scope of a deployed honeypot. We describe in detail both the flaw and how it can be used by an attacker. Our technique relies on a set of specially crafted packets which are able to elicit a response from a Honeyd-based honeypot. Simple experiments show that this method is extremely accurate and effective in detecting the presence and the scope of a Honeyd deployment. Moreover, due to the low level of effort and bandwidth required, it is possible to perform honeypot reconnaissance easily prior to launching a malicious attack on a network, even for large address spaces. We also discuss a simple fix for this flaw as well as other factors that can affect the effectiveness of our approach.

1 Introduction

Honeypots play an extremely important role in Internet security. In conjunction with distributed Internet sensors[1][7][11][12], honeypots are able to provide early visibility into emerging security threats, attack tools, and evolving attack techniques. However, just like Internet sensors, honeypot deployments must remain hidden in order to be useful. An attacker who is aware of the location of a monitor can avoid it, mislead it by feeding it erroneous information, or even target the monitor itself.

Recent work has further highlighted the importance of maintaining the anonymity of Internet sensors. In two separate studies, [6] and [13], the authors have presented probe/response techniques to illustrate the inadequacy of current monitoring sensor deployments. In these papers the authors describe how potential attackers can easily determine the location and scope of currently deployed sensors. These detection algorithms illustrate the importance of sensor anonymity.

In addition to protecting the anonymity of currently deployed Internet monitoring sensors, it is also important to ensure that the monitoring software and tools themselves are not vulnerable to attack. Detecting and patching software vulnerabilities in these systems are extremely critical as they themselves may be compromised or can reveal the presence and extent of the monitoring system. One particular area where vulnerabilities have frequently been found in the past is the handling of packet fragments. Packet fragment reassembly has long been a source of various security problems related to various IP stacks. Reported problems include the Tiny Fragment Attack, Overlapping Fragment Attack as well as their variants [3] [4]. Attacks can be easily initiated by crafting malicious packet fragments that can bypass firewall rules. Over the years, significant attention has been paid to these problems such that most modern operating systems are able to correctly handle maliciously crafted packet fragments. However, due to this behavior, any system that does not correctly handle packet fragments can be quickly recognized by this non-standard behavior. The Honeyd detection procedure described in this paper is based upon maliciously constructed fragments combined with an incorrect implementation of the fragment reassembly procedure in Honeyd.

```
#define DIFF(a,b) do { \
        if ((a) < (b)) return -1; \
        if ((a) > (b)) return 1; \
} while (0)

int
fragcompare(struct fragment *a, struct fragment *b
{
        DIFF(a->ip_src, b->ip_src);
        DIFF(a->ip_dst, b->ip_dst);
        DIFF(a->ip_id, b->ip_id);

        return (0);
}
```

Figure 1: Honeyd's ipfrag.c

In this paper we describe a procedure by which an attacker can easily determine if a network is running Honeyd [9], one of the most commonly used honeypot software tools, as well as the range of addresses the honeypot is monitoring. An attacker can run a remote scan that will reveal the presence and scope of a Honeyd-based honeypot. This vulnerability places the effectiveness of several monitoring system that rely on Honeyd at risk. The Network Situational Awareness (NetSA) Internet monitoring system described in [12] is an excellent example of a large scale system that relies on Honeyd.

The rest of this paper is organized as follows: In Section 2 we describe in detail the Honeyd vulnerability that allows our detection technique to work, as well as a fix of this vulnerability; Section 3 contains a description of a proof-of-concept tool that can be used to scan a network for Honeyd deployments. We also describe some of our experiments and discuss some related issues; Finally, Section 4 provides our conclusions and ideas for future work.

2 Honeyd Virtual Honeypot Software

The Honeyd project [10] was initially developed by the Center for Information Technology Integration (CITI) at the University of Michigan. Honeyd allows users to configure virtual honeypots on a network and monitor their activity to gain insight into developing security threats. The Honeyd software is publicly available and widely used by various threat tracking and monitoring projects around the world. Attackers can be tricked into interacting with a Honeyd honeypot as if they were interacting with a vulnerable host. This interaction can be monitored and provides invaluable data to security researchers. Honeyd is generally used in conjunction with another freely available utility called arpd [2], which allows a single host to monitor multiple addresses on a network by responding to ARP requests for unclaimed IP addresses.

In order to avoid being fingerprinted and identified based on the host operating system that it is running on, the Honeyd software interacts directly with the network. Therefore it is responsible for performing its own packet fragment reassembly, and does not rely on packet fragment operations of the host operating system. In the following subsections we describe a flaw in the packet fragment handling code of Honeyd as well as a solution to fix the problem.

2.1 IP Fragment Reassembly

According to the Internet Protocol standard specification [5], a correct IP stack implementation must identify corresponding fragments by matching the source address, destination address, identification number, and protocol number. Figure 1 shows the relevant section of code from the Honeyd source code. The fragcompare() function is responsible for determining whether a fragment should be reassembled. The figure clearly shows that the the function is only performing a comparison on the source and destination IP addresses and the IP identification number. The protocol number is not being used. The end result is that Honeyd will incorrectly reassemble fragments that have a matching source address, destination address, and identification number, but a differing value in the protocol field. This flaw does not normally create a problem in the behavior or functionality of the honeypot, as it is extremely unlikely that a host with a modern network stack would send IP packet fragments where only three of the four fields would match. However, it is trivial to craft customized packet fragments which will trigger this bug and result in packet reassembly occurring where it should not.

One way to expose this flaw is to split a TCP SYN packet into several fragments and set the protocol field of the IP header of one of the fragments to some protocol other than TCP. A stack that correctly implements IP fragment reassembly on receiving this collection of fragments will end up dropping

them, as it will identify the packet with the differing protocol number as not belonging to the others. However, Honeyd on the other hand will receive these fragments and reassemble them into a complete TCP SYN packet. It will then respond to the sender of the fragments with a SYN/ACK. An attacker can send out these fragments to a large number of hosts and simply listen for any returning SYN/ACK packets. Since this reassembly flaw does not exist in most common operating system IP stacks, the attacker can be reasonably suspicious of any machines that do respond as they are likely to be Honeyd-based honeypots.

```
[root@ns1 root]# ./winnie 198.108.62.0/24
pcap listening on eth0 with filter "tcp and src net 198.108.62.0/24
and tcp[13] = 18"

winnie - honeyd scanner
target range: 198.108.62.0/24
scanning 256 unique addresses
repeating in max chunks of 256

scanning to begin in:
3...
2...
1...

scanning 256 addresses at 198.108.62.0
repeating 256 addresses at 198.108.62.0
*** response from 198.108.62.144
*** response from 198.108.62.145
*** response from 198.108.62.146
*** response from 198.108.62.147

scanning complete, 256 addresses took 10.453581 seconds
waiting for delayed replies...

exiting!
```

Figure 2: Winnie sample output

2.2 Patching Honeyd

As seen in Figure 1, the fragcompare() function determines whether the attributes of two fragments are identical but does not compare the protocol number. Eliminating the vulnerability in Honeyd is as simple as including the protocol number field into the fragcompare() function. We have provided the maintainers of Honeyd with a patch that implements this fix and expect that this issue will be resolved quickly. Until this patch is deployed, older versions of Honeyd will continue to be vulnerable to detection by attackers via a simple scan.

3 Scanner Implementation and Experiments

In order to demonstrate the feasibility of utilizing the flaw in Honeyd's reassembly for detection purposes we decided to build a proof-of-concept scanner. In the following subsection we describe some details about how our scanner operates as well as some experiments we have been able to conduct. We also discuss some related issues that affect the success of our scanner.

3.1 winnie - A Honeyd Scanner

Using winnie, we were able to perform some experiments in order to determine whether the flaw in Honeyd could actually be triggered in a remote scan. Winnie allows us to automate the construction of the fragmented packets as well as the scanning of target networks. It uses routines provided by libdnet to build and inject the necessary packet fragments into the network and libpcap to watch for the resulting SYN/ACK replies.

The basic operation of winnie is simple. We start by constructing a valid TCP SYN packet split into two fragments. The entire IP packet payload is 32 bytes, 20 bytes of TCP header and 12 of random data. The 32 bytes of payload are split into two fragments of 24 bytes and 8 bytes, each having identical randomly generated IP identification numbers. For the first fragment, the protocol number in the IP header is set to TCP (6). For the second fragment, we use an alternate protocol number in the IP header. We chose UDP (17) as the alternate protocol number as opposed to some other atypical protocol that could possibly be filtered out on the way to the target. This fragment pair, when sent to any RFC-conforming host, would not be reassembled by the IP stack into a valid SYN due to the differing protocol numbers in the IP header. However, when sent to a host running Honeyd, the Honeyd software will reassemble the two fragments into a valid SYN request and therefore respond back with a SYN/ACK.

An example of tcpdump output during a winnie scan can be seen in Figure 3, showing the outgoing fragment pairs with identical identification numbers but differing protocol numbers. In the snippet provided, one can see that there is no response from hosts 142 or 143, but there are SYN/ACK replies from hosts 144 and 145, indicating that they are running Honeyd.

```
6:59:24.860101 67.101.10.34.14745 > 198.108.62.142.ssh: S
825593105:2825593109(4) win 5840 (frag 14920:24@0+)
6:59:24.860154 67.101.10.34 > 198.108.62.142: udp (frag 14920:8@24)
6:59:24.880091 67.101.10.34.46523 > 198.108.62.143.ssh: S
159669245:4159669249(4) win 5840 (frag 17554:24@0+)
6:59:24.880142 67.101.10.34 > 198.108.62.143: udp (frag 17554:8@24)
6:59:24.900091 67.101.10.34.22194 > 198.108.62.144.ssh: S
912179748:1912179752(4) win 5840 (frag 30473:24@0+)
6:59:24.900143 67.101.10.34 > 198.108.62.144: udp (frag 30473:8@24)
6:59:24.920092 67.101.10.34.23783 > 198.108.62.145.ssh: S
013482849:1013482853(4) win 5840 (frag 57629:24@0+)
6:59:24.920144 67.101.10.34 > 198.108.62.145: udp (frag 57629:8@24)
6:59:24.924290 198.108.62.144.ssh > 67.101.10.34.22194: S 0:0(0) ack
912179749 win 16000 <mss 1460>
6:59:24.924294 67.101.10.34.22194 > 198.108.62.144.ssh: R
912179749:1912179749(0) win 0 (DF)
6:59:24.939426 198.108.62.145.ssh > 67.101.10.34.23783: S 0:0(0) ack
013482850 win 16000 <mss 1460>
6:59:24.939429 67.101.10.34.23783 > 198.108.62.145.ssh: R
013482850:1013482850(0) win 0 (DF)
```

Figure 3: tcpdump trace of winnie scan

3.2 Experiments

In order to determine whether a remote scan would be successful, we conducted the following experiment. We used Honeyd to setup a honeypot on Merit's network, limited to only five unused IP addresses. The scans were conducted from an offsite PC at a remote location. Through this experiment we wanted to determine whether other devices on the network would respond to our scan thereby resulting in false positives. The network we were scanning was a fairly heterogeneous network, consisting of Windows, Linux, BSD, Mac, and Solaris based computers as well as network printers and copiers. The network was a /14 network which consists of 2^{18} total IP addresses. Winnie was used to scan each individual IP address.

The total scan took roughly three hours to complete. All five IP addresses that had been configured with Honeyd were correctly identified. Among all the different types of devices configured on the network, there was only one false positive report. We discuss this as well as other factors that can affect the effectiveness of winnie in section 3.4.

Figure 2 shows sample output from an experiment where we scan a /24 subnet. Our scan consists of two fragments with an ethernet (14 bytes) and IP header (20 bytes) each and a total of 32 bytes of payload for each IP address. This implies that we create 100 bytes of scan traffic for each IP address we scan. Given that we scan each IP address twice to increase accuracy, this translates to 51200 bytes of traffic in order to scan a /24 subnet. The time to scan the /24 subnet is approximately 10 seconds, therefore we generate roughly 40 Kbps of scan traffic.

3.3 Scan Detection

An important issue when developing a scanning tool is the amount of network resources used and the access patterns that may alert sensors watching the network. While winnie was designed as a simple proof-of-concept and does not contain any intentional routines to avoid detection, the nature of the vulnerability actually lends itself to low detection risk. In fact, as the two fragments sent by winnie are never reassembled at hosts with a correct stack, they will simply be dropped silently when the reassembly timer expires. By having a total payload size of 32 bytes and only eliciting a response from the targets of interest, the amount of noise generated on the network is greatly reduced.

A sensor watching for standard TCP SYN port scans may not pick up winnie's scans as the sensor's stack would not see a valid SYN packet passed up from the IP layer. However, a sensor that is watching for unusual IP fragments on the network may, in fact, detect our scan. In our experimentation, we did come across at least one IDS sensor that was tripped because of the suspiciously small fragments. However, even if a network operator is alerted to the strange fragmented scanning, the damage has already been done as the attacker will have gleaned valuable information regarding any deployed honeypots.

3.4 Discussion

3.4.1 ARP Issues

arpd is a tool that is often used in conjunction with Honeyd to respond to ARP requests for unallocated IP addresses within an address space. When arpd sends out its ARP reply to claim an IP address, it addresses the reply to the broadcast address (ff:ff:ff:ff:ff:ff). While some hosts accept this reply and record the entry in their ARP cache, others will reject broadcast replies in an attempt to prevent ARP spoofing attacks. To resolve this issue, we modified arpd to include an alternate mode where ARP replies are addressed only to the hardware ad-

dress of the initial requester as opposed to the broadcast address.

Another feature of arpd is to send out its own broadcast ARP request to be sure that an IP address is truly unused before claiming it with an ARP reply. During this discovery stage, winnie's fragmented packets may be dropped as the gateway device does not receive an ARP reply in time. To compensate for this problem, winnie was modified to scan a block of 256 addresses of the target network and then repeat the scan in the hope that the first scan of an address will cause arpd to begin its discovery process and by the time the repeat scan comes around, arpd will have replied and the gateway device will have the ARP entry cached. This issue is illustrated in Figure 2 which shows that the first scan did not find any instances of Honeyd but the repeat scan successfully detected them.

3.4.2 Packet Normalization

Another issue that can hinder the effectiveness of winnie is inline devices that perform packet normalization or reassembly, such as OpenBSD's PF and norm [8]. Traffic normalizers are often deployed to prevent attacks and host fingerprinting resulting from various ambiguities in network stacks. When the fragments sent by winnie reach a normalizer, they will be correctly dropped as invalid fragments. The existence of such a device in the path to the target host may be unknown and is an important issue to keep in mind when performing Honeyd detection experiments.

3.4.3 Ethereal Packet Fragment Reassembly

During our experimentation, we also discovered that Ethereal, a popular protocol analyzer, suffered the same reassembly issues as Honeyd. While this bug may be frustrating when attempting to analyze a trace containing fragments incorrectly assembled by Ethereal, it is not terribly concerning. A patch was provided to the development mailing list on January 2nd and was subsequently committed on the 4th.

3.4.4 False Positives

It is possible to receive false positive responses when using winnie from any host whose IP reassembly routines contain the same flaw as observed in Honeyd. One such host discovered in our scanning experiments is HP's Lights-Out remote management device. However, from an attacker's point of view, any host responding to the malformed fragments is not an ordinary host with a RFC-conforming stack. While the attacker may not be absolutely sure that the host is indeed a honeypot, they do know that it is an anomalous host and should avoid targeting it in an attack.

4 Conclusions and Future Work

In this paper we have presented a technique that can be used by potential attackers to detect the presence of any Honeyd-based honeypots. They can subsequently modify their attack to either avoid the honeypot or send misleading traffic to the honeypot to hide their actions. The technique outlined in this paper is based on a flaw in the Honeyd packet fragment reassembly procedure.

We have implemented a proof-of-concept tool called winnie that demonstrates how an attacker can exploit this flaw. We have also presented results from simple experiments that illustrate the feasibility of performing network reconnaissance in order to detect Honeyd-based honeypot deployments. Our experiments reveal that it would only take an attacker less than 3 hours to scan a medium to large network containing 2^{18} hosts with a low amount of scan traffic. During our experiments we also uncovered some mitigating factors that affect and limit the scope of this vulnerability. We have implemented a patch for Honeyd that fixes this security flaw and have provided it to the maintainers of the Honeyd suite.

We are looking into the feasibility of conducting large scale Internet wide scans to detect unpatched Honeyd implementations. This would be an interesting experiment in order to determine the scope and coverage of various Honeyd-based honeypot deployments worldwide.

References

[1] D. MOORE, C. SHANNON, G. VOELKER, AND S. SAVAGE. Network Telescopes: Technical Report. *CAIDA Technical Report* (2004).

[2] D. SONG AND N. PROVOS. arpd. *http://www.honeyd.org/tools.php* (Feb 2003).

[3] G. ZIEMBA, D. REED, AND P. TRAINA. RFC: 1858 - Security Considerations for IP Fragment Filtering. *http://www.ietf.org/rfc/rfc1858.txt* (Oct 1995).

[4] I. MILLER. RFC: 3128 - Protection Against a Variant of the Tiny Fragment Attack. *http://www.ietf.org/rfc/rfc3128.txt* (June 2001).

[5] IETF. RFC: 791 - Internet Protocol DARPA Internet Program Protocol Specification. *http://www.ietf.org/rfc/rfc791.txt* (Sep 1981).

[6] J. BETHENCOURT, J. FRANKLIN, AND M. VERNON. Mapping Internet Sensors with Probe Response Attacks. *In Proceedings of the 14th USENIX Security Symposium* (Aug 2005), 193–208.

[7] M. BAILEY, E. COOKE, F. JAHANIAN, J. NAZARIO, AND D. WATSON. The Internet Motion Sensor: A Distributed Blackhole Monitoring System. *In Proceedings of Network and Distributed System Security Symposium* (Feb 2005).

[8] M. HANDLEY, C. KREIBICH, AND V. PAXSON. Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics. *In Proceedings of the 10th USENIX Security Symposium* (Aug 2001).

[9] N. PROVOS. A Virtual Honeypot Framework. *In Proceedings of the 13th USENIX Security Symposium* (Aug 2004).

[10] N. PROVOS. Honeyd Project. *http://www.honeyd.org* (Jan 2005).

[11] TEAM CYMRU. The darknet proejct. *http://www.cymru.com/Darknet* (Jun 2004).

[12] V. YEGNESWARAN, P. BARFORD, AND V. PAXSON. Using Honeynets for Internet Situational Awareness. *In Proceedings of HOTNETS* (Nov 2005).

[13] Y. SHINODA, K. IKAI, AND M. ITOH. Vulnerabilities of Passive Internet Threat Monitors. *In Proceedings of the 14th USENIX Security Symposium* (Aug 2005), 209–224.