Real-Time Emulation of Intrusion Victim in HoneyFarm

Xing-Yun He[1], Kwok-Yan Lam[1], Siu-Leung Chung[2], Chi-Hung Chi[3], and Jia-Guang Sun[1]

[1] School of Software, Tsinghua University, Beijing, P.R. China
{hexy02,lamky,sunjg}@tsinghua.edu.cn
[2] School of Business Administration, The Open University of Hong Kong
slchung@ouhk.edu.hk
[3] School of Computing, National University of Singapore
chich@comp.nus.edu.sg

Abstract. Security becomes increasingly important. However, existing security tools, almost all defensive, have many vulnerabilities which are hard to overcome because of the lack of information about hackers techniques or powerful tools to distinguish malicious traffic from the huge volume of production traffic. Although honeypots mainly aim at collecting information about hackers' behaviors, they are not very effective in that honeypot implementers tend to block or limit hackers' outbound connections to avoid harming non-honeypot systems, thus making honeypots easy to be fingerprinted. Additionally, the main concern is that if hackers were allowed outbound connections, they may attack the actual servers thus the honeypot could become a facilitator of the hacking crime. In this paper we present a new method to real-time emulate intrusion victims in a honeyfarm. When hackers request outbound connections, they are redirected to the intrusion victims which emulate the real targets. This method provides hackers with a less suspicious environment and reduces the risk of harming other systems.

Keywords: Honeypot, intrusion, interception proxy, reverse firewall

1 Introduction

Defensive security tools, such as firewall and intrusion detect system (IDS) may have difficulties in protecting organizations from intrusion. These tools only passively wait for hackers and a large volume of data is collected by firewall or IDS daily, they may fail to distinguish accurately malicious activities from production activities however. Also, we cannot observe the whole process of real attacks because hackers have attacked the target servers and run away before we see the logged data.

In order to overcome the vulnerabilities of existing security tools and secure our system, knowledge of the hackers' behavior is important. Hackers' behavior includes what hackers do before attacking, in the process of attacking and after attacking in cyberspace, such as actions which the hackers perform in computer

C.-H. Chi and K.-Y. Lam (Eds.): AWCC 2004, LNCS 3309, pp. 143–154, 2004.

systems or networks, tools which the hackers use to attack others, targets which the hackers want to find, motivations that cause hackers to attack, tactics that the hackers take in order to compromise one system and avoid being prosecuted.

Understanding hackers' behavior requires the design of special tools. A honeypot, especially high-interaction honeypot, is an adaptive tool. Honeypots mainly aim at collecting information about hackers' behavior by providing an emulated environment that lures the hacker to attack. Honeypots and the data collected by honeypots can be of great value to improve overall security of the system.

However, nowadays nearly all honeypot designs limit hackers' outbound connections, many simple honeypots do not even allow any outbound connection. If hackers manage to penetrate into honeypots, they will soon realize that they can do nothing further. For example, hackers typically need to download tools from the Internet. If the honeypots do not allow them to do so, then they cannot proceed further and will run away. Thus, it is difficult to collect detailed information about their hacking techniques. Even if hackers are allowed outbound connections, these connections are also limited by the honeypot, and thus such honeypot is easy to be fingerprinted. If a honeypot is fingerprinted and hackers will bypass it, then the honeypot is of no security value. Furthermore, the risk of harming other systems exists if we permit hackers' connections to actual targets when the hackers compromise honeypots and want to connect to outside.

In this paper, we present a new honeypot design that can lure all levels of hackers (including high-skilled hackers) to do what they want, and at the same time largely reduce the risk of harming other systems. The proposed design is based on real-time emulation of intrusion victims in a honeyfarm. For example, if a hacker requests a connection to target website, we can provide an intrusion victim environment to simulate this website. The emulated intrusion victim looks like the actual target website, and thus hackers can hardly fingerprint it. To achieve this emulation, our method uses an interception proxy and a reverse firewall to control hackers' outbound connections and together with the website copying tools to real-time emulate intrusion victims. The rest of the paper is organized as follows. Section 2 introduces some basic concepts of honeypot. Our new design about real-time emulating intrusion victims in a honeyfarm is described in Section 3. Section 4 describes other issues pertinent to the real-time emulation of intrusion victims. Section 5 presents the analysis and evaluation of our method. We conclude the paper in Section 6 by summarizing the paper and pointing out further research directions.

2 Overview of Honeypot

Honeypot is an information system resource whose value lies in unauthorized or illicit use of that resource [1]. There is no production value in honeypots, so any connection to honeypots will be regarded as malicious activities. If the honeypots are never probed or attacked, then it has little or no value. Honeypots mainly aim at capturing hackers' behavior and can help other security tools by collecting data about hackers.

Honeypots have immense advantages. Firstly, usually data collected by honeypots is of higher value but in smaller amount. One of the challenges that the

security community faces is to obtain valuable information from a large number of data. Organizations collect vast amounts of data every day, including firewall logs, system logs, and intrusion detection alerts. The sheer amount of data can be overwhelming, and makes it extremely difficult to derive any value from the data. Honeypots, on the other hand, collect very little data, but what they do collect is normally of high value. The honeypot concept of no expected production activity dramatically reduces the noise level. Instead of logging gigabytes of data every day, most honeypots collect several megabytes of data per day. Any logged data is most likely scan, probe, or attack information of high value. Secondly, honeypots can be used to study new attacks. New tools and methods come into being in hackers community at regular intervals. Traditional defensive security tools are almost based on rule, such as IDS (based on misuse detection) and firewall. They generally only detect or prevent known types of attack. Although IDS based on anomaly detection can find new attacks, it spends too much time building the normal behavior profile, and may have high false alarm rate. For honeypots, however, any traffic to and from them is suspicious and is all logged in detail. Thus we can analyze logs to find out the unknown attacks. Thirdly, honeypots can deal with hackers' encrypted packets. For example, honeypots can use kernel-based rootkits to capture the data in honeypot's kernel. There have been some tools available, such as SEBEK [3].

Honeypots do have disadvantages. For example, honeypots may give hosting organization more risks. All security technologies have certain level of risks. Firewalls have the risk of being penetrated, encryption has the risk of being broken, and IDS sensors have the risk of failing to detect attacks. Honeypots are no exception. Specifically, honeypots have the risk of being taken over and being used to harm other systems by the hackers.

According to the level of interaction afforded to attackers, honeypots can be classified as low-interaction honeypots and high-interaction honeypots [1].

Low-interaction honeypots [2] simply emulate a few services and are easy to install. A hacker is limited to interacting with these pre-designed services (such as fake ftp or http service). Thus low-interaction honeypots have a low level of risk, but only capture known behaviors and do not do well in interacting with or discovering unknown or unexpected behaviors or attacks. E.g. BOF, specter and honeyd [2] are low-interaction honeypots.

High-interaction honeypots [2], however, are systems with full-blown operating systems and applications. There is also a far greater level of risk compared with low-interaction honeypots. They can get a large amount of information about hackers and give insight into hackers and be proficient at discovering unknown attacks, thus being primarily used for research purposes. But high-interaction honeypots are extremely difficult and time-consuming to install and configure. Honeynets such as GEN I and GEN II [2] belong to the category of high-interaction honeypots.

From the view of users, honeypots can be categorized into production honeypots and research honeypots. When used for production purposes, honeypots can protect an organization or help mitigate risk. For example, helping orga-

nizations respond to an attack. Low-interaction honeypots are often used for production purposes. When used for research purposes, honeypots are used to collect information about hackers. High-interaction honeypots are often used for research purposes.

A honeyfarm [4, 5] (or a honeypot farm), a group of many honeypots deployed together in a single consolidated location. Hackers are redirected to the honeyfarm, regardless of what network they are on or probing. A redirector is needed to transport a hacker's probes to a honeypot within the honeyfarm, without the hacker ever knowing it. The hacker thinks he is interacting with a target victim, when in reality he has been transported to the honeyfarm. We can build and deploy a single centralized honeyfarm to simplify the deployment of distributed honeypots, instead of having individuals all over the world build, customize, deploy, and maintain a separate honeypot for every network (keeping in mind many organizations have thousands of networks). Updating and administering honeypots, especially high-interaction honeypots, also become far easier.

Honeypot is a new concept, and therefore research on emulating intrusion victims is also very little. At present, the systems emulated or run on honeypots do not aim at specific websites or hosts. That is, they emulate the common characters of many websites or hosts, but not a specific website or host.

If honeypots are compromised, hackers may connect to other networks from the honeypots. Until now, tactics of dealing with outbound connections has always been to block or limit them. Alternatively, honeypots may let hackers alone regardless of the risk. Both of these tactics are undesirable as they either cause honeypots to be figured out or put systems at excessive risk.

GEN I honeynets [2] take two available methods to limit outbound connections. One is to block whatever a hacker requests. The other is to set a limited number of outbound connections. When hackers' connections reach the limit, further connections will be blocked. It is obvious that experienced hackers can fingerprint them as "honeypot" by trying more connections. GEN II honeynets [2] modify or throttle outbound connections to non-honeynet systems. Likewise, if hackers compare their packets flowing through honeynet with the original, they will find the difference.

3 The New Design

The objective of the design is to give hackers a less suspicious environment just like the real targets for hackers to intrude, and reduce the risk of harming non-honeypot systems from compromised honeypots. An intrusion victim is the victim service or system or device intruded by hackers. Here, we real-time emulate the intrusion victim (the hackers' target) in a honeyfarm, and redirect hackers to the intrusion victim.

There are three requirements to be satisfied in order to real-time emulate intrusion victims in a honeyfarm.

- Real-time: the latency between hacker's request and the response to the hacker should be small enough to lure highly skilled hackers.

- Emulation: the emulated victim must be less suspicious, or at least looks like the real target server, thus is more difficult to fingerprint.
- Control of outbound connections of hackers: it includes transparent redirection and access restriction so as to prevent harming other systems.

After a hacker is attracted to a honeypot, the hacker may look for interesting data or further targets. In most situations, the hacker most likely requests outbound connections from the honeypot. This is what we want to deal with.

When a hacker requests the outbound connection from honeypot for the first time, we allow the connection in order to learn the hacker's target and then emulate the target victim. But access restriction must be imposed on the connection, such as limiting network bandwidth, thus making the hacker feel the target host or network is slow, rather than of suspecting the honeypot. The main advantage of access restriction is to slow down hacker's connection in order to earn time to emulate the intrusion victim and reduce the risk of harming non-honeypot systems. While we allow hacker's outbound connection, at the same time, we also set up the website copying tools to duplicate the target web pages at the local honeyfarm in order to real-time emulate the intrusion victim. When the hacker wants to connect to outside again, he is likely to request the next layer of the web page, so we can immediately redirect the hacker to the fake victim. Additionally, the access restriction does not totally block the hacker' request, but slow it down, so the hacker may not suspect the honeypot, and may think the target server is simply slow or overloaded. Even if he is impatient for the low traffic speed and runs away, he may still come back again in future. Thus we can also redirect the hacker to the emulated intrusion victim.

After intrusion victims are emulated in our honeyfarm, if hackers request outbound connections, we check whether their targets have been emulated in the honeyfarm. If the target servers have been emulated, we immediately redirect hackers to the corresponding intrusion victims. Otherwise, by following the same steps, the new intrusion victim is emulated.

Architecture of real-time emulating intrusion victims in a honeyfarm is illustrated in Figure 1. In this model, the interception proxy, including interceptor and cache, plays an important part in redirecting hackers' requests to the emulated intrusion victim in our honeyfarm and control the outbound requests together with a reverse firewall. A reverse firewall blocks known attacks and sets access restriction to outbound connections.

Honeyfarm is positioned close to the compromised honeypot. Thus if the target victim is accessed by the hacker, the response to the hacker's request will be very fast. In addition, today we have many existing website copying tools available, and the process of emulation is very simple, and the emulated intrusion victim can be made identical to the target server, at least on the surface, hence hackers may feel they are in actual targets. Finally, we select an interception proxy and a reverse firewall to help us control hackers' outbound connections.

An interception proxy [7–9], also known as transparent proxy, is located on the path of traffic. A proxy [6] is an application that sits somewhere between the client and the original server. It behaves like both a client and a server,

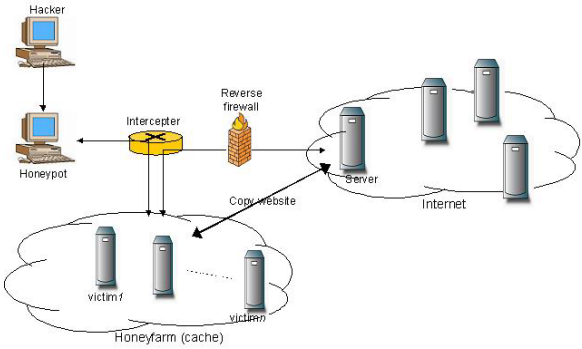

Fig. 1. Architecture of honeyfarm using interception proxy

that is, it acts like a server to clients, and like a client to servers. It receives and processes requests from clients, and then forwards those requests to original servers. A caching proxy [6, 8–11] is a proxy with a cache, and it is often referred to as "proxy cache" or simply "cache". It can alter the path of packets flowing through the network and split a web request into two separate TCP connections, one to the client and the other to the server. Transparent deployment [7] of a proxy relies on some network element (a switch or a router) to intercept all traffic from web clients to web servers and divert it to a cache instead of its actual destination. The cache pretends to be the original server. When the cache sends packets back to the client, the resource IP address is that of the original server. This tricks the client into thinking it's connected to the original server. For this reason, proxies deployed in this manner are called interception proxies. The network elements that intercept and divert packets to interception proxies are called intercepting elements or simply interceptors.

Since the proxy forwards requests to original servers, it hides the client's network address. The key feature of a caching proxy is its ability to store responses from the original servers for later use. When the requested data is found in the cache, we call it a hit. Similarly, referenced data that is not cached is known as a miss. The performance improvement that a cache provides is based mostly on the difference in service times for cache hits compared to misses. The percentage of all requests that are hits is called the hit ratio. Also, any system that utilizes caching must have mechanisms for maintaining cache consistency. This is the process by which cached copies are kept up-to-date with the originals. However, one of the most difficult aspects of operating a caching proxy is getting clients to use the service. For example, users might not configure their browsers correctly, and even they can disable the caching proxy.

Unlike a regular proxy, an interception proxy requires no browser configuration. They are transparent to the users, and most users are not even aware that their requests are going through or perhaps even served by an interception proxy. Even when the client has the IP address of the original servers it should contact, it might never reach it. Along the network path between the client and

the server, there might be a network switch or router that directs all Web requests transparently to an interception proxy cache [8]. Thus administrators have greater control over the traffic sent to each cache, and users have no choice but to use interception proxy. However, the main limitation of interception proxies is that they can only work properly if all packets from a given client to a given destination flow through the same intercepting element [7]. In our method, you will see that this limitation becomes an advantage to deal with hackers.

In our design, we implement the honeyfarm as the cache in the interception proxy. When a hacker has compromised a honeypot and wants to connect to outside, all the hacker's requests must pass through an interceptor. The task of an interceptor is to divert hackers' requests to honeyfarm, regardless of HTTP or FTP requests. The interception proxy checks whether hacker's target website has been emulated in the honeyfarm (or cache). If not finding the fake target, that is, cache (honeyfarm) miss, the interception proxy will transmit the hacker's request to the original target server from the honeyfarm, as shown in Figure 2. Here only one honeypot is shown for simplifying, and in fact, all honeypots in an organization must connect to Internet through the interception proxy.

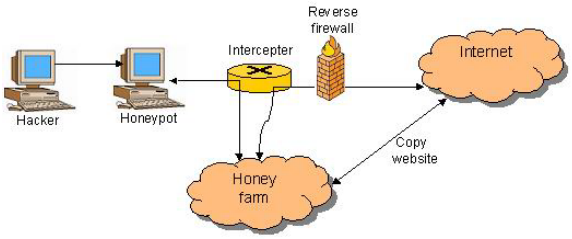

Fig. 2. Cache (or honeyfarm) miss using interception proxy

A reverse firewall is set in order to control the connection further. The reverse firewall filters the request and blocks a known attack, such as Denial of Service (DOS), Distributed Denial of Service (DDOS). Access restriction to the request will be set by the reverse firewall. For example, limiting the outbound bandwidth to a certain low value, like several bytes per second or tens of bytes per second, so that the hacker cannot do harm to other systems or fingerprint the honeypot.

The real response from the hacker's target is also redirected to the honeyfarm by the interceptor, and here the interception proxy can check the contents of response, and if necessary, it replaces the critical contents to avoid the risk of exposing information.

While we impose restrictions on the hackers' outbound connections, at the same time, we copy quickly the target web pages or services to the honeyfarm in order to build the intrusion victim. There are several tools available to copy website. For example, teleport pro is for windows systems; and wget is for both linux and windows. From Figure 2, we can see that the process of copying website does not pass through the interceptor and the reverse firewall. The purpose

is to speed the copying of website to make the emulation of intrusion victim much faster, and make the reverse firewall deal with only hackers' requests thus simplifying the rules of the reverse firewall. However, the channel of copying websites must be protected. Additionally, we can set the number of layers of copying website for saving memory space because hackers generally only access very few web pages in a website. After the intrusion victim is completed, the interception proxy can redirect hackers to it. Unfortunately, if a hacker is impatient with the access restriction, he may run away. However, because the hacker may trust the honeypot and suspect the target is in question, he generally comes back to the honeypot again in the near future, and he has a high possibility to request the same target server.

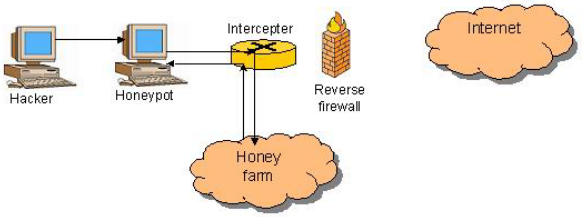

Fig. 3. Cache (or honeyfarm) hit using interception proxy

Otherwise, if the intrusion victim has been emulated, that is, cache (honeyfarm) hit, interceptor will immediately redirect the hacker to the corresponding intrusion victim in the honeyfarm, as shown in Figure 3. The response is sent back to the hacker from the emulated intrusion victim, but the source address of the response is that of the target server which the hacker requests. This tricks a hacker into thinking he is communicating with the actual target server.

4 Other Issues

In our method, there is a large difference between our interception proxy and a traditional interception proxy. It is the responses from original servers that the cache of a traditional proxy stores. However, the intrusion victim cannot be emulated, if we also wait for responses of the original server. So we must copy web pages much faster than hackers do, in order to real-time emulate the victim.

Besides, a traditional interception proxy generally only diverts HTTP requests. It assumes traffic from or to port 80 is HTTP. There is no need to distinguish HTTP from other traffic in our proxy. By the definition, all traffic from honeypot is malicious. What we need to do is only to divert all of them. Also, from the Figure 2, we can see that the interceptor will intercept all the responses from original servers to the honeyfarm (cache), and the responses will be sent to hackers from honeyfarm. Here, important content of responses can be identified replaced by useless data. It is important to reduce the risk of information exposure. In the process of real-time emulating intrusion victims, we

can take the same measure to avoid exposing information. If the contents of files or pages are vital or critical (e.g. financial reports and credit card numbers), we can modify or replace them by fake data.

We take the means of prefetching [9] in order to enhance the hit ratio further. At the beginning of emulation, there is no intrusion victim in our honeyfarm. When hackers request outbound connections, the hit ratio of intrusion victims is very low. However, at first we can collect some information that is usually visited by hackers. According to the information, we can copy web pages and emulate some intrusion victims in our honeyfarm before hackers come to our honeypots.

The web pages in the original servers may have been changed. Therefore, the web pages in the honeyfarm may be old or stale. We need to maintain our intrusion victims consistency or keep them up-to-date with the original servers [9]. The honeyfarm can send requests to original servers to check whether the web pages have been changed since the last copy. However, because hackers are not authorized, we have no obligation to keep the victims fresh. Under the condition that hackers cannot fingerprint the honeypots, we may postpone the refreshing of web pages. Consequently, in order to simplify the process of emulating and reduce the latency responding to hackers' request, the refresh time span can be set longer, such as one day, even one week.

Additionally, hackers may request new targets continuously, and then there will be overloaded in the honeyfarm, that is to say, the cache is full. The new intrusion victim cannot be simulated any more. Here, we can take the replacement technique [11] just like those in operation system, such as First-In First-Out algorithm (FIFO) and Least Recently Used algorithm (LRU). When the space of honeyfarm is not big enough, some emulated intrusion victims will be deleted, and replaced by the new intrusion victim.

In our design, there may be legal and copyright issues. When we download other websites to emulate intrusion victims, we should also think about copyright issues. This is less of a technical issue nevertheless.

5 Analysis of the Real-Time Emulation of Intrusion Victim

In our design, an interception proxy is deployed close to the content consumer (consumer-oriented) [9]. When hackers request outbound connections through our honeypots, it can quickly redirect all requests to the honeyfarm, regardless of HTTP or FTP. Figure 4 [12] shows the comparison of latency between caching, redirection, direct and proxy (There are detailed definitions about caching latency, redirection latency, direct latency, and proxy latency in [12]). If the requested contents are in the cache, the latency is lowest. When caches miss, it has to get the contents from the original servers. The latency includes checking caches, transmitting the request to the original target, storing and returning response, so it has the highest latency.

Here, redirection means it has to transmit hackers' requests to the original servers. So the latency of an interception proxy should be the weighted average

of cache and proxy latency. The hit ratio on caches can be up to furthest 50% [11] even if no other assistant techniques, such as profetching and replacement. The more the hit ratio, the less the latency. So, generally, with the reasonable hit ratio, we can make the latency lower. Moreover, by combining perfect caching and perfect prefetching between proxies and web servers, the proxy can at least reduce the client latency by 60% for high bandwidth clients [11]. In addition, the cache replacement technique can also enhance the hit ratio of interception proxy. So we conclude that the latency of interception proxy is also lower than that directing the requests to original servers.

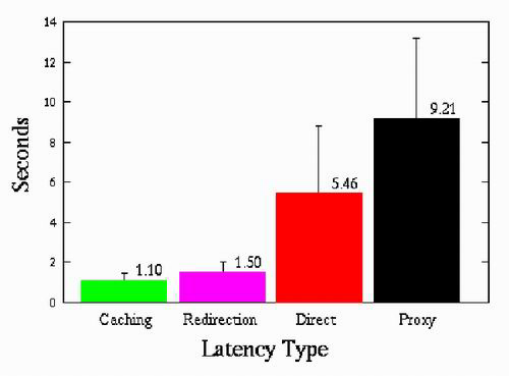

Fig. 4. The latency of caching, redirection, direct and proxy

If the hit ratio is higher than 46.2%, then the speed of real-time emulation is also quicker than that directing to original server. The following is the methods to enhance the hit ratio used in the paper.

- Prefetching: A central issue in prefetching is how to predict user future web-access patterns. There are many policies described in [10]. We take the prefetching technique between proxy and servers. Without prefetching, proxy caching with an unlimited cache size resulted in a 26% reduction in latency. However, with a basic prefetching strategy in place, that reduction in latency could be improved to 41%, and with a more sophisticated prefetching strategy, that number could be further improved to be 57% [13].
- Replacement: It is concerned with how to utilize the limited cache storage capacity to achieve the best caching performance. No known replacement policy can outperform others for all web access patterns [11], because the performance of replacement policies depends highly on the traffic characteristics of WWW accesses. Here, before knowing about hackers, we add only traditional replacement policies (E.g. LRU) to our interception proxy. Regardless of which policy, it may further improve the hit ratio of our honeyfarm.
- No delay of checking whether HTTP or FTP: All traffic from honeypots is malicious according to the definition of honeypot, so the interception proxy

can be configured to divert all of them. This is the difference between our interception proxy and traditional interception proxies, and such an implementation reduces the overload of interceptor. And the time wasted in distinguishing HTTP from others was saved.
- Website duplication: The special channel may improve the speed of responses to hackers' requests. If we used traditional interception proxy, there would be a challenge that we cannot real-time emulate intrusion victims, because the traditional interception proxy always wait for responses from original servers. The specific channel solves this problem. It not only fetches the currently requested web page, but also prefetches the next few layers of the web page. Since a hacker's request has a close correlation with the next request, thus the hit ratio also enhance, we may gain a much lower latency than the traditional interception proxy.

Compared with other honeypot designs, our method provides a more attractive environment for hackers. We duplicate target web pages to emulate hackers' destinations. It gives hackers a virtual websites or servers to intrude, in some sense, the environment is a replica, hackers can do everything just as in the actual one, but they cannot harm the targets. It not only real-time emulates a target server, but also real-time responds to a hacker's request.

6 Conclusion

At present, honeypots technology, especially high-interaction honeypots, has large risk of being taken over by hackers and being a facilitator to harm non-honeypot systems in that they allow hackers to connect to the actual target servers after they are compromised. In this paper, we have presented a new method to real-time emulate intrusion victims in a honeyfarm. When hackers compromise a honeypot and connect to Internet by the honeypot, they are secretly intercepted and redirected to a honeyfarm where intrusion victims were emulated.

This method provides hackers with an environment which is hard to be fingerprinted, and thus it may lure more advanced hackers compared with those honeypots which block or limit hackers' outbound connections or modify their packets. Also, when our honeyfarm responds to the hackers' requests, it replaces the source IP addresses of the response packets by the IP addresses of original servers, therefore further deceiving hackers to trust the emulated intrusion victims. More important, this design distracts the hackers' attention from the original servers, thus mitigating the risk of harming the real target websites or hosts. The information logged from the intrusion victims can be used to improve performance of existing security tools (E.g. IDS and firewall), and help us protect ourselves from attacks.

In future, we can design self-adapting emulation of intrusion victims. By analysis of the logs collected on existing intrusion victims, we may predict where hackers want to go next step. So we can emulate intrusion victims before the

hackers request outbound connections, and the latency of real-time emulation will be reduced further. In addition, there may be many honeypots in the world, or even in one organization. Thus some distributed honeyfarms may be required and the caching structure of honeyfarm needs to be improved, because too many honeypots will bring much more traffic than before, and the honeyfarm may become a "bottleneck". Finally, caching dynamic web objects is another part that should be thought about to add to our method.

Acknowledgement

This research was partly funded by the National 863 Plan (Projects Numbers: 2003AA148020), P. R. China and the Internet Security project of PrivyLink International Limited (Singapore).

References

1. Lance Spitzner. Honeypots Definitions and Value of Honeypots, http://www.tracking-hackers.com/, May 29, 2003.
2. Lance Spitzner. Honeypots: Tracking Hackers. Addison-Wesley. Boston. 2002.
3. Lance Spitzner. Know Your Enemy: Sebek2 A kernel based data capture tool, http://www.honeynet.org/, Sept 13, 2003.
4. Lance Spitzner. Hitting the Sweet Spot, Jul 2003.
5. http://www.phrack.org/fakes/p62/p62-0x07.txt
6. Duane Wessels. Web Caching, the O'REILLY press, Nov 2002
7. M. Rabinovich and O. Spatscheck. Web Caching and Replication, Chapter 8. Reading, MA: Addison Wesley, 2002.
8. Brian D.Davison, Rurgers. A Web Caching Primer, IEEE Internet Computing, vol. 5, pp. 38-45, Jul/Aug 2001.
9. Greg Barish, Katia Obraczka. World Wide Web Caching: Trends and Techniques, IEEE Communications Magazine Internet Technology Series, May 2000.
10. Daniel Zeng, Fei-Yue Wang, and Mingkuan Liu. Efficient Web Content Delivery Using Proxy Caching Techniques, IEEE Transactions on Systems, Man, And Cybernetics—Part C: Applications and Reviews, VOL. 34, NO. 3, Aug 2004.
11. Jia Wang. A Survey of Web Caching Schemes for the Internet, ACM Computer Communication Review, vol. 29, no. 5, pp. 36-46, Oct 1999.
12. Radhika Malpani, Jacob Lorch, and David Berger. Making World Wide Web Caching Servers Cooperate, Proceedings of the 4th International WWW Conference, Boston, MA, Dec 1995 (http://www.w3.org/Conferences/WWW4/Papers/59/)
13. Thomas M. Kroeger, Darrell D. E. Long, Jeffrey C. Mogul. Exploring the Bounds of Web Latency Reduction from Caching and Prefetching, Proceedings of the Symposium on Internet Technologies and Systems, 1997.