Undergraduate thesis in Computer Science

HoneyBuddy: a honeypot architecture for detecting IM threats in the wild

Heraklion, Crete, 2009

Petsas Athanasios

Computer Science Department

University of Crete

Supervisor: Evangelos P. Markatos

Report Outline

ABSTRACT **5**

1 INTRODUCTION **6**

2 RELATED WORK **7**

3 CATEGORIES OF STUDIED IM ATTACKS **8**

3.1 Malware Infection ..... Error! Bookmark not defined.

3.2 Harvesting Accounts 9

3.3 Issues of weak privacy settings 10

3.4 Vulnerabilities in client software 11

4 IMPLEMENTATION **11**

4.1 HoneyBuddy Architecture 12

4.1.1 The harvesting module 12

4.1.2 MSN messenger clients' handler 12

4.1.3 The inspection module 13

4.2 Contact Sources 14

5 COLLECTED DATA ANALYSIS **14**

5.1 MSN phishing sites 16

5.1.1 Phishing site recognition 16

5.1.2 URL blacklists' response 17

5.2 Malware Analysis 18

5.2.1 HoneyBuddy VS Malware databases 18

6 MYIMHONEYPOT, A DETEDTION SERVICE **20**

7 CONCLUSIONS **21**

REFERENCE: **22**

APPENDIX A **25**

Table of Figures

Figure 1. Screenshot from an MSN pshishing site.....17

Figure 2. Classification of collected URLs.....18

Figure 3. CVF of uptime of URLS per category.....19

Figure 4. Detection delay of collected samples compared to the VirusTotal database.....20

Figure 5. Cumulative distribution function of detection rate for collected samples based on VirusTotal reports 20

Abstract

Instant messaging (IM) services have become extremely popular because of their momentary activity. The vast majority of Internet users prefer to use instant messaging for their communications instead of sending emails, because it permits the exchange of real-time text messages. Due to their popularity and acceptance, attackers try to exploit such kind of services by sending malicious URLs or files to the contact lists compromised instant messaging accounts or clients. In this work we studied the behaviour of IM attacks through the design and implementation of HoneyBuddy, a honeypot like infrastructure for detecting malicious activities in IM networks. HoneyBuddy finds and adds contacts to its honeypot messengers by querying popular search engines for IM contacts or by advertising its accounts on contact finder sites. Our deployment has shown that with nearly three thousand contacts we can gather between 50 and 110 malicious URLs per day that belong to 10-15 unique domains. 21% of our collected executable samples were not gathered by other malware collection infrastructures, while 87% of the identified IM phishing domains were not recorded by popular blacklist mechanisms.

1 Introduction

Instant messaging is one of the most popular Internet activities. There are hundreds of millions of people around the world that use an instant messenger for communication, as well as millions are and the instant messages that are exchanged every day. Due to the fact that instant messaging services are large-scale deployed, as they are supported by different kinds of computer systems, from the personal computers of the usual type to more sophisticated portable devices, such as the modern mobile phones etc., attackers find them very attractive for their malicious purposes. For this reason, attackers try to exploit vulnerabilities of the IM client software (in the most popular are included Microsoft Windows Live Messenger, Yahoo! Messenger, A.O.L. and Skype), or try to delude IM users through phishing techniques. Once a user account has been compromised, attackers utilise the user's credentials (username and password) to connect to the particular IM service and continue the same procedure to the friends of the user, so as the attack propagates by targeting the victim's contacts. The attack vectors are either file transfers or instant messages that contain URLs of websites controlled by the attacker. As users tend to trust content sent from their contacts, the probability of users accepting the transfer or clicking the URL is higher than in the case of traditional phishing campaigns or malicious websites. This work focuses on the detection of such kind of attacks against IM users. Throughout the experience with the study of these types of attack, came as a result the idea of the implementation of HoneyBuddy, a honeypot mechanism for detecting such attacks. Honeypots are closely monitored decoy machines that are not used by a human operator, and their role is to attract the interest of malicious users and aggregate data of their behaviour. Based on this HoneyBuddy operates a number of IM accounts automatically and is equipped with some significant functions such as, harvesting IM contacts and requesting them to join the IM accounts' lists, accept file transfers, finding in log files URL matching patterns etc.

2 Related Work

Xie et al. propose HoneyIM [32], a system that uses decoy accounts in users' contact lists, to detect content sent by IM malware. HoneyIM can be deployed in an enterprise network and alert network administrators of malicious content, provide attack information, and perform network-wide blocking. HoneyIM has a limited view of the IM attack landscape due to its passive architecture and enterprise deployment. To overcome these disadvantages, HoneyBuddy is an active architecture that constantly adds new "buddies" to its decoy accounts, transcending the narrow confines of an enterprise level deployment, and monitors a variety of instant messaging users for signs of contamination. Furthermore, the use of pidgin [11] prevents their system from detecting attacks that exploit vulnerabilities in dominating instant messaging software such as the MSN live messenger [9]. Trivedi et al. address the problem of instant messaging spam (spim) and how to utilize honeypots to extract network and content characteristics of spim [30]. They set up an open SOCKS proxy that only allows outbound connections to IM servers. The analysis of the collected data reveals several characteristics of spim campaigns. An interesting result is that advertised URLs lead to a small number of websites, something that is confirmed by our findings. However, there are several major differences with our work. While they focus on spim campaigns, our honeypot detects all types of instant messaging threats mentioned in section 3, and also handles malicious file transfers. Furthermore, they propose a passive architecture that waits for spimmers to connect to their open proxy while our system actively broadens its view by connecting with a diverse and wide-spread set of IM users. Finally, their approach will not work with encrypted instant

messaging traffic, such as Skype traffic. Mannan et al. conduct a survey and provide an overview of threats against instant messaging users and existing security measures [27]. Several scenarios of attacks against IM users are presented, as well as the weaknesses of default security and privacy features provided by IM client software. They conclude that existing public and enterprise IM systems fail to provide sufficient security and protect users from existing IM threats. Hindocha [22] provides an overview of several IM clients and protocols, threats to instant messaging like worms and trojans, and issues regarding IM blocking. Liu et al. [25] propose an architecture, for detecting and filtering spim, that incorporates widely deployed spam-filtering techniques and new techniques specific to spim based on the analysis of spim characteristics. In follow-up publications [26, 24], the authors focus on instant messaging worms. In [26] worm propagation is modeled and traced through multicast event tree tracing, while in [24] a formal IM worm modeling based on branching process is presented. Williamson et al. [31] apply virus throttling as a mitigation measure against viruses and worms that spread through instant messaging. They explore how several throttle parameters delay propagation without interfering with normal traffic.

3 Categories of studied IM Attacks

As mentioned above, attackers get attracted by the high population of IM networks, so they try to exploit such networks to fulfil their malicious purposes, such as spreading malware, scamming and other similar issues. What follows are the most common scenarios of attacks that take place in IM networks.

3.1 Malware Infection

There are many types of malware instances [21] that their role is to be attached to a victim's instant messaging client and start spreading themselves to random contacts of user's account that use that IM client by sending executables or by sending URLs that point to malicious websites. The most common procedure that these malware pieces follows is to make login to the IM network, to send the malicious URLs or files, and afterwards to log out as expeditiously as possible. In order to appear more believable to an everyday user, the sending URLs point to domains whose name contains the username of the recipient, for example *http://contact_username.party-pics.com* .

3.2 Harvesting accounts

Another way for attackers to obtain access to IM accounts and do their malicious stuff is by using compromised credentials which can harvest either through phishing sites, through keyloggers or through social engineering. The fact that many services, such as the MSN, that use unified credentials for email and instant messaging, have made the life of attackers easier. The most common phishing sites, ask for a victim to fill a username and a password field, in order to see some photographs that want to share with her someone of her contactlist. The victim enters her IM credentials in the website and she is redirected to another domain that nothing happens,

however her account is stolen. A screenshot of a phishing site is displayed below, in Figure 1.

Figure 1: Screenshot from an MSN phishing site.

3.3 Issues of weak privacy settings

Even in the absence of malware infection or stolen credentials, s some messengers provide the option to allow incoming messages from people who are not in the user's contact list. By a look at the latest client versions of the most popular IM services: MSN live messenger, Skype, Yahoo and AIM, one can find out that MSN live messenger is the only IM client which

has a privacy setting enabled by default that blocks messages from accounts not contained in the contact list. Skype, Yahoo and AIM by default allow anyone to send instant messages to our account, but this setting can be opted-out. Attackers exploit these settings to send unsolicited messages to IM users.

3.4 Vulnerabilities in client software

IM client software suffers from the problem of monocultures. Once an exploit is discovered, then automatically millions of clients can be infected immediately [20]. While in the case of malware infection exploits take advantage of the IM client to spread, this case involves the attack where the IM client is used to infect the rest of the machine.

4. Implementation

HoneyBuddy was designed taking into consideration the four attack scenarios mentioned above. It uses the latest versions of the original clients, the same software that most users install. The main reason for this choice is that direct attacks on IM client will be detected. The main idea is to run harvested accounts to a decoy account, or to advertise this account to famous contact finder sites and to monitor any incoming message that is by default suspicious, as HoneyBuddy is a honeypot architecture. In this implementation was chosen the MSN service as it’s the most popular, but HoneyBuddy is generic enough to allow the fast implementation and in other IM services. Furthermore, MSN live messenger 2009 inter-operates with Yahoo, and is planned to introduce interoperability with Google Talk, AIM and other services, rendering our architecture

deployable to all major instant messaging services. All deployed messengers run in a fully patched Windows XP SP3 system.

4.1 HoneyBuddy Architecture

In this chapter is presented the main components of HoneyBuddy architecture, which are a harvesting module, a script-based engine that handles the MSN messenger clients and the inspection module.

4.1.1 The harvesting module

The harvesting module is responsible for gathering accounts that will later be added to the decoy accounts. All harvested accounts are inserted in CTT files (MSN contact files) that are imported in the messengers and all accounts listed are automatically invited. In the Appendix A, there is the source code of creation of such lists from the accounts that is collected.

4.1.2 MSN messenger clients' handler

The script-based engine starts the messengers and invites all contacts gathered from the harvesting module. Based on the AutoIt software [3], we can automatically start the application, import CTT files and invite other accounts to our friend list. The

AutoIT software allows the manipulation of the windows of an application the same way a user would manually click, monitor the status of the application and check for new windows (in order to check for incoming messages). When an incoming message comes and includes a request for a file transfer, the engine automatically understands that there is a transfer by matching strings in the conversation windows and accepts it. As each messenger can only have a limited number of friends in its contact list, MSN by default allows only 1000 contacts, it is preferable to run multiple messengers. For resource efficiency reasons, we used MSN Polygamy [10] in order to run multiple MSN messengers on a single platform without the need of additional virtual machines. Moreover, when a decoy account has been advertised in an account finder site, in case there are requests from other users that have visited this site, the engine accepts all these requests. Sample code of how the engine works is presented in the Appendix A.

4.1.3 The inspection module

The inspection module monitors the logs of the messengers for malicious URLs. It additionally checks the default download folder for new file transfers. An interesting finding is that there were URLs and malware in the Hotmail inboxes of our accounts. Thus, the inspection module was extended to also fetch and analyze e-mails, so as to extract URLs and executable attachments. All malicious URLs are stored in a database and are queried every one hour to check their uptime status.

4.2 Contact Sources

Were used two major sources for finding and adding contacts. The first one was queries for contact files and e-mail accounts belonging to the @hotmail.com and @live.com domains. Simple queries like “filetype:ctt msn” or “inurl:’@hotmail.com” were able to provide us thousands of contacts. Were invited 6554 contacts to become friends with the decoy accounts. 946 of those (14%) accepted the invitation.

Other potential sources are sites where users advertise their MSN account, such as `addmymsn.com`[8], orbuddyfetch.com/. The `addmymsn.com` site contains more than 25,000 active messenger contacts that are advertised by their owners for social networking purposes. Were also advertised some accounts on this site and were instructed the honeypot messengers to accept any friend request. Between two weeks 1990 contacts had been added while this number increases daily.

5 Collected data analysis

In this chapter is provided an analysis of one-month data collected by the HoneyBuddy infrastructure, from the 1st to the 31st of March 2009. During the collection period, the HoneyBuddy collected 1801 unique URLs that belong to 277 unique top-level domains, and the majority of the malicious URLs had not been detected by popular detected mechanisms.

At first it took place a simple classification of the URLs. The four main categories were phishing, porn, dating and adware (adware are characterized sites that promote third-party addons

for the MSN messenger like extra winks, emoticons etc.). As we can see in the figure 2, bellow, 617 of the URLs in 62 top-level domains were phishing that pose a security danger for IM users.

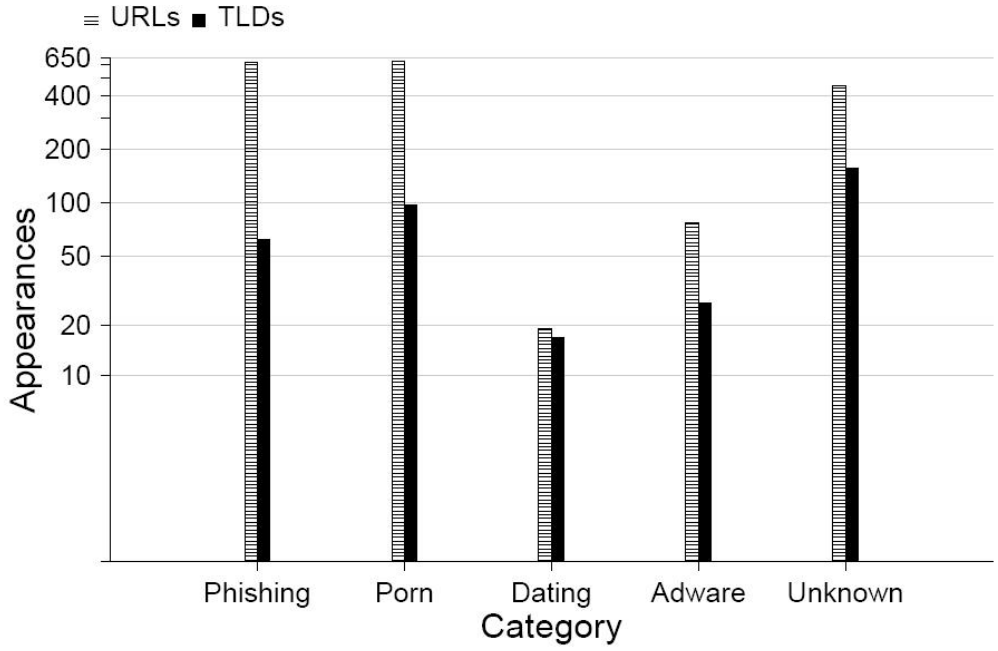

Figure 2: Classification of collected URLs

Afterwards, was done an analysis of the uptime of the collected URLs that can be seen in Figure 3. On average, a site is functional approximately for 240 hours (10 days). Was also plotted the uptime graph for each category. As can be noticed, porn and MSN phishing sites present much higher uptime than adware and unclassified sites. Half of the MSN phishing sites were alive for up to 250 hours (ten and a half days), while adware present a shorter lifetime of up to 80 hours (three and a half days).

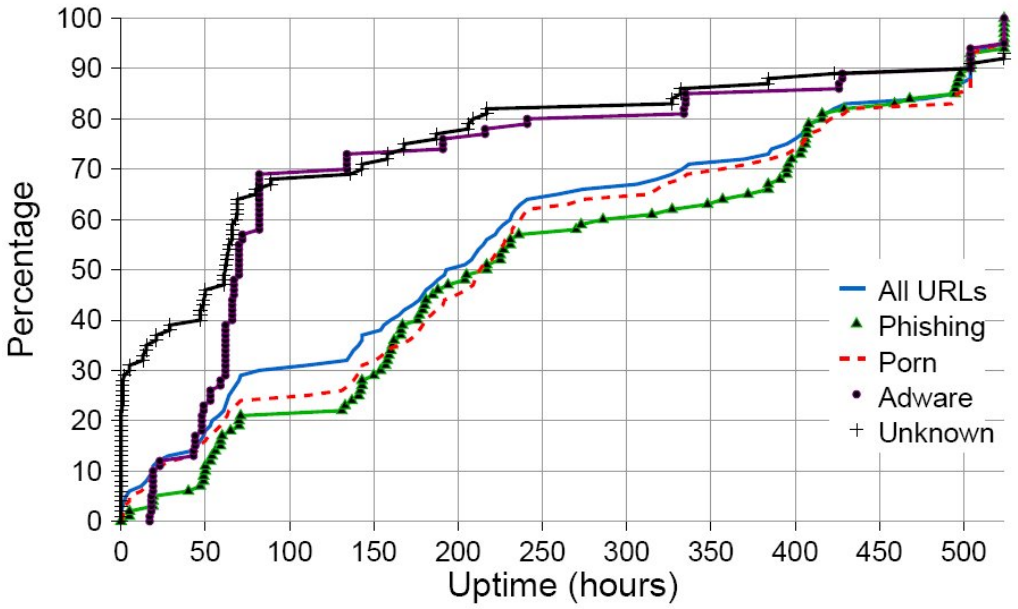

Figure 3: CDF of uptime of URLs per category

5.1 MSN Phishing sites

Attackers try to gather MSN credentials by tricking the user into entering her MSN e-mail and password in a bogus site. To validate that these phishing sites actually steal user credentials, were created several MSN accounts and were entered into the phishing sites. Each account had one of the decoy accounts as a friend. The decoy account received messages from the stolen MSN accounts that advertised the phishing site.

5.1.1 Phishing site recognition

All phishing sites that were gathered look exactly the same. A screenshot of such a site is shown in Figure 1. Was analyzed the source HTML code of all phishing sites and there was absolutely zero difference. All the phishing pages were 6K long and contained the same images and forms. Was also detected a localized phishing site which had translated content, a technique used in e-mail spam campaigns [16]. The number of syntactical and grammatical errors revealed that the text translation was done automatically. For the time being, simple pattern matching for specific text segments is efficient for detecting these sites. Another detection mechanism is to query the various URL blacklists.

5.1.2 URL blacklists' response

The Google blacklist was queried through the Google Safe Browsing API [7] to check if it included the phishing sites that were discovered. From the 62 unique top-level domains (TLD) that hosted phishing sites and were detected by HoneyBuddy, only 8 were listed by Google blacklist. That means that 87% of the domains captured by HoneyBuddy were not listed elsewhere, making HoneyBuddy an attractive solution for MSN phishing detection. The average delay from when our system detected one of the 8 sites until it was included in the Google blacklist was around two weeks, leaving a time window of 15 days for attackers to trick users. Firefox, one of the most popular browsers uses the Google Safe Browsing API as an anti phishing measure. It was done the same with the blacklist maintained by SURBL [17] and URL- blacklist.com [18]. SURBL detected only 1 out of the 62 MSN phishing domains (1.5%) and none of the adware domains. None of the phishing or adware sites were listed by URLblacklist.com. A very interesting fact is that all 62 top-level domains translate to only five different IP addresses. Also, was queried Spamhaus.org

[15] but none of the IP addresses were included. 47% of the domains translate to the first IP address, 28% to the second IP address, 18% to the third address and the rest of the domains to the other two.

5.2 Malware Analysis

HoneyBuddy infrastructure collected 19 unique malware samples either through direct file transfers (uncommon case) or by visiting URLs that were redirected to executable files. In the case of URLs, the e-mail account of the victim was always appended as a parameter to make it look more realistic. In some cases attackers used popular keywords, like Facebook. Due to the small volume of files, it was easy to manually check these files using the Anubis analysis center [1]. All of them were characterized as dangerous, while some of them were bots that connected to an IRC C&C server. By joining the IRC network, we downloaded even more malware samples (not listed in this section).

5.2.1 HoneyBuddy VS Malware databases

In order to verify how original these samples are, were submitted to the VirusTotal [19] service. VirusTotal is a large malware collection center with the primary goal of providing a free online virus and malware scan report for uploaded samples. Every day VirusTotal receives around 100,000 samples. Four collected samples had not been seen by VirusTotal before, that is 21% of the samples were zero-day malware instances. Figure

4 shows the relative detection delay compared to the date the samples entered the VirusTotal database. The base bar of the stack graph (solid white) shows how many samples were detected with a delay of one or more days, the middle bar (solid black) displays the number of samples that were detected the same day as VirusTotal while the top bar shows the number of samples not included in the VirusTotal database. Five samples (26%) were collected the same day they entered the VirusTotal database, while the maximum detection delay was five days. We also checked the VirusTotal analysis reports for the collected samples. 42% of the samples were detected by half of the anti-virus engines, while the maximum detection rate was 77%. However, the dates of the analysis reports were one month after the collection date as we did not submit the samples the day they were captured. The one month delay means higher detection rates for the anti-virus engines. Even in that case, it can be observed that there are samples that are recognized only by one third of the anti-virus products. The cumulative distribution function of detection rates can be seen in Figure 5.

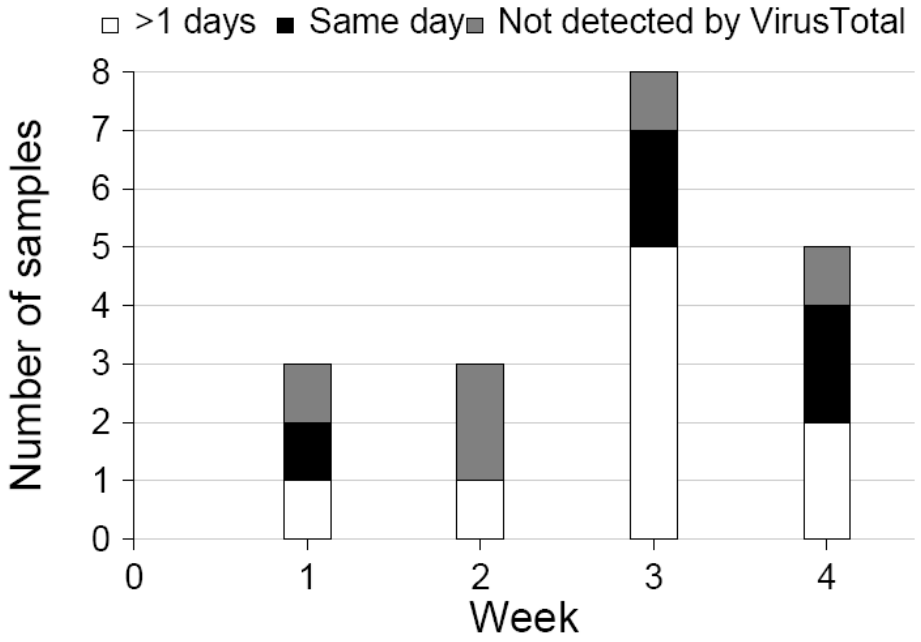

Figure 4: Detection delay of collected samples compared to the VirusTotal database.

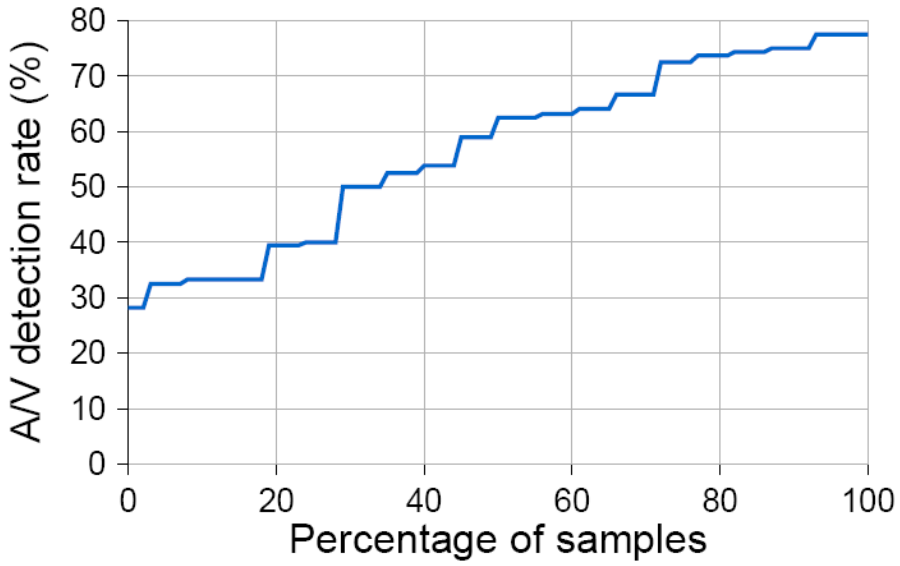

Figure 5: Cumulative distribution function of detection rate for collected samples based on VirusTotal reports.

6. MyIMhoneypot, a detection service

From all the study that was described, came out the idea of setting up an early detection service that can inform users if their accounts or IM clients have been compromised. MyIMhoneypot is orthogonal to existing defence mechanisms and works as follows. Any user that wants to check if her account is compromised registers with the myIMhoneypot service. Upon registration, the service creates a unique IMhoneypot account (for example, a new MSN account that will be used as a decoy account) and informs the user to add that honeypot account to her contact list. As the user will never start a conversation with the honeypot account but an IM attacker will, the user can check if something is wrong by visiting the website of the service and checking the conversation logs with her unique honeypot account. If there are entries in the conversation log of her decoy account, then there is a strong

indication that her IM client or credentials have been compromised. There is an implementation of MyIMhoneypot for the MSN platform, named myMSNhoneypot and can be found at `http://mymsnhoneypot.dyndns.org`.

7. Conclusions

In this work is being proposed HoneyBuddy, an active honeypot infrastructure designed for detecting malicious IM activities in the wild. HoneyBuddy automatically finds user accounts for incoming messages and file transfers, and extracts suspicious executables and URLs. The suspicious data gathered by HoneyBuddy is correlated with existing blacklists, and malware collection center databases. Despite the simplicity of our system, deployment for the MSN service showed that 87% of the identified phishing domains were not listed by popular blacklist mechanisms. Furthermore, 21% of collected malware samples were also not listed by other infrastructures. These findings confirm that existing security measures of instant messaging services are insufficient, and also indicate the effectiveness of our system as a complementary detection infrastructure.

It was also deployed myMSNhoneypot, a prototype implementation of a service that is open to the public and creates dedicated IM honeypots for users. This service provides an early alerting mechanism for users whose IM accounts or clients are compromised. It provides decoy accounts for users that register with the service to add to their contact list. A message from the user to a decoy account is an indication that the user’s credentials or IM clients are compromised, as the user would never initiate a conversation with the decoy contact.

References

[1] Anubis: Analyzing unknown binaries. http://anubis.iseclab.org/.

[2] AQABA Search Engine Demographics. http://http://www.aqaba-sem.com/search ed.htm/.

[3] AutoIt. http://www.autoitscript.com/autoit3/index.shtml/.

[4] BuddyFetch. http://buddyfetch.com/.

[5] CAPTCHA: Telling Humans and Computers Apart Automatically. https://captcha.net/.

[6] Europe surpasses north america in instant messenger users, comscore study reveals. http://www.comscore.com/press/release.asp?press=800.

[7] Google safe browsing api. http://code.google.com/apis/safebrowsing/.

[8] MSN Contacts Finder. http://addmymsn.com/.

[9] Msn messenger. http://messenger.live.com/.

[10] MSN Polygamy. http://www.softpedia.com/get/Internet/Chat/Instant-Messaging/MSN-Messenger-7-8-Polygamy.shtml/.

[11] Pidgin, the universal chat client. http://www.pidgin.im/.

[12] Planetlab, an open platform for developing, deploying and accessing planetary-scale services. http://www.planet-lab.org.

[13] Scraping facebook email addresses. http://kudanai.blogspot.com/2008/10/

scraping-facebook-email-addresses.html.

[14] Skype Fast Facts, Q4 2008. http://ebayinkblog.com/wp-content/uploads/2009/01/skype-fast-facts-q4-08.pdf.

[15] The spamhaus project. http://www.spamhaus.org/.

[16] The state of spam a monthly report august 2007. http://www.symantec.com/avcenter/reference/Symantec Spam Report - August 2007.pdf.

[17] Surbl. http://www.surbl.org.

[18] Urlblacklist.com. http://www.urlblacklist.com/.

[19] Virustotal, online virus and malware scan. http://www.virustotal.com/.

[20] Vulnerability in PNG Processing Could Allow Remote Code Execution. http://www.microsoft.com/technet/security/bulletin/MS05-009.mspx.

[21] W32.Bropia. http://www.symantec.com/security response/writeup.jsp?docid=2005-012013-2855-99&tabid=2.

[22] HINDOCHA, N. Threats to instant messaging. *Symantec Security Response* (2003).

[23] LESKOVEC, J., AND HORVITZ, E. Planetary-Scale Views on a Large Instant-Messaging Network. In *Proceedings ofWWW2008* (April 2008).

[24] LIU, Z., AND LEE, D. Coping with instant messaging worms - statistical modeling and analysis. pp. 194–199.

[25] LIU, Z., LIN, W., LI, N., AND LEE, D. Detecting and filtering instant messaging spam - a global and personalized approach. pp. 19–24.

[26] LIU, Z., SHU, G., LI, N., AND LEE, D. Defending against instant messaging worms. In *In Proceedings of IEEE GLOBECOM*

2006 (2006), pp. 1–6.
[27] MANNAN, M., AND VAN OORSCHOT, P. Secure public instant messaging: A survey. In *Proceedings of the 2nd Annual Conference*

on Privacy, Security and Trust (PST04), pp. 69–77.
[28] PORTOKALIDIS, G., SLOWINSKA, A., AND BOS, H. Argos: an Emulator for Fingerprinting Zero-Day Attacks. In *Proceedings of ACM SIGOPS Eurosys 2006* (April 2006).

[29] PROJECT, H. Know your enemy: Learning about Security Threats. *Pearson Education, Inc.* (2004).

[30] TRIVEDI, A., JUDGE, P., AND KRASSER, S. Analyzing network and content characteristics of spim using honeypots. In *Proceedings of the 3rd USENIX SRUTI* (2007).

[31] WILLIAMSON, M., PARRY, A., AND BYDE, A. Virus throttling for instant messaging. In *Virus Bulletin Conference* (2004), pp. 38–4.

[32] XIE, M., WU, Z., AND WANG, H. HoneyIM: Fast detection and suppression of instant messaging malware in enterprise-like networks. In *Proceedings of the 2007 Annual Computer Security Applications Conference (ACSAC07).*

Appendix A

What follows are a small piece of HoneyBuddy architecture:

Harvesting module:

Code for searching in Google ctt files and download them, to be imported later to the decoy accounts (Python):

```
#!/usr/bin/python

from Google import Google, search
import os, sys, threading

def download(url):
        """
        Copy the contents of a file from a given URL
        to a local file.
        """
        import urllib
        webFile = urllib.urlopen(url)
        localFile = open(url.split('/')[-1], 'w')
        localFile.write(webFile.read())
        webFile.close()
        localFile.close()

"""
create a file named "search results.txt" to save
some information there about the search procedure
"""
#del sys
#sys.stdout = open("out.txt", "w")
class DownloadUrls(threading.Thread):
   def __init__(self, result):
      threading.Thread.__init__(self)
      self.result = result
   def run(self):
        try:
           print "downloading url: %s ...\n" % (result.url())

           #sys.stdout = open("out.txt", "a")
           download(result.url())
        except IOError:
           print "The server is taking too long to respond. Connection Timeout!"


"""
search in google for 'filetype:ctt "msn""
122: is the number of results that google gives
"""
results = search('filetype:ctt "msn"', 60)
```

```
total_urls=0


"""
create a directory with name setted in folder name,
and change the current directory to the created one.
(Checks if already exists in the begining)
"""
foldername = "msn_ctt_files"

folder_exists = 0

listdirs= os.listdir("./")

for directory in listdirs:
  if directory == foldername:
    folder_exists = 1

if folder_exists == 0:
  folder = os.mkdir(foldername)

os.chdir("./"+foldername)


for result in results:
        #create one thread for each url download...
        background = DownloadUrls(result)
        background.start()

        background.join()    # Wait for the background task to finish

        total_urls += 1

print "Total number of urls: %d" % (total_urls, )
```

Code that converts a file that contains mails, to ctt format to be imported to a decoy account (Python):

```
#!/usr/bin/python

import sys, string, os

''' Converts a file that contains e-mails to ctt format '''

''' Return True if file already exists, else False'''
def fileExists(f):
    try:
        file = open(f)
    except IOError:
        exists = False
    else:
        exists = True
    return exists

# --- The Main Program ---
```

```
if sys.argv[1] == "":
  print "\n\t Usage: file2ctt.py <emails_file>"
  sys.exit()

mails_file = open(sys.argv[1], 'r')

# if file exists, erase the old file..
if fileExists("accounts.ctt") == True :
  os.system("rm accounts.ctt")

cttfile = open("accounts.ctt",'w')

cttfile.write("<?xml version=\"1.0\"?>\n")
cttfile.write("<messenger>\n")
cttfile.write("  <service name=\".NET Messenger Service\">\n")
cttfile.write("    <contactlist>\n")


for line in mails_file:
  print line.rstrip('\n')
  cttfile.write("      <contact>"+line.rstrip('\n')+"</contact>\n")

cttfile.write("    </contactlist>\n")
cttfile.write("  </service>\n")
cttfile.write("</messenger>\n")
```

Inspection module:

Code of fetching the decoy account mailboxes and searching them for URLs in the mailboxes (Python):

```
#!/usr/bin/python

import poplib
import os
from email.Parser import Parser
import string
import re
import errno
import datetime as dt
import MySQLdb

#convert a string month to its an integer representation
def month2number(m):
  month_no = {
    'Jan': '01',
    'Feb': '02',
    'Mar': '03',
    'Apr': '04',
    'May': '05',
    'Jun': '06',
    'Aug': '08',
    'Sep': '09',
    'Oct': '10',
```

```
   'Nov': '11',
   'Dec': '12',
   }[m]
  return month_no

#convert a date from format'4 Mar 2009' to '2009-3-4' (datetime for sql)
#s should be 11 chars long.if day is < 10, then s[10] should be ' '
def to_dateTime(s):
  if s[10] == ' ':
    return '%s-%s-0%s'%(s[6:10], month2number(s[2:5]), s[0])
  else:
    return '%s-%s-%s'%(s[7:11], month2number(s[3:6]), s[0:2])

#split an sql dateTime to python time
def datetime_slice(s):
  return int(s[5:7]), int(s[8:10]), int(s[0:4])

def hotmail_parser (username, password, url_list):
  print "* examine mailbox of %s"%username
  M = poplib.POP3_SSL('pop3.live.com')
  M.user(username)
  M.pass_(password)

  p=Parser()

  try:
            os.mkdir("attachments")
  except OSError, e:
            # Ignore directory exists error
            if e.errno <> errno.EEXIST:
                        raise

  urlfinders = [
     re.compile("([0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}\\.[0-
9]{1,3}|(((news|telnet|nttp|file|http|ftp|https)://)|(www|ftp)[-A-Za-z0-9]*\\.)[-A-Za-z0-9\\.]+)(:[0-
9]*)?/[-A-Za-z0-9_\\$\\.\\+\\!\\*\\(\\),;:@&=\\?/~\\#\\%]*[^]'\\.}>\\),\\\"]"),
            re.compile("([0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}\\.[0-
9]{1,3}|(((news|telnet|nttp|file|http|ftp|https)://)|(www|ftp)[-A-Za-z0-9]*\\.)[-A-Za-z0-9\\.]+)(:[0-
9]*)?"),
            re.compile("'\\<((mailto:)|)[-A-Za-z0-9\\.]+@[-A-Za-z0-9\\.]+")
  ]


  numMessages = len(M.list()[1])
  for i in range(numMessages):

            #get the whole e-mail into a single string
            fullmsg = string.join(M.retr(i+1)[1],'\n')

            #parse the string to construct a Message
            msg = p.parsestr(fullmsg)

            #we get the Received header as a timestamp
            datereceived = ""
            if msg.__contains__('Received'):
                        datereceived = msg.__getitem__('Received')

            ar = datereceived.split(';')
            try:
```

```
            datereceived = ar[1]
except:
            #print "Malformed Received header"
            continue

datereceived = datereceived.strip()

#iterate through all parts of the email
for part in msg.walk():
            # multipart/* are just containers
            if part.get_content_maintype() == 'multipart':
                        continue

            filename = part.get_filename()
            #if we have an attachment , dump it to the folder
            if filename:
                        fp = open(os.path.join("attachments", filename), 'wb')
                        fp.write(part.get_payload(decode=True))
                        fp.close()
            else:
                        #text parts must be searched for URLs
                        body = part.get_payload(decode=True)
                        for matcher in urlfinders:
                                    m = matcher.search(body)
                                    if m is not None:
                                                #conver date to sql timestamp
                                                url_date = to_dateTime(datereceived[5:16])
                                                url = m.group(0).strip()
                                                #create a dictionary with url details
                                                url_dict = {}
                                                url_dict["url"] = url
                                                url_dict["date"] = url_date
                                                if len(url_list) == 0:
                                                  url_list.append(url_dict)
                                                else:
                                                            url_found = False
                                                      for i in range(len(url_list)):
                                                             if url_list[i]["url"] == url:
                                                               url_found = True;
                                                               break
                                                            if url_found == True:
                                                              dt1 = url_list[i]["date"]
                                                              dt2 = url_date
                                                              m1, d1, y1 = datetime_slice(dt1)
                                                              m2, d2, y2 = datetime_slice(dt2)
                                                              date1 = dt.date(y1, m1, d1)
                                                              date2 = dt.date(y2, m2, d2)
                                                              dateDiff = date2 - date1


                                                              if dateDiff.days < 0:
                                                                url_list[i]["date"] = url_date

                                                            else:
                                                               url_list.append(url_dict)

M.quit()
```

```
#list with dictionaries of unique url domains and their timestamps
urls=[]

print "* Start to examine the mailboxes"
hotmail_parser("johanna_1990@windowslive.com","Password!.",urls)
hotmail_parser("gina_sweeet@windowslive.com","Password!.",urls)
hotmail_parser("natallia_88@live.com","Password!.",urls)
hotmail_parser("claudia__87@live.com","Password!.",urls)
hotmail_parser("lucia__86@live.com","Password!.",urls)
hotmail_parser("milena86@windowslive.com","Password!.",urls)
hotmail_parser("emma88@windowslive.com","Password!.",urls)
hotmail_parser("emily86@windowslive.com","Password!.",urls)
hotmail_parser("sofia__89@windowslive.com","Password!.",urls)
#hotmail_parser("maria__88@windowslive.com","Password!.",urls) #other pass!!!
hotmail_parser("christina___89@windowslive.com","Password!.",urls)
hotmail_parser("ada_1990@windowslive.com","Password!.",urls)
#hotmail_parser("helen__88@windowslive.com","Password!.",urls)
hotmail_parser("miss_anna_90@live.com","Password!.",urls)
hotmail_parser("christinaa_1990@windowslive.com","Christ!Ina.",urls)
hotmail_parser("miss_honey@windowslive.com","Miss!Honey.",urls)
hotmail_parser("mr_honey@windowslive.com","Mr!Honey.",urls)
hotmail_parser("mariaa_1990@windowslive.com","Mar!Ia.",urls)

url_sum=0
urls_added=0

 # connect to the MySQL server
try:
  print "\n* Connecting to database..."
  conn = MySQLdb.connect (host = "localhost",
                          user = "root",
                          passwd = "mv2ghasz",
                          db = "im_honeypots")
except MySQLdb.Error, e:
  print "Error %d: %s" % (e.args[0], e.args[1])
  sys.exit (1)

#try:
cursor = conn.cursor ()

for i in range(len(urls)):
  #see if this url already exists in the database
  cursor.execute("select * from mail_url where url='%s'"%urls[i]["url"])
  row = cursor.fetchone ()
  if row == None :
    cursor.execute ("""
            INSERT INTO mail_url (url, timestamp, category, source)
            VALUES
              ('%s', '%s', null, '%s')
     """ % (urls[i]["url"],urls[i]["date"], "email"))
    url_sum+=1
    urls_added+=1

  else:
    url_sum+=1

#except MySQLdb.Error, e:
#    print "Error %d: %s" % (e.args[0], e.args[1])
```

```
cursor.close ()
print "\n---> total URLs:%d\n"%url_sum
print "---> URLs added:%d\n"%urls_added
for i in range(len(urls)):
  print urls[i]["url"]+""+urls[i]["date"]
```

Code of msn handler:

Code for parsing the log files of the decoy accounts, extracting the URLs and store them to a database (Python):

```
#!/usr/bin/python

import xml.dom.minidom

import sys, string, os

import subprocess

import re

import datetime as dt

import MySQLdb

'''
  different types of date:
      m/d/yyyy   len=8
      m/dd/yyyy  len=9
      mm/d/yyyy  len=9
      mm/dd/yyyy len=10
'''
def date_slice(s):
  if len(s) == 8:
    return int(s[0:1]), int(s[2:3]), int(s[6:8])
  elif len(s) == 10:
    return int(s[0:2]), int(s[3:5]), int(s[8:10])
  elif len(s) == 9:
    if s[1] == '/':
      return int(s[0:1]), int(s[2:4]), int(s[7:9])
    else:
      return int(s[0:2]), int(s[3:4]), int(s[7:9])

def slice_dash(s):
  if len(s) == 8:
    return int(s[5:6]), int(s[7:8]), int(s[2:4])
  elif len(s) == 10:
    return int(s[5:7]), int(s[8:10]), int(s[2:4])
  elif len(s) == 9:
    if s[6] == '-':
      return int(s[5:6]), int(s[7:9]), int(s[2:4])
    else:
      return int(s[5:7]), int(s[8:9]), int(s[2:4])

#def datetime_slice(s):
#  return int(s[6]), int(s[9]), int(s[3])
#def datetime_slice(s):
#  return int(s[5:7]), int(s[8:10]), int(s[0:4])
```

```
# Pattern for fully-qualified URLs:
url_pattern = re.compile(  r'"
            (?x)( # verbose identify URLs within text
    (http|ftp|gopher|www.) # make sure we find a resource type
             (://){0,} # ...needs to be followed by colon-slash-slash
      (\w+[:.]?){2,} # at least two domain groups, e.g. (gnosis.)(cx)
              (/?| # could be just the domain name (maybe w/ slash)
          [^ \n\r"]+ # or stuff then space, newline, tab, quote
             [\w/]) # resource name ends in alphanumeric or slash
    (?=[\s\.,>)'"\]]) # assert: followed by white or clause ending
                ) # end of match group
                  '")


l = []

def getText(nodelist):
   rc = ""
   for node in nodelist:
      if node.nodeType == node.TEXT_NODE:
         rc = rc + node.data
   return rc

def handleLog(log):
   messages = log.getElementsByTagName('Message')
   handleMessages(messages)

def handleMessages(messages):
   for message in messages:
           bitmess = message
           date = bitmess.attributes["DateTime"]
           #print date.value
      handleMessage(message, date)

def handleMessage(message, date):
   handleMessageText(message.getElementsByTagName("Text")[0], date)

def handleMessageText(text, date):
   all = url_pattern.findall("%s\n" % getText(text.childNodes).encode("utf-8"))
   if all == []:
           return
   for i in all:
    d = {}

    #if url begins with 'w' ("www"... ) add in the begining "http://"
    if i[0][0] == 'w':
           url = "http://"+i[0]
    else:
           url = i[0]

    d["url"] = url
    datetime = date.value
    this_date = datetime[0:10] #keep only the date
    md, dd, yd = datetime_slice(this_date)
    d["date"] = this_date
    if len(l) == 0:
           l.append(d)
```

```
            return

        url_found = False
        for j in range(len(l)):
            if l[j]["url"] == url:
              url_found = True
              break
        if url_found == True:
            dt1 = l[j]["date"]
            dt2 = this_date
            m1, d1, y1 = datetime_slice(dt1)
            m2, d2, y2 = datetime_slice(dt2)
            date1 = dt.date(y1, m1, d1)
            date2 = dt.date(y2, m2, d2)
            dateDiff = date2 - date1


            if dateDiff.days < 0:
              l[j]["date"] = this_date

        else:
              l.append(d)



def fileExists(f):
    try:
       file = open(f)
    except IOError:
       exists = 0
    else:
       exists = 1
    return exists


# --- The Main Program ---
logspath = "/home/petsas/project/msn logs/"

dirList=os.listdir(logspath)

print "* Start to examine msn log files"

for fname in dirList:
  print "* examine msn logs for %s" % fname

  scriptpath = os.getcwd()+"/"
  os.chdir(logspath+fname+"/History/")

  path="./"
  total_files = 0
  dirList=os.listdir(path)
  for fname1 in dirList:
     if os.path.splitext(fname1)[-1] == ".xml":

            logfile = open(fname1,"r")

            dom = xml.dom.minidom.parse(logfile)

            handleLog(dom)
```

```
            logfile.close()
            total_files = total_files + 1

print "\n---> total log files examined:%d\n"%total_files

url_sum=0
urls_added=0
res = open("res","w")
 # connect to the MySQL server

try:
  print "\n* Connecting to database..."
  conn = MySQLdb.connect (host = "localhost",
                          user = "root",
                          passwd = "mv2ghasz",
                          db = "im_honeypots")
except MySQLdb.Error, e:
  print "Error %d: %s" % (e.args[0], e.args[1])
  sys.exit (1)



for i in range(len(l)):
            res.write("%s %s\n" % (l[i]["url"], l[i]["date"]) )
#try:
cursor = conn.cursor ()



for i in range(len(l)):
            #res.write("%s %url_sus\n" % (l[i]["url"], l[i]["date"]) )
  cursor.execute("select * log_url from url where url='%s'"%l[i]["url"])
  row = cursor.fetchone ()
  if row == None :
    #print "None\n"
    cursor.execute ("""
            INSERT INTO log_url (url, timestamp, category)
            VALUES
              ('%s', '%s', null)
     """ % (l[i]["url"],l[i]["date"]))
    url_sum+=1
    urls_added+=1

  else:
    url_sum+=1
    #print cursor.rowcount

#except MySQLdb.Error, e:
#    print "Error %d: %s" % (e.args[0], e.args[1])
#res.close()
cursor.close ()
print "\n---> total URLs:%d\n"%url_sum
print "---> URLs added:%d\n"%urls_added
```

MSN Handler:

Accept files and contacts procedure (AutoIt):

```
    ;Check if file with contacts exists
    If FileExists($file) Then

        $contacts = FileOpen($file, 0)

        ; Check if file opened for reading OK
        If $contacts = -1 Then
            MsgBox(0, "Error", "Unable to open contacts.txt file.")
            Exit
        EndIf

        $i = 0

        ; Read in lines of text until the EOF is reached
        While 1
            ;Or $i = 500 to add maximum 500 contacts
            $line = FileReadLine($contacts)
            If @error = -1 Or $i = 500 Then ExitLoop

            add_contact($line)


            $i += 1
        WEnd

        FileClose($contacts)


    EndIf

EndFunc   ;==>add_contacts_from

Func IsVisible($handle)
    If BitAND(WinGetState($handle), 2) Then
        Return 1
    Else
        Return 0
    EndIf

EndFunc   ;==>IsVisible
```

```
;accept all files that send contacts
Func Accept_files()
    $var = WinList()
    For $i = 1 To $var[0][0]
        ; Only display visble windows that have a title

        $livewin = StringInStr($var[$i][0], "Live Messenger", 0)
        $location = StringInStr($var[$i][0], "@", 0)

        If $location > 0 And IsVisible($var[$i][1]) Then

            WinActivate("[TITLE:" & $var[$i][0] & "; CLASS:IMWindowClass]")
            ;WinWaitActive("[TITLE:" & $var[$i][0] & "; CLASS:IMWindowClass]")
            SendKeepActive("[TITLE:" & $var[$i][0] & "; CLASS:IMWindowClass]")
            ;go to text field where are the send messages
            Send("+{TAB}")
            ;select all the text of send messages
            Send("^a")
            ;copy it to clipboard
            Send("^c")
            $clipbrd = ClipGet()
            ;MsgBox(0, "Clipboard contains:", $clipbrd)

            ;Doing some procedures, like closing a video call,
            ;by detect spesific patterns in conversation
            $result = StringInStr($clipbrd, "Alt+C")
            $isCall = StringInStr($clipbrd, "is calling you.")
            $isInvitation = StringInStr($clipbrd, "is inviting you to start")
            $isWebcamCall = StringInStr($clipbrd, "wants to start a Video Call with you")

            ;to take off Alt+A event...
            Send("{RIGHT}")

            Send("{TAB}")

            If $result > 0 Then

                If $isWebcamCall > 0 Or $isCall > 0 Or $isInvitation > 0 Then
                    Send("!D")
                    WinKill("[TITLE:" & $var[$i][0] & "; CLASS:IMWindowClass]")
                Else
                    Send("!C")
                EndIf

                ;Avoid Warning Window
                WinActivate("[CLASS:DUIDialog]")
                SendKeepActive("[CLASS:DUIDialog]")

                Send("{ENTER}")
            Else
                WinClose("[TITLE:" & $var[$i][0] & "; CLASS:IMWindowClass]")

                ;if there is a file transfer ==> don't close the conversation
                ;until the tranfer is completed
                If WinExists("[CLASS:#32770]") Then
                    WinActivate("[CLASS:#32770]")
                    SendKeepActive("[CLASS:#32770]")
                    Send("{ENTER}")
                EndIf

            EndIf



        ; accept contact
        ElseIf $livewin > 0 Then ;without isVisible
            If WinExists("[CLASS:Contact Add Invite]") Then
                WinSetState("[TITLE:" & $var[$i][0] & "; CLASS:Contact Add Invite]", "", @SW_SHOW)
                WinSetState("[TITLE:" & $var[$i][0] & "; CLASS:Contact Add Invite]", "", @SW_ENABLE)
                WinActivate("[TITLE:" & $var[$i][0] & "; CLASS:Contact Add Invite]")
                SendKeepActive("[TITLE:" & $var[$i][0] & "; CLASS:Contact Add Invite]")


                Send("{ENTER}")
                Sleep(250)

                WinSetState("[TITLE:" & $var[$i][0] & "; CLASS:Contact Add Invite]", "", @SW_SHOW)
                WinSetState("[TITLE:" & $var[$i][0] & "; CLASS:Contact Add Invite]", "", @SW_ENABLE)
                WinActivate("[TITLE:" & $var[$i][0] & "; CLASS:Contact Add Invite]")
                SendKeepActive("[TITLE:" & $var[$i][0] & "; CLASS:Contact Add Invite]")
```

```
                Send("{TAB}{ENTER}")
            EndIf
        EndIf

    Next
EndFunc   ;==>Accept_files_and_contacts



Func ClosePopUpFromTransfer()
    ;if this there is a file transfer ==> don't close the conversation
    If WinExists("[TITLE:Windows Live Messenger; CLASS:#32770]") Then
        WinActivate("[TITLE:Windows Live Messenger; CLASS:#32770]")
        SendKeepActive("[TITLE:Windows Live Messenger; CLASS:#32770]")

        ControlClick("[TITLE:Windows Live Messenger; CLASS:#32770]", "", "[CLASS:Button; TEXT:&No; INSTANCE:2]")
    EndIf
EndFunc


Func IsVisible($handle)
    If BitAND(WinGetState($handle), 2) Then
        Return 1
    Else
        Return 0
    EndIf

EndFunc   ;==>IsVisible

Func IsMsnActive()

    If ProcessExists("msnmsgr.exe") Then
        return 1
    EndIf

    return 0

EndFunc
```

add contact from a file procedure (AutoIt):

```
1
2   ; Prevent Duplicates From Running
3   If ProcessExists("msnmsgr.exe") Then
4       Dim $iMsgBoxAnswer
5       $iMsgBoxAnswer = MsgBox(4, "Attention", "Messenger is already running. Do you want to close it due to continue the script?")
6       Select
7           Case $iMsgBoxAnswer = 6 ;YES
8               MsgBox(0, "Info", "Restarting MSN...", 1)
9               ProcessClose("msnmsgr.exe")
10              ;RunMSN()
11
12
13          Case $iMsgBoxAnswer = 7 ;NO
14              MsgBox(0, "Info", "Script is being terminated...", 1)
15              ProcessClose("AutoIt3.exe")
16
17      EndSelect
18  EndIf
19
20  ; Run windows Live Messenger
21  ;Run(@ProgramFilesDir & "\Windows Live\Messenger\msnmsgr.exe")
22
23  ;Wait for the MSN become active - it is titled "Windows Live Messenger" on English-language systems
24  ;WinWaitActive("Windows Live Messenger")
25
26  If FileExists(@ScriptDir & "/user_details.txt") Then
27
28      $usr_det = FileOpen(@ScriptDir & "/user_details.txt", 0)
29
30      ; Check if file opened for reading OK
31      If $usr_det = -1 Then
32          MsgBox(0, "Error", "Unable to open user_details.txt file.")
33          Exit
34      EndIf
35
36      ; Read in lines of text until the EOF is reached
37
38      $username = FileReadLine($usr_det)
39      $password = FileReadLine($usr_det)
40      ;MsgBox(0, "Info", $password)
```

```
41      FileClose($usr_det)
42
43  Else
44      MsgBox(0, "Error", "Unable to open user_details.txt file.")
45  EndIf
46
47  ; Run windows Live Messenger
48  Run(@ProgramFilesDir & "\Windows Live\Messenger\msnmsgr.exe")
49
50  ; Wait for the MSN become active - it is titled "Windows Live Messenger" on English systems
51  WinWaitActive("Windows Live Messenger")
52
53
54
55  ;Fill up the username and password fields
56  ;send("<username>@hotmail.com")
57  Send($username)
58  Send("{TAB}")
59  Sleep(1000)
60  ;send("<password>")
61  Send($password)
62  Send("{TAB}")
63  Sleep(1000)
64  ;choose "online state"
65  Send("{ENTER}")
66  Sleep(1000)
67  Send("{DOWN}1")
68  Sleep(1000)
69  Send("{ENTER}")
70
71  Sleep(1000)
72  ;shift+TAB to go to pass field
73  Send("+{TAB}")
74  Sleep(1000)
75  Send("{ENTER}")
76
77  ;wait to login
78  Sleep(25000)
79
80  If WinExists("Today") Then
```

```
81      WinClose("Today")
82  EndIf
83
84  ;file with contacts to add named as "contacts.txt" in directory
85  ;where the script is
86  $file_existance = FileExists(@ScriptDir & "/contacts_added.txt")
87  If $file_existance == 0 Then
88
89      add_contacts_from(@ScriptDir & "/contacts.txt")
90      $cnt_added = FileOpen(@ScriptDir & "/contacts_added.txt", 10)
91      FileClose($cnt_added)
92  EndIf
93
94
95
96  Func add_contact($strContact)
97      If WinExists("Windows Live Messenger") Then
98          WinActivate("Windows Live Messenger")
99          WinWaitActive("Windows Live Messenger")
100
101         Send("{ALT}")
102         Send("{RIGHT}")
103         Send("{DOWN}")
104         Send("{ENTER}")
105         Send($strContact)
106         Sleep(250)
107         Send("{ENTER}")
108         Send("{ENTER}")
109     EndIf
110 EndFunc   ;==>add_contact
111
112
113
114
115
116 Func add_contacts_from($file)
117     ;Check if file with contacts exists
118     If FileExists($file) Then
119
120         $contacts = FileOpen($file, 0)
```