Application Presence Information based Source Address Transition Detection for Edge Network Security and Management

Jun Bi and Jianping Wu

Tsinghua University, Beijing 100084, China

Summary
Source address transition technologies, such Network Address Translation and Proxy, can be used to provide unauthorized private address space. The source address of packets originated in the private address space will be changed by NAT gateway or proxy server, which is hard for service providers to manage the edge network and trace source of attacks. This paper presents the source address transition detection methods based on application presence information to enhance the edge network security.

Key words:
Source address transition, Application presence Information, Edge network

Introduction

Today, Internet is an important information infrastructure worldwide. Internet provides a low-cost and open data transmission approach for various applications. However, it is realized that the security and management of the Internet is very weak to meet security requirements, due to the problem of network architecture. At the initial stage of Internet when the users and sub-networks can be trusted within the academic community, it was assumed that most network problems came from the link breaking or node failure. However, such situation changed in middle of 1990s when Internet was commercialized. The edge network is not managed by ISPs, so users are no longer trusted. ISPs need to know the source of attacks, need to know who is using the Internet for accounting and management purposes. Because the Internet is a huge distributed system, security and management problems have to be resolved distributely and locally in the edge network.
Source Address Transition technologies, such as NAT (Network Address Translator) [1] and Proxy, change the source address of packets in the middle of communication. They can be used to provide private address space. However the unauthorized private address space brought serious problems in past years:

(1) Management Issues. NAT and Proxy break the end-to-end model of Internet. ISPs need to know the real topology, which is hidden by private address space. The deployment of p2p applications will be accelerated and the performance will be enhanced without NAT and proxy.
(2) Security issues. IP source address, which is an important identifier to trace end users, is changed by NAT and proxy. Therefore it's hard to accurately trace attackers inside the private address space.
(3) Billing issues. Some service providers that charge fixed monthly fees on each authorized IP address can not bill to hosts in unauthorized private address space. An unauthorized proxy server could be used to access restricted network resources.
Therefore, monitoring the usages of NAT gateways and proxy servers are helpful for service providers to administer private address space according to their policies, and enhance the network management and security of edge networks.
The existing detection approaches mainly use network layer or transport layer information; therefore, those methods could be undermined by modifying implementations of gateways or the TCP/IP stacks in hosts inside the private address space. In this paper, NAT and Proxy detection methods are proposed based on passively monitoring application presence information, which is usually not easily modified by NAT gateways or hosts having private addresses.
The rest of this paper is organized as follows: Section 2 discusses related works. Section 3 presents the NAT detection algorithm based on instant messaging information. Section 4 presents passive proxy detection method based on application fingerprints. Section 5 introduces experiments and Section 6 summarizes the paper.

2. Related Work

The current NAT detection methods can be summarized as follows:

Manuscript received January 5, 2007
Manuscript revised January 25, 2007

(1) To find some characteristic from a packet that forwarded by a NAT gateway;
(2) To find out the number of users on one IP address.
There were four major methods proposed: checking the TTL field in IP header [2]; checking IPid field in the IP header [3]; checking OS fingerprints [4][5][6]; and checking Clock skew [7].
The basic assumption of TTL method is: for a specific operating system, the initial TTL value is determined. After an IP packet passing a NAT gateway, the TTL value is decreased by 1. This method is quite simple but it is easy to be avoided if NAT gateway ignores the processing of TTL field.
The basic assumption of IPid method is: the IPid value is increased by 1 for every IP packet sent out from a host. So if there are multiple hosts behind a NAT gateway, then multiple IPid sequences will be observed. So we can know not only the existence of NAT gateway, but also the number of private hosts behind the NAT gateway. However it has to check every packet and setup IPid sequence for each potential host, so it is unrealistic for a larger network or a high-speed link.
The basic assumption of OS fingerprinting method is that different OS has different fingerprints. The available fingerprints include TTL field, initial TCP window size, DF field in IP header, etc. This method counts the number of hosts in a private address space by counting the number of different OS fingerprints in packets coming from the same source address. However, if all users have the same operating systems, then this method will fail.
The clock stew method counts the number of hosts in a private address space by partitioning packets that coming from the same source address into sets corresponding to different sequences of time-dependent TCP or ICMP timestamps and applying a clock skew estimation technique on the sets. One possible way to defeat this method is to make modification on NAT gateway to delete the timestamp option in TCP SYN packets. Then both side of the TCP would not use TCP timestamp option any more and thus this method fails.
The common drawback of the above methods is that they rely on network layer, or transport layer information, which are possible for NAT vendors or users inside private address space to elude these detection methods by making modification on NAT gateways or TCP/IP stacks on hosts.
In this paper, we propose to use application presence information for NAT detection.
There are many studies and tools on proxy detection. This paper focuses on unauthorized proxy inside an edge network. There are four situations:
(1) External Client accesses external Server via an internal proxy.
(2) External Client accesses internal Server via an internal proxy.
(3) Internal Client accesses external Server via an internal proxy.
(4) Internal Client accesses internal Server via an internal proxy.
Sometimes, the administration policy of an edge network restricts external users to access internal network resources (e.g. the internal library server) or external resources (e.g. a purchased service provided by an external server) by the unauthorized internal proxy.
Sometimes, administration policy of an edge network restricts internal users to use the unauthorized internal proxy to access internal or external servers to avoid billing or access control.
Currently, network administers can use active detection method to scan proxy servers, such as the tool *proxycheck* [8]. The main problems of active detection method are:
(1) It will take a long time to scan a large network.
(2) It brings extra detection packets into the network by consuming the bandwidth.
(3) It will fail if the proxy servers are configured with access control.
In this paper, we propose a passive Proxy detection method based on application layer information.

3. Detection on Network Address Translator

3.1 Application Presence Information

Some network applications are user-oriented and designed to be used by an individual on one host. Therefore, normally users run only one application instance on one host. If there is more than one instance of such applications (we call it application presence information) running on one IP address, it is likely that there is a NAT gateway on this IP address.
From the prevalent network applications, such as Web, Email, FTP, IM (Instant Messaging), etc., we choose IM as the application for detection, for the following reasons:
(1) Usually, only one instance of one type of IM application runs on a host. Some IM applications (e.g., Microsoft MSN Messenger) have the limitation that only one instance can be running on one desktop. It is also reasonable that people usually do not run two or more instances of each type of IM at the same time.
(2) Popular IM applications (e.g., MSN Messenger, Yahoo Messenger and Google Talk) have a large number of users.
(3) IM users often keep the IM clients running for a relatively long period.
Those characteristics make the IM based information can be used for detection, and make detectors have more chance of detecting the private address space. In this paper

we choose Google Talk, Microsoft MSN Messenger, and Tencent QQ (a popular IM tool in China) as the IM applications for NAT detection.

3.2 Typical IM Presence Information

Google Talk client sets up a TCP connection to the Google Talk servers at service port 5222 to transfer instant messages and presence information. This TCP connection will last the whole session.
The MSN Messenger client will connect to three kinds of servers: Dispatch Server (DS), Notification Server (NS) and Switchboard Server (SS). MSN Messenger servers use port number 1863 as service port. The MSN Messenger client periodically sends the ``PNG" command to NS. This command is used to ensure that the TCP connection to be alive. The format of the command is:
PNG\r\n
Tencent QQ has a majority of Instant Messaging users in China. A client of QQ can use either TCP or UDP to communicate with the server. QQ also has a keep-alive mechanism: the client sends a keep-alive packet to the server every 60 seconds.

3.3 Detection Algorithm

Definition 1: Presence Packet. Presence Packet denotes the packets that carry the application presence information.
Definition 2: Presence Channel is defined by a 5-tuple <source IP address, destination IP address, source port, destination port, payload characteristic>. Given the payload characteristic, the source IP (the suspicious target address *i* we observing) and the destination port number (service port number of a specific IM application IM_j), then the presence channel c_{ijkl} can be determined by destination IP *k* and source port number *l*.
Definition 3: A timer t_{ijkl} is set for each c_{ijkl} to denote the final updated time of that presence channel.
Definition 4: $TMAX_j$ is set for the maximum idle time of the presence channel for each IM application IM_j.
Definition 5: A threshold TH_j is set for the maximum number of allowed concurrent presence channels for each IM application IM_j.
Figure 1 shows the list of IM presence channel records in NAT detector. For each target IP address in the edge network, a list of presence channels is maintained for each kind of IM application. A presence channel record contains destination IP address, source port, and a timestamp set for each channel to denote the latest update time for that channel.

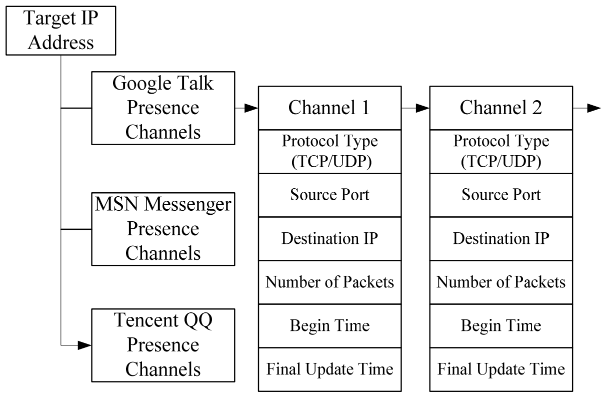

Fig. 1. Presence channel table.

Based on the IM presence characteristics discussed in section 2, we assume that there is more than one host running the same IM application behind a NAT gateway and there is only one instance for each type of IM application on one host. A NAT detector captures IM presence channel packets and counts the number of presence channels, if the number of presence channels exceeds a threshold, it will report detection results. The expired channel records will also be removed, if the channel record hasn't been updated in the maximum value of the time gap between two packets for one presence channel. Figure 2 shows the NAT detection algorithm.

Step 1: For each target IP address *i* in the monitored access network, the detector maintains a presence channel list C_{ij} for each IM_j.

Step 2: When a presence packet of IM_j coming from target IP address *i* is captured, the detector checks destination IP address *k* and source port number *l* to get the presence channel c_{ijkl} and determines whether belongs to an existing channel.
if $c_{ijkl} \in C_{ij}$, then $t_{ijkl} = 0$ (reset the update time);
else, create a new presence channel c_{ijkl} , $C_{ij} = C_{ij} \cup \{ c_{ijkl} \}$.

Step 3: The detector counts the number of current presence channels of target *i* and IM_j to make the verdict.
n_{ij} = number of members in C_{ij} .
if $n_{ij} \geqq TH_{ij}$, then makes a verdict that this IP address *i* is a NAT gateway address.

Step 4: The detector checks each t_{ijkl} to remove expired presence channel c_{ijkl}.
if $t_{ijkl} \geqq TMAX_j$, then $C_{ij} = C_{ij} - \{ c_{ijkl} \}$.

Fig. 2 NAT detection algorithm.

To capture the presence channel packets for Google Talk, we apply IP address and port number of its server as the packet filtering criteria. Google Talk clients connect to only one server in the whole log-on process. The total number of all the Google Talk servers is not large. We collected the IP addresses of the servers by domain name "talk.google.com". When a packet passes through the detection point, we check whether the destination address in this packet is one of the Google Talk servers and whether the destination port is 5222, to judge whether it is a presence channel packet for Google Talk.

To capture the presence channel packets for MSN Messenger, we apply port number and payload characteristic as the packet filtering criteria. From the observation on MSN Messenger discussed in last section, we can see that TCP packets between the target and one of the Notification Servers are what we want to filter out. Since there are fairly a large number of notification servers and it is difficult to collect all addresses of NS servers, we did not use address as a filtering condition. We filter presence channel data of MSN Messenger by checking whether the port number is 1863 and whether there is a string ”PNG” in the payload.

Similarly as MSN messenger, to capture the presence channel packets for Tencent QQ, we use port number and the payload characteristic as the packet filtering criteria. We filter presence channel data of Tencent QQ by checking whether the port number is 8000 (UDP) or 80 (TCP) and whether there is a keep-alive command in the payload.

There are some IM applications (such as Tencent QQ) that use UDP in transmitting presence information. For these IMs, as long as the client port number is not often changed and NAT doesn't use different port number to transfer UDP packets for the same presence channel, their packets can be treated in the same way as TCP packets.

After capturing an IM presence channel packet, the NAT detector can find the channel list according to the source address (target IP address in the edge network under detection) and IM type (based on the characteristics of destination address, destination port or payload). Then the presence channel record is updated by the source port and destination address information of packet. If the channel record exists (the source port and destination IP address can be found in the channel list), change the “latest update time”. Otherwise, create a new channel record and append it to the list.

4. Proxy Detection

4.1 Socks and HTTP CONNECT Proxy Detection

Besides NAT, hosts in private network could use the *socks proxy* [9] and HTTP proxy in *CONNECT mode* [10] to access the Internet. The socks proxy and HTTP CONNECT proxy act quite like NAT: they all simply relay the data. If there are many users using IM behind a socks proxy or HTTP CONNECT proxy, there would also be many presence channels in existence. Therefore the NAT detection method based on Instant Messaging can be also used to detect the presence of socks proxies and HTTP CONNECT proxies.

4.2 Proxy Detection

Unlike NAT, an HTTP proxy encapsulates forwarded user data, so we can not directly used the method proposed in last section.

One method to detect HTTP proxy is to find HTTP proxy fingerprints. As figure 3 shows, a packet forwarded by HTTP proxy contains HTTP proxy fingerprints: “Via” and “X-Forwarded-For”. “Via” is inserted by a HTTP proxy to indicate that this packet has been forwarded by that proxy and it is used for avoiding loop. “X-Forwarded-For” is used by software *squid* [11] to express this packet is forwarded for this source address. Although it’s not standardized, some other software also uses “X-Forwarded-For”.

```
GET / HTTP/1.0
Accept: */*
Accept-Language: zh-cn,en-us;q=0.5
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US;
rv:1.8.0.1) Gecko/20060124 Firefox/1.5.0.1
Host: www.google.com
Via: 1.0 proxy.domain:9981
(squid/2.5.STABLE13)
X-Forwarded-For: client.host.name
Cache-Control: max-age=259200
Connection: keep-alive
```

Fig. 3 A packet example.

We noticed that some applications other than WEB also support accessing application servers via the HTTP proxy. The application’s data are encapsulated in HTTP packets. Therefore another detection method is checking whether the data carried in HTTP packets is another application’s data, as shown in figure 3.

In this paper, we still choose Instant Messaging as the non-WEB application, because IM is the third most popular application.

In section 3, we analyzed characteristics of an Instant Messaging application working without a HTTP proxy. In the HTTP proxy mode, the characteristics of MSN messenger are:

(1) A MSN Messenger client connects to the server gateway.messenger.hotmail.com, which acts as a notification server, at service port 80. A Session ID is assigned to keep the connection stable. All control commands are encapsulated in HTTP packets. The client uses HTTP POST to let the server run a script “/gateway/gateway.dll”.

(2) The server can only reply requests from MSN Messenger clients, but can not actively send out commands. Therefore, the client has to periodically send empty request to the server so that the server can send out the command by replying to the empty request.

```
POST
/gateway/gateway.dll?SessionID=217136341.5271
HTTP/1.0\r\n
Accept: */*\r\n
Content-Type: text/xml; charset=utf-8\r\n
…
```

Fig. 4. A HTTP request sent by MSN messenger client.

As shown in figure 4, we can use "POST /gateway/gateway.dll" as the fingerprint of presence channels and use the algorithm presented in section 3 to count the number of channels and detect HTTP proxies. Because the fingerprint contains enough information, even when we set the threshold as 1, the detection results are still accurate.

5. Analysis and Comparison

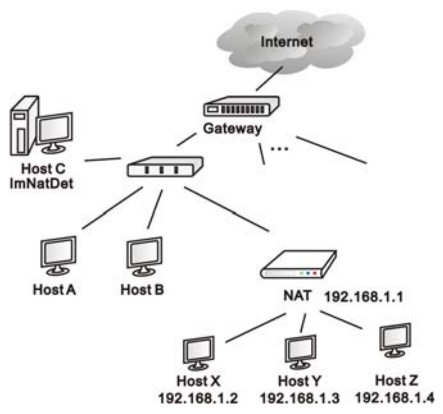

Fig. 5 Experiments on NAT detection.

We implemented the NAT and Proxy detector on Linux. The captured length of a packet is:

$$Len_{IP} + Len_{TCP} + Max(Len_{IM_1} + Len_{IM_2} + ... + Len_{IM_n})$$

Len$_{IP}$ denotes the length of IP header. *Len*$_{TCP}$ denotes the length of TCP header. Len_{IM_i} denotes the length of a type of IM application payload used for detection. For MSN Messenger, the length of payload "PNG\r\n" is 5 bytes. For Tencent QQ, the length of keep-alive command is 7 bytes. For HTTP proxy detection, we need to capture the whole packet.
The timeout timer and maximum number of presence channels are set as the value shown in table 1.

Table 1: Parameter value setting

	Timeout timer	*Threshold*
Google Talk	50 seconds	2
MSN Messenger	100 seconds	2
Tencent QQ	180 seconds	3

We did primary experiments with the detector in the environment shown in figure 5. We ran detector and the well-known tool *p0f* [4] on host C and ran MSN Messenger/Tencent QQ on host A, B, X, Y, and Z (X, Y, and Z are hosts inside a private address space). The NAT detection results are shown in table 2. The reason that the last experiment fails is because the total number of IM users is below the threshold. We noticed that *p0f* failed in all test cases, because host X, Y and Z uses the same Windows operating system. The experimental results validate the detection method proposed in this paper.

Table 2: NAT detection results

Hosts X, Y and Z in private address space	*Normal host A and B*	*Results*
X and Y run Google Talk, Z runs MSN	A and B run Google Talk	Detectio n
X, Y, and Z run Tencent QQ (single instance)	A and B run Tencent QQ (single instance)	Detectio n succeeds
X and Y run MSN Messenger	A and B run MSN Messenger	Detectio n succeeds
X runs Google Talk and Y runs MSN Messenger	No IM application running on A or B	Detectio n fails

We also did some experiments in Tsinghua campus network. According to the detection results, we found 4254 IM users and 162 NAT gateways out of 33860 active IP addresses. The max number of IM application used in one private address space is 75 QQ presence channels and 14 MSN presence channels. Compared with experimental results we did in the same environment using *p0f* version 2.06 (threshold is set to 30%), our detector found 54 more NAT gateways, which *p0f* could not detect because hosts in the private address space use the same operating system. Figure 6 shows the experimental environment for Proxy detection. We ran both detector and *p0f* on host D and ran the socks proxy *CCProxy* [12] on host C. The detection results are shown in table 3. The reason that the second experiment fails is because the total number of IM users is below the threshold. The reason *p0f* fails in all test cases is because it can only observe TCP/IP stack fingerprints of host C.

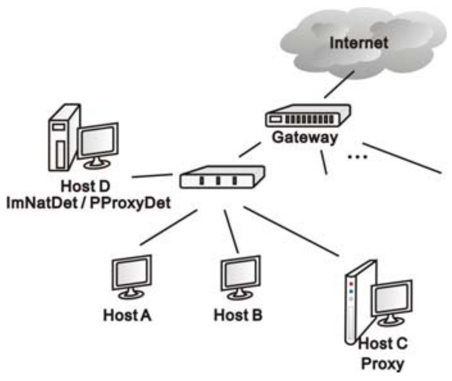

Fig. 6. Experiments on proxy detection

Table 3: Socks proxy detection results

Scenarios	*Results*
Host A and B run MSN Messenger using socks proxy on host C.	detector succeeds, p0f fails.
Host A runs MSN Messenger using socks proxy on host C. Host B runs MSN Messenger without using socks proxy.	detector fails, p0f fails.

We installed HTTP Proxy squid on host C and detector on host D. As shown in table 4, we noticed that even if only one host runs an IM application, we can still detect the HTTP Proxy based on the fingerprint.

Table 4: HTTP proxy detection results

Scenarios	*Results*
Host A and B run MSN Messenger using HTTP proxy on host C.	detector succeeds
Host A runs MSN Messenger using HTTP proxy on host C. Host B runs MSN Messenger without using HTTP proxy.	detector succeeds

6. Conclusion

This paper presents the detection on unauthorized source address transition by capturing and analyzing application presence information to enhance edge network security and management for service providers. To the best of our knowledge, it is the first time that application presence information has been used in NAT and Proxy detection. In addition to passive method, we are working on the active methods and the enhancement of algorithm efficiency.

References

[1] P. Srisuresh and K. Egevang, Traditional IP Network Address Translator (Traditional NAT), RFC3022, Jan 2001.
[2] P. Phaal, Detecting NAT Devices Using Sflow, URL http://www.sflow.org/detectNAT, 2003.
[3] S.M. Bellovin, A Technique for Counting NATted Hosts. proc. of 2nd Internet Measurement Workshop, 2002: p. 267-272.
[4] M. Zalewski, Passive OS Fingerprinting Tool, URL http://lcamtuf.coredump.cx/p0f.shtml.
[5] W. Kaniewski, Detect NAT Users in Your LAN, URL http://toxygen.net/misc/.
[6] M. Ulikowski, NAT Detection tool, URL http://elceef.itsec.pl/natdet/.
[7] T. Kohno, A. Broido, and K.C. Claffy, Remote Physical Device Fingerprinting, proc. of IEEE Symposium on Security and Privacy 2005, May 2005.
[8] M. Tokarev, Proxycheck: Open Proxy Checker, URL http://www.corpit.ru/mjt/proxycheck.html.
[9] M. Leech, SOCKS Protocol V5, RFC 1928, Mar 1996.
[10] R. Khare, S. Lawrence. Upgrading to TLS Within HTTP/1.1. RFC 2817, May 2000.
[11] Squid, URL http:// www.squid-cache.org/.
[12] CCProxy, URL http://www.youngzsoft.net/ccproxy.

Jun Bi received the B.S., M.S. and Ph.D. degrees in Computer Science from Tsinghua University, Beijing, China. From 1999 to 2003, he worked for Bell Laboratories Research and Bell Labs Advanced Technologies, New Jersey, USA. Currently he is a full professor and director of Network Architecture & IPv6 research laboratory, Network Research Center, Tsinghua University. His research interests include IPv6 next generation network architecture and protocols.

Jianping Wu received the B.S., M.S. and Ph.D. degrees in Computer Science from Tsinghua University, Beijing, China. Currently, he is a full professor in the Computer Science Department, Tsinghua University. He is also the director of the China Education and Research Network. His current research interests include computer network architectures, next generation Internet, and formal methods.