A secure and covert communication channel for HTTP tar pits to implement dynamic web page blocks to bar spammer's harvesters

TOBIAS EGGENDORFER
Institut für Technik Intelligenter Systeme ITIS e.V.
An-Insititut der Universität der Bundeswehr München
Werner-Heisenberg-Weg 39
85579 Neubiberg
GERMANY

http://www.unibw.de/tobias.eggendorfer/

Abstract: Unsolicited commercial email (UCE, spam), scam and phishing emails make up for more than 90% of all emails sent world-wide. Most anti spam methods known rely on filtering emails. Meanwhile, even web browsers check URLs against blacklists to avoid fraud. However, all those methods are reactive, ergo they are only able to deal with known attack patterns. A preventive approach is to stop spammers from collecting mail addresses with their harvesters. Beside obfuscation of email addresses, HTTP tar pits have proven their efficiency in catching harvesters. This paper presents a method to use HTTP tar pits to identify harvesters and how to use this knowledge to dynamically block their access to regular web pages.

Keywords: Spam, HTTP tar pit, proactive anti-spam-measures, access control, distributed

1. Introduction

By now, the vast majority of all emails are spam. It is just a matter of the definition of spam and the time of measurement, whether it is 82% [1] for 2004, 93% [2] in January 2005 or, as the German provider T-Online states, even 97% – in early 2006 [3]. Although T-Online is likely to use a quite general definition of spam that also includes viruses, worms and Trojans, there is no doubt left that spam is a serious thread to email communication.

2. Reactive methods

As of now, to reduce the percentage of unsolicited commercial email (UCE) in any user's inbox, spam filters are implemented. They usually try to filter spam using keywords, spam signatures and heuristics like those used to identify viruses, worms and Trojans. This is sufficient and likely to be one of the few available methods to identify malware, because according to Rice's theorem [4], there is no program that is able to predict what another program does. But first spam is different to malware – it is easier individualised. And second: There are also some preventive anti-malware measures, like installing safer operating systems, using techniques like canaries [5], non-executable-flags for memory pages or simply randomising the memory layout. Those techniques try to deal with typical programming errors and to reduce their impact on system security. All of them present additional obstacles to attackers and thus increase security. The best solution would be to educate all programmers so that they are aware of the security risks resulting from security-unaware programming.

Education would also stop spam – if people would finally understand that spam only exists because products advertised therein are bought, the spam problem would be solved. But it seems more realistic to educate programmers than to educate users, as they seem to click on anything that might be attractive to them. Therefore, technical methods to reduce spam are urgently required.

However, filtering is reactive, it is based on an understanding of how previous spam looked like. It is also a heuristic approach and therefore prone to a certain rate of bad guesses, i.e. spam filters will always misidentify spam as ham, as the opposite of spam is called, or vice versa. The later being the worse alternative, as quite a lot of users trust in their spam filter and do not manually check it for ham.

A better approach would be to identify bulk mailers' typical behaviour and stop spam before it abuses resources on a mail server. This is necessary because filtering also consumes huge amounts of computing power. Current anti-spam appliances run on high-end server hardware, equipped with double multi-core-processors and gigabytes of RAM [6].

A more promising approach would be, if spammers were unable to find email addresses. If they do not know their victims, they cannot spam them. For this reason, [7] analysed and tested several techniques to obfuscate email addresses on web pages and proposed a HTTP tar pit to lock harvesters in.

3. Organisation of this paper

Section 4 gives an overview over existing anti-spam techniques and explains why their efficiency is limited. Section 5 explains the principle of a HTTP tar pit and describes how it could be combined with a SMTP tar pit to become even more effective. In the next section 6, an analysis of how to use IP addresses collected by those tar pits is presented. Then in section 7, two implementations of an IP based filter on an Apache web server are shown. Section 8 discusses how tar pits could communicate with a blacklist database. In the last section 9, we conclude and give an outlook on our ongoing research.

4. Reactive anti spam techniques

4.1. Blacklisting

Probably the first anti-spam-filter was blacklisting bad sender's “from”-addresses, a countermeasure spammers evaded by using a faked “from”-address. The alternative to identify ham is white listing, here, only known good senders are listed. This does make sense to reduce the risk of false positives if combined with other spam filtering techniques but is not the key to get rid of spam, as it inhibits new contacts.

To resolve the problem of forged "from"-addresses, blacklisting was extended to list IP addresses from were spam is sent and to disallow those machines to connect to a mail server. This has been especially useful because spammers used to use so called open relays, i.e. mail servers configured to accept mail for any domain and to forward those messages to the respective mail servers. They later also used so called open proxies. Those are HTTP proxies allowing to push SMTP-commands to a remote SMTP server and thereby obfuscate the real senders IP address and identity.

When invented back in the late 1990s, those blacklists, often called open relay blacklist, both helped filtering spam and supported the demand of switching off those open relays. But they were also known for having heavy side-effects: Almost all important email providers have already been blacklisted on at least some of the widely available blacklists [8][9][10].

Now, the increasing usage of so called zombie PCs, i.e. mostly windows computers infected with worms, to send spam [11] made those black lists more and more useless, because usually zombies do not have static IP addresses but dynamically assigned ones: Those machines are mainly PCs used by individuals and are therefore connected via dial-in or a DSL-line to the Internet. The usage of those zombies does not require any special knowledge, because those "bot-nets" are available for rent for comparable small amounts of money [12], so it has evolved to be common practice.

Thus blocking spam based on the sender's IP requires either a very dynamically changing blacklist to reflect the fast change of IP addresses or to block entire subnets known to be used by dial-in providers to prevent potential abuse. This again has heavy side effects, because this also blocks thousands of legitimate mail users that run their mail transfer agents (MTA) on mostly Unix machines at home. With the increasing use of bot-nets to send spam from, open relays are becoming less interesting to spammers, but are still in use.

4.2. Content Filtering

Another, well known and widely implemented way to filter spam are content-filters that might be both applied to the header and / or the body of a mail message. Some of those filters are based on a "bad-word-list", i.e. a list of words which is likely to occur only in spam. Besides the well-known pharmaceutical products, those lists often also include phrases as for example "click here" considered to be typical spam phrases. Modern content filters allow to weight their bad list words and only consider a mail to be spam, if an overall score computed from the individual word weights has been reached. This is necessary because to different users the same word might both be a typical spam indicator or a terminus technicus used in daily work like "mortgage" for a bank clerk or certain drugs trade names for physicians.

Bayesian filters implement a self learning mechanism to identify typical spam words. They then calculate a probability off the message being spam, based on what they learned. However, they basically implement nothing else then a bad and good word list.

All those bad word lists filters failed when, in early July 2006, spammers started to conceal their message in images sent via email. Often, those images where cut in random pieces and then readjusted using HTML tables or style sheets [6][13].

Content filtering also includes to analyse the headers of a mail message for implausible "received"-headers. Those headers indicate the way the message took through the Internet and are often forged to mislead both spam filters and anti-spammers on their quest to identify spammers [14].

More advanced products also analyse the proportion of HTML-tags to text or look for images loaded from external web pages, so called web bugs, used to verify that a spam message has been read [15]. Some even check for URLs quoted in the message and compare them to a list of web pages known to be spam-vertised [6]. Those lists are maintained the same way sender IP blacklists are.

Also those indicators for spam are usually weighted and a score is computed. If the score is above a defined limit, the message is considered to be spam. Good anti spam filters again allow users to adjust this weighting mechanism according to their needs.

Unfortunately, spammers know about those techniques and register accounts with mail service providers offering spam filters. They then test their spam against their mail filters and fine tune it, until it passes through [16][17].

This is the reason, why filtering is always one step behind, no matter how advanced content-filtering becomes [18].

4.3. Collaborative Filtering

Collaborative filtering is yet another approach to identify spam: To do so, big mail providers analyse mails their customers get and compare them to both mails to other customers and mails received on special honeypot addresses. This requires the filter operator to either store incoming messages for comparison or to compute a checksum over an incoming message. To store entire messages requires huge storage capacities: A spammer might wait between delivering two spam messages to different accounts at the same provider for some time. According to a test, in only 92% of the cases, the two messages were received within 15 minutes of each other [19]. Therefore incoming messages should be stored and delayed for at least 15 minutes to identify it with a certain probability as a part of a spam run. Comparing each stored message to an incoming message, also consumes computing power and rises important privacy questions.

Using checksums is also difficult, as spam messages are more and more individualised and therefore differ one from another although they have basically the same content. Compared to the usual requirement of cryptographically secure checksum algorithms, i.e. checksums that change significantly even on small alterations of the message [20], those checksum algorithms required to compare mail messages should remain constant on small alterations, but they should also be secure enough to only identify messages with semantically identical content and should not generate false positives [21].

4.4. Grey listing

Collaborative filtering should implement a certain waiting time to allow some messages out of a spam run to arrive. This results in a delayed delivery of the message, the same as it would with grey listing. Grey listing is to force the sending MTA of a message to resend it after a short time.

As of now, this solution is quite potent, as most spam is sent through zombies. Those worms contain their own SMTP engine, which is usually quite simple and only implements a subset of SMTP. Most of them are still unable to handle the temporary unavailable condition used in grey listing and therefore consider this condition as a fatal error and stop delivery. Grey listing has two major disadvantages: It slows email communication down and it is likely to be useless when those worms will implement better SMTP-engines, which is to be expected soon. As of beginning June 2006, there are already anecdotal reports of bots being capable of dealing with grey listing [22].

4.5. Authenticated SMTP

Another common suggestion is to fix SMTP's lack of authentication, which most people believe to be the one and only reason for spam. The basic concept is to add an additional record to the Domain Name Service (DNS) indicating which IP addresses are allowed to send email for a certain domain [23]. It implements some kind of reverse Mail Exchange (MX) lookup, where a MX lookup is used to identify a domain's mail server to send mail to.

Unfortunately, this new technology breaks important and intended email features. Email forwarding, used probably as often as call forwarding, becomes very complex. Also using a company's mail server to send private mails from with the private mail address as sender address becomes impossible, because the company's mail server is not listed as a trusted relay in the private domain's DNS record. Nota bene, the company's mail server does not need to be an open relay to provide this functionality, as long as the user has an IP within the range of IPs the server relays for.

The “Sender Policy Framework”-based (SPF) Sender-ID technology discussed at MARID [24] was foredoomed for using Microsoft's proprietary, licensed technology. Looking at the described disadvantages of SPF and also Yahoo's competing “Domain Keys”, it is likely, that it will fail too. Considering that spammers were among the firsts to implement and use SPF [25][26][27], it offers no advantage over any other spam filtering technology.

The problem with SPF is, that it only requires to register a domain and list IP addresses that are allowed to send emails with sender addresses containing that domain name. Spammers register domains regularly to promote their products, they often use so called “bullet proof”, i.e. spammer friendly hosters to do so. Therefore, SPF neither helps to identify the sender, although SPF advocates claim it.

Last but not least, all those changes to SMTP require a broad installed base of mail servers supporting them. Besides being hampered by competing standards, those changes to SMTP need to be deployed world wide. Rough estimations based on the amount of open relays world wide and their percentage of SMTP servers indicate, that there is an installed base of at least 22.5 million SMTP servers world wide [28], that all need to be updated and registered. Compared to the simpler task of just configuring a MTA to not be an open relay, installing those authentication enhancements to the protocol is rather difficult.

Although since a few years most MTAs are non open relays by default configuration, i.e. out of the box, still 1% of all MTAs are open relays. Considering that open relays are blacklisted and banned since at least ten years, world wide adoption of any SMTP authentication scheme would require at least ten years from when a suitable authentication standard emerged.

4.6. Stopping email address collection

Other anti spam techniques are based on an understanding of how the spam business works. In [12] and [29] an ex-spammer shared insights on how spammers organise their work. According to those sources, collecting email addresses to spam to and sending spam itself are two distinct spheres of business.

By using Google or looking at ebay, it is easily verified, that there are people selling mail addresses to spammers. There are even spam mails advertising email addresses to spam to. However, if spam is sent through zombies, the locally installed worms could search local hard disk drives for email addresses and spam to them. Some worms actually offer this, but the problem of worms infecting a computer is probably easier addressed by the operating system manufacturer and is not spam specific. The abuse of worms to send spam is only a symptom of broad installed base of insecure computers.

Most email address vendors collect the mail addresses in two simple ways: They either offer web pages where they ask people to subscribe themselves and / or some friends with their respective email addresses to receive either some kind of information or join sweepstakes, or they collect email addresses from web pages using spidering technology known from search engines. The programs they use to do so are called “harvesters” or “email spider”, the later being their providers' preferred term, and are easily available from major download sites in the web.

In [7] several methods to prevent the collection of email addresses using different obfuscation techniques are discussed, [30] analyses their efficiency. To offer a fast and easy to install solution, [31] proposes a way to automatically obfuscate email addresses on both static and dynamically generated web pages, thereby solving the problem to modify or redo existing web pages.

Besides obfuscation of email addresses, it would be useful, if harvesters could easily be identified and their access to web pages might be blocked. If it is possible to do so, harvesters would not find any useful piece of information on a web page. A way to achieve this might be using access information from HTTP tar pits.

5. HTTP tar pit

Their main intent is to bar harvesters from collecting mail-addresses by trapping them in a tar pit. The basic concept is to create random web pages containing links on the same or other tar pits. This pollutes the list of web pages to visit the harvester has and keeps the harvester returning and finally staying in the tar pit. As soon as the harvester is caught, all of it's resources are attracted to the tar pit, thereby preventing it to visit any other web page and collect email-addresses there.

Setting up a functional and safe tar pit is not as easy as it might seem at first glance: First, “honest” spiders, such as GoogleBot, should be kept out. Second: If the tar pit publishes links to itself, they need to be different. And last but not least the tar pit needs to make sure it is not hit by a denial of service condition if a harvester runs in circles through the site.

The first problem is solved by using the robots.txt-Standard

[32], which all serious spiders obey but harvesters ignore.

Publishing different links to the tar pit-script means to alias all of those generated URLs to the same script. There are different approaches to do so: One would be to build a small web server on its own to satisfy those requests, the other, easier deployable solution is to use features of existing web servers like Apache. Apache alone offers three ways to associate one script with many URLs: mod_rewrite, mod_alias and ErrorDocument. Each of them has its own specific advantages and disadvantages:

mod_rewrite for example sends a "Location"-header back to the browser. This could betray the tar pit. mod_alias requires access to the main httpd.conf, which is impossible if the tar pit should be implemented on some kind of shared virtual web server. By contrast, the ErrorDocument-directive is usually available through .htaccess-files at most providers. Here, the tar pit has to take care of the HTTP status sent back, which is no problem.

To solve the maximum load problem, again, different approaches exist. The probably most elegant way is to use semaphores initialized with the maximum amount of parallel threads to run [33]. This reduces busy-waiting-logic within the tar pit, thus relieving the processor and is simple and efficient.

A tar pit should not only have the harvester return to it every few moments, but it should also slow down the connection, thereby reducing download speed and attracting the harvester even longer to it. A simple method is to slow down the connection on the application level. A commonly used implementation is to deliver content character by character or line by line sleeping for a few seconds between sending each. This gives the impression of a very slow server, but if done carefully, gives no hint on the existence of a tar pit.

Timing is crucial to this kind of slowdown: Some harvesters do set very short default time-outs to avoid being caught in a tar pit like this. Therefore, tar pits should be tested against existing harvesters to identify those limits and adjust their timing accordingly.

A very basic implementation of a HTTP tar pit using PHP4 has been published in [34], a little more elaborated solution and first real-world test impressions have been presented in [35]. The tar pit described there uses some further obfuscation techniques: Some are using randomly generated sub domains to give the impression of across-host-links and the use of different domains and IP addresses to obfuscate it's existence even further.

Although in real-world experiments this tar pit proved to be efficient, tests with off-the-shelf harvesters available in the web gave some hints on how to modify the tar pit to be even more effective. Most harvesters implement some kind of progress meter by listing the last email addresses found. The first tar pit implementation did not deliver any email addresses. Therefore, a human operator could realise that his harvester got caught by a tar pit. He could even blacklist the tar pit and inform other spammers of its existence.

To have harvesters stick longer to the tar pit, the tar pit should offer some email addresses to the harvester. But those addresses need to be existent: Random addresses under random domains might easily contain existing email addresses belonging to someone else who then will receive spam.

The other downside to random addresses is the so called bounce spam [36]. This is spam sent to a non-existent address seeming to originate from another domain or email address than the one the spammer has. For each undeliverable spam message an error message is created and sent to the supposed sender's address, and, if it is also non-existent, to the postmaster of his domain.

Considering this, email addresses published by the tar pit should be existent and a mail server should accept messages to them. To achieve this, the authors suggested in [2] to use a SMTP tar pit as pseudo-MTA for the HTTP tar pit.

6. Identifying harvesters with a tar pit

Due to previous experiments with HTTP and combined tar pits, the test tar pits are heavily linked from many web pages in the Internet. They therefore attract enough harvesters that follow links on web pages. Humans are unlikely to accidentally follow a link to the tar pit, because web master have been instructed to link them with a CSS-style "invisible". If for any reason a human would follow a link to it, he will soon notice that the page he came across is not intended for human visitors. Harvesters by contrast will stay in the tar pit, as field experiments proved.

Therefore, the tar pit is a useful method to tell apart humans from machines: As soon as a visitor stays for more than a few visits in the tar pit, it is very likely to be a machine.

If the visitor is a machine, it did not obey the robots.txt standard [11], [12], that protects good spiders from being trapped in the tar pit. This is the method of choice to distinguish between search engines' spiders and spammers' harvesters.

To understand harvesters' behaviour, the log files of the tar pits were analysed and evaluated. As expected, there were no time patterns to be identified – harvesters seem to wait for a random time between two visits and they also have different length lists of pages to visit that also influence when they will visit the next link to the tar pit in their list. The longer harvesters have already been trapped in the tar pit, the more frequent return visits become. Considering the links-to-visit-list harvesters use, this is expected behaviour.

Accidental human visitors by contrast usually visit the tar pit for less than an average of two links. Those that stayed longer seemed to have analysed the tar pit's behaviour. This at least is made plausible by looking at entry points harvesters used when visiting the tar pit: Some web masters who linked the tar pit understood well how it worked and crafted their own, specific links to it.

As harvesters' timing is unpredictable, but humans' is, the first piece of information to identify a harvester is that it is visiting the tar pit more than twice. If by this piece of information, it was possible to tell apart humans from harvesters, this could be used to block access to web pages for harvesters, at least for a certain time period.

An obvious, yet important requirement is, that humans should not be blocked from visiting other web sites. Only harvesters should. It might be helpful, that, if our assumption drawn from the log file is true, human visitors who click on more than one link in the tar pit are likely to be people who try to understand how the tar pit works. Thus, it might be safe to also assume that those people will understand why their access to other web pages has been blocked for a certain time.

So the real problem are "one link visitors" who by accident came across the harvester trap. To avoid them to be banned from web page access for too long, the ban is imposed depending of the amount of visits during a certain time period.

Another important piece of information derived from log file analysis was that a non negligible fraction of harvesters are operated from dynamic dial-in IPs, i.e. their IP address is changing at least every 24 hours. Therefore, it should be avoided to block IP addresses for more than 24 hours, if, after 24 hours after the first harvesting report from this address has occurred, harvesting suddenly ceased.

To both accomplish the requirement not to block humans and to only block dynamic IPs for the time harvesting occurs from them, we suggest to calculate a ban time according to Formula 1.

We suggest to first increase the block time until a certain limit (m) of visits to the tar pit has been reported to prevent accidental human visitors from being blocked for to long. After m visits, access for the harvester's IP will be blocked for the next 24 hours.

$$t_{ban}(n)=\begin{cases} n\cdot 0{,}25 & for\ n\leq m \\ 24 & else \end{cases}[h]$$

$$t_{ban}: ban\ time,\quad n: visit\ count,\quad m: maximum\ visits$$

Formula 1: Ban time

We added an additional check after 25 hours of the first harvesting report from an IP to unlist the IP if during the last hour no harvesting activity was reported. If not, we added another 24 hours of ban time for this IP. It would be tested again after 24 hours. This is necessary to avoid dynamic IPs to be blocked for too long.

7. Dynamically blocking the Harvesters

As soon as a harvester's IP has been identified, the harvester could be disallowed access to web pages. To do so, different methods exists: One is to totally ban this harvester from a web page, i.e. showing it an error message. The other would be to not only obfuscate mail addresses but to dynamically totally remove them from a web site.

Although a thoroughly tested obfuscation algorithm might be used, that has not been broken yet and is unlikely to be any time soon [7], totally removing an address is for obvious reasons even safer. Another advantage of this approach is besides not displeasing visitors that accidentally clicked into a tar pit, to obfuscate the obfuscation mechanism even more: To spammers trying to investigate why their harvesters do not find any email addresses any more, the web page presented looks rather unsuspicious. The small changes introduced by totally removing mail addresses are not as obvious as an error message would be.

By contrast, a total block reduces work load for the web server if it was to produce complex dynamic web pages whose content is of no use for harvesters. Adding to this, the error message displayed could contain a random link to any tar pit to send the harvester to. If the error message does not contain any other links, the harvester does not add new links to its list of links to visit. Thereby, the percentage of tar pit links in its list grows, in turn helping to increase the tar pits effectiveness and also protecting other web pages, that do not implement the blocking mechanism, because the harvester is busy working his way into the tar pit.

From our point of view, both approaches offer their specific advantages and disadvantages, so it should be left to the local administrator to choose which one is better suitable to the environment the web server runs in.

Based on the Apache output filter presented in [37] that provided a solution to dynamically obfuscate email addresses, we propose an enhanced output filter able to look up the client's IP in a database of known harvesters whether it is listed. If a blocked IP has been identified, the module does not only obfuscate email addresses, but removes them from the web page.

Email addresses in "mailto:" links are replaced with a link to a contact form. [7] also suggested methods to implement a spammer save contact form, that avoids both abuse and the harvesting of email addresses from the form. Both the URL of the form and the text used to replace email addresses are easily configurable.

The database used is populated with data from the combined SMTP HTTP tar pit and regularly maintained in a way, that, if an IP is found in the database, the output filter knows, that this IP is to be blocked. This simplifies access rules and allowed us to easily modify the ban regime during testing.

Instead of just removing email addresses, access to the web server could be blocked for known harvesters. As soon as a harvester tries to access a web page, an error message is displayed, indicating that the client's IP has been blocked because it was reported to take part in harvesting activity.

To do so, we implemented an Apache filter in Apache's URI translation phase [38]. In this phase, the web server tests, whether an URL is valid locally or needs some changes to be made. If changes are necessary, they are done in a filter module. Obviously, the simplest way to redirect a harvester to an error page would be to change the URI to the URI of such an error page. This is exactly what we did based on the client's IP.

In our test setup, we stored the IP black list in a local MySQL database. Without any optimisation of the database, we were able to identify a performance reduction for database access of approximately 1 ms. This test was run on an Intel Centrino 735 MHz system with Linux, Fedora Core 2. We tested the performance using Apache Bench, a benchmark programme included in Apache.

A delay of 1 ms is likely to be within the measuring tolerance if the request is sent across an Internet connection and not locally as we did. Obviously, the blocking mechanism is constant in time, independent on how large the file sent by the server would be.

With the output filter, the obfuscation overhead grows with the size of the files. The additional overhead for database access however remains constant with approximately 1 ms.

For our test setup, we had the tar pits store the IPs of harvesters accessing them in a MySQL database. A cron job on the database server periodically removed IPs that were not longer to be blocked.

8. Communication with the blacklist database

In a test environment, a direct MySQL connection between the tar pits and the black list database server might be acceptable. For real world applications, security becomes an important requirement. As the tar pits are likely to be the first aim to attack because they are exposed being the communication partner for harvesters, it is mandatory to also secure the connection against attacks issued by a cracked tar pit. Because tar pits are likely to be installed on cheap virtual root servers, that are often poorly maintained, crackers might also have access to other machines in the same network as the tar pit. This could allow them to eavesdrop communication between the database server and the tar pits. Therefore, we also suggest to steganographically hide their data exchange, thereby reducing the probability of it being identified.

To increase security in a test setup, [37] first suggested to use a SSH tunnel to connect to the MySQL server and limit the MySQL user the tar pits use to insert privileges. A SSH tunnel has a few disadvantages: It needs to be monitored, because it might disconnect due to network time outs or other problems. Therefore, the SSH connection needs to be established without human interaction. For the required authentication upon reconnection, SSH keys without a pass phrase might be used [39].

If the the database server had the tar pit's SSH keys in its local authorized keys file, anyone with access to the tar pit could start a SSH connection to the database server. Although the tar pit might be hardened by using a specific user without any other rights but connecting to the server, a security hazard is left over.

Therefore, reverse SSH tunnels are better. In this case, the database server's key is stored in the tar pits' authorized keys file, and the server connects to the tar pit. Then, a SSH tunnel from the tar pit to the MySQL database is established. In this case, if the tar pit is cracked by an attacker, one more obstacle is added.

However, the attacker might use security flaws in MySQL, be it privilege escalations for the MySQL user with insert privileges, or be it something like a buffer overflow on the MySQL network connector.

Putting all this together and adding that the tar pits are what an attacker sees first, making them the preferred target for an attack, we decided that allowing direct access to MySQL from the tar pits is to dangerous from a security point of view.

In [40] we analysed several other ways to allow a secure communication between the tar pits and the database server. Because direct database access is to dangerous, another protocol would be preferable. We came up with the idea to use DNS to send IPs to the database.

From a steganographic point of view, this is a very promising approach: A lot of web servers will automatically send a reverse lookup request for any client that connects to the machine. So a DNS reverse lookup is standard behaviour of a web server and therefore unsuspicious to anyone sniffing traffic.

Implementing DNS reverse lookups and answers to those requests is also quite simple, because there are libraries both for Perl [41] and PHP, the languages used for this project.

Unfortunately, DNS does not support any authentication for client requests. [42] explicitly states that DNS is an open protocol, therefore, client requests do not need any authentication.

A first solution would be to only allow certain IPs to send DNS requests that result in a blacklist update. But this would require the database server to maintain a list of all tar pits' IPs. This again is difficult, because our suggestion is to also run those tar pits on DSL lines with dynamic IPs. This increases their invisibility and make blacklisting them for harvesters more difficult.

Furthermore, DNS requests are usually send via UDP. A UPD request's sender's IP is easily forged. Compared to TCP, there is no need to take care of answer packets, because there is no three way handshake required.

If it was feasible to use an IP lists as an access control to the database server, the server should behave like a regular DNS server, i.e. also send answers to requests from machines not the tar pit IP list. This further obfuscates the double function of this server. To increase the time and effort for IP forging, tar pit's should use DNS via TCP.

To update the IP list, a tar pit should periodically establish an IPsec connection to the database server. Then, tar pits are authenticated and their IP is stored for a certain, short period of time. This tar pit is then allowed to add harvester IPs with TCP-DNS requests.

It is also possible, to directly send those DNS reverse lookup requests over an IPsec secured channel. But this might be more obvious to an attacker sniffing traffic from and to the tar pit.

Although DNS seems to be the perfect way to steganographically hide the communication between the tar pit and the database server, some small security issues are left. However, we believe this to be the best available compromise between steganographic demands and security.

9. Conclusion

In this paper we propose to use combined HTTP and SMTP tar pits to identify spammers' harvesters while they extract email addresses from web pages. To protect regular web pages from harvesters, we propose to Apache filter modules. One of them totally blocks access for harvesters to web pages, but might annoy humans still blacklisted because their IP was in use by a harvester before, although our proposed ban time algorithm reduces this risk to a bare minimum. The total block also reduces work load on the web server, because dynamic web pages are not rendered for harvesters.

Our other example implementation modified an existing email address obfuscation output filter module for Apache to not only obfuscate email addresses but to remove them entirely from the web page as soon as the client's IP is on the list of harvesters. This has the advantage to neither disgruntle human visitors nor give spammers any obvious clues.

Whichever method is chosen, the disappearance of an email address from a web page reduces the amount of spam received on this address by 50% within one year [43].

We currently researching even more secure and hidden ways our tar pits could communicate with the black list.

References

[1] Gaudin, Sharon, Record Broken: 82% of U.S. Email is Spam, http://itmanagement.earthweb.com/secu/article.php/3349921, 2004

[2] McGann, Rob, The Deadly Duo: Spam and Viruses, January 2005, http://www.clickz.com/stats/sectors/email/article.php/3483541, 2005

[3] Kuri, Jürgen, T-Onine verzeichnet eine Milliarde Spam-Mails pro Tag, http://www.heise.de/security/news/meldung/72324.html, 2006

[4] Asteroth, Alexander; Baier, Christel, Theoretische Informatik, Pearson Studium, München, 2002

[5] Peikari, Cyrus, Chuvakin, Anton, Security Warrior. Know Your Enemy, O'Reilly, Sebastopol, 2004

[6] Schulz, Carsten, Erstelllen eines Konzepts sowie Durchführung und Auswertung eines Tests zur Bewertung unterschiedlicher Spam-Filter-Mechanismen bezüglich ihrer Langzeiteffekte Masterthesis, Universität der Bundeswehr München, Neubiberg, 2006

[7] Eggendorfer, Tobias, Methoden der präventiven Spambekämpfung im Internet Masterthesis, Fernuniversität in Hagen, München, Hagen, 2005

[8] McWilliams, Brian, SpamCop blocking some Gmail servers, http://spamkings.oreilly.com/archives/2006/01/, 2006

[9] Bleich, Holger, GMX landet auf Open-Relay-Blacklist, http://www.heise.de/newsticker/data/hob-27.05.03-000/, 2003

[10] Jacob, Philip, The Spam Problem: Moving Beyond RBLs, http://theory.whirlycott.com/~phil/antispam/rbl-bad/rbl-bad.html, 2003

[11] Kuri, Jürgen, Aufgedeckt: Trojaner als Spam-Roboter, http://www.heise.de/newsticker/meldung/44869, 2004

[12] Spammer X, Talk by Spammer X in Proceedings of , , 2006

[13] Graham-Cumming, John, Die Tricks der Spammer in: Hackin9, Nr. 3 / 2004, 30 ff., Software-Wydawnictwo Sp. z.o.o, Warschau, 2004

[14] Hochstein, Thomas, FAQ. E-Mail-Header lesen und verstehen, http://www.th-h.de/faq/headerfaq.php3, 2003

[15] Davy, Michael, Feature Extraction for Spam Classification Masterthesis, University of Dublin, Dublin, 2004

[16] McWilliams, Brian, Spam Kings. The Real Story Behind the High-Rolling Hucksters pushing porn, pills, and @*#?% Enlargements, OReilly, Sebastopol, 2005

[17] Wittel, Gregory L.; Wu, Felix S., On Attacking Statistical Spam Filters in Proceedings of CEAS 2004, Moutainview CA, 2004

[18] Gansterer, Wilfried et. al., Anti-spam methods - state of the art in: , , 99, Institute of Distributed and Multimedia Systems, University of Vienna, 2005

[19] Donelli, Giovanni, Email Interferometry in Proceedings of Spam Conference 2006, Cambridge, MA, 2006

[20] Schwenk, Jörg, Sicherheit und Kryptographie im Internet. Von sicherer E-Mail bis zur IP-Verschlüsselung, Vieweg, Braunschweig, 2002

[21] Gray, Alan; Haahr, Mads, Personalised, Collabortive Spam Filtering in Proceedings of CEAS 2004, Moutainview, CA, 2004

[22] Kühnast, Charly, Auftragskiller. Spam-Botnetz überfällt Charly in: Linux Magazin 07/06, , 68 f., Linux New Media, München, 2006

[23] Wong, M.; Schlitt, W., Sender Policy Framework (SPF) for Authorizing Use of Domains in E-Mail, Version 1, http://www.ietf.org/rfc/rfc4408.txt, 2006

[24] Lyon, J.; Wong, M., Sender ID: Authenticating E-Mail, http://www.ietf.org/rfc/rfc4406.txt, 2006

[25] Sreekumaran, Jonathan, Those spammers are at it again!, http://www.techtree.com/techtree/jsp/article.jsp?article_id=53789, 2004

[26] Varghese, Sam, Spammers ahead of the pack again, http://www.smh.com.au/articles/2004/09/09/1094530732236.html, 2004

[27] Claburn, Thomas, Spammers Hijack Sender ID, http://www.informationweek.com/story/showArticle.jhtml?articleID=47102042, 2004

[28] Eggendorfer, Tobias, Comparing SMTP and HTTP tar pits in their efficiency as an anti-spam-measure in Proceedings of Spam Conference 2006, Cambridge, MA, 2006

[29] Spammer X, Inside the spam cartel. Why spammers spam, Syngress Publishing, , 2004

[30] Eggendorfer, Tobias, Spam proof homepage design. Methods and results of an ongoing study in Proceedings of , Stuttgart, 2005

[31] Eggendorfer, Tobias, Dynamic obfuscation of email addresses - a method to reduce spam in Proceedings of AUUG, Melbounre, 2006

[32] W3C, W3C Recommendations. Appendix B: Performance, Implementation and Design, http://w3.org/TR/REC-html40/appendix/notes.html, o. A.

[33] Tanenbaum, Andrew S., Modern Operating Systems, Prentice Hall, Upper Saddle River, 2001

[34] Eggendorfer, Tobias, Ernte - nein danke. E-Mail-Adressenjägern auf Webseiten eine Falle stellen in: Linux Magazin, , 108 ff., Linux New Media, München, 2004

[35] Eggendorfer, Tobias, Stopping Spammers' Harvesters using a HTTP tar pit in Proceedings of AUUG 2005, Sydney, 2005

[36] Graham-Cumming, John, Bounce Spams, http://www.jgc.org/antispam/01312005-5decf14cb-cec9687db9aa705789b55e0.pdf, 2005

[37] Eggendorfer, Tobias; Keller, Jörg, Dynamically blocking access to web pages for spammers' harvesters in Proceedings of IASTED CNIS 2006, Cambridge, MA, 2006

[38] Stein, Lincoln D., MacEachern, Doug, Writing Apache Module with Perl and C, O'Reilly, Sebastopol, 1999

[39] Anonymous, Maximum Linux Security. A Hacker's Guide to protecting your Linux, Sams Publishing, Indianapolis, 2001

[40] Tietze, Frank, Konzeption und Entwicklung einer sicheren Infrastruktur für eine verteilte Blacklist Masterthesis, Universität der Bundeswehr Neubiberg, Neubiberg, 2006

[41] Siever, Ellen; Spainhour, Stephen; Patwardhan, Nathan, Perl in a Nutshell, O'Reilly, Köln, 2000

[42] Eastlake, Donald, DNS Security Extensions, http://www.ietf.org/rfc/rvc2535.txt, 1999

[43] Eggendorfer, Tobias, Curbing spam. Fending off spam before it reaches your filter in: Linux Magazine (International Edition) 03/2007, , 25 ff., Linux New Media, München, 2007