Computer Networks 56 (2012) 448–461

Contents lists available at SciVerse ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Secure instant messaging in enterprise-like networks ☆

Mengjun Xie [a], Zhenyu Wu [b], Haining Wang [b,*]

[a] Computer Science Department, University of Arkansas at Little Rock, Little Rock, AR 72204, United States
[b] Computer Science Department, College of William and Mary, Williamsburg, VA 23185, United States

ARTICLE INFO

Article history:
Received 12 July 2010
Received in revised form 16 May 2011
Accepted 7 September 2011
Available online 17 October 2011

Keywords:
Instant messaging
Security
Enterprise networks

ABSTRACT

Instant messaging (IM) has been one of most frequently used malware attack vectors due to its popularity. However, previous solutions are ineffective to defend against IM malware in an enterprise-like network environment, mainly because of high false positive rate and the requirement of the IM server being inside the protected network. In this paper, we propose a novel IM malware detection and suppression mechanism, HoneyIM, which guarantees almost zero false positive on detecting and blocking IM malware in an enterprise-like network. The detection of HoneyIM is based on the concept of honeypot. HoneyIM uses decoy accounts to trap IM malware by leveraging malware spreading characteristics. Fed with accurate detection results, the suppression of HoneyIM can conduct a network-wide blocking. In addition, HoneyIM delivers attack information to network administrators in real-time so that system quarantine and recovery can be quickly performed. The core design of HoneyIM is generic, and can be applied to the scenarios that either enterprise IM services or public IM services are used in the protected network. Based on open-source IM client `Pidgin` and client honeypot `Capture`, we build a prototype of HoneyIM and validate its efficacy through both simulations and real experiments. Our results show that HoneyIM provides effective protection against IM malware in enterprise-like networks.

1. Introduction

Instant messaging (IM) has been widely used in enterprise environments. According to [2], the daily number of instant messages sent within enterprises around the world is 15 billion in 2009, and will be tripled in 2013, reaching 46 billion. However, large user base and communication immediacy also attract malware to land on IM, which is particularly ideal for malware propagation. By virtue of IM features and social engineering tricks, IM malware can spread quickly and stealthily, which poses a serious security threat not only to home IM users but also to organizations which allow the use of instant messaging in workplace. The IM malware studied in this paper refers to the malicious code that spreads through the Internet-based IM networks such as Windows Live Messenger (formerly named MSN Messenger) and AOL Instant Messenger (AIM), which have dedicated servers for account management and message relay. Bropia [3] that attacks MSN Messenger and Opanki [4] that attacks AIM are two examples of IM malware. Most of known IM malware spreads through public IM networks. Security breaches caused by IM malware not only result in individual system damage and financial losses, but also often seriously degrade the usability of IM service. For example, in November 2010, the spread of IM malware forced Microsoft to temporarily turn off active link functionality in Windows Live Messenger 2009 because the malware propagates through instant messages with malicious URL links [5]. IM malware can also penetrate enterprise IM systems such as IBM Lotus Sametime [6] and Microsoft Lync Server [7] as these corporate IM services usually provide connectivity and

☆ The preliminary version of this paper was appeared in the proceedings of ACSAC 2007 [1].

* Corresponding author.
E-mail addresses: mxxie@ualr.edu (M. Xie), adamwu@cs.wm.edu (Z. Wu), hnw@cs.wm.edu (H. Wang).

1389-1286/$ - see front matter © 2011 Published by Elsevier B.V.
doi:10.1016/j.comnet.2011.09.006

interoperability with public IM services. In 2005, the outbreak of a variant of Kelvir worm even forced Reuters to shut down its IM service [8].

File transfer and URL (Uniform Resource Locator)-embedded message are two major spreading vectors of IM malware. After compromising an IM client, the malware propagates itself by either making a malicious file transfer or sending a text message containing a malicious URL to the online users[1] in the victim's contact list. The contact list is also called buddy list. Once those invigilant contacts click the file or URL, malicious code will be triggered to execute or be downloaded from the URL and executed, and subsequently the malware propagation continues at an exponentially increasing speed.

Although the threat of IM malware, especially the outbreak of zero-day IM malware, is on the rise, network administrators still lack effective solutions to protect enterprise-like networks such as campus networks and corporate networks. Conventional protections using firewalls and anti-virus products are insufficient to defend against IM malware due to the unique propagation feature of IM malware. Most of popular IM protocols are able to circumvent firewalls if their default ports are blocked. Signature-based anti-virus products cannot detect zero-day IM malware. Meanwhile, anomaly detection techniques, such as Norman Sandbox technology [9], may also be ineffective in catching evasive malware which behaves differently in the sandbox environment. Compared to malicious file transfers, malicious-URL-embedded IM messages are even harder to be identified by firewalls and anti-virus programs. Although there exist many URL blacklists such as Google Safe Browsing API [10], SURBL [11], and URIBL [12], a recent IM threat characterization study shows that the majority of malicious URLs sent from IM malware slip through those blacklists [13].

IM providers may take quick responses, e.g., releasing patches and mandating client upgrade, to newly discovered vulnerabilities in their products. They may even proactively block potentially malicious file transfers. However, these filtering mechanisms still could be bypassed [14,15]. Moreover, it is extremely hard for IM providers to protect against malicious URLs that exploit the vulnerabilities of Web browsers or other related applications [16]. While some protection schemes, such as CAPTCHA and virus throttling for IM [17,18], can enhance IM security, the incurred overhead and usability degradation could be significant, and thus prohibit IM providers from using them in near future.

Motivated by the shortage of effective defense against IM malware, we propose HoneyIM, a framework for automating the process of IM malware detection and suppression in an enterprise-like network. Based on the concept of honeypot, HoneyIM detects IM malware by leveraging its inherent spreading characteristics. Specifically, HoneyIM uses decoy accounts in normal users' contact lists as sensors to capture malicious content sent by IM malware, which achieves almost zero false positive. With accurate detection, HoneyIM suppresses malware by performing network-wide blocking. In addition, HoneyIM delivers attack information to network administrators for system quarantine and recovery. The core design of HoneyIM is generic and can be applied to a network that uses either private (enterprise) or public IM services. We implement a prototype of HoneyIM for public IM services, based on open-source IM client `Pidgin` [19] and client honeypot `Capture` [20]. We validate the efficacy of HoneyIM through both simulations and real experiments. The simulations show that even only a small portion, e.g., 5%, of IM users in the network have decoys in their contact lists, HoneyIM can detect the IM malware as early as after 0.4% (on average) of IM users are infected. The experimental results demonstrate that the prototype system succeeds in detection, suppression, and notification of IM malware within seconds.

The remainder of the paper is structured as follows. Section 2 describes the major spreading mechanisms of IM malware and related work. Section 3 presents our measurement study on IM user communication. Section 4 details the framework of HoneyIM, followed by the implementation and evaluation of HoneyIM in Sections 5 and 6, respectively. Section 7 discusses possible evasion to HoneyIM and the countermeasures. Finally, we conclude the paper in Section 8.

2. Background and related work

2.1. IM malware

IM malware propagates mainly through two ways: malicious file transfer and malicious URL in text message. Usually the malware infection is triggered by the victim's action such as clicking the transferred file or the received URL. IM malware could also spread without victim's involvement, e.g., by exploiting the vulnerabilities in IM clients. However, this type of spreading vector is rare.

In the file transfer mechanism that has been used since early 2000s, IM malware propagates by initiating malicious file transfers to remote contacts. Malicious files are usually renamed to attract victims or to evade network filters. Once a victim clicks the file, the malware is invoked and will attempt to infect more victims in the contact list. To counter this type of malware spreading, some IMs such as MSN forbid IM clients to transfer certain types of files such as *.pif* files. While the actual file transfer is normally carried out directly between two IM clients, the messages for transfer establishment still go through IM server. Therefore, IM servers can easily detect the messages for establishing malicious file transfers and silently drop them to block malware propagation.

Nowadays malicious URL messages become much more popular than malicious file transfer for IM malware propagation. Instead of sending a file, IM malware sends a text message containing a malicious URL to remote contacts. Once a victim clicks the link, either a malware binary is downloaded and executed or some malicious web scripts run to exploit the vulnerabilities of the Web browser or other related applications. Compared to malicious file transfers, malicious URL messages have several advantages

[1] Offline contacts may also be attacked but this type of attack is rare.

in propagation. First, malicious URL messages have more means to compromise a system. File downloading is just one of its attacking vectors. Second, malicious URLs can be used to collect victims' information by exploiting Web functionality. For instance, the URL sent by Kelvir [21] points to a php script and contains the contact's email address. The email address is harvested as soon as the URL is clicked. Last but not least, IM malware can play more social engineering tricks on URLs. For example, a malicious URL can be crafted to mimic the link on a reputable Web site [22]. The IM clients supporting HTML scripts also provide a playground for IM malware to fake URLs at their will. Those forged URLs appear normal but in fact point to malicious webpages.

After infection, IM malware may take different actions for propagation. Many types of malware start spreading immediately after they compromise IM clients, while others wait until they receive instructions to spread. The latter usually install certain bot programs on compromised machines, through which the malware is controlled by the remote bot herder.

2.2. Related work

The security threats posed by IM malware have been studied in [23,24]. In [23], the spreading speed of IM malware is estimated, showing that 500,000 machines could be infected within a minute.

The network properties and communication characteristics of instant messaging have been extensively studied and leveraged by previous defense schemes against IM malware. Avrahami and Hudson [25] explored communication characteristics of 16 IM users using five-month user logs. Xiao et al. [26] investigated IM traffic characteristics of hundreds of users by analyzing one-month network traffic trace of two popular IM systems (AIM and MSN) captured within a large enterprise. Leskovec and Horvitz [27] studied structural properties and communication patterns of the global MSN network using a full-month dataset of the whole MSN system. Several independent measurement studies [18,28,29] have revealed that IM social networks formed by IM contacts are scale-free, that is, the IM network connectivities follow power-law distributions. However, Xiao et al. [26] suggested that Weibull distributions may be more appropriate for describing the connectivity of IM social networks. Leskovec and Horvitz [27] found that the degree distribution of the global MSN network formed by actual IM communication is heavy tailed but does not follow a power-law distribution.

For scale-free networks, a small portion of nodes that are highly connected have significant effect on mitigating malware spread. Based on this observation, Smith [29] proposed to delay the propagation of IM malware by disabling the accounts of most connected IM users on the network. This scheme needs to be deployed on IM servers. It only reduces the spread speed and may have significant side-effects. Williamson et al. [18] applied their virus throttling mechanism to instant messaging and demonstrated its effectiveness through simulation. The throttling to IM also needs to be conducted at server side. The throttling becomes blind blocking if its threshold is very restrictive, which degrades IM usability. Mannan and van Oorschot [17] proposed two defense methods, namely limited throttling and CAPTCHA-based challenge-response. They also provided a usage study on per-user frequency of IM text messages and file transfers to support the applicability of the CAPTCHA-based challenge-response scheme. Liu et al. modeled the spread of IM malware using multicast tree [30] and analogous branching process with varied lifetime [31]. HoneyIM is orthogonal to all the aforementioned schemes and therefore is compatible with them.

Honeypot technologies [32,33] have been widely used in computer security research. A honeypot is a system or resource that is deployed to lure malicious activities. Seifert et al. provided a detailed taxonomy of honeypots in [34]. Traditionally, honeypots are servers (or devices that expose server services) configured passively to be attacked. In [35], Wang proposed the concept of client honeypot and introduced the first open-source client honeypot implementation, honeyclient. A client honeypot is a system or tool that mimics a user-driven network client application and interacts with the server in question to examine whether the server contains malicious content. Due to the popularity of Web, the focus of client honeypots is often on Web browsers. In [36], Wang et al. developed an Internet Explorer based high-interaction client honeypot called HoneyMonkey and demonstrated its effectiveness on finding a large number of malicious Web sites in a fully automatic manner. `Capture` [20] is an open-source high-interaction client honeypot supporting major Web browsers and HTTP aware applications. Besides high-interaction client honeypot systems that are fully functional systems, there are a number of low-interaction client honeypot implementations, which are also called virtual honeypots. A low-interaction client honeypot does not use a real client application such as Web browser, but rather uses a lightweight or emulated client. HoneyC [37] is a low-interaction client honeypot that drives a Web browser emulator to interact with Web servers. Recently, Nazario [38] developed another open-source virtual client honeypot named PhoneyC. PhoneyC is capable of deobfuscating malicious content in Web pages and emulating specific vulnerabilities.

Honeypot technologies have been applied to measuring and characterizing IM threats. Trivedi et al. studied the network and content characteristics of spim, the spam messages on IM networks, by using a proxy server as honeypot [39]. Their work is different from HoneyIM, since [39] is a measurement study and it targets spim but not IM malware. The honeypot used in [39] refers to a SOCKS proxy, which is exploited by spimmers to conceal their identities. Antonatos et al. conducted a more systematic characterization of IM threats using HoneyBuddy [13]. HoneyBuddy is quite similar to HoneyIM in the design: both systems leverage decoy accounts to trap malicious file transfer requests and IM messages and detect IM malware. However, HoneyBuddy is designed for measuring IM threats over the Internet. Therefore, HoneyBuddy does not have suppression and notification functionality. Being deployed in an open environment, HoneyBuddy is equipped with the functionality of actively searching and adding new "buddies" to its decoy accounts, while this

functionality is not necessary for HoneyIM as HoneyIM is designed for a closed environment.

Yan et al. proposed two algorithms that are based on change-point detection to detect both fast scanning and self-restraining IM malware [40]. The algorithm for detecting fast scanning IM malware applies the CUSUM algorithm to discover the abrupt increase of file transfer requests or URL embedded IM messages; The algorithm for detecting self-restraining IM malware leverages the social intimacy characteristic of IM users to identify the file transfer requests or malicious IM messages issued by stealthy IM malware. Both algorithms are designed for server-side deployment.

3. Measurement

Effective countermeasures against IM malware require a thorough understanding of normal IM user behaviors. However, most of existing IM measurement studies focus on either structural properties of IM networks (e.g., [29,26]) or high-level communication patterns of IM users (e.g., [27]). In [25], a behavior characterization of 16 IM users was conducted to help build a prediction model of interpersonal relationship. To date, a study of IM user behavior characterization for the purpose of IM security is still missing but much needed.

In this section, we give an analysis of IM user communication patterns based on the log histories of 34 MSN users. Compared to [25], our data set is much larger. The total number of messages we collected is over 1 million while the number of messages studied in [25] is only around 92 thousand. The maximum timespan of log histories is five months in [25] while ours is over seven years. The long-time history logs and the large number of messages greatly strengthen our analysis, and we believe that the revealed results can shed light on the design of future IM malware countermeasures.

We chose MSN as the IM system for characterization due to its worldwide popularity. For those volunteers who would like to contribute their MSN log history files, we sent them a special program to anonymize data. All names in a log file were replaced by numbers and message contents were removed. The anonymized information was then stored into an XML file and sent back to us. In total, we have 34 volunteers who contributed their chatting histories. Among them, 18 are female and 16 are male. The data set contains the logs from both home and workplace (office or school).

Table 1 provides a brief summary of the collected data. For each contributor, we calculate the numbers of active buddies, sessions, and messages, the aggregated size of messages, and the timespan of the chatting history from his/her history logs. Then, we aggregate the data of all contributors and compute the following statistics: sum, min, max, median, and mean/std, for each of the five attributes and list them in Table 1. An active buddy refers to such a person with whom at least one conversation (either sending or receiving a message) was recorded. Clearly, the number of active buddies is the lower bound of the number of all contacts in the buddy-list. For each active buddy, MSN messengers maintain a log file that contains all the chats with the buddy. For MSN messengers, messages displayed in the same chatting window are grouped into a session, and the timestamp of the first (last) message is the start (end) time of the session. Each message is a string of characters ended with a return character, and its size only accounts for the content in plain text. All HTML tags in a chat are removed and not considered in size calculation. The timespan of a contributor's chatting history is determined by the earliest and latest messages in his/her logs.

The collected chatting histories contain a total of 1,237,150 incoming and outgoing instant messages, which belong to 60,421 sessions between 34 contributors and their 2,301 buddies and have 16,589,882 bytes. From the table, we can see that each attribute manifests a high variance, as indicated by the coefficient of variation (CV) of each attribute being greater than 1. For example, a contributor can have as few as six active buddies but can also have as many as 394 buddies for chatting. We further find out that aggregation is not a major cause of the high variation. The numbers of sessions/messages averaged over buddies and/or days still significantly vary across contributors. The root cause we believe lies in the very different user behaviors.

To better understand the session-level user behavior, we aggregate all the sessions together from the history logs of 34 contributors and compute the statistics per session. Fig. 1 shows the CDF of message number (in solid line) and the CDF of message size in terms of byte (in dashed line). Note that the X axis is log-scaled. We can see that 20% of sessions have only one message, which means in those sessions the chatting window is immediately closed after sending or receiving a message line. About 65% of sessions have ten messages or less and 96% of sessions have one hundred messages or less, indicating that in general a chatting window do not stay for long time. The majority of sessions (over 90%) contain messages of one kilobyte (KB) or less. The message number per session and the message size per session both present high variance. Their corresponding CVs are 3.6 an 3.1, respectively.

Table 1
Overview of the collected MSN log information.

Statistics	Active buddies	Sessions	Messages	Size (byte)	Timespan (day)
Sum	2301	60,421	1,237,150	16,589,882	20,573
Min	6	14	293	3720	16
Max	394	9911	254,542	3,180,019	2695
Median	52	947	13,680	208,800	365
Mean ± std	67.7 ± 73.5	1,777.1 ± 2,145.9	36,387 ± 56,539	487,940 ± 683,810	605.1 ± 641.6

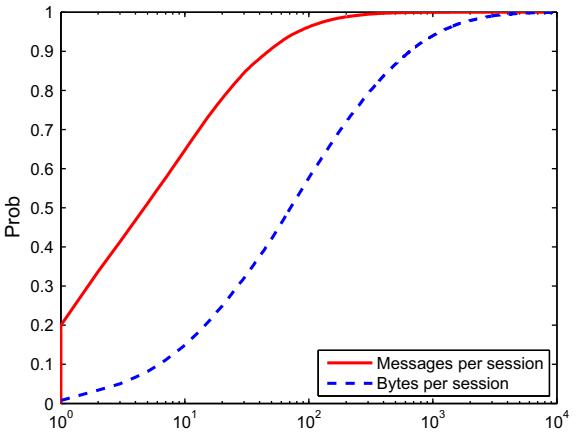

Fig. 1. CDFs of message number and message size per session.

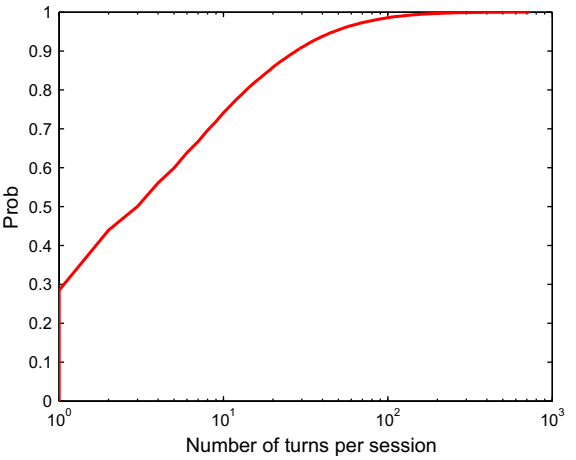

Fig. 3. CDF of the number of turns per session.

We further compute the CDFs of session interval and session duration (both in second) and display them in Fig. 2. Because around 20% of sessions contain only one message, their corresponding session duration is zero. Clearly, session duration is at least one order of magnitude shorter than session interval. The average session duration is 3,404 s (close to one hour) while the average session interval is 802,980 s (over nine days). Both session duration and session interval are highly dynamic and their CVs are 10.1 and 5.0, respectively.

As instant messaging provides an online communication medium, through which text-based human interaction occurs in nearly real-time, we are particularly interested in the interactive characteristics of IM users. In each session, we group consecutive messages sent by the same user into a single turn and dissect the session into one or more turns. In total, we obtain 710,045 turns from all the sessions. Although the recent versions of MSN messenger support multi-user session in which more than two users participate in the conversation, we find that the percentage of multi-user sessions is negligible in our collected data. The CDF of the number of turns per session is presented in Fig. 3. From the figure, we can see that about 30% of sessions contain one turn. Therefore, the majority (~70%) of sessions have two or more turns. In other words, most of the time a chatting window will not be closed before a response is received. Simply put, high communication interactivity manifests at the session level.

Fig. 4 depicts the CDFs of message number per turn and message size per turn. Notably, 97% of turns are composed by 100 or less bytes; 62% of turns have one message and 99% of turns consist of five or less messages, which further reveals the interactive nature of IM communication. For those sessions including multiple turns, the communication is conducted in a manner very similar to that of normal face-to-face conversation: speaking by turns and in short sentences. Similar to session-level measures, the distributions of turn-level message numbers and message sizes also show large variation. The CVs of message numbers and message sizes are 8.6 and 6.7, respectively.

Another visible feature of interactive conversation is that usually either party tries to speak concisely and be responsive. Instant messaging also demonstrates this

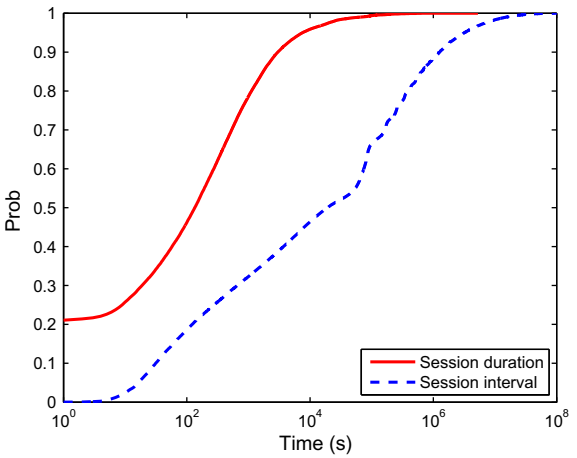

Fig. 2. CDFs of session interval and session duration.

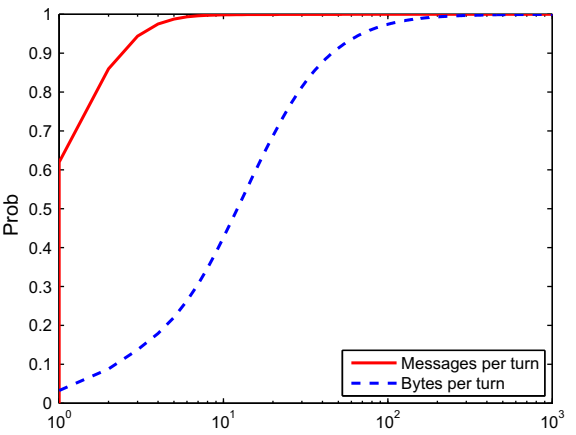

Fig. 4. CDFs of message number and message size per turn.

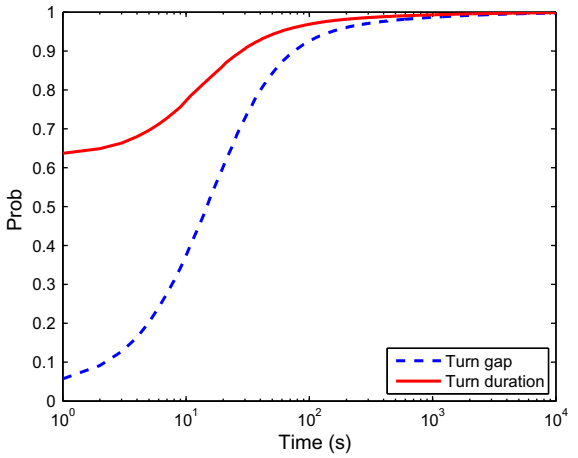

Fig. 5. CDFs of turn gap and turn duration.

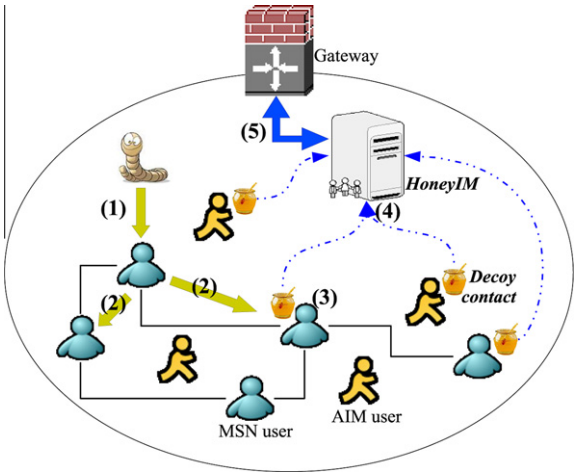

Fig. 6. Working mechanism of HoneyIM.

feature. Fig. 5 shows the CDFs of turn gap (the interval between two consecutive turns) and turn duration (the interval between the first message and the last one in a turn). Since 62% of turns are made up of a single message, in which the message is both the first and the last, the duration of those turns becomes zero. The fact that 97% of turn durations and 93% of turn gaps are two minutes or less, clearly indicates the highly interactive feature of IM usage.

4. HoneyIM framework

HoneyIM aims to assist network administrators in IM malware defense by automating the process of malware detection and suppression in an enterprise-like network. Utilizing the innate spreading characteristics of IM malware and applying the concept of honeypot, HoneyIM can detect and block unknown IM malware at its early stage of spreading, which greatly facilitates network filtration and system quarantine and recovery. In this section, we first give an overview of HoneyIM, how and why it can detect IM malware early. Then, we discuss several issues that need to be considered when using HoneyIM in practice. After that, we present the design of HoneyIM and the functionality of its components. Finally, we describe the deployment of HoneyIM in an enterprise-like network.

4.1. Overview

HoneyIM is based on the concept of honeypot. As an effective intrusion detection technology, honeypot has been used widely [32,33]. According to [32], *a honeypot is an information system resource whose value lies in unauthorized or illicit use of that resource*. Not only can a honeypot be a physical machine or a specialized program, which is a common case, but it can also be an e-mail address, or even an IM decoy user. Since IM malware always attempts to infect other users on the victim's contact list, HoneyIM captures this essence and exploits decoy users to detect IM malware. Under normal circumstances, a client user does not initiate a conversation with a decoy user. Therefore, if the decoy user receives a file transfer request or a URL-embedded text message originated from a client user, it is highly probable that malware is spreading and the request/message sender has been compromised. Thanks to decoy users, HoneyIM can achieve almost zero false positive in detection. This strong guarantee, which is rarely offered by other schemes, relieves network administrators from worrying about possible interruption to normal IM users caused by the protection technique. In addition, HoneyIM can block malicious content that has been detected and inform network administrators of the attack information, e.g., the IP address of the compromised machine, in real-time.

Fig. 6 illustrates the working mechanism of HoneyIM. The IM user with an icon of honeypot is the one whose contact list contains a decoy user. The events happen in the following sequence. (1) Some IM malware compromises an IM client and (2) propagates. However, (3) when it tries to spread again, it hits a decoy user and (4) is detected by HoneyIM. (5) HoneyIM blocks the malicious content in IM traffic (either at the edge gateway or at the IM server if the IM service is provided within the network) and non-IM traffic[2] instantly, and notifies the attack information to the network administrator.

HoneyIM is designed to be independent, with no restriction on the type and location of IM servers. Therefore, the framework of HoneyIM can be flexibly realized under the context of either public IM services or private (enterprise) IM services being used in the protected network. The core of HoneyIM is the same for both server-enhanced realization (HoneyIM with private servers) and serverless realization (with public servers). The difference lies in the implementation and deployment, which will be discussed in Section 4.4. The framework of HoneyIM consists of several modules and these modules can be deployed in a single machine or at different places.

[2] Doing this is to block accesses to malicious contents, e.g., malicious URLs.

4.2. Design issues

The success of HoneyIM largely depends on the use of decoy users. In the following, we discuss three issues of HoneyIM that are much related to decoy users, including initialization, sensitivity, and compatibility.

The initialization of HoneyIM mainly refers to the creation and addition of decoy user accounts. Strictly speaking, it is a deployment issue. If public IM services are used in the protected network, the administrators need to create decoy accounts and solicit the IM users within the organization who are willing to add those decoy users into their contact lists. The process is basically the same as a normal IM user adding a new buddy. The administrators are also responsible for authorizing and authenticating the volunteer users to prevent external IM users from becoming a decoy user's friends. On the other hand, if an enterprise IM service is employed, the creation and addition of decoy users can be done automatically by programming the IM server. However, the IM system must notify users of the purpose and usage of decoy accounts, and provide a disable (or opt-out) option. The decision of how many decoy accounts to set up should consider both the number of IM users under protection and the overhead of setup and maintenance. On one hand, more decoy accounts provide more flexibility of mimicking human users, helping improve detection rate. On the other hand, more decoy accounts make account setup and maintenance more complicated.

The initialization of HoneyIM is a one-time fulfillment and its overhead can be much reduced by using tools that automate administrative tasks such as AutoIt [41]. The maintenance of decoy accounts, for example, changing profiles of decoy accounts, adding or deleting decoy accounts, can be performed when necessary. In addition to the volunteer policy for IM user cooperation, administrators may require those IM users who have high connectivity degrees (in other words, the super nodes in IM networks) to include decoy accounts in their contact lists. These highly connected IM users are the critical points along the malware propagation path.

The sensitivity of HoneyIM is measured by the ratio between the number of infected users and that of all IM users in the protected network when the spreading of IM malware is first detected. The key factor affecting the sensitivity of HoneyIM is the coverage of HoneyIM—the portion of the IM users equipped with decoy user accounts among all IM users within the network. It is obvious that HoneyIM cannot detect malware for those users who do not include decoy accounts in their contact lists. Moreover, IM malware may intentionally or inadvertently bypass HoneyIM by not hitting decoy users in the infected users' contact lists. The word "intentionally" does not mean that the IM malware knows the decoys in advance, but reflects its capability of distinguishing decoys from other contacts. Here we assume that the threat comes from the outside of the protected network and the inside IM users do not collude with the outside attackers. Given the coverage of HoneyIM, which is usually determined by the network administration policy, we will consider how to counter evasive IM malware to improve HoneyIM sensitivity in Section 7.

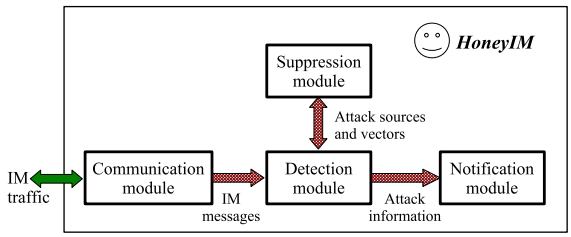

Fig. 7. Framework of HoneyIM.

Compatibility is not an issue if HoneyIM is deployed on an enterprise IM server, since the server can maintain the compatibility with supported IM clients. However, the compatibility has to be taken into account if public IM services are used in the protected network. Under this circumstance, various types of public IM systems may coexist. This is especially true on the networks with less strict IM usage policies such as campus networks. Thus, HoneyIM should be able to communicate with different types of IM clients when deployed for public IM services. Today, this requirement can be easily fulfilled as the majority of IM protocols are accessible and many open-source IM clients such as `Pidgin` have the capability of simultaneously joining multiple IM networks.[3]

4.3. System components

Fig. 7 shows the general framework of HoneyIM, which comprises four modules each performing specific functionality. These modules can be deployed on either the same machine or different hosts (or network devices). As displayed, the communication module is responsible for handling IM traffic. It parses the IM traffic to decoy users and delivers it to the detection module. The detection module extracts attack vectors and related information from IM messages, and then feeds them into the suppression and notification modules. The suppression module sifts through network traffic and filters out malicious traffic containing attack vectors. Meanwhile, the notification module informs administrators of the detected malware spreading.

4.3.1. Communication module

The communication module is the base of HoneyIM. Decoy accounts use it to join IM networks and communicate with normal IM clients. This module realizes all necessary functions of a normal IM client, such as adding/deleting a friend, signing on/off, setting presence status, receiving messages and files, etc. These functions are automatically executed by default and can also be manually operated by an administrator. The module only accepts the messages from "friends", that is, the authenticated and authorized users in the contact list. In other words, messages from external IM users will be regarded as "spim" (the term for spam in IM networks) and discarded. Accepted

[3] Close-source IM systems are not a problem to HoneyIM as long as their IM protocols are known. Skype is an exception. It not only is close-sourced but also applies strong encryption to all communications.

messages are forwarded to the detection module. For a file transfer request, the communication module accepts the request and saves the file. At the same time, the module also records the IP address of the sending host, which is attainable as a file is usually transferred directly between two IM clients. After receiving the whole file, the communication module forwards both the file and sending host information to the detection module. For serverless realization of HoneyIM, the communication module is required to support all the IM protocols of the protected IM services and allow multiple decoy accounts to simultaneously log into different IM networks when necessary, which can be achieved by taking advantage of existing open-source IM projects.

4.3.2. Detection module

The detection module serves three purposes: (1) detecting compromised IM clients, (2) identifying attack vectors, and (3) validating attack vectors. It accomplishes the first two tasks by consulting the suppression module and scrutinizing IM messages delivered by the communication module, and attains the last task by conducting deep-inspection.

In principle, the detection module classifies a sending IM client as compromised if a decoy account receives a file transfer request or a text message with URL from the IM client. The reason is that it is very rare for a normal user to issue such a request or message to a decoy account.[4] The detection is not affected by encryption of client-to-client or client-to-server communication because IM messages received by a decoy (as a client) must be in plain-text. For IM malware that spreads via file transfer, the detection module obtains both the attack source (the IP address of the compromised host) and malicious file from the communication module. For IM malware that spreads through URL message, the detection module extracts the URL from the message and further derives the address of the sending host by consulting the suppression module, which will be described shortly.

Furthermore, the detection module performs deep-inspection to verify the virulence of the received file or URL. There are many techniques available to achieve this purpose. For example, for a received file, we can employ dynamic taint analysis based techniques such as TaintCheck [42] and Argos [43] to (1) examine whether the file can compromise system and (2) generate the corresponding signature if a compromise occurs. We can also adopt client honeypot techniques such as HoneyMonkey [36] and `Capture` [20] to examine received URLs. Both HoneyMonkey and `Capture` can detect Web exploits by browsing a URL inside a virtual machine and monitoring the change of system states. In general, any effective and efficient host-based anomaly detection technique can be used for deep-inspection. HoneyIM does not contain any specific technique for analyzing IM malware, but rather provides a platform to apply existing techniques to malware dissection and leaves the decision of what technique to use to administrator. The adopted techniques are implemented as plug-ins of the detection module, and the deep-inspection is conducted in a contained environment such as a virtual machine to prevent HoneyIM itself from being compromised.

It might be too expensive for those organizations that are lack of resources to set up and maintain a dedicated on-site deep-inspection module. An alternative option is to outsource the deep-inspection function to third parities. There exist a few anomaly detection websites such as Anubis [44] and VirusTotal [45] that provide services of examining suspicious files and URLs and reporting analysis results. To use these services, the detection module only needs to send the file/URL to the indicated address via public API and then wait for the result. Oursourcing deep-inspection certainly simplifies the detection module and saves the cost. However, possible side-effects such as delayed service response should also be considered.

The incorporation of deep-inspection is justified by the following considerations. First, deep-inspection can further reduce false positives. It is possible that innocent URLs or files could be sent with malicious content by IM malware to disguise their malice. Second, deep-inspection helps discover additional or real attack vectors used by IM malware. For example, file deep-inspection can generate the signature of malware binary, based on which the filtering is much more robust against evasion than based on file name. IM malware can also use different URLs in its spreading, which in fact are doorway webpages redirecting traffic to the same website that hosts real exploits. With URL deep-inspection, the protection can be further enhanced because not only doorway URLs but also real exploit URLs can be discovered. Last but not least, deep-inspection uncovers the IM malware activities, such as the infection mechanism and the infected files, for network administrators.

After attack vector extraction and validation, the detection module supplies the validated attack vectors and sources to the suppression module for immediate network traffic filtration. In the meantime, the detection module feeds all collected attack information into the the notification module, which informs network administrators of the occurrence of an attack in real-time for prompt system quarantine and recovery.

4.3.3. Suppression module

The suppression module in essence is a network filter. It takes as input the attack source and vector information from the detection module. Then, it blocks any traffic from attack sources and filters out network traffic that contains attack vectors. Different from other modules that have no requirement for deployment location, the suppression module should be installed at a network vantage point, where it can monitor all traffic passing through the protected network. The location of the suppression module will be further discussed in Section 4.4.

The suppression module consists of two components: non-IM traffic filter and IM traffic filter. These two components are logically independent for flexible implementation and deployment. The non-IM traffic filter fulfills two tasks: blocking attack sources and filtering non-IM network traffic. For the former, the filter simply drops any packet from the attack sources to terminate malware prop-

[4] Even if a normal user accidentally sends a message to the decoy account, the message is usually a pure text message.

agation. For the latter, the filter examines contents of inbound and outbound packets to identify if an internal user is attempting to access a malicious webpage or transfer a virulent file. Any packet containing a matched attack vector will be discarded.

The IM traffic filter also provides two functionalities. The first is traffic filtration, which weeds out the IM messages that either come from (or go to) the compromised clients or contain identified malicious file names or URLs. Although a file is usually transferred between two clients, the IM messages for establishing transfer connections are relayed through servers in plain-text for mainstream IM products. Therefore, blocking malicious file transfer by dropping connection establishment messages is not affected by client-to-client encryption. The second functionality of the IM traffic filter is to help identify malicious URL sending hosts within the protected network. Because messages are relayed through server, the detection module cannot identify the sources of malicious URL messages. To track the IP address of the compromised host, the IM traffic filter records the URLs and the corresponding IP addresses of their senders. With this information, the detection module can easily pinpoint the malicious URL senders.

4.3.4. Notification module

The notification module plays the role of messenger. Its job is to inform network administrators of the occurrence of IM malware spread upon the detection of an attack. Given the fast spread of IM malware, the notification to network administrators should be made in real-time or near real-time by means of SMS (Short Messaging Service) or IM. The notification module can also notify the victim about the fact that his machine has been infected with IM malware via IM or email.

4.4. Deployment

As mentioned in the overview section, HoneyIM can be deployed with a private IM server inside the protected network (server-enhanced deployment) or with public IM services outside the network (serverless deployment). The major differences between the two deployments lie in the function location and system initialization of HoneyIM. In serverless deployment, the non-IM and IM traffic filters of the suppression module have to be placed on the network edge device. However, in server-enhanced deployment, while the non-IM traffic filter still needs to be on the network edge device, the best place for the IM traffic filter is the private IM server, where the filter can see all IM traffic. Moreover, in practice many IM servers already include the message filtering functionality, making IM traffic filtering much easier there.

The deployment of HoneyIM also involves system initialization, i.e., the creation and addition of decoy accounts. In serverless deployment, network administrators need to register accounts for decoy users on public IM services before running HoneyIM. Due to the maximum size of contact list (e.g., 600 for MSN) and the protection consideration, the administrators can create multiple decoy accounts and use them for different groups of IM users. Then, the decoy accounts are added into the volunteer IM users' contact lists with their cooperation. By contrast, the server-enhanced deployment saves the efforts of network administrators and IM users by automating the creation and addition of decoy accounts, just like the use of AIM Bots for shopping and movie guide. This can be achieved by adding a decoy account management module to the private IM server. The module can also be used to (1) provide IM users with the information of decoy accounts and the option to enable/disable them, and (2) update decoy accounts periodically against potential evasion.

5. Prototype

To demonstrate the efficacy of HoneyIM, we have built a prototype of the serverless HoneyIM, which can be easily transformed to the server-enhanced HoneyIM prototype with minor changes in function location and system initialization. We implement the HoneyIM modules using different techniques. We use a full-fledged open-source IM client `Pidgin` (formerly known as `Gaim`) [19] to build the communication module. The detection module employs `Capture` [20], a high interaction client honeypot on Windows systems, for URL deep-inspection. The detection module extracts URLs from the communication module and feeds them into `Capture`, which decides whether a URL is malicious by comparing the system states such as registry and running processes before and after the URL is accessed. For any file transfer request HoneyIM does not perform deep-inspection but immediately fires an alert instead, given that the file transfer method is relatively unpopular in IM malware spreading and most IM users and programs are vigilant to this type of threat. HoneyIM receives the delivered file and sends it to network administrators via email. In the construction of the suppression module, we use Perl IPQueue module for iptables [46] to perform URL logging and pattern-matching. We implement the notification module with two communication means: email and SMS. The suppression module communicates with the detection module via network socket, and thus can be deployed on a separate machine.

Because `Pidgin` supports multi-protocol and multi-account, HoneyIM can log into multiple accounts on multiple IM networks simultaneously. Therefore, it can provide protection for multiple public IM networks. Note that the choice of `Pidgin` and `Capture` is mainly due to the availability of their source code. Upon the accessibility of source code, any IM clients or anomaly detection systems can be used to construct HoneyIM.

6. Evaluation

In this section, we first evaluate the detection sensitivity of HoneyIM under different coverage settings via simulation. Then, we validate the applicability of HoneyIM through real experiments.

6.1. Simulation

When adding decoy accounts is voluntary for IM users on the protected network, it is very possible that HoneyIM does

not cover all IM users. Under this circumstance, how effective would HoneyIM be? Because we cannot carry out a large-scale experiment in practice, we turn to simulation for answering this question. We adopt the simulation model from [47] due to the similarity in propagation between IM malware and Email worms [47]. The major metric we use is the percentage of IM users being infected by the time the IM malware is firstly detected by HoneyIM (the percentage of infected IM users for short), and we investigate its variation under different HoneyIM coverage configurations.

6.1.1. Simulation model

The simulation model of IM malware propagation is described as follows. First, when an IM user receives an IM message, she may or may not read the message immediately. The reading delay for user i, denoted by T_i, is a stochastic variable. When the user receives a message with a malicious URL,[5] she clicks the URL with a clicking probability denoted as C_i. We assume that C_i is a constant for user i. If the malicious URL is clicked, the malicious code is downloaded and executed immediately. It infects the current IM client and sends malicious URLs to all the victim's contacts with no delay. The malware will not spread again unless the user receives the same URL and clicks it again.

Before we start the simulation, we need to determine the IM network topology and the values of each C_i and T_i. Here the IM network refers to the virtual network composed by the contact lists of the IM users on the protected network. According to [29] that studies an IM network containing 50,158 users, over 80% of the user contacts are bidirectional, indicating that most of users are also in the contact lists of their buddies. Thus, we model the IM network topology by an undirected graph $G = \langle V, E\rangle$. For $\forall v \in V$, v denotes a node (IM user), and for $\forall e = (u, v) \in E$, u, $v \in V$, e represents an edge that connects two users, u and v, who are in each other's contact list. $|V|$ is the total number of nodes, and $D(i)$ is the degree of node i, i.e., the number of edges connected to node i. The size distribution of contact lists has been identified as scale-free by [28,29,18], except that [26] claims that Weibull distribution has a better fit. However, [26] does not give the parameters of Weibull distribution and the number of their monitored IM users is small compared to [28,29,18]. Therefore, we model the IM network topology as power law and set the power law exponent α to 1.7, based on the measurement results from [28, 29]. The network is generated by using GLP power law generator [48] with the given α, the number of nodes $|V|$, and the average node degree $E[D]$. We generate three IM networks with 200, 1000, and 5000 nodes to study the effectiveness of HoneyIM within small, medium, and large enterprise networks, respectively. The average node degree $E[D]$ is 20 for all three networks. The maximum node degrees of the generated networks are all below 600, the maximum size of a contact list for MSN.

Similar to [47], we assume that IM users have independent behaviors. Due to the large number of users $|V|$ and independent behaviors, the mean values of user reading delay T_i and clicking probability C_i, denoted by $E[T_i]$ and $E[C_i](i = 1, 2, \ldots, |V|)$, can be assumed to follow Gaussian distribution. That is, $E[T_i] \sim N(\mu_T, \sigma_T^2)$ and $E[C_i] \sim N(\mu_C, \sigma_C^2)$. We also assume that T_i follows exponential distribution and C_i is a constant for user i, and the generation of T_i and C_i is constrained by $T_i \geqslant 0$ and $C_i \in [0, 1]$. In simulation, we use $N(20, 10^2)$ and $N(0.5, 0.3^2)$ to generate $E[T_i]$ and $E[C_i]$, respectively.

6.1.2. Simulation results

Given the network topology, we *randomly* deploy decoys in the network with different coverage $\mathfrak{R}$ and run simulation experiments. Each simulation run stops once IM malware hits a decoy user (blocking is in effect immediately) or timeout occurs. The number of infected users and detection time are the simulation output. For each coverage $\mathfrak{R}$, we vary the decoy deployment 10 times and run simulation 100 times for each deployment, and have the mean and median values derived from these 1,000 simulation experiments.

With the increase of HoneyIM coverage, the corresponding percentages of infected IM users on three different IM networks are shown in Fig. 8, in which the solid curves are for mean values and the dashed curves are for median values. The mean curves are above the median curves for very small coverage values, and both types of curves drop sharply and converge to zero with the increase of coverage. This clearly demonstrates the effectiveness of HoneyIM. From the figure, we can see that when a decoy account is deployed on 10% of all IM users, on average, HoneyIM can detect IM malware with around 5% of users (10 users) being infected for the small network, and 1% and 0.2% of users being infected for the medium and large networks respectively.

As IM throttling [18] is an important technique for curtailing the spread of IM malware, we are interested in the performance comparison between HoneyIM and IM throttling. The throttling of IM malware is generally conducted on IM servers. We use the "no-delay" mode of IM throttling and configure the working set size and threshold to 5 and 2, respectively, as suggested. Since it is difficult to simulate the working set for each user at run time, we simplify the propagation model by (1) randomly determining a node's working set between 0 and 5 right before the node is propagating and (2) blocking the node after its propagation (no matter whether the delay queue is overflowed or not). Therefore, the maximum number of the nodes that a compromised node could infect is its working set size plus 2 (the threshold). Note that this model is conservative compared to the original scheme, as we block an infected node permanently once it starts spreading.

Fig. 9 shows the performance comparisons between HoneyIM (coverage $\mathfrak{R} = 10\%$) and throttling on the three IM networks. The solid curves represent HoneyIM and the dotted curves represent throttling. Note that the y-axis is logarithmic, and the results for throttling are the mean values for 100 runs. Fig. 9 demonstrates that HoneyIM outperforms IM throttling in all three network environments and the superiority of HoneyIM over IM throttling is more evident for larger networks. The simulation results show that HoneyIM can detect the spread of IM malware with

[5] The situation for malicious file transfer is similar.

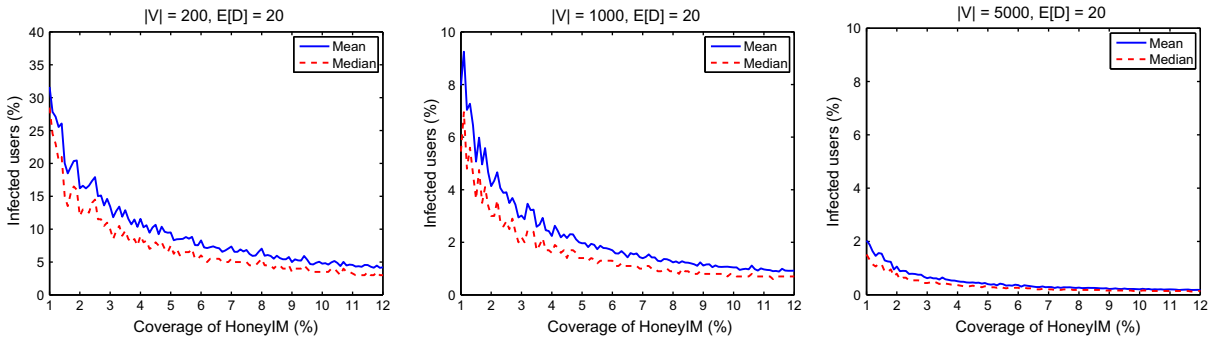

Fig. 8. HoneyIM coverage settings and their corresponding percentages of infected users.

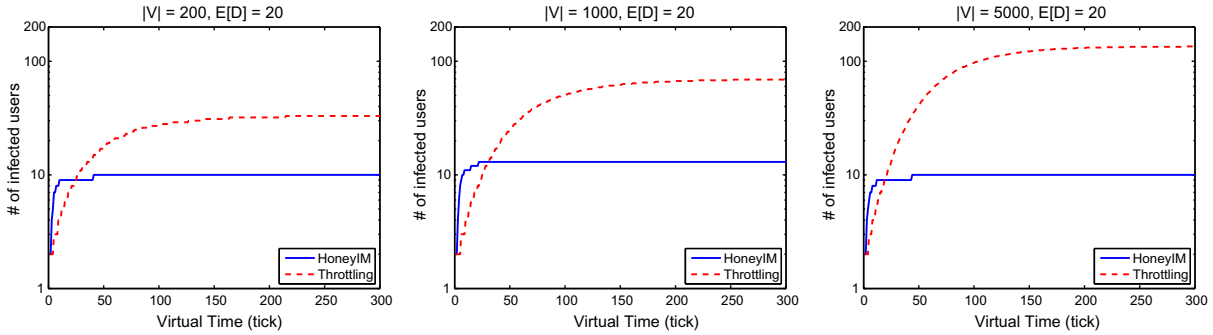

Fig. 9. Comparisons between HoneyIM and IM throttling.

less than 20 victims no matter the network has 200 nodes or 5,000 nodes, while the number of malware victims under IM throttling is already over 30 for the small network. More importantly, HoneyIM can accurately detect the malware and block its spread right after detection, while throttling cannot differentiate malicious traffic from normal traffic, let alone block them in an effective manner.

6.2. Real experiment

We set up a small testbed comprising three machines. We use one machine as the IM client and the other two as HoneyIM and the network gateway. The suppression module of HoneyIM is deployed on the network gateway. Both the IM client and HoneyIM run inside virtual machines for security and ease of experimentation. We first use real IM malware binaries we have collected to test HoneyIM by running malware on the IM client machine. We test Jitux-A [49], Kelvir-F [50], Kelvir-M [51], and Kelvir-Q [52], respectively, all of which spread through malicious URL messages on MSN platforms. The URLs for Jitux-A and Kelvir-F lead to .exe and .scr file downloading, while the URLs for Kelvir-M and Kelvir-Q point to .php scripts which also harvest victim's email addresses. Unfortunately, due to the legal reaction taken by the IM providers and security community, the webpages pointed by these known malicious URLs are either invalid or have been removed by the hosting websites.[6] The URL message sent by Kelvir-F is not even received by HoneyIM, because of the filtering in MSN servers. No detailed information about IM malware is given by deep-inspection. Thus, we reconfigure the detection module to skip the deep-inspection step and rerun the tests. The suppression and notification modules work well as expected.

We also test the prototype using a generic approach which overcomes the difficulty caused by the invalidity of the known malicious URLs. We mimic IM malware by sending malicious URLs collected by ourselves to decoy accounts. The malicious URLs we used, in principle, have no difference from those carried by known IM malware in terms of Web exploits. Thus, they should have the same effect on normal IM clients and HoneyIM. The URL process time of HoneyIM is mainly determined by deep-inspection, which is usually finished within 30 s. Overall, HoneyIM successfully detects all malicious URLs, updates the URL blacklist, and sends the attack information to the designated recipient via SMS and email. For emulated malicious file transfers, HoneyIM automatically receives files, reveals file names to the suppression module, and sends file payloads to the designated recipient via email. The whole process takes seconds to complete, since no deep-inspection is performed for file transfer.

7. Discussion

In previous sections, we assume that IM malware always attempts to infect all online contacts by either initiating a file transfer or sending a malicious URL during its

[6] This situation also applies to other known IM malware.

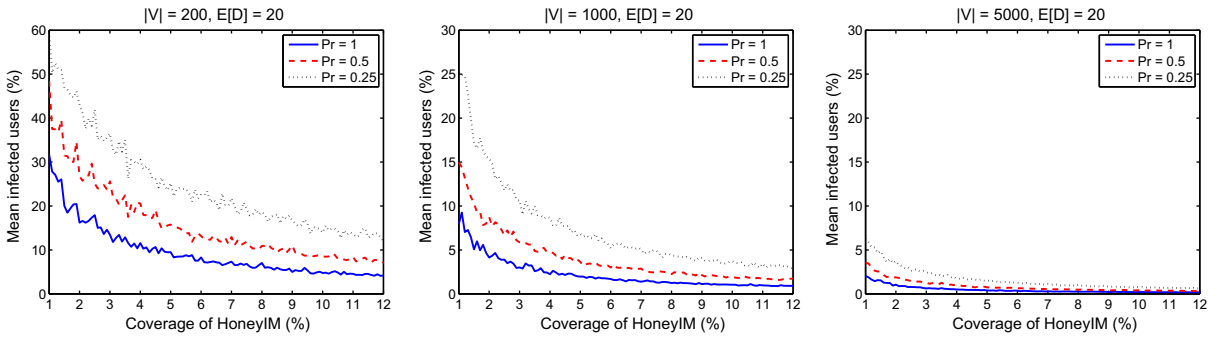

Fig. 10. Effects of randomly selecting infection targets on HoneyIM.

spread. This hit-all propagation strategy, however, might not always be used. For example, "smart" IM malware may send malicious URLs or files only to the active online contacts, i.e., those contacts that the infected IM client is talking to; or the propagation is activated only after the infected client receives a message. Taking the non-hit-all strategy, IM malware might not hit the decoy contact even if the contact list of the infected IM user includes the decoy accounts.

IM malware can realize the non-hit-all propagation strategy by either intentionally or randomly selecting a part of all online contacts as targets. To prevent decoys from being easily distinguished, we can enhance HoneyIM with interaction functionality. As a countermeasure, HoneyIM uses the interaction functionality to mimic human users for decoys by initiating chat sessions with normal users, making it much harder for IM malware to tell decoys from others. The chat content can be important security notices or other user interested information. We readily agree that IM malware can still avoid decoy contacts even with the interaction functionality, for example, by infecting the most active contacts. However, the spread of this type of IM malware could be significantly reduced. According to a recent IM traffic measurement [26], IM users only contact a small portion of users in their contact lists. On average an AIM user chats with only 1.9 users and an MSN user chats with 5.5 users.

The random selection of infection targets may also help IM malware bypass decoy contacts. To study the effect of the random selection on HoneyIM, we conduct the following experiments based on the previous simulation for HoneyIM. We apply a probabilistic propagation strategy to the experiments. That is, when IM malware propagates, it will send malicious content to each contact with a probability p. With the probabilistic infection, the number of users that malware will contact becomes $p \times n$ on average, where n is the total number of the online contacts of the infected user.

We test and compare the effects of random target selection on HoneyIM with three different probabilities $p = 1$, 0.5, 0.25 on the three IM networks, respectively. Here $p = 1$ refers to the aforementioned deterministic infection. The comparison is displayed in Fig. 10, in which the curve of $p = 0.5$ is above the curve of $p = 1$ but below the curve of $p = 0.25$. Fig. 10 depicts that the average number of infected users increases with the decrease of the probability value, indicating the elusiveness of the probabilistic propagation strategy. However, the figure also shows that the differences among three curves shrink quickly with the increase of coverage for all three networks. The effect of the random target selection becomes insignificant and even negligible when HoneyIM coverage is high.

8. Conclusion

In this paper we have proposed HoneyIM, a novel detection and suppression mechanism to defend against IM malware for enterprise-like networks. Distinct from all previous defense schemes, HoneyIM introduces decoy users for IM malware detection. It exploits the basic spreading characteristics of IM malware and guarantees almost zero false positive. With accurate detection, the suppression of HoneyIM achieves instant network-wide blocking. Moreover, HoneyIM notifies network administrators of the infected machines and the infection features of IM malware in real-time. The generic design of HoneyIM enables its flexible realization on a network that uses either enterprise IM services or public IM services. We have built a prototype of HoneyIM that works with public IM services using open-source IM client `Pidgin` and client honeypot `Capture`. The simulation studies demonstrate that even with a small portion of IM users equipped with decoy accounts, HoneyIM can still detect and block IM malware in the early stage of its spread. The real experiments on the prototype further demonstrate that HoneyIM is competently capable of detecting and suppressing the spread of IM malware.

Acknowledgments

This work was partially supported by NSF Grants CNS-0627339 and ECCS-0901537.

References

[1] M. Xie, Z. Wu, H. Wang, Honeyim: Fast detection and suppression of instant messaging malware in enterprise-like networks, in: Proceedings of ACSAC 2007, Miami Beach, FL, December 2007.
[2] Radicati, Instant messaging market, 2009-2013, <http://www.radicati.com/?p=4654>, 2009 (accessed 18.11.2010).
[3] Symantec, W32.bropia, <http://www.symantec.com/security_response/writeup.jsp?docid=2005-012013-2855-99>, (accessed 18.11.2010).

[4] Symantec, W32.opanki, <http://www.symantec.com/security_response/writeup.jsp?docid=2005-051810-1834-99>, (accessed 18.11.2010).
[5] J. Scarrow, Security alert: Active links in messenger 2009 temporarily turned off to prevent a malicious worm, <http://windowsteamblog.com/windows_live/b/windowslive/archive/2010/11/12/security-alert-active-links-in-messenger-2009-temporarily-turned-off-to-prevent-a-malicious-worm.aspx>, 2010 (accessed 18.11.2010).
[6] IBM, Lotus Sametime, <http://www-01.ibm.com/software/lotus/sametime/> (accessed 18.11.2010).
[7] Microsoft, Microsoft lync server, <http://lync.microsoft.com/en-us/Pages/default.aspx> (accessed 18.11.2010).
[8] M. Hicks, Reuters suspends im service due to kelvir worm, <http://www.eweek.com/c/a/Messaging-and-Collaboration/Reuters-Suspends-IM-Service-Due-to-Kelvir-Worm/>, 2005 (accessed 18.11.2010).
[9] Norman, Norman sandbox whitepaper, <http://download.norman.no/whitepapers/whitepaper_Norman_SandBox.pdf> (accessed 18.11.2010).
[10] Google, Google safe browsing api, <http://code.google.com/apis/safebrowsing/> (accessed 25.11.2010).
[11] SURBL, <http://www.surbl.org/> (accessed 25.11.2010).
[12] URIBL, Realtime URI blacklist, <http://www.uribl.com/> (accessed 25.11.2010).
[13] S. Antonatos, I. Polakis, T. Petsas, E.P. Markatos, A systematic characterization of IM threats using honeypots, in: Proceedings of the 17th Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA, February 2010.
[14] R. Schouwenberg, Do you like photos?" <http://www.securelist.com/en/weblog?weblogid=199354341>, 2006 (accessed 18.11.2010).
[15] R. Schouwenberg, MSN filter bypassing – part 2, <http://www.securelist.com/en/weblog?weblogid=199850358>, 2006 (accessed 18.11.2010).
[16] C. Raiu, The IM worms armada, <http://www.securelist.com/en/weblog?weblogid=203678309>, 2006 (accessed 18.11.2010).
[17] M. Mannan, P.C. van Oorschot, On instant messaging worms, analysis and countermeasures, in: Proceedings of WORM 2005, 2005, pp. 2–11.
[18] M.M. Williamson, A. Parry, A. Byde, Virus Throttling for Instant Messaging, HP Lab Bristol, Technical Report, May 2004.
[19] Pidgin, the universal chat client, http://pidgin.im/ (accessed 18.11.2010).
[20] R. Steenson, C. Seifert, Capture: A high interaction client honeypot, <https://projects.honeynet.org/capture-hpc> (accessed 18.11.2010).
[21] R. Schouwenberg, Kelvir changes its approach, <http://www.securelist.com/en/weblog?weblogid=162243612>, 2005 (accessed 18.11.2010).
[22] A. Gostev, Social engineering: the latest chapter, <http://www.securelist.com/en/weblog?weblogid=168136245>, 2005 (accessed 18.11.2010).
[23] N. Hindocha, E. Chien, Malicious threats and vulnerabilities in instant messaging, <http://www.symantec.com/avcenter/reference/malicious.threats.instant.messaging.pdf>, 2003 (accessed 18.11.2010).
[24] M. Mannan, P.C. van Oorschot, Secure public instant messaging: A survey, in: Proceedings of the 2nd Annual Conference on Privacy, Security, and Trust, 2004, pp. 69–77.
[25] D. Avrahami, S.E. Hudson, Communication characteristics of instant messaging: effects and predictions of interpersonal relationships, in: Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work (CSCW '06), 2006, pp. 505–514.
[26] Z. Xiao, L. Guo, J. Tracey, Understanding instant messaging traffic characteristics, in: Proceedings of the 27th ICDCS, Toronto, Canada, 2007.
[27] J. Leskovec, E. Horvitz, Planetary-scale views on a large instant-messaging network, in: Proceeding of the 17th International Conference on World Wide Web (WWW '08), 2008, pp. 915–924.
[28] C.D. Morse, H. Wang, The structure of an instant messenger network and its vulnerability to malicious codes, in: Proceedings of ACM SIGCOMM 2005 Poster Session, Philadelphia, PA, 2005.
[29] R.D. Smith, Instant Messaging as a Scale-Free Network, http://arxiv.org/abs/cond-mat/0206378v2, 2002 (accessed 18.11.2010).
[30] Z. Liu, G. Shu, N. Li, D. Lee, Defending against instant messaging worms, in: Proceedings of IEEE GLOBECOM 2006, San Francisco, CA, 2006, pp. 1–6.
[31] Z. Liu, D. Lee, Coping with instant messaging worms – statistical modeling and analysis, in: Proceedings of the 15th IEEE Workshop on Local and Metropolitan Area Networks, Princeton, NJ, 2007.
[32] The Honeynet Project, Know Your Enemy: Learning about Security Threats (2nd Edition). Addison-Wesley Professional, May 2004.
[33] N. Provos, T. Holz, Virtual Honeypots: From Botnet Tracking to Intrusion Detection, Addison-Wesley Professional, 2007.
[34] C. Seifert, I. Welch, P. Komisarczuk, Taxonomy of honeypots, <http://www.mcs.vuw.ac.nz/comp/Publications/CS-TR-06-12.abs.html>, Victoria University of Wellington, Technical Report, 2006, (accessed 18.11.2010).
[35] K. Wang, Using honeyclients to detect new attacks, <www.synacklabs.net/honeyclient/Wang-Honeyclients-RECON.pdf>, 2005 (accessed 18.11.2010).
[36] Y.-M. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, S. King, Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities, in: Proceedings of the 13th NDSS, San Diego, CA, Feb. 2006.
[37] C. Seifert, I. Welch, P. Komisarczuk, Honeyc - the low interaction client honeypot/honeyclient, https://projects.honeynet.org/honeyc (accessed 18.11.2010).
[38] J. Nazario, Phoneyc: A virtual client honeypot, in: Proceeding of the 2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET '09), 2009.
[39] A.J. Trivedi, P.Q. Judge, S. Krasser, Analyzing network and content characteristics of spim using honeypots, in: Proceedings of the 3rd USENIX SRUTI, Santa Clara, CA, 2007.
[40] G. Yan, Z. Xiao, S. Eidenbenz, Catching instant messaging worms with change-point detection techniques, in: Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats (LEET), San Francisco, CA, 2008.
[41] Autoit, http://www.autoitscript.com/, (accessed 18.11.2010).
[42] J. Newsome, D. Song, Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software, in: Proceedings of the 12th NDSS, San Diego, CA, 2005.
[43] G. Portokalidis, A. Slowinska, H. Bos, Argos: an emulator for fingerprinting zeroday attacks, in: Proceedings of the EUROSYS 2006, Leuven, Belgium, 2006.
[44] International Secure Systems Lab, Anubis: Analyzing unknown binaries, http://analysis.iseclab.org/ (accessed 25.11.2010).
[45] Virustotal, <http://www.virustotal.com/> (accessed 25.11.2010).
[46] Netfilter, iptables project, <http://www.netfilter.org/projects/iptables/> (accessed 18.11.2010).
[47] C.C. Zou, D. Towsley, W. Gong, Modeling and simulation study of the propagation and defense of internet email worm, IEEE Transactions on Dependable and Secure Computing 4 (2) (2007) 105–118.
[48] T. Bu, D. Towsley, On distinguishing between internet power law topology generators, in: Proceedings of the 2002 IEEE INFOCOM, New York, NY, 2002, pp. 638–647.
[49] Sophos, W32/Jitux-A, <http://www.sophos.com/security/analyses/viruses-and-spyware/w32jituxa.html> (accessed 18.11.2010).
[50] Sophos, W32/Kelvir-F, <http://www.sophos.com/security/analyses/viruses-and-spyware/w32kelvirf.html> (accessed 18.11.2010).
[51] Sophos, Troj/Kelvir-M, <http://www.sophos.com/security/analyses/viruses-and-spyware/trojkelvirm.html> (accessed 18.11.2010).
[52] Sophos, W32/Kelvir-Q, <http://www.sophos.com/security/analyses/viruses-and-spyware/w32kelvirq.html> (accessed 18.11.2010).

Mengjun Xie received the Ph.D. degree in Computer Science from College of William and Mary, Williamsburg, in 2009. He is an Assistant Professor of Computer Science at University of Arkansas at Little Rock, Little Rock. His research interests include network security, information security, network systems, and operating systems.

Zhenyu Wu received his M.Sc. degree in Computer Science from the College of William and Mary in 2005. He is currently a Ph.D. candidate in Computer Science at the College of William and Mary. His current research area focuses on data center resource management and network optimization. His research interest also lies in system and network security, including but not limited to malware analysis, packet filters, and Internet chat and online game security.

Haining Wang received his Ph.D. in Computer Science and Engineering from the University of Michigan at Ann Arbor in 2003. He is an Associate Professor of Computer Science at the College of William and Mary, Williamsburg, VA. His research interests lie in the area of security, networking systems, and distributed computing. He is a senior member of IEEE.