IPoIM: Internet Protocol over Instant Messaging

Ariel Sabiguero Yawelak, Pablo Rodríguez-Bocca and Laura Rodríguez Mendaro

Instituto de Computación - Facultad de Ingeniería
Universidad de la República - Montevideo, Uruguay
Architecture et Modèles de Résaux (ARMOR)
Institut de Recherche en Informatique et Systèmes Aléatoires - Rennes, France
e-mail: {asabigue,prbocca}@{fing.edu.uy,irisa.fr}, lrodrigu@irisa.fr
asabigue@ieee.org

***Abstract*— This work proposes a novel utilization of Instant Messaging (IM) services as an ubiquitous network layer for protocol encapsulation and transmission. As more than 70% of organizations and almost all home users do not block IM traffic, this work evidences a latent vulnerability of the technology that can be exploited. A working prototype was implemented using Open Source Software. The prototype combines an IM client with a tunnel that was used through public IM services to transmit arbitrary content. The measured bandwidth is of the same order of the one provided by a dial-up modem. Preliminary experimental results are presented and initial security considerations suggested.**

Keywords: Instant Messaging, Internet Protocol, tunnel, VPN, OpenVPN, GAIM

I. Introduction

Instant Messaging (IM) is a new set of communication applications and protocols that have widely spread all over Internet. It is a new medium of communications over the Internet. Instant Messaging is a means for sending small, simple messages that are delivered immediately to online users. Acceptable delay and jitter of the applications allows users to establish one-to-one or one-to-many written conversations, which complement classical e-mail and phone communication mechanisms. The speed of this deployment is not only due to the fact that it is a useful application, but that it was from the beginning freely available and simple to use. A regular user can download and install an IM client and set it up. IM clients are designed to require almost no configuration and they can adjust their communication mechanism to bypass firewalls or HTTP proxies. This facts inverted the way applications are deployed in organizations. Network administrators found themselves with a new application already deployed, and being used not only by employees but managers. The general misconception is that text messages are harmless: it is only text. Thus, the most common security scenario for IM deployment within organizations is to allow only text messages and block other exchanges.

With this in mind a proof of concept prototype was developed: IPoIM. The IPoIM is a modified IM client that allows IP traffic tunneling through publicly available IM servers. When a IM connection is established between two IPoIM clients, the IM channel is turned into an IP tunnel. Moreover, TLS is used to provide end-to-end encryption and privacy. The IP tunnel allows to route IP traffic between connected sites.

This paper is organized as follows. Section II introduces relevant concepts and motivates the work. In Section III, a conceptual architecture of the components and the complete solution is presented. The software components used are presented in Section IV. In Section V first results of experiments are introduced. Main contributions of this work are collected in Section VI and the paper concludes in Section VII.

II. Motivation

Instant Messaging has become ubiquitous and is expected to extend the reach of e-mail and make overall messaging market grow. IM plays a key role in the Interactive Collaborative Environments (ICE) markets and the focus of its development is based on commercial aspects like market penetration. A wide variety of problems exist and can include IM messages sent in clear text, no local routing, no namespace control, no auditing or logging, easy introduction of threats and more, depending on the IM protocol.

Currently, 34% of e-mail users also uses IM and penetration into organizations raised to 54% after 40% in 2002 [1]. Despite of the growth in the organizations, consumer IM clients continue to dominate the use of IM in the workplace and in the Internet.

During Q1 2005, 71.2% of the organizations did not block IM traffic. Moreover, most home computers do not block IM traffic either. The most common threat associated to IM is the loss of productivity, and organizations are mostly concerned of constraining personal contacts to work-related contacts. In some special fields, like some banking organizations, there are special regulations that force all the traffic to be recorded, for auditing purposes. As IM is *only text*, it is mostly considered to be harmless.

The other immediate key feature of the IM is it's *instantaneity*. A normal chat session requires a minimal communication delay. There is already enough delay on the fact that messages get transmitted only when the user sends it.

The broad diffusion, lack of control and instantaneity suggests that all the requirements for data transmission are already present. In the following Subsections will develop this concepts together.

A. Conceptual grounds

The security related aspects, together with the capabilities of the technology, suggested that *anything* can be transmitted. The reader can perform a simple experiment, consisting in `uuencoding`[1] an arbitrary file to obtain a *textual* representation of the file. Afterward, using cut-n-paste on the *sender* side into the IM client message window, it is possible to transmit the image. On the *receiver* side, again using cut-n-paste, it is possible to reassemble the possibly various instant messages, and `uudecode` to exact a copy of the original file. Most IM clients provide means for file-exchange, but the implementation is *outbound*, an additional connection is created, generally between both ends, for such purpose. This experiment, and IPoIM, shall be understood as *inbound*. We will explore in Section III-A further aspects of general IM architecture.

This naïve confirmation allows us to practically see that we can turn digital content into some IM-compatible representation format and transmit it as a standard IM sequence of messages.

What different types of content can be transmitted? What is this good for? The first answer we found was to develop a communication-like API wrapper that automatizes the coding/decoding process. The API (named IM sockets) would provide a Berkeley-like set of socket primitives, but has the drawback that software has to be developed to use it. The design decision was taken so as to favor fast development and software reuse both for development and testing. The simplest way of reusing existing communication software is to provide communication services at the OS level, so client applications would communicate through it. Interfacing the solution to existing software is automatically achieved. Adding an interface allows any network application to ”just” use it.

B. Design considerations

Key issues during the development of the solution were software reuse and portability. The prototype implementation shall concentrate on communication aspects more than on OS integration and IM client implementation.

Several problems are faced when connecting to public IM services. There is no up-to-date or standard definition of most publicly available IM protocols. The exception is the protocol Jabber [2]–[5] which has a standard and up-to-date definition. Services based on it (like Google Talk) can be accessed with clients based on a standard specification. Protocols like Microsoft MSN [6] and Mirabilis ICQ OSCAR [7] were standardized in their original versions, but protocol further evolution was not standardized. Implementations of standard MSN protocol do not interoperate with current on-line servers. The standard solution is to reverse-engineer the protocols, which was out of the scope of the work. To work around the lack of standards and to avoid the complexity of protocol development, the decision was taken in favor of software-reuse.

[1] A binary to ASCII converter.

Provisioning of a software network device was also addressed by software reuse. There exist several projects that provide the required network services.

We opted for Open/Free solutions that provide platform independence, clean and documented implementation.

III. Architecture

Architectural decisions were based on software-reuse and not on a from-scratch design. The solution is built as the composition of existing software components. Their combination allows the encapsulation and transmission of network traffic through IM. Before presenting the details of selected building blocks, we will present general characteristics of instant messaging and tunneling tools.

A. General IM architecture

Following description deals with instant message traffic, leaving aside services like VoIP services or file transfers. The general framework and terminology for IM is taken from [8]. Instant Messaging applications use non standard and non interoperable protocols from different vendors. Despite of that fact, commercially deployed IM services share several common characteristics. There exist a group of Instant Messaging solutions which are classified as *serverless*. They are generally deployed inside small organizations and their protocols do not scale properly.

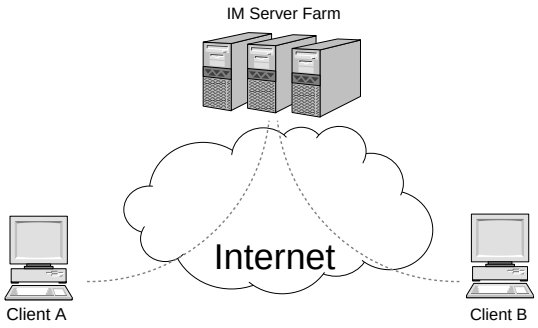

Fig. 1. General Instant Messaging Architecture

Internet wide solutions are based on servers, as represented in Figure 1. Due to load requirements, the general case is that several servers are deployed and different tasks are distributed amongst them.

When a `Client A` initiates a connection to another `Client B`, both of them connect to the IM server pool, not directly between them. Each message exchanged is relayed through the IM servers. Most protocols open a TCP connection between client and server, which is closed after some seconds of inactivity to minimize stale connections at the servers. Many protocols provide some level of encryption in the connection between the client and the server, but messages are decrypted and encrypted back while being relayed. From the client’s privacy point of view, the relaying process introduces a vulnerability in the end-to-end communication.

Also related to privacy is the fact that endpoints of the communication do not know the IP address of their counterpart. They only need to access well-known IM Servers. The user trusts the IM Service provider his originating connection address.

B. Tunnel overview

Tunneling techniques are widely deployed and currently mostly associated with security. In the general case, a *tunnel* implements a *tunneling protocol*, a network protocol which encapsulates one protocol inside another. When protocol P is encapsulated within protocol Q, P treats Q as it would be a data link layer as it provides point-to-point services. Details of Q network are isolated from P. Tunneling can be used for providing Virtual Private Network (VPN) functionality features like encryption and privacy through public networks. Different tunneling/VPN solutions exist like IPSec, GRE, IPoIP, L2TP, TLS, etc.

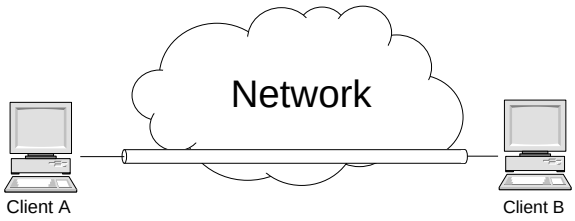

Fig. 2. General Tunneling Architecture

The general case is that both ends of the tunnel are connected using services of the network P without a central or dedicated special component. The general tunnel architecture can be graphically depicted in Figure 2. If compared to Figure 1, tunneling does not present a single convergence point of all concurrently connected users. This leads to the fact that there is no single point that connects all the tunnels. Moreover, there is no intermediate that decrypts and encrypts communication flows. The immediate privacy result is that, provided proper encryption schemes, an adequate level of privacy is achieved, without the need of trusting an intermediate service provider. As endpoints of a tunnel are directly connected, they need to know each other's IP address, with the loss of privacy generated by the fact that both endpoints know where is their counterpart.

C. IPoIM architecture

IPoIM is built merging a tunnel with an instant messaging service. Basically, the tunnel is used to perform network interface level handling and provide end-to-end privacy. The decision of providing end-to-end privacy is not a requirement for IPoIM encapsulation, but it is a powerful value added. The end-to-end privacy combined with the fact that the IM architecture abstracts the actual IP address of each endpoint combines strengths of both approaches.

The tunnel is interfaced with the IM client, so that every packet exchange is redirected through the IM server farm. IM messages are transmitted inside a TCP stream, over IP packets. An IP packet that is about to be transmitted through the tunnel is caught at the tunnel interface, encrypted and compressed by the tunnel. Afterward it is converted into an IM compatible representation and sent as an IM message inside a TCP/IP stream.

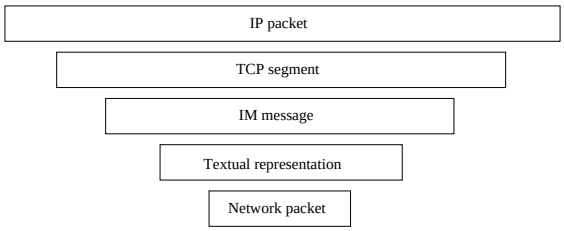

Fig. 3. Abstract traffic encapsulation representation

Figure 3 shows the successive encapsulation stages performed. It is notorious that any particular implementation of this solution would pay an important overhead price.

From the architectural point of view, the resulting solution will provide the same end-to-end services of the tunnel, but messages are interchanged through IM servers. Figure 4 shows a conceptual representation of IPoIM. The tunnel is built on top of the IM service and provides the same end-to-end features as a standard tunnel.

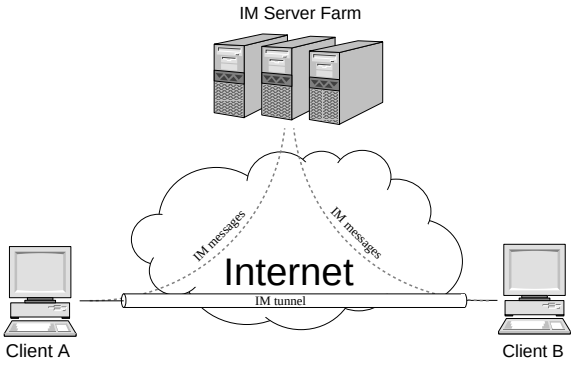

Fig. 4. Conceptual architecture of IPoIM

IV. Building Blocks

This Section introduces the basic building blocks directly used to implement our prototype. Focus is placed on the facts that guided our decisions, which proved to be accurate by means of results.

A. GAIM

Gaim [9], [10] is a multiplatform, multi-protocol instant messaging client. It is compatible with MSN Messenger [11], Yahoo! Messenger [12], AOL Instant Messenger [13] and ICQ [14], IRC [15], Jabber [16] and Google Talk [17], among others.

The Gaim project starts in 1998. It is an Open Source development licenced under the GNU General Public License

(GPL). Currently, it is in active development, and it has good documentation for end users and developers [10].

The core communication functionality, located in `libgaim`, is implemented as a software library. It can be invoked and used independently from the graphical interface, making it suitable for a system-like daemon process. Based on the API documentation, it is possible to use the core library without considering the particular details of the IM protocols.

To use this prototype in real networks it is specially important to choose a multi-platform software, like Gaim. It runs on a number of platforms, including Linux, MacOS X, and Microsoft Windows.

Considering source code and good documentation availability, Gaim is a multi-protocol and multi-platform software perfectly suitable for our needs.

B. OpenVPN

OpenVPN [18] is an Open Source project licensed under the GNU General Public License (GPL). OpenVPN is a software solution which allows networks at different locations to be securely connected (tunneled), using a public network as the communication layer. In each endpoint, OpenVPN uses TUN/TAP [19] virtual interfaces, that can be manipulated by programs in the user space. The end-to-end connection uses the industry standard SSL/TLS [20] to provide privacy and authentication. OpenVPN works as follow: after two hosts are authenticated and connected, it creates a virtual interface (TUN or TAP), that others programs can easily use like any other interface. When a IP packet arrives to the TUN/TAP virtual interface, OpenVPN encrypts, encapsulates and sends it to the remote connected machine. At the remote machine, the IP packet is decrypted, authenticated, and de-encapsulated, and finally put in its virtual TUN/TAP interface.

OpenVPN has clean source code and good documentation. It runs on different platforms, including Linux, MacOS X, and newest Microsoft Windows. It is easy to configure in these platforms, and smoothly integrate same software ports (like winpcap [21] for Windows).

Basing our solution on Gaim and OpenVPN projects, we achieve a simple and portable implementation.

V. Implementation and first results

The following two Subsections give a more specific and technical insight in how the solution was developed and present some of the preliminary results and benchmarks. The first Subsection deals with some implementation details, including required modifications and usage patterns of the building blocks. The last one specifies some of the different test case configurations deployed, together with initial measurements gathered during the execution of these tests.

A. IPoIM implementation

The core of the prototype is based on a modification of OpenVPN's main block. Behavior regarding local node processing is preserved, what is bypassed through the IM server is inter-node communication.

OpenVPN creates a virtual interface in the host that is executed. According to the assigned network address and the defined routing table in the host, packets might need to be transmitted through that interface. OpenVPN establishes a SSL/TLS connection between endpoints, which is used for all required communications. Packets *entering* the virtual interface are forwarded through the secure connection to the counterpart for further routing. IPoIM implementation replaces the direct SSL/TLS connection between OpenVPN clients with IM messages. Berkeley socket handling routines like `accept`, `connect` etc. are replaced by IM connection handling primitives.

OpenVPN presents some distinctive characteristics compared to other tunneling technologies. Particularly, as it uses SSL/TLS for transport, it utilizes services of the application layer. OpenVPN allows the configuration of SSL/TLS connections both over UDP or TCP. This enhances usability, as it is pretty simple to configure connections over firewalls.

For the implementation purposes, it is beneficial the fact that it is able to run over UDP: it already splits the traffic into a single sequence of datagrams. A simple to implement approach is to bypass UDP communication using a sequence of text messages through an IM service. This configuration permit us abstract communication issues like persistence, fragmentation, retransmission, congestion control, etc., which is already implemented in OpenVPN over UDP. Implementation delegates this work to OpenVPN implementation, which is already tested and validated by its user community.

The only constraint that has to be imposed is that IM services does not provide arbitrary length messages. Valid UDP messages range from 0 to $2^{16} - 1$ bytes in length, which does not apply for IM messages. As we do not want to implement some fragmentation protocol when converting UDP datagrams into IM messages, we opted to limit maximum UDP message size. We indirectly limit UDP size by configuring appropriately the MTU size of the interface. This parameter depends on the IM service, but all of them impose some limits to maximum message size. The set of software building blocks and their interconnection are depicted in Figure 5.

Modifications performed to OpenVPN's source code comprises three different sections of the code. The first modification concerns access to *configuration* file. IM connection descriptors are required. Configuration parameters added are:

- protocol: IM protocol name to be used in the connection.
- account: name of the IM account to be used by current endpoint.
- account-to: name of the IM counterpart for the communication link.

The rest of the necessary information for the IM communication (e.g.: account's password) is gathered from Gaim's configuration parameters through the pair account-protocol.

The second modification was introduced at the beginning of the main communication block. It involves the *initialization* and setup of Gaim module. It establishes the IM connection to

the IM server, authenticates the given account and initiates a wait loop until the communication counterpart gets connected.

The third processing block that was altered is associated with *IM message reception and transmission*. IM transmission corresponds to the reception of an IP packet through the tunnel interface that has to be sent to the other end of the tunnel. In this case the implementation must transmit the received packet through the IM connection. Additional information needs to be appended to the packet so as to mock counterparts OpenVPN and simulate message arrival. The necessary information for the other side of the communication link to understand the delivery, includes the virtual IP address of the original sender and the network packet itself.

IM Messages are of structured types referred as *information-blocks*. Information-blocks are composed by:

- from-addr: local (virtual) IP tunnel address
- data-len: length of the packet
- to-send: original packet to be sent

The new object, composed of this three items, is copied to memory. As transmission through an IM service only allows data in text format, the messages have to be converted into some textual representation. We opted for the well known and extensively used `Base64` [22]. Using `Base64` the information-block is converted into its textual representation and transmitted.

Upon reception of the information-block at destination, it is decoded and accessed in its binary format. Based on the information-block received, the OpenVPN block is feed with all the signals required to mimic a standard packet reception. Afterward, control is passed to the OpenVPN, which can continue the local processing.

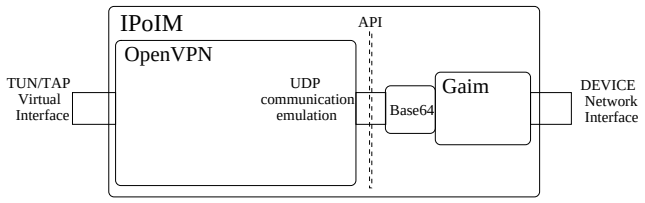

Fig. 5. General architecture of IM Software

To meet asynchronous requirements of the signals handled, the implementation was decoupled in two threads; one for each service. The first thread is in charge of attending the IM service, it has to send and receive all the messages (encapsulated packets) through the IM communication. The second one is in charge of attending the virtual communication link, and it has to send and receive the packets from/to the virtual interface. These two threads have to share the application's context, which contains all required communication parameters including tunnel and IM static information and also information about the different states (p.e.: received new IM message). Access to this structure has to be serialized to avoid inconsistencies due to concurrent access.

The methodology followed based on the interfacing of existing solutions provides a modular implementation with several benefits. First of all, it abstracts our implementation from the underlying problems (i.e. no IM communication protocol was implemented). This also allows execution over any IM protocol that Gaim implements. Additionally, it allows the individual replacement of some of the different involved blocks. This implementation could be altered by replacing, the IM block (Gaim) or the codification algorithm. For replacing the IM block the only modification needed is the `API` for the IM service. All the necessary information and functionality is encapsulated there. In this case we use the Gaim block for the service but it is quite easy and understandable to change it for another IM service implementation. The replacement of the codification algorithm is quite different because there is an abstract interface that allows the programmer to have several implementation algorithms at the same time. Each time algorithm related functions are called, the programmer must specify the algorithm.

As an implementation conclusion its important to remark that the design decisions were made based on the idea of making an initial prototype for testing and not for end-users. It was implemented for POSIX and Windows platforms, and tested on Linux and Microsoft Windows. Interfacing issues are almost identical for both implementations.

B. Preliminary test results

During development phase a stable and working prototype was developed. One of the initial concerns was regarding the efficiency of the solution. Conversion to `Base64` and further encapsulation into IM messages adds overhead. In Figure 6 we plot the overhead added to incoming traffic. Size of IM transmitted messages over MSN service is plotted over the size of original packets. Data was gathered sniffing real traffic, produced using ICMP messages with different sizes.

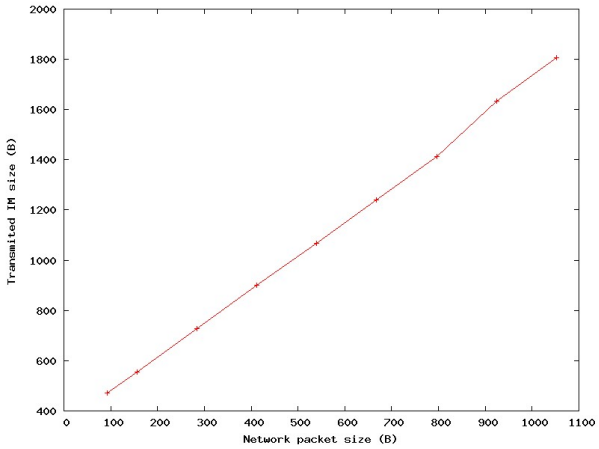

Fig. 6. IPoIM overhead

A simple linear approximation of the overhead is given by $o(x) = 1.3x + 355$, where x is the length of the IP packet to be transmitted. It is not completely simple to model the overhead function, as it is dependent of the payload. OpenVPN performs LZO compression to the traffic before

its transmission. Structure of a standard IM message adds an almost fixed-size control structure. We conclude that the burden added to small messages is very big, compared to real network protocols, reaching an overhead of about 500% for 64 byte ICMP messages.

The next logical step was to validate communication capabilities of the solution and to obtain some benchmark results. Testing tools were selected from the RFC 2398 "Some Testing Tools for TCP Implementors" [23], which suggests a group of tools for this purpose. Not all the tools presented there were used, as some of them were not ported to our target OS and others were not suitable for our purposes.

The selected tools were: NetPerf [24], TcpTrace [25] and TTcp [26]. These tools are all known, maintained and suitable for this kind of tests. Test tools generate and observe traffic between two processes running on different hosts. Figure 7 shows the network configuration used during the tests.

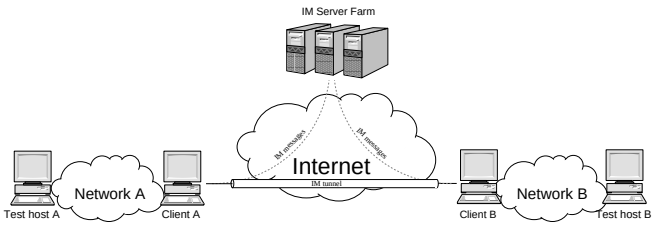

Fig. 7. Test configuration.

Even though it is possible to run selected validation tools from client hosts, we decided to separate the hosts that run the IPoIM client from those used for testing. IPoIM client run on client A and Client B hosts, turning them into routers which connect networks A and B through MSN service.

The following tests were executed:

- Test 1 - Basic Functionality: The objective of this test is to get the first performance indicators under a normal situation. Both IPoIM clients A and B uses GNU/Linux as Operating System. The IM service used was Microsoft MSN.
- Test 2 - Interoperability: IPoIM Linux Client vs Microsoft Windows Client: The main objective of this test is to validate cross-platform capabilities of the solution. The IM service used was Microsoft MSN.
- Test 3 - Local Area Network Test - Wide vs Local: The objective of this test is to evaluate prototype performance using a local IM server. The IM service used was a local Jabber server.

All previous tests were executed without any abnormal result, and gave consistent results through different tools and configurations. A remarkable fact is that results were consistent all through the different tests, time of day and geographical distribution of networks A and B. Tests executed using Microsoft's MSN service produced a consistent throughput of about 22Kbps, with mean Round Trip Time (RTT) of 550ms. The measured throughput was consistent using TCP and UDP transport protocols. In different tests, Networks A and B were physically close to each other and also, deployed in different continents. In all cases, RTT and throughput were consistent. It suggests that the bandwidth limit is imposed by the IM service provider and not by the solution.

The third test using a local IM server showed better performance results than the first two executed through the Internet. Measured bandwidth was of 138Kbps and RTT was 100ms.

This initial results validates the concept and show that it is possible to use IM for *serious* communication purposes. For our team, it is a clear confirmation that it is worth making the effort of releasing IPoIM as a product and completing evaluation of communication capabilities.

VI. Final remarks

The first relevant and remarkable fact is that it works. During the development process there always existed a shadow of a doubt about the actual possibility that there might exist some classification rules from the server side. We were not able to doubtlessly know it until we finally implemented our first version of the tool. After we managed to transfer several megabytes of information through publicly available IM services, we confirmed our guess that it is not only theoretically possible but practically usable. Bit-rate, latency and jitter were better than expected.

Even though the implementation cannot be considered more than a proof-of-concept prototype implementation, it allowed us to do several tests using this novel transport. First of all, after having the connection established, we concentrated in validating the communication channel as already presented in V-B. Despite of that, we also *played around* with the tunnel. We managed to run smoothly interactive `telnet` and `ssh` sessions with very good responsiveness. Another test performed was to browse Internet through the IPoIM tunnel by setting up a proxy on one end of the connection and configuring the browser accordingly on the other end. Browsing experience is not completely satisfactory for up-to-date standards, but can be compared to dial-up, modem based access. Another application that was tested was `ftp`, with satisfactory results.

We were not able to establish an adequate VoIP connection through the IPoIM tunnel. We tested H.323 and SIP based soft-phone clients, with TCP and UDP transports, but we did not manage to obtain a flux with the required quality for real-time audio. Ongoing work is addressing the technical issues at this point, but we believe that there is some buffering problem that delivers messages in bursts instead of keeping the original cadence. The soft-phone discards most packets for arriving out of time. We are still unsure if the buffering problem is inside our solution or from the IM server side. While performing these tests, we found that when the amount of traffic transmitted through the tunnel gets close to the bandwidth limit, the jitter grows. While this behavior is normal for low-bandwidth connections, the user perception is that it behaves worse that an equivalent connection. Also ongoing work addresses this issue to determine where the buffering, and thus, the jitter is introduced and to try to correct it as

much as possible. It is possible to attach some packet scheduler to the tunnel device, but we need to know better the traffic characteristics to be accurate and use as much as possible of the available bandwidth.

Our prototype evidenced some random stability problems. They could not be associated to traffic patterns or other usage factors so far and is being addressed currently.

An important benefit found due to the fact that we used an existing IM client instead of developing the protocol inside our solution, is that we are able to adapt easily to different protocol versions and details easily. As an example, we were able to use Microsoft's MSN service both with the direct connection to the servers and by proxying the traffic through a HTTP Proxy. From the connection establishment point of view, the only requirement is to modify configuration files and launch the client that is behind the proxy. The immediate result is that jitter grew from about 500ms to about 4000ms. This is due to the fact that messages reach the client through an HTTP-polling mechanism that is constantly checking if there are new instant messages relayed at the IM servers waiting to be transmitted. In this scenario everything that worked before still works, but interactive sessions are difficult to use and bandwidth is noticeably reduced.

Despite of previous facts, we believe that results are an adequate proof of the idea and also are able to point out a latent security issue regarding IM usage.

It is possible to imagine a user leaving his IM session open before going home. From his house, he can access internal resources of the company, provided that he NAT the traffic or alter some routing definitions. This usage can be performed without knowledge of the administrators using accepted services.

Even though there are companies that forbids IM or audit it's traffic, more than 70% of companies and home users does not perform any particular control of the traffic. There is a breach that can be used by viruses, Trojans or other sorts of malware to transmit information from an user computer without being noticed. This fact, with the additional fact that there is no way of auditing the destination of the communication, as the destination of the messages is some sort of an IM user Id. This IM user Id is not bound to a physical IP address in other place than in the IM server provider. Moreover, some IM services do not authenticate the IM user Id, making it even more difficult to trace. Tracing this kind of attack would be very difficult. Security related findings open new lines of applied research. The implementation can be a useful tool for research groups in the area.

VII. Conclusion

This work shows empirically that IM service can be used for transport any kind of information. Despite of the overhead of the encapsulation process, the presented initial results gives enough information for ascertaining that IPoIM is a novel communication means. The results of the initial tests suggests a stable throughput 22 Kbps through different tests using the Microsoft's MSN protocol as the IM service. This communication speed could be compared to a dial-up modem connexion.

For users of IM services this work exposes an existing latent risk. The possible malware usage of the IM service, can be achieved using the initially conceived safe service, that is "only text". It is important to mention that it is not an exploit, but a simple usage of the service. Considering the fact that approximately the 70% of enterprises do not control IM traffic, this work shows a security vulnerability that affect most of Internet connected user systems.

Presented results provide grounds for future research, and security testing tools. Team's ongoing work addresses the newly exposed security challenges.

References

[1] Osterman Research Inc. (2006) http://www.ostermanresearch.com/.

[2] P. Saint-Andre. (2004) Extensible Messaging and Presence Protocol (XMPP): Core. ftp://ftp.rfc-editor.org/in-notes/rfc3920.txt.

[3] ——. (2004) Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. ftp://ftp.rfc-editor.org/in-notes/rfc3921.txt.

[4] ——. (2004, Oct.) Extensible Messaging and Presence Protocol (XMPP): . ftp://ftp.rfc-editor.org/in-notes/rfc3922.txt.

[5] ——. (2004, Oct.) Extensible Messaging and Presence Protocol (XMPP): Extensible Messaging and Presence Protocol (XMPP). ftp://ftp.rfc-editor.org/in-notes/rfc3923.txt.

[6] R. Movva and W. Lai. (1999) MSN Messenger Service 1.0 Protocol. http://www.hypothetic.org/docs/msn/ietf_draft.txt.

[7] (2006, May) Oscar Protocol - Open System for Communication in Realtime. http://iserverd.khstu.ru/oscar/.

[8] M. Day and J. Rosenberg and H. Sugano. (2000) A Model for Presence and Instant Messaging. ftp://ftp.rfc-editor.org/in-notes/rfc2778.txt.

[9] (2006, May) Gaim - A multi-protocol instant messaging (IM) client. http://gaim.sourceforge.net/.

[10] S. Egan, *Open Source Messaging Application Defelopment: Building and Extending Gaim*, ser. For professionals by professionals. Apress, 2005.

[11] (2006, May) Microsoft Online Services - MSN Messenger Overview. http://messenger.msn.com/.

[12] (2006, May) Yahoo! Messenger. http://messenger.yahoo.com/.

[13] (2006, May) AIM - AIM homepage. http://www.aim.com/.

[14] (2006, May) ICQ - community, people search and messaging service! http://www.icq.com/.

[15] (2006, May) IRC.org - Home of IRC. http://www.irc.org/.

[16] (2006, May) Jabber - Open Instant Messaging and a Whole Lot More, Powered by XMPP. http://www.jabber.org/.

[17] (2006, May) Google Talk - Talk and IM with your friends for free. http://talk.google.com/.

[18] James Yonan. (2006, May) OpenVPN - An Open Source SSL VPN solution. http://openvpn.net/.

[19] Maxim Krasnyansky. (2006, May) Virtual Point-to-Point(TUN) and Ethernet(TAP) devices. http://vtun.sourceforge.net/.

[20] Eric A. Young. (2006, May) OpenSSL - The Open Source toolkit for SSL/TLS. http://openvpn.net/.

[21] Van Jacobson, Craig Leres and Steven McCanne. (2006, May) WinPcap - The Windows Packet Capture Library. http://www.winpcap.org/.

[22] S. Josefsson. (2003) The Base16, Base32, and Base64 Data Encodings. ftp://ftp.rfc-editor.org/in-notes/rfc3548.txt.

[23] S. Parker and C. Schmechel. (1998) Some Testing Tools for TCP Implementors. ftp://ftp.rfc-editor.org/in-notes/rfc2398.txt.

[24] (2006, May) NetPerf - Benchmarking Methodology Working Group (BMWG). http://www.netperf.org/netperf/NetperfPage.html.

[25] S. Ostermann. (2006, May) tcptrace - Official Homepage. http://www.tcptrace.org/.

[26] (2006, May) TTCP is a benchmarking tool for determining TCP and UDP performance between 2 systems. http://renoir.csc.ncsu.edu/ttcp/.