Application-based TCP Hijacking and Its Application to Windows Live Messenger

Abstract

We present Application-Based TCP Hijacking (ABTH), a technique that exploits flaws within the transport and application protocols to inject data into an application session without either end noticing it. Following the injection of a TCP packet, ABTH is used to re-synchronize the TCP stacks of both the server and the client. To demonstrate the effectiveness of ABTH, we present an attack on Windows Live Messenger in which we use ABTH to perform command spoofing and user impersonation. Due to its generic nature, ABTH can be mounted on a variety of application protocols. We also propose specific countermeasures to thwart and/or limit ABTH, such as strict Ethernet switching prevention and cryptographic protection of messages.

1 Introduction

Since its first specification in 1974 [1], the Transmission Control Protocol (TCP) has grown to become the core transport protocol for a vast number of applications including HTTP, FTP, SMTP, and TELNET. The security properties of these application protocols partially depend on the security of TCP and the underlying Internet Protocol (IP). Many network attacks have shown prominence over the past decades in regards to vulnerabilities of the TCP design [2]. While preventive mechanisms have been developed to throttle or even eliminate most of these attacks [3], the last item on the list of TCP vulnerabilities is yet to be written.

In this paper, we present and suggest countermeasures for a new way of attacking a TCP-based communication. Our technique is similar to TCP hijacking [4] but dwells on application-layer protocols. Traditional TCP hijacking attacks exploit vulnerabilities of the transport and network layers. However, the majority of these attacks have been circumvented through the use of hardware switches and routers [3], which provide countermeasures against these direct low-level attacks. On the other hand, Application-Based TCP Hijacking (ABTH), the exploit method presented in this paper, utilizes loopholes in the logistics of application-level communication to evade policy enforcement for the transport and IP layers. Trivial design features of application protocols become fatal vulnerabilities that can be exploited by ABTH.

To demonstrate the feasibility and effectiveness of ABTH, we present a case study of attacking communications of Microsoft Windows Live Messenger (WLM). With instant messaging (IM) quickly becoming ubiquitous at home and at work [5], WLM represents one of the largest IM networks, with 27 million users in the US alone [6].

We show that ABTH can be used to compromise the privacy and confidentiality of WLM users. By attacking Microsoft Notification Protocol (MSNP) using ABTH, an attacker is able to spoof any command available to a WLM client and impersonate any contact known to the victim. As a result, unauthorized messages can be delivered to various contacts. Such an attack could result in at minimum user inconveniences or embarrassment but in extreme cases it could even lead to devastating results.

Even though the reported case study is limited to WLM, we believe that due to its generic nature, Application-Based TCP Hijacking can be mounted on a variety of application protocols.

To effectively prevent ABTH, application protocols could encrypt packets and thus put off TCP security exploits. Application developers

no doubt would need time to redesign their applications in order to deploy these changes. In the meantime, Internet service providers (ISPs) could employ stricter security controls on the network layer.

The remainder of the paper is organized as follows. Existing attacks on TCP and related work are discussed in Section 2. The theory and general operation of ABTH is discussed in Section 3. The feasibility of ABTH on MSNP to spoof a command and impersonate a contact is demonstrated in Section 4. The limitations of ABTH and countermeasures against it are discussed in Section 5. The paper is concluded in Section 6.

2 Background and Related Work

In this section, we provide background on TCP necessary for understanding the rest of the paper. Additionally, details of existing security flaws are presented.

2.1 Overview of TCP

TCP is a connection-oriented transport protocol that guarantees reliable in-order delivery of packets and minimizes network congestion [1]. One way the protocol achieves these properties is through the use of sequence numbers and acknowledgement (ack) numbers. All TCP packets contain a *sequence number*, which represents the n-th byte of data transferred, and an *ack number*, which represents the last known n-th byte of data received. Packets containing data have the SYN and ACK flags set; packets with no data may have the ACK flag set to denote an empty acknowledgement packet. Many attacks, including ABTH, on TCP and the upper-layer application protocols exploit vulnerabilities of the *sequence* and *ack numbers* mechanism. In the following section, we discuss those attacks that are most relevant to ABTH.

2.2 Attacks on TCP

Extensive research on TCP has resulted in the development of a variety of techniques for exploiting TCP/IP protocols. These attacks include ARP poisoning, IP spoofing, ICMP redirection, DNS poisoning, TCP man-in-the-middle, and combinations of thereof [2], [7], [8]. In contrast to denial of service attacks (e.g., TCP SYN flood, IP "ping of death"), the purpose of these attacks is to penetrate and compromise systems. The aforementioned attacks are widely documented and there exists numerous tools (e.g., Hunt, Snort, Ettercap, and Juggernaut) to scan for or exploit TCP vulnerabilities in systems [9].

Many commercial Internet service providers (ISPs) and corporate networks have adopted protection mechanisms provided by switches and routers, which effectively thwart these attacks. The success of these attacks typically depends on the exploitation of two or more vulnerabilities. For example, TCP man-in-the-middle, also known as TCP session hijacking, requires the capability to view network traffic and to execute ARP poisoning [4], the latter of which is normally disabled by modern switched equipment [10].

In the case of TCP session hijacking, by eavesdropping on a TCP session, an attacker is able to calculate the expected *sequence* and *ack* numbers and can than proceed to inject a spoofed TCP packet. The spoofed packet would contain the numbers expected by the recipient and the source IP address of the legitimate sender. Although the packet was sent by a spoofed sender, the recipient would accept the packet as an authenticate packet. However, following this spoof, the connection is effectively broken as the expected *sequence* and *ack* numbers of the sender and receiver are no longer in sync. ABTH improves upon conventional TCP session hijacking allowing an attacker to re-synchronize TCP state machines of the legitimate sender and receiver.

Vanilla TCP session hijacking has also the disadvantage of ensuing an ACK storm [11], in which both the sender and receiver continuously transmit ACK packets, each trying to correct the other with its own *sequence* and *ack* numbers. An RST packet typically follows an ACK storm causing a disconnection of the TCP connection. As a result of the disconnection, the user, either the sender or the receiver, would notice a hang in the application indicating the attack. Thus, vanilla TCP session hijacking is practical only if a packet containing one command could accomplish the objective(s) of the adversary. In order to prevent an ACK storm, the attacker can ARP poison both ends of the connection and then relay messages for the rest of the session. However, as mentioned before, most Ethernet switching equipment provides means of preventing ARP poisoning.

3 Application-Based TCP Hijacking

We developed an extension of the existing TCP session hijacking attacks. Application Based TCP Hijacking (ABTH) re-establishes the validity of the connection between the client and the server without utilizing ARP poisoning or any other techniques below the transport layer. ABTH achieves this objective through the exploitation of vulnerabilities found in the application-layer protocol; therefore, ABTH design is specific to each application protocol. ABTH has two requirements on the capabilities of the adversary: the abilities to view network traffic and to spoof IP packets. Both of these requirements are satisfied when the attacker and the victim (i.e., either the client or the server but not necessarily both) are co-located on the same local area network (LAN).

Many application protocols intended for two-way communication, such as Telnet and FTP, maintain a persistent TCP connection. The persistent TCP connection serves two purposes: it maintains an application session and bypasses network address translation (NAT) restrictions. The former purpose allows applications to limit the frequency of relatively costly steps, such as authentication, to once-per-session and allows future commands to be sent without verification of the sender's identity. The latter purpose enables bi-directional commands that are capable of traversing through NAT devices, a critical requirement for some applications, e.g., instant messaging.

Furthermore, application protocols typically specify trivial commands that do not modify the state of applications. An example of such a command is *ping*, which is used to ensure the validity of the open connection. ABTH exploits these seemingly trivial commands to perform session hijacking and TCP stack re-synchronization.

ATBH repairs the broken connection by realigning the *sequence* and *ack* numbers of both the server and client using seemingly trivial commands. Besides ping command (or its equivalents), application protocols typically provide the client and server with asymmetric commands, which leads to different commands for the client and the server. If the server ping and the client ping have different data lengths, the injection of certain multiples of each ping could counterbalance the original difference created by the injected packet. As each ping to the server and client inflate the *sequence* and *ack* numbers of the server and client, respectively, the injection of a carefully calculated number of pings directed to each side of the connection can result in the re-synchronization of the server and the client. Once this is achieved, the connection is "repaired" and new legitimate and illegitimate application commands can be received and processed successfully. It is assumed that the server and client pings have different lengths, but as the WLM case study will show, this assumption is not particularly important if *whitespace padding* is allowed to arbitrarily increase the size of packets.

During the "repair" phase, once the recipient receives a ping, it replies with an ACK packet. As the ACK packet does not correspond to the expected *sequence* and *ack* numbers of the spoofed sender, the sender quickly replies with its own ACK packet, thus initiating an ACK storm. However, this ACK storm is short-lived as an attacker then proceeds to inject a new ping with correctly calculated numbers to comfort the server and/or client. This ACK storm stops immediately, and no RST packets are sent to disconnect the session. This process continues until the *sequence* and *ack* numbers of each side of the connection are re-aligned.

In comparison to typical TCP hijacking methods which involve exploits on lower layers of the network stack [4], for which detection and prevention mechanisms are commonly put in place, the exploitation of the application protocol allows ABTH to avoid the use of ARP poisoning and other exploits prohibited and impractical in well-controlled networks as well as to evade existing intrusion detection systems.

To demonstrate the feasibility and effectiveness of ABTH, we next describe our experience with employing our attack to compromise Windows Live Messenger.

4 Case Study: Attacking Windows Live Messenger with ABTH

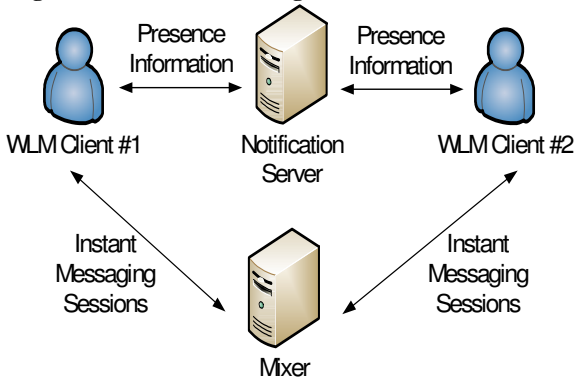
First released in 1999, Windows Live Messenger (WLM) has since grown to become one of the world's most popular instant messaging (IM) services. Current version of WLM utilizes Microsoft Notification Protocol (MSNP) to communicate to servers within the .NET Messenger Service [12].

4.1 .NET Messenger Service

The .Net Messenger Service consists of a centralized cluster of servers where they provide WLM clients with "Presence and Rendezvous services required for user-to-user communication" [13]. Two types of servers within the cluster include the notification servers (NS) and mixers [14].

When a user logs into their WLM account, a persistent TCP connection to the NS is opened. The connection between the WLM client and the NS must always remain active. A lost connection with the NS results in the client being logged out and disconnected from the messaging service. Through this connection, the user's contact list, presence data (e.g., user status, email notification), and access management information (e.g., forward list, reverse list) are synchronized between the NS and the WLM client. Another function of the NS is to setup connections to the mixer server. Mixer servers handle all IM sessions between clients.

Figure 1. The .NET Messenger Service Infrastructure



4.2 Microsoft Notification Protocol

MSNP is a text-based application protocol employed for communications between WLM clients and servers in the .NET messenger service. Although the protocol was first intended to be an open standard [13], it has since become proprietary and has undergone numerous revisions. Due to the unencrypted nature of the protocol, attempts to reverse-engineer MSNP have been proven successful [16, 17].

The protocol consists of a series of UTF-8 and URL/XML encoded commands. Each MSNP command is represented by three capital letters (e.g., command PNG represents ping).

Table 1. List of MSNP Commands

MSNP Command Name	Command Breakdown	Description	Size (bytes)
PNG	PNG\r\n Command Name	Client command to ping the NS server	5
QNG	QNG 12\r\n Command Name: QNG; Transaction Identifer: 12	NS response to the PNG	8
CHL	CHL 0 18359197213018043653\r\n Command Name: CHL; Challenge String: 18359197213018043653	NS command to ping the client	28
QRY	Transaction Identifer: 12; Client ID String: msmsgs@msnmsgr.com; Payload Size: 32 QRY 12 msmsgs@msnmsgr.com 32\r\n 8f2f5a91b72102cd28355e9fc9000d6e MD5 Fingerprint	Client response to the CHL	60
QRY	QRY 12\r\n Command Name: QRY; Transaction Identifer: 12	NS response to the client's QRY	8
PRP	PRP 123 MFN hacked\r\n Command Name: PRP; Transaction Identifer: 123; My Friendly Name: MFN; New Display Name: hacked	Client command to change display name	20
RNG	Type of Authentication: CKI; Session ID of Mixer Session: 1312697181; IP address & port of Mixer: 65.217.175.28:1863 RNG 1312697181 65.217.175.28:1863 CKI 17021696.318151 msnp_bob@hotmail.com Bob U messenger.msn.com 1\r\n Email Address and Display name of client that initiated IM session: msnp_bob@hotmail.com Bob; Authentication String: 17021696.318151	NS command to invite client to a mixer session	102
IRO	Participant Number: 1; Total number of participants: 1; User Handle: 1; Initial Roster: IRO; Participant's email: msnp_bob@hotmail.com IRO 1 1 1 msnp_bob@hotmail.com Bob 1985855532\r\n Participant's display name: Bob	Mixer server command to clients to notify clients of new participants in mixer session	48

MSNP contains a design feature of allowing asynchronous bi-directional pings; both the client and the NS have the ability to initiate a ping. Pings from the client to the NS are achieved through the PNG command; the NS, on the other hand, uses the CHL command to ping the client. In order to avoid disconnection, the receiver of the ping is required to respond; disconnection typically occurs within a few seconds if a response to the ping was not received. Note that the ping-response initiated by the NS entails three transactions (CHL-QRY-QRY). Table 1 specifies the structure of those MSNP commands that are relevant to our ABTH-based attack on the WLM.

In the official MSNP IM application, the client pings the NS approximately every 45 seconds, and approximately every minute in alternative MSNP IM applications such as Pidgin [18]. Within this timeframe, ABTH can exploit these two sets of ping-response commands to eliminate the discrepancy between the *sequence* and *ack* numbers of the client and the NS TCP stacks created by the injection of an MSNP command. Although some strings within the MSNP commands contain pseudo-random numbers or hash values (e.g., challenge string, MD5 fingerprint), we found that the exact these values do not have to be correct for the purpose of mounting an ABTH attack.

In addition, MSNP commands are terminated by carriage return (\r\n or 0x0d0a), which has a size of 2 bytes. The specification's flexibility with white-space padding greatly simplifies the calculations and allows reducing the number of packets necessary to re-synchronize the underlying TCP connection. By applying the ABTH technique, an attacker has the capability of injecting a spoofed MSNP command without disconnecting the underlying TCP session.

In what follows, we describe two examples of spoofing MSNP commands without causing disconnection of the MSNP connection: display name spoofing and identity spoofing. To demonstrate the feasibility of ABTH, we implemented both attacks using real WLM deployment and official Microsoft WLM clients. In the second attack, we implemented our own rogue WLM mixer server, which the official Microsoft WLM clients were connected to.

4.3 Display Name Spoofing

In this section, we describe how we were able to modify a victim's display name to an arbitrary string. For this attack, the MSNP command PRP was employed; its format is shown in Table 1.

We spoofed a PRP command to the NS. Once the packet was received by the NS, it responded to the client by sending back the exact same command but without the additional white space that was added in the original PRP following the carriage return. The NS then updated the user's information resulting in presence updates pushed to all of the user's contacts. While all of the user's contacts were notified of the change, the victim user was completely oblivious to the situation and did not notice the change in their display name locally.

Following the injection of the packet, the TCP stacks of the WLM client and the NS were resynchronized through ABTH. For demonstration purposes, the sequence numbers of the client and the NS are assumed to be 100 and 500 prior to the spoofed packet. During normal operation, NS *ack* number should match client *sequence* number and client *ack* number should match NS *sequence* number, as seen in the first row of Table 2.

Table 2. Display name spoofing: Client and NS seq and ack numbers after each spoof packet. Bold indicates a change in value. Direction indicates whether the command was sent to or received by the attacker.

Command	Direction	From/to	Data Size	Client Seq	Client Ack	NS Seq	NS Ack
Before attack				100	500	500	100
PRP	Sent to attacker	NS	24	100	500	500	**124**
PRP	Received by attacker	NS	20	100	500	**520**	124
Begin ABTH:							
PNG	Sent to attacker	NS	6	100	500	520	**130**
QNG	Received by attacker	NS	8	100	500	**528**	130
Repeat PNG/QNG exchange for 5 more times							
CHL	Sent to attacker	Client	60	100	**560**	**568**	**160**
QRY	Received by attacker	Client	60	**160**	560	568	160
QRY	Sent to attacker	Client	8	160	**568**	568	160

While we could not control the sequence number of the NS or the client, we could alter the *ack* numbers of both sides by spoofing a whitespace-padded packet. In this case, the number of spoofed packets to send to each side of the connection was pre-calculated. The fact that MSNP commands allow white-space-padding greatly simplified our calculations. Table 3 shows the amount of padding that used for this example. Each command was sent or received by the attacker in the order listed in Table 2.

As the last row of Table 2 shows, the client *sequence* number was equal to the NS *ack* number, and the NS *sequence* number was equal to the client *ack* number. ABTH had completed its series of application specific commands and the TCP connection had returned to a synchronized state and was ready to transmit further legitimate and illegitimate packets.

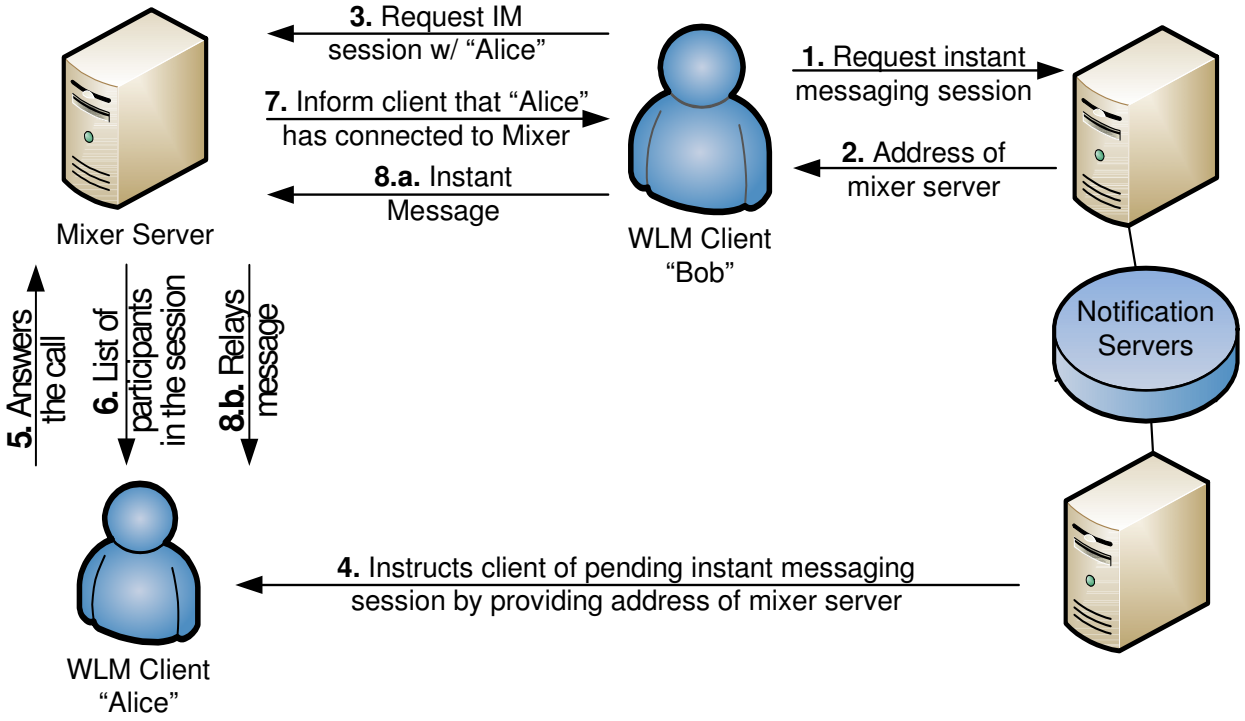
4.4 Identity Spoofing

By exploiting the same vulnerability, ABTH can be used to spoof the identities of WLM participants. As illustrated in Figure 2, when a user wishes to initiate a conversation with another contact, the user first requests an IM session through the NS. The NS returns a session authentication ID and network destination IP address and port that points to a mixer. The client then establishes a persistent TCP connection to the designated mixer. Once the client's connection to the mixer has been established, the client proceeds to invite other contacts to join the conversation. Each invited contact receives an invite through their NS connection to connect to the specified mixer. Once all parties are connected to the mixer, it begins to relay all instant messages between the WLM clients.

Table 3. Padded commands and their sizes.

	Command	Size Before Padding	Size After Padding
Name change	PRP 123 MFN hacked\r\n	20	24
Client ping	PNG\r\n	5	6
Server ping	CHL 0 835919721301804365 3\r\n	28	60
Server ping response	QRY 12\r\n	8	8

Figure 2. Initiating an IM Session



In order to spoof an instant messaging session, we sent a spoofed invite message to the victim as the NS, which correlated to step 4 of Figure 2, instructing the victim to connect to a spoofed mixer. As shown in Table 1, the invite message contained the address of the spoofed mixer server and the identity of the contact that we wished to impersonate, in this case Bob. When the victim, Alice, connected to our own mixer server, the authentication string and the type of authentication (CKI) specified in the invite message was ignored.

Upon receiving the invite message, the victim established a connection to the address specified in the invite message, which in this case was a rogue mixer server. The TCP connection with the NS was now out of order and Alice would soon be disconnected and unable to communicate with the attacker unless the connection was resynchronized. Thus, following the injection of the RNG command, we employed ABTH to mask the effects of the injected packets. Our calculations indicated a minimum spoofing of approximately 30 packets, as shown in Table 4 and Table 5.

In cases such as this, it is crucial that

Table 4. Identify spoofing: Padded commands and their sizes.

	Command	Size Before Padding	Size After Padding
Mixer invite	RNG 1312697181 65.217.175.28:1863 CKI 17021696.318151 msnp_bob@hotmail.com Bob U messenger.msn.com 1\r\n	102	120
Client ping	PNG\r\n	5	5
Server ping	CHL 0 835919721301804365 3\r\n	28	28
Server ping response	QRY 12\r\n	8	8

Table 5. Identity spoofing: client and NS seq and ack numbers after each spoof packet. Bold indicates a change in value. Direction indicates whether the command was sent to or receievd by the attacker. From/to indicates whether the command was sent or received from the NS or the client.

Command	Direction	From/to	Data Size	Client Seq	Client Ack	NS Seq	NS Ack
Before hack				100	500	500	100
RNG	Sent to attacker	Client	120	100	620	500	100
Begin ABTH:							
PNG	Sent to attacker	NS	5	100	620	500	**105**
QNG	Received from attacker	NS	8	100	620	**508**	105
Repeat PNG/QNG exchange 23 more times							
CHL	Sent to attacker	Client	28	100	**648**	**692**	**220**
QRY	Received from attacker	Client	60	**160**	648	692	220
QRY	Sent to attacker	Client	8	160	**656**	692	220
CHL	Sent to attacker	Client	28	160	**684**	692	220
QRY	Received from attacker	Client	60	**220**	684	692	220
QRY	Sent to attacker	Client	8	220	**692**	692	220

ABTH is completed prior to the next client ping, which occurs in roughly 45 seconds. Otherwise, the client would repeatedly ping the NS and very quickly timeout before more packets could be sent to fix the imbalance in the sequence numbers. Depending on the resources of the attacker, this attack can be reasonably accomplished within this timeframe, as our experiments demonstrated.

Once connected to the victim, our rogue mixer server then sent the identities of the participants already in the session through the IRO command, which only included Bob, as shown in Table 1. At this point, we had established a separate and legitimate TCP connection with the victim, who believed the other end of the connection was an authenticated mixer. No additional attack packets were required to send messages to Alice on behalf of Bob, beyond sending correctly formatted MSNP commands.

The victim's WLM client recognized the participant's email and display name as a contact on their contact list and assumed that an authentic instant messaging session had been established with Bob (msnp_bob@hotmail.com). We could now send any messages to the victim through this spoofed connection and there was not much Alice could do to reveal Bob's real identity.

5 Discussion

Our experiments demonstrate that MSNP could be successfully attacked by employing ABTH. The technique, however, has several caveats.

A limitation of ABTH is that the number of pings to send to the client and the server are calculated prior to the restoration phase and are therefore static. If unexpected packets were sent and received to the server and/or client during the restoration phase, this would result in ABTH to fail in re-synchronizing the TCP stacks of the client and the server. As a result, no unexpected network traffic can occur while ABTH is in the process of sending its own packets. This limitation of ABTH can be fixed through the introduction of more complex algorithms capable of making adjustments dynamically.

As ABTH relies on the ability to inject packets that lack authenticity and integrity protection, one countermeasure to circumvent ABTH attacks is to protect application traffic cryptographically. As a case in point, SimpLite-MSN [19], according to its marketing description, sets up RSA keys and encrypts all

IM traffic for MSNP, providing protection against spoofing.

Another defence against ABTH could possibly be intrusion detection logic built into the application. A possible approach would be to limit the number of pings per one unit of time, and thus limiting the attacker's ability to impair the application. However, such a mechanism could possibly be evaded, as the methods of ABTH could adapt. Also, such a restriction on the usage of the protocol would hinder application development.

ABTH reveals the inherent flaw in network protocols that all packets are sent transparently on a shared channel that attackers have access to. Although it is impractical to isolate communication channels for specific applications using hardware, cryptographic isolation should be considered. Even if packets are sent in plain text, a simple regulatory mechanism of the protocol should present mutual authentication with each message, such as sequence numbers for the application-level protocol messages. It would be difficult to inject a packet and expect further messages to be accepted since the application also keeps track of the list of messages. This straightforward yet non-trivial detail also alleviates the application protocol from its dependence on the network layer and the assumption that packets on the network layer are authentic.

Application protocols vulnerable to ABTH, such as MSNP, were not designed to offer secure communication channels. It is imperative that, as protocols become more widely adopted, they are improved upon and revised to accommodate their broadening uses. On the other hand, the initial design of the protocol should be given consideration of various future uses to provide backward compatibility in case a need for modifications arises.

For network service providers, the most effective method to disable ABTH and message spoofing all together would be to restrict IP packets containing designated source IP addresses for each port of a switched network [20].

6 Conclusion

In this paper, we introduced Application-Based TCP session Hijacking (ABTH). By combining the innocuous features of the transport and application layers and the lack of cryptographic protection at either layer, ABTH offers an elegant way of exploiting certain application protocols. We presented a proof of ABTH concept by successfully experimenting with two sample attacks on Windows Live Messenger, Microsoft's popular instant messaging application, which runs over Microsoft Notification Protocol (MSNP).

While some protocols, such as MSNP, were designed to suit flexible environments, as the environment evolved, these features have become subtle vulnerabilities. Although hardware Ethernet equipment has been updated to obstruct lower level attacks (e.g., ARP poisoning), the realm of attacks is reaching beyond the protection capabilities of network devices.

References

[1] *Transmission Control Protocol*, RFC 793, 1981.

[2] B. Harris and R. Hunt. (1999, June). "TCP/IP security threats and attack methods." *Computer Communications* [Online]. 22(10), pp. 885-897. Available: http://dx.doi.org/10.1016/S0140-3664(99)00064-X

[3] I. Dubrawsky, "Safe Layer 2 Security In-Depth," Cisco Systems, Inc., San Jose, CA, 2004. http://cisco.com/warp/public/cc/so/cuso/epso/sqfr/sfblu_wp.pdf

[4] M. Stamp. "Network Security Basics" in *Information Security: Principles and*

Practice. Hoboken, NJ: John Wiley and Sons, 2006, pp. *349-350*.

[5] America Online, Inc. "AOL's Third Annual Instant Messenger Trends Survey," http://www.aim.com/survey/

[6] Microsoft, "WLM Ads Reach Millions of Consumers," http://advertising.microsoft.com/windows-live-messenger

[7] M. Gregg, S. Watkins, G. Mays, C. Ries, R. Bandes, and B. Franklin, *Hack the Stack: Using Snort and Ethereal to Master the 8 Layers of an Insecure Network*. Rockland, MA: Syngress, 2006.

[8] L. Joncheray, "A Simple Active Attack Against TCP" at USENIX, Salt Lake City, UT, 2005.

[9] J. Scrambray, S. McClure, G. Kurtz, "Session Hijacking" in *Hacking Exposed: Network Security Secrets & Solutions*, 2nd ed. NY: McGraw-Hill, 2001, pp. *530-533*.

[10] R. Spangler. (2003, Dec.). "Packet Sniffing on Layer 2 Switched Local Area Networks," Packetwatch Research. www.packetwatch.net/documents/papers/layer2sniffing.pdf

[11] D. Anderson and B. Teague, "ACK Storms and TCP Hijacking," unpublished.

[12] P. Piccard, B. Baskin, C. Edward, G. Spillman, and M. H. Sachs. "MSN Messenger Architecture and Protocol" in *Securing IM and P2P Applications for Enterprise*. Rockland, MA: Syngress, 2006, pp. *96-103*.

[13] T. Fout. (2001, Oct. 1). Inside Windows Messenger – How it Communicates. *Microsoft Technet*. [Online]. Available: http://technet.microsoft.com/en-us/library/bb457041.aspx

[14] C. Torre. (2006, July 13). Windows Live Messenger – What. How. Why. *Channel 9*. [Online] Available: http://channel9.msdn.com/Showpost.aspx?postid=215459

[15] R. Movva and M. Lai. *MSN Messenger Service 1.0 Protocol.* Microsoft. August 1999. tools.ietf.org/id/draft-movva-msn-messenger-protocol-00.txt (accessed February 19, 2008).

[16] M. Mintz and A. Sayers. (2003, Dec. 19). *MSN Messenger Protocol.* [Online]. Available: http://www.hypothetic.org/docs/msn/index.php

[17] 2007, June 4. MSNPiki: Unofficial MSN Protocol Documentation. [Online]. Available: http://msnpiki.msnfanatic.com/index.php/

[18] Pidgin. [Online]. Available: http://pidgin.im/

[19] Secway Global. [Online]. Available: http://www.secway.fr/us/products/simplite_msn/home.php

[20] S.J. Templeton and K. E. Levitt, "Detecting Spoofed Packets" at DARPA Information Survivability Conference and Exposition (DISCEX), Arlington, VA, 2003.