

# HoneySpam: Honeypots fighting spam at the source

Mauro Andreolini Alessandro Bulgarelli Michele Colajanni Francesca Mazzoni

*Dip. Ingegneria dell'Informazione  
Università di Modena e Reggio Emilia  
Modena, Italy, 41100*

{andreolini.mauro, bulgarelli.alessandro, colajanni, mazzoni.francesca}@unimo.it

<http://weblab.ing.unimo.it/people/>

## Abstract

In this paper, we present the design and implementation of HoneySpam, a fully operating framework that is based on honeypot technologies and is able to address the most common malicious spammer activities. The idea is that of limiting unwanted traffic by fighting spamming at the sources rather than at the receivers, as it is done by the large majority of present proposals and products. The features of HoneySpam include slowdown of the e-mail harvesting process, poisoning of e-mail databases through apparently working addresses, increased spammer traceability through the use of fake open proxies and open relays.

## 1 Introduction

Internet has become the preferred architecture for the most popular user-oriented services. As any popular resource, the Internet-based services are subject to an increasing level of malicious actions which may have bad consequences for both the users and the services providers. In this paper, we focus on the amount of unwanted traffic due to unsolicited e-mails [12, 3], which is widely known as *spam*. Spamming activities often turn into remarkable losses of user time and productivity, and a waste of user and service provider resources.

The most diffused proposals and products for fighting against spam are *receiver-oriented* in the sense that the mail server tends to receive all e-mails and then to determine through some filtering technique(s) whether an e-mail is valid or not (e.g., [19, 16, 7]). Similar techniques can be applied at the client level by the user (e.g., [18, 21, 11]) or can be integrated (e.g., [22]). Independently of the applied techniques that are becoming more sophisticated and valid, these approaches suffer of two main problems: the percentage of low positives is nowadays very low, but there are still false negatives [26, 17]; even worse, receiver-oriented approaches do not reduce the Internet traffic and do not limit the waste of network and disk resources of the service providers.

The alternative idea proposed in this paper is to

address the previous issues by working on the *spam sources* instead of the *spam destinations*. A first source-oriented solution can be achieved through filtering mechanisms, such as the use of mail server *black lists* [25]. The destination e-mail server checks whether the IP address of the source e-mail server is inserted into the black list and, in case, denies the receipt of messages. This is a brute force approach with limited effects because an increasing number of spammers use forged SMTP headers in order to hide the real sender. Another source-oriented approach aims to build *white lists* of friendly addresses from which the receipt of spam is highly unlikely; all other sources that are not included in the white lists are not accepted (e.g., [8]). It should be clear that, although more modern white list are dynamically built and self-adaptable, they are really effective just for specific user communities. Apart from these issues, black and white listing do not help in decreasing the amount of unwanted traffic through the network, but only at the destination.

We follow the idea that fighting against the spam sources cannot be done through one tool, because sending unsolicited e-mails is just the last step of a complex set of operations that are carried out by spammers (see Section 2). Hence, an adequate fight against spam sources requires a framework of different tools and, possibly, the cooperation among interested organizations. The proposal of this paper does not yet include cooperation, but it describes a framework (*HoneySpam*) that is built up of many components among which the pivot is represented by the honeypot technology. The traditional goal of honeypots is to trace the activity of an attacker, locate him through a *traceback* operation, recognize common operational patterns in attacks with the purpose of automatically detecting them in the future. HoneySpam faces the following spamming actions at the source:

- *e-mail harvesting*;
- *anonymous operations*.

E-mail harvesting that is, the collection of valid e-mail addresses through automated crawlers [2, 27], is fought

through emulated Web services. Two different damages are inflicted to crawlers: *crawler slowdown* and *e-mail database poisoning*. First, the crawlers are slowed down into endless loops of Web page retrievals, through the creation of Web pages with many hyperlinks. Second, databases used by spammers to collect the e-mail addresses are polluted. In particular, the pages generated by the Web service contain special, valid, dynamically created e-mail addresses that are handled by emulated SMTP services present in HoneySpam. As soon as the spammer starts to use these freshly collected addresses, it can be traced back and blacklisted.

Fraudulent activities such as spamming are often achieved anonymously, through the use of open proxy chains. HoneySpam provides fake open proxy and open relay services [12, 14] that log every activity and block spam-related traffic.

HoneySpam fights common spamming actions for one organization, although its real potential should be tested in a cooperative context, such as [10].

The rest of the paper is organized as following. Section 2 presents a brief overview of some common spamming activities. Section 3 describes the requirements of an anti-spam tool that emerge from a detailed study of spamming activities and presents the design and some implementation details of the HoneySpam architecture. Section 4 presents an overview of possible countermeasures that the spammers might carry out against HoneySpam and how they are handled. Section 5 discusses related work. Section 6 illustrates future research perspectives. Finally, Section 7 presents some conclusive remarks.

## 2 Spammer activities

In this section, we present a brief summary of the most common actions performed by a spammer.

**E-mail harvesting.** Figure 1 shows one of the first actions performed by spammers that is, *e-mail harvesting*. An automated browsing software (called *crawler*) issues a HTTP request to fetch a Web document (*page 1*). Once the HTTP response is retrieved from the Web server, the crawler parses its content, extracting links to e-mail addresses and to other Web pages. Links to Web pages are used to continue the crawling process (*link 1* to *link k*), while links to e-mail addresses (*address 1* to *address n*) are used to compose lists or (more usually) to populate databases of potential *customers*. E-mail harvesting is an important step, because a spammer has to build its list of victims before sending unsolicited messages to them. It should be pointed out that (thanks to existing collections of e-mail addresses, usually available on CDs or databases), e-mail harvesting is not performed by every spammer, since long lists of (presumably valid) e-mail addresses can be bought from the Internet. Although

harvesting actions have softened recently, they still remain one of the major sources of e-mail addresses for spammers [13].

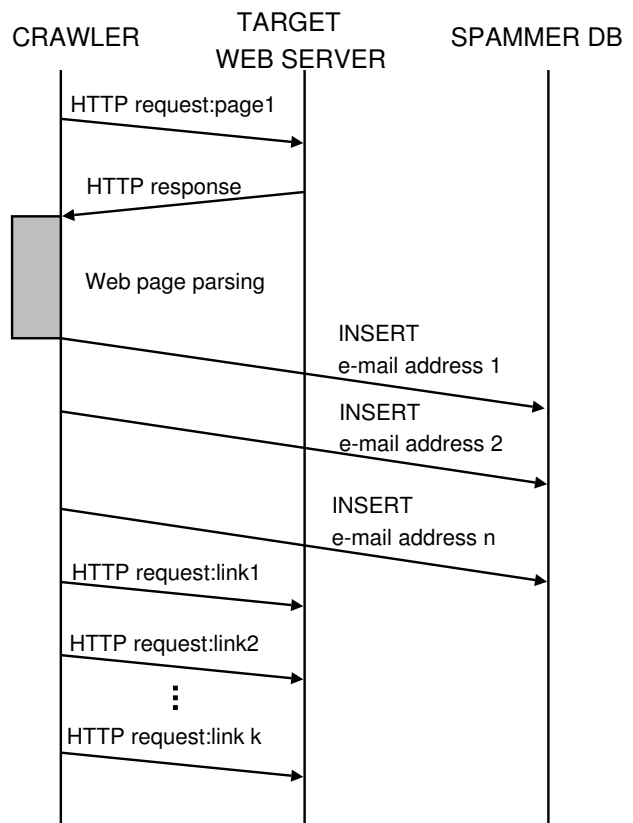


Figure 1: E-mail harvesting

**Operating anonymously.** Spamming activities are illegal in many (but not every) countries, thus *anonymity* is one of the most important goals pursued by a spammer. Figure 2 shows the actions undertaken by a spammer to gain anonymity when sending e-mails.

A common technique to hide traces of malicious activities is the use of an *open relay*, that is an SMTP server which does not need authentication to forward e-mail messages. The open relay is contacted by the spammer, receives the e-mail message, and forwards it to the destination SMTP server. This path is detailed through cross-hatch arrows in Figure 2. In theory, a single open relay is sufficient to provide anonymity because every SMTP request appears to be coming from the open relay IP address. However, if a spammer contacts an open relay that is configured to log any activity, the IP address of the spamming machine gets stored in the open relay log files. Using a single open relay is considered a “risky” operation for the spammer.

For this reason, spammers often use one or more *open proxies* (which form a *proxy chain*) to hide their activ-

ities. An open proxy is an intermediary node that forwards traffic between a client and a server without the need of authentication. As shown by straight-arrow path in Figure 2, the spammer establishes a TCP connection with the first open proxy (*open proxy 1*). Next, the spammer uses this connection to establish a new proxy connection with another open proxy (e.g., *open proxy 2*) through the *CONNECT* method. In this way, a chain of proxy connections is built, until *open proxy n*. Finally, the spammer uses the proxy chain to send an e-mail message to an open relay, which is forwarded to the destination SMTP server.

The use of open proxy chains makes it much more difficult to trace spammers back, because, even if the logs of every open proxy are available, the whole path of requests has to be reconstructed. The problem is augmented because public lists of open proxies can be easily found online, for instance at [23].

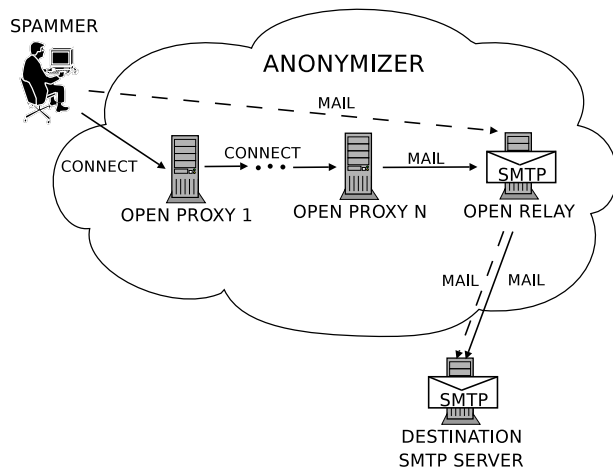


Figure 2: Sending spam anonymously

### 3 Architecture of HoneySpam

Once the principal activities of a spammer are detailed, in this section we outline the requirements that a system fighting against spam sources has to satisfy. Next, we propose the architectural design and some implementation details of HoneySpam.

#### 3.1 Requirements of an anti-spam system

To effectively fight against spam sources, it is necessary to contrast the operations described in Section 2. In particular, we identify the following three goals.

**Reduce the efficiency of crawlers.** Spammers rely on up-to-date lists of valid e-mail addresses in order to reach as many recipients as possible. Thus, it is crucial to reduce the efficiency of the e-mail harvesting process. This goal can be obtained in several ways, as shown be-

low.

A first step to reduce the effectiveness of crawlers consists in the possibility of forcing them into a (possibly endless) loop of Web page retrievals. In particular, if each Web page in the loop does not contain any parsable e-mail address, the final goal of the crawler (that is, collecting valid e-mail addresses) is defeated.

A further damage that can be inflicted to crawlers is the possibility of populating the Web pages with invalid addresses (*poisoning*). These are parsed by the crawlers and are usually fed to the spammer databases. The end result is a database that contains many invalid entries (*polluted database*). The drawback of possibly increased network traffic due to the spammer attempts to contact all those fake addresses can be used to better trace spam sources.

Web pages are browsed not only by malicious crawlers, but also by normal ones (e.g., *Googlebot*). To permit legitimate crawling, the *robot exclusion protocol* [5] has to be implemented at the Web server. It tells crawlers which portions of the site are available to indexing purposes and which are not. The vast majority of legitimate crawlers complies to this protocol and is not involved in the Web page retrieval loop.

**Identify spammers.** To identify spammers, it is necessary to encourage them to use honeypot services to their advantages. This is done through the deployment of fake servers, such as open proxies and open relays.

Letting spammers use honeypot services is not enough. To ensure traceability of their actions, *logging* must be enabled for every deployed service (open proxy, open relay, Web server).

A further measure that helps in the identification is the use of valid e-mail addresses in the Web pages returned by the fake Web server. These addresses are handled by a honeypot SMTP server, which logs every activity and every message. This information is parsed to obtain valuable information about the spam sources.

**Block spam e-mails** To reduce the end effects of spam that is, the deliver of unsolicited messages, the associated traffic has to be blocked. A good anti-spam tool must not block valid e-mail messages (*false positives*) and should pass the least amount of unsolicited messages (*false negatives*).

#### 3.2 Architecture

Figure 3 shows the architecture of HoneySpam. The framework is placed into the DMZ area of an enterprise network. HoneySpam includes several emulated components, each one dedicated to fight one specific aspect of spamming. We distinguish the following components: *fake Web servers* (to allow e-mail harvesting), *fake open proxies*, *fake open relays* (to provide spammers with services that can be used to their advantage), *destination*

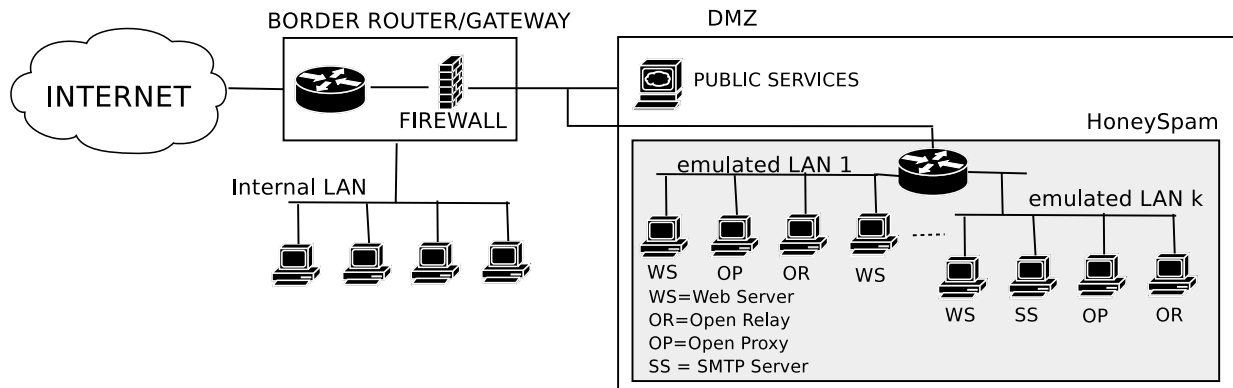


Figure 3: Architecture of HoneySpam

SMTP servers (to increase spammers traceability). Service emulation is obtained through honeypots. In the following we detail the design choices and some implementation details regarding honeypots and each considered component.

### 3.3 Honeypots

The first design choice concerns the deployment of honeypots. Many alternatives exist.

A possibility is to use real software on real hardware, for example an instance of an Apache Web server running on a node. This solution, though feasible, is definitely not advisable, since software typically has security bugs, which could be used to penetrate the host in order to obtain valuable information, to install “unwanted” services, even to obtain a privilege escalation. Besides making a service unavailable, a compromised host could also be used to attack third-party nodes. In this case, it is often necessary to install both the operating system and the applications from scratch.

Another possibility is to use real software on emulated hardware, for instance by using User-Mode Linux (UML), as explained in [24]. UML is an open source virtual machine. It allows to run multiple instances of Linux, as a user process, on the same system at the same time. Real services are executed on emulated hosts. Using emulated hardware instead of real hardware is a better approach, because if the host is compromised, it is only necessary to replace the copy of the emulated host and to restart it to obtain the previous situation. Still there is the possibility to use the security holes of the UML software to acquire superuser privileges and to drive attacks to third party hosts.

Finally, it is possible to run emulated software on both real and emulated hardware. Depending on the amount of the available system nodes, emulated services are executed on real or emulated hardware. In this way no security holes can be exploited, simply because there is no real service running, only an emulation.

This possibility seems the most promising for implementing honeypots, because it is the most immune solution from exploits, which must be avoided at all costs. The previous considerations motivate the implementation of the honeypots of our framework by using emulated software on both real and emulated hardware. We use the `honeypd` tool [14], which is a small daemon that creates virtual hosts on a network. The hosts can be configured to run arbitrary services (e.g., an Apache Web server, a sendmail SMTP server), on predefined operating systems (Windows 2000 Pro, GNU/Linux with a 2.4.x kernel). These services are implemented through Perl scripts.

### 3.4 System components

**Fake Web server.** To protect against harvesting we deploy fake Web sites, whose goal is to slow down the process and to pollute the spammer databases. The idea is to provide a (potentially endless) sequence of links, thus catching the crawlers in a very long process. The pages of HoneySpam Web servers provide “special” e-mail addresses that, if used by the spammer, help trace him back. Furthermore, each action is logged in order to understand who is accessing the Web servers.

The *fake Web servers* generate dynamic pages containing many links and e-mail addresses which cannot be identified as fake by common spammer tools [15]. This idea resembles the behavior of Wpoison [6], with some improvements. The number of links, as well as that of the addresses is randomly chosen in a given range (for instance there is a minimum of 2 and a maximum of 20 links per document). We also provide many text files to be randomly used as the body of the html document. Random generation of Web pages greatly reduces determinism and, consequently, the risk of being identified as a honeypot. The e-mail addresses are not randomly chosen, because otherwise one cannot be sure to create a valid address. Instead, the user names are created as a composition of as much information as pos-

sible about the crawler from the Web server it is visiting. We collect the source IP address as well as the timestamp of the request and create a user name of the form `crawler-IP_timestamp`. The domain is randomly chosen among the list of domains belonging to our system. This approach is useful in two ways. First of all these addresses cannot be recognized as fake by software like e-mail Verifier [15], whose goal is to clean polluted e-mail databases. Next, spammers send e-mails to many of our honeypots, thus increasing the effectiveness of traceability.

An interesting aspect concerns the creation of fake links. These are randomly distributed among different Web servers. At each Web server the same script is executed, regardless of the requested resource, except for the requests directed to `robots.txt` to implement the robot exclusion protocol.

**Fake open proxies and relays.** In order to destroy the anonymity of the spammer, we provide virtual open proxies and open relays, whose main goal is to intercept illegal traffic operated by spammers. Connections to our open proxy and open relay servers are logged, to understand where the unwanted traffic is coming from. Furthermore, the traffic is blocked, thus reducing the number of unwanted e-mails reaching target users.

A connection that reaches an open proxy is usually generated by a spammer or another open proxy, in both cases enemy entities. Consequently, the corresponding source IP address is logged (and, optionally, inserted into a black list). We also log the IP address of the following node, since usually it is another open proxy or an open relay [12]. This allows HoneySpam to catch two enemy entities at a time. A connection that reaches an open relay is almost always generated by an open proxy, mainly for anonymity reasons. In this case, the corresponding source IP address is logged (and, possibly, blacklisted).

The *fake open proxies* emulate a subset of the HTTP protocol. Requests made with methods other than GET and CONNECT are answered with an error message. GET requests are answered with a randomly generated page. CONNECT requests to port 25 are internally redirected to an emulated open relay. The motivation behind this redirection is that the spammer may think nothing went wrong and he is connected to the SMTP server he requested, while he actually is connected to one of our honeypots. CONNECT requests to ports other than 25 are served with a “Request Timeout” message.

The *fake open relays* emulate a `sendmail` SMTP server. All the main commands of the SMTP protocol have been implemented, so that no one could notice the difference with a real server. When an e-mail is sent through the open relay, it actually does not reach destination, since all messages are logged but not forwarded,

except the very first one. This is done in order to fool a spammer who sends a first probe message to himself to see if the service is properly running.

**Fake destination SMTP server.** The destination SMTP server is specific to the domains protected by HoneySpam. If the spammer has harvested e-mail addresses from Web servers present in HoneySpam, these addresses will correspond to mailboxes in HoneySpam’s destination SMTP servers. The idea is to redirect all the traffic coming to any of the special address to a single mailbox, in which all the messages are logged and stored for analysis and backtracking purposes. The *fake destination SMTP server* is implemented similarly to the open relay; as a matter of fact, the set of SMTP commands is the same. This time the server is configured not to send any message, if requested to.

## 4 Possible countermeasures to HoneySpam

In this section we present a brief survey of malicious activities that can be directed to HoneySpam and related responses.

**Honeypot identification.** One of the main problems when using honeypots is the possibility to be discovered. Attackers use *footprinting* techniques [4, 9] in order to understand whether the services they are using are real or emulated. The `honeypot` framework is extremely useful to this purpose, since it is capable of returning different messages based on which operating system and services it is emulating. We tested its capabilities of emulating network topologies with many tools, including `ping`, `hping`, `traceroute`. All IP addresses emulated by the system are reachable through these tools, thus we can say `honeypot` is effectively able to emulate network topologies. Furthermore, we tested the capability of emulating in a proper way the features of various operating systems (Windows 2000 Professional, and Linux with a 2.4.7 kernel, among others) and of the services (a `sendmail` SMTP server, an Apache Web server). Network scanners such as `nmap` and `amap` have been fooled by `honeypot`. Indeed, they were not able to recognize that the operating systems and the services were emulated and not real. Thus, we can conclude that, with the tools usually adopted by spammers, it is rather difficult to recognize our servers as honeypots.

Besides white hats, spammers and attackers also maintain their blacklists (in this case, of honeypots); they are called *reverse blacklists*. In the actual implementation, HoneySpam is not able to address this problem, because each emulated server uses only one IP address. Each blacklisted server is no longer available for honeypotting. A possible improvement, which does not solve this problem entirely, is to use as many IP addresses as possible. An authoritative DNS server will

map entire IP ranges into one hostname in a random way. Thus, if one of the IP addresses of a host is blacklisted, the host will still be reachable thanks to the other IP addresses it is related to.

**Intrusion.** Another possible malicious action is to penetrate the host in order to get valuable information, to install “unwanted” services or even to attack a third party. This is usually accomplished by using a software or protocol vulnerability. For instance, if the attacker gets to understand that the servers are running `honeyd`, he could take advantage of its security holes to penetrate one of HoneySpam’s hosts. Removing all security vulnerabilities in `honeyd` is difficult, but these risks may be kept limited through simple actions. For example, `honeyd` has to be run with superuser privileges in order to let it manage networking; this is quite dangerous. If, on the other hand, `honeyd` runs in a restricted environment (such as a *chroot* jail), the privileges obtained through an exploit are limited only to the restricted area. As a consequence, the attacker is prevented from having access to sensible information such as system configuration files. To improve protection against intrusion attacks, HoneySpam can be integrated with a file alteration monitoring service, such as [28]. Its main purpose is to check for the creation of new files, which is one of the most common steps in installing new unwanted software. Changing or just reading existing files should also be cared, with the exception of the log files, because they are constantly changed by HoneySpam.

## 5 Related Work

The idea of using honeypots to fight spam is not quite novel. There are many works in literature describing how to use honeypots to detect suspicious traffic in order to understand how black hats behave, which include (but are not limited to) spam traffic detection.

A recent work describing the main steps followed by spammers in sending e-mails is presented in [12]. The author also explains how it is possible to use honeypots to fight the various spammer activities, for instance by deploying fake Web servers in order to pollute the spammers databases, after the e-mail harvesting process, or by deploying fake open proxies and relays in order to trace the spammer back. The author explains how to use real Web servers and how to turn them into honeypots.

In [14] Provos suggests to implement fake open proxies and relays through the `honeyd` framework, in an emulated network environment. HoneySpam not only implements all the suggestions of this paper, but also adds an anti-harvesting mechanism that poisons the spammer databases. The goal is to drive spammer activity to other honeypots of the HoneySpam framework.

In [10] a mobile honeypot mechanism is proposed, that allows unwanted traffic to be detected significantly

close to the origin of spam. This is obtained through a strict cooperation among Autonomous Systems (ASs). The main goal of Mohonk is to detect and possibly block the routing of messages through the use of dark address spaces. HoneySpam tries to reach the same goal, that is to block spam close to its source.

Real software can also be used to provide honeypots. As an example, a description on how to use *sendmail* as a honeypot SMTP server is presented in [20]. In [1], an open proxy honeypot (Proxypot) is deployed through an Apache Web server compiled with additional security modules. Wpoison [6] is a tool that attempts to solve the e-mail harvesting problem. It is a CGI script that generates randomized invalid addresses and pseudo-hyperlinks. The goal is to pollute the spammers databases and to capture the crawlers in a possibly endless loop. There are two main differences with the approach of HoneySpam: the e-mail addresses are bogus, thus preventing the interaction with further honeypot facilities, such as HoneySpam’s destination SMTP servers. Furthermore, being it a CGI script, this tool has to be run on a real Web server, which is a security treat.

## 6 Future work

Though it is far from being perfect, HoneySpam can be extended to improve its effectiveness in fighting spam sources. There are two possible directions to look for improvement.

**Scalability and fault-tolerance** Given the increasing amount of attacks, a single honeypot is more likely to become the bottleneck of the entire framework. Honeypot overloads have to be definitively avoided, since an overloaded host loses appreciation to spammers, and its attracting potential decreases. A future direction towards scalability consists in the replication of the honeypots, both at the local and at the geographical level.

**Limiting the network throughput of spammers** Another way to avoid overloads is to take advantage of traffic shapers, which limit both incoming and outgoing bandwidth to given thresholds. Traffic shaping also carries another benefit: it helps slowing down the throughput associated to spammer activities. In order to better emulate the network conditions, variable routing delays should be considered. In this way it is much more difficult for a spammer to recognize that the traffic is artificially slowed down, since these network conditions resemble a real environment.

## 7 Conclusions

In this paper we have shown the design and implementation of a fully working anti spam framework. Through the deployment of honeypot services, we are able to face the main spammers’ malicious actions. The e-mail harvesting phase is slowed down and the addresses

databases are polluted with special e-mail addresses, which lead the spammer to interact with other honeypot services (destination SMTP servers). This improves spammer traceability. We also deploy honeypot open proxies and relays, with the goal of intercepting and blocking unwanted traffic. Furthermore all interactions are logged with analysis purposes to increase the spammer traceability. Finally, we take into account the main attacks spammers could drive against our framework, and how we can face them. We are currently testing the implementation of the HoneySpam framework. We are running honeyd on a few machines to test the functionality and effectiveness of the implemented services.

## Acknowledgements

The authors acknowledge the financial support of the FIRB project "Wide-scale, Broadband, Middleware for Network Distributed Services" (Web-MiNDS).

## References

- [1] BARNETT, R. C. Open Proxy Honeypots, Mar 2004. [http://honeypots.sourceforge.net/open\\_proxy\\_honeypots.pdf](http://honeypots.sourceforge.net/open_proxy_honeypots.pdf).
- [2] CENTER FOR DEMOCRACY & TECHNOLOGY. Why Am I Getting All This Spam?, Mar 2003. <http://www.cdt.org/speech/spam/030319spamreport.pdf>.
- [3] CHANNEL MINDS. Spam Traffic Risen 40% Since November Says Email Systems, Feb 2005. [http://www.channelminds.com/article.php3?id\\_article=2464](http://www.channelminds.com/article.php3?id_article=2464).
- [4] COREY, J. Advanced Honey Pot Identification And Exploitation, Jan 2004. <http://www.phrack.org/fakes/p63/p63-0x09.txt>.
- [5] COSTER, M. A Method for Web Robots Control, Nov 1996. <http://www.robotstxt.org/wc/norobots-rfc.html>.
- [6] E-SCRUB TECHNOLOGIES, INC. Wpoison, 2000. <http://www.monkeys.com/wpoison/>.
- [7] ERIDANI STAR SYSTEM. MailStripper, 2005. <http://www.eridani.co.uk/MailStripper/>.
- [8] HANNA, J. Anti-Spam SMTP Proxy, 2004. <http://assp.sourceforge.net/>.
- [9] KOHNO, T., BROIDO, A., AND CLAFFY, K. Remote physical device fingerprinting. In *Proc. of the 2005 IEEE Symposium on Security and Privacy* (May 2005).
- [10] KRISHNAMURTHY, B. Mohonk: Mobile honeypots to trace unwanted traffic early. In *NetT '04: Proceedings of the ACM SIGCOMM workshop on Network troubleshooting* (Aug-Sep 2004).
- [11] OPEN FIELD SOFTWARE. Ella for Spam Control, 2005. <http://www.openfieldsoftware.com/>.
- [12] OUDOT, L. Fighting Spammers With Honeypots: Part 1 and 2, Nov 2003. <http://www.securityfocus.com/infocus/1747>.
- [13] PRINCE, M., KELLER, A. M., AND DAHL, B. No-Email-Collection Flag. In *Proceedings of the First Conference on Email and Anti-Spam (CEAS)* (Jul 2004).
- [14] PROVOS, N. A Virtual Honeypot Framework. In *Proc. of 13th USENIX Security Symposium* (Aug 2004).
- [15] SEND-SAFE. Email Verifier, 2005. <http://www.send-safe.com/verifier.php>.
- [16] SERVICE STRATEGIES INC. MailSWAT SPAM Filter, 2003. [http://www.ssimail.com/SMTP\\_spam\\_filter.htm](http://www.ssimail.com/SMTP_spam_filter.htm).
- [17] SNYDER, J. What is a false positive?, Dec 2004. <http://www.networkworld.com/reviews/2004/122004spamside3.html>.
- [18] SOPHOS. MailMonitor, 2005. <http://www.sophos.com/products/es/gateway/>.
- [19] SOPHOS. PureMessage, 2005. <http://www.sophos.com/products/es/gateway/>.
- [20] SPENCER, B. Fighting Relay Spam the Honeypot Way, Mar 2002. <http://www.tracking-hackers.com/solutions/sendmail.html>.
- [21] SUBMIT SERVICES AND WEB WORLD DIRECTORY. MailWasher PRO, 2005. <http://www.submit-services.com/mailwasher.html>.
- [22] THE APACHE SOFTWARE FOUNDATION. The Apache SpamAssassin Project, 2005. <http://spamassassin.apache.org/>.
- [23] THE ATOMINTERSOFT TEAM. AliveProxy, 2005. <http://www.aliveproxy.com/>.
- [24] THE HONEYNET PROJECT & RESEARCH ALLIANCE. Know your Enemy: Learning with User-Mode Linux, Dec 2002. <http://www.honeynet.org/papers/uml/>.
- [25] THE SPAMHAUS PROJECT. SpamHaus Block List. <http://www.spamhaus.org/sbl/index.lasso>.
- [26] UNIVERSITY OF ALBERTA. SpamAssassin FAQ. <http://www.ualberta.ca/HELP/email/spamfaq.html>.
- [27] UNSPAM, LLC. Project Honey Pot, 2005. <http://www.projecthoneypot.org>.
- [28] WARDLE, M. FAM - File Alteration Monitor, 2004. <http://savannah.nongnu.org/projects/fam/>.