Towards Proactive Spam Filtering (Extended Abstract)

Jan Göbel Thorsten Holz Philipp Trinius
{goebel|holz|trinius}@informatik.uni-mannheim.de

Laboratory for Dependable Distributed Systems
University of Mannheim, Germany

Abstract. With increasing security measures in network services, remote exploitation is getting harder. As a result, attackers concentrate on more reliable attack vectors like email: victims are infected using either malicious attachments or links leading to malicious websites. Therefore efficient filtering and blocking methods for spam messages are needed. Unfortunately, most spam filtering solutions proposed so far are *reactive*, they require a large amount of both ham and spam messages to efficiently generate rules to differentiate between both. In this paper, we introduce a more *proactive* approach that allows us to directly collect spam message by interacting with the spam botnet controllers. We are able to observe *current* spam runs and obtain a copy of latest spam messages in a fast and efficient way. Based on the collected information we are able to generate *templates* that represent a concise summary of a spam run. The collected data can then be used to improve current spam filtering techniques and develop new venues to efficiently filter mails.

1 Introduction

In the recent years, we observe a shift how attackers proceed to compromise system on a larger scale: instead of using random scanning and remote exploits against common Windows network services, more and more attacks use email messages as propagation vector. These spam messages either contain a malicious attachment or a link to a malicious web page to compromise victims by exploiting client side applications, like for example the victim's browser [8,11,18].

Current approaches to deal with email spam have the problem that they are commonly *reactive*: Given a large collection of email messages collected at end-user mailboxes or dedicated mailboxes (so called *spamtraps*), the algorithms extract features of all messages that can be used to distinguish spam from ham messages, for example by using a Bayesian model [2,14] or other machine learning techniques [3]. A complementary approach is to generate a blacklist of IP addresses that are known to be related to spammers. Such blacklists can for example be constructed by extracting frequently appearing sender IP addresses from spam email headers [6]. Another example are URIBLs (*Uniform Resource Identifier Blacklists*) that list domain names that appear in URIs such as web sites mentioned in message bodies more than a given threshold of times [7]. All these approaches have the drawback that they need a larger collection of email messages in order to generate precise rules, to distinguish ham and spam emails.

In this paper, we present an approach to deal with spam in a more *proactive* way: instead of waiting at the end-user's mailboxes or spamtraps for mail messages to arrive and then decide whether or not this is spam, we directly *interact* with the servers that are used to send spam messages. The basic idea is that we execute *spambots*, i.e., malicious software dedicated to sending spam emails, in a controlled environment and collect all email messages sent by the bots. This enables us to directly interfere with botnet control servers to collect *current* spam messages sent by a specific botnet.

Based on the collected information, we can generate models of how spam messages look like (so called *spam templates*), identify unique spam runs, and extract information that can be used to enhance spam filtering techniques. Current research shows that the actual number of unique spam botnets out there is not very large (in the order of hundreds or thousands [5,15,16]), and thus it may be feasible to continuously collect information about a significant number of spam botnets in the wild with only a limited amount of resources.

The contributions of this paper are threefold. First, we propose a proactive approach to filter mail messages. With the help of controlled spambot executions, we are able to collect spam messages in the very early phases of a spam run and can continuously observe different kinds of bots. Second, we show how the collected information allows us to generate filter rules which can then be used to detect spam messages, or to generate blacklists to efficiently block spam, way ahead of current techniques to mitigate spam. Third, we implemented the system and present first results of running the system for a limited amount of time.

2 Related Work

Concurrent with our work, Xie et al. [20] and John et al. [5] introduced similar techniques to study current spambots. The basic idea of all three projects is to execute spambots in a controlled environment and collect current spam messages by observing the behavior of the bots. In contrast to the other two projects, our focus is on generating spam templates that can then be used to filter incoming spam. We use the complete email text and not only embedded URLs to be able to detect a wider range of spam. For example, image-based spam or spam messages that contain links to popular websites can potentially not be handled correctly by the other two projects.

Venkataraman et al. introduced a method to use the network structure for proactive spam mitigation [17]. Their approach is orthogonal to ours and could be combined: we use information about the actual spam sources and the spam messages sent, whereas their approach focusses on filtering known bad IP netblocks. Ramachandran et al. studied DNSBLs and the techniques used by spammers in detail [13,12]. These techniques also complement our work, since we focus on directly interacting with the spam botnets to extract information about new spam runs as early as possible.

The effectiveness of blacklists was studied by Jung et al. [6]. Since spammers behave very dynamic and change their tactics frequently, it is questionable whether or not the results from 2004 still hold. Especially due to the spammer's move to (reverse) SOCKS proxies and template-based spamming, the effectiveness of blacklists is limited. Kreibich et al. studied Storm Worm in great detail and provide more information about template-based spamming [8], one of the most popular techniques used by spammers.

3 Overview of Current Spamming Techniques

Traditional Spamming Techniques. The traditional techniques for sending spam is *direct spamming*: a spammer uses a set of machines under his control to send spam mails to the intended recipients directly. This specific behavior can easily be detected by an ISP (e.g., a large amount of outgoing SMTP connections or many complaints about a specific IP address), who then shuts down the spammer's account or blocks further SMTP requests from the specific IP addresses.

As a result, spammers started to use *open email relays* to send out spam mails [6]. The basic idea is that misconfigured mail servers can be easily abused to send out large amounts of spam: a spammer has to scan the network for such open mail relays and then send all his spam emails using this server. The open relay will send all emails to the intended recipients. This technique was mainly used by spammers several years ago, thus efficient techniques exist to block spam sent via open relays, e.g., blacklists as explained in Section 4. *Proxy pots* (i.e, honeypots that act as open relays [1,10]) can be used to study spammers that still use open relays and provide additional input to spam detection filters.

A similar technique to send out spam emails is based on *open proxies*: the spammer scans the network for open proxies (typically either SOCKS protocol version 4 or 5). Once he has found an open proxy, he uses this machine to relay SMTP commands to the recipient's mail server via the proxy. The open proxies thus acts as a intermediary between the spammer and the mail server, efficiently hiding the spammer's true identity. Nevertheless, there exist several blacklists which contain IP addresses of current open proxies that can be used to efficiently block spammers using open proxies.

(Reverse) SOCKS Proxy-based Spamming. To counter the success of blacklists, spammers use compromised machines as proxies: the attacker installs a SOCKS proxy on the compromised machine and then uses these proxies to send SMTP commands to the mail server (see Figure 1(a) for an illustration). Since the IP address of the bots changes frequently — for example due to reboots of the infected machine or other DHCP effects — blacklists have a hard time to keep up with the dynamic changes. In order to be more efficient, the attackers invented the concept of *reverse proxy-based spamming* [4]. The bot connects in the first phase to the controller and establishes a reverse SOCKS proxy connection. All SMTP commands are then relayed through this tunnel from the controller to the actual mail server. The basic concept is the same, the main advantage is

that the bots announce to the controller once they are available and can then immediately be used for spamming purposes. Furthermore, the approach also has the fundamental advantage that bots behind a NAT gateway can be used by the spammer as well: these machines are typically not directly reachable and can thus not be used for proxy-based spamming. However, if they use a reverse proxy approach, also these machines, which are typically end-user machines running behind a DSL router, can be used for spamming purposes. Again, due to the dynamic nature of these machines, blacklists are often not able to accurately list these machines. Therefore, reverse proxy-based spamming is an interesting method from an attacker's point of view to send out spam emails.

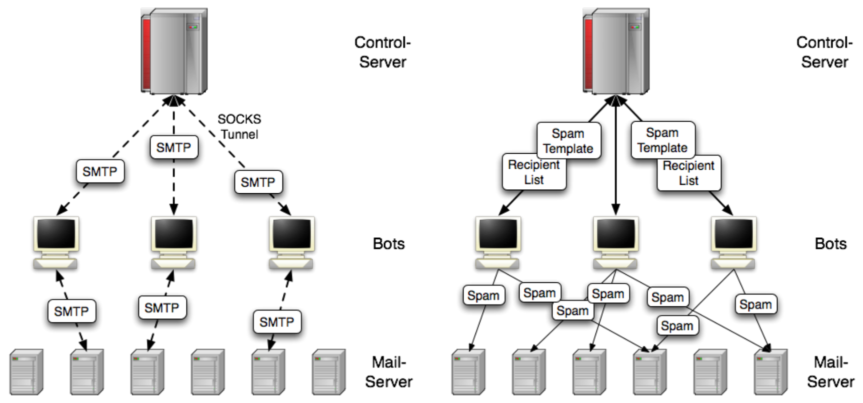

Fig. 1: Schematic overview of (a) SOCKS proxy-based spamming (b) template-based spamming, two common spamming techniques nowadays.

Template-based Spamming. Another modern technique used by spammers is *template-based spamming*, as illustrated in Figure 1(b). Instead of relaying the SMTP commands through the compromised machines, the attacker sends the bots a *spam template* that describes the structure of the spam message to be sent. Furthermore, the attacker sends meta-data like recipient list, subject list, and a list of URLs that are used to fill in variables in the template. The bots then construct an email based on the template and the meta-data, and send this email to the targets. As a result, the actual work of handling the SMTP communication is moved from the control server to the bots. Nowadays this technique is used by most large spam botnets, like Storm Worm, Bobax, Rustock, and a lot of the other major spam botnets [15,16].

One indication that this technique is widely used by spammers can be seen by the huge drop in spam emails after the web hosting service provider McColo was shutdown in November 2008 [9]. This provider is suspected to have supported spammers by hosting many control servers used in template-based spamming [16]. When the provider McColo was disconnected on Tuesday, November 11, 2008, the average number of detected spam mails dropped more than half on many networks [9].

4 Towards Proactive Spam Filtering

Currently, there exist quite a few techniques to filter spam mails, which we discuss in this section. A *DNS blacklist* (*DNSBL*) contains a list of IP addresses that mail servers should block and not accept mail from. A DNSBL can be queried with the help of the DNS system and allows an efficient way to perform lookups. DNSBLs typically list either open mail relays or open proxies. However, there exist also DNSBLs that list other suspicious hosts, e.g., machines running within specific networks or supposedly infected machines. Closely related to DNSBLs are URIBLs (*Uniform Resource Identifier Blacklists*). An URIBL contains domain names and IP addresses that appear in URIs such as web sites mentioned in message bodies more than a given threshold of times. In contrast to DNSBLs, which are used to check the sender's address, an URIBL checks the *content* of mail messages for suspicious links and complements the DNSBL check.

Another approach to detect spam mail are *hashing systems*: the intuition behind this approach is that if the same message body is sent to many people, it is bulk and should be filtered. Unfortunately, these systems require that many people receive the same mail in order to be effective. This can be defeated by a spammer with the help of random content which is added to the mail body.

Bayesian spam filtering [14] and rule-based filtering systems such as SpamAssassin focus on the mail content to detect spam. Given a large repository of both spam and ham messages, these approaches extract features of all messages that can be used to distinguish spam from ham messages, e.g., via a Bayesian model [2,14] or other machine learning techniques [3].

An orthogonal approach to prevent spam messages is *greylisting*: a mail server does not accept an incoming messages from an unknown sender and responds with a temporarily reject message (often SMTP return code 450 or 451). A legitimate mail server will typically re-send the message after a short timeout, whereas current bots commonly try to send a message only once. Therefore, messages sent via bots are effectively blocked. Unfortunately, this approach only works since bots do not try to re-send messages. Once enough mail servers use greylisting, the spammers will presumably adopt and also re-send messages.

4.1 Infrastructure for Proactive Spam Collection

Most approaches to filter mail as discussed in the previous section require a corpus of both spam and ham messages. Based on these two sets, filtering rules are extracted. These rules can for example be either a model how spam messages look like (Bayesian or rule-based filtering) or a threshold-based approach to classify a given message as spam if it contains certain URIs. To be more efficient, it is desirable to obtain spam messages as accurately and early as possible when the bots start spamming. In this section, we introduce an approach to collect spam messages in an automated way by directly communicating with the spam botnet control servers. This approach can be used to study both (reverse) proxy- and template-based spamming operations in an automated way.

A straightforward approach to collect spam mails is based on high-interaction honeypots: we execute a spambot on a native Windows machine and let it communicate with the controller such that the bot can either establish a SOCKS tunnel or receive the template and meta-information from the controller. However, we prevent outgoing spam messages by intercepting SMTP communication at the local gateway and emulating the behavior of the target mail server. For example, if the bot wants to connect to `gsmtp183.google.com`, we first intercept this communication and redirect it to the local mail server running at the gateway. Furthermore, we grab the banner from the intended server and then replay the banner to the bot. As a result, the bot is tricked into thinking that it actually communicates with the intended server, whereas it only communicates with our mail server. We can collect all mail messages at the gateway and obtain an overview of the *current* spam messages sent via this spam botnet.

After a certain amount of time, we reset the machine to a clean system using a software-based restore mechanism and execute the next spambot. Different strategies exist to determine when we have collected "enough" unique email messages to reset the honeypot. Simple heuristics include using a predefined timeout or a predefined number of email messages that should be collected, e.g., resetting the honeypot once 5,000 spam messages are collected. Advanced techniques analyze the collected spam messages and reset the honeypot once a complete spam run was observed: our preliminary results indicate that one spam run typically consists of 10 – 100 different domains being advertised in different kind of spam emails. These domains are advertised in a round-robin fashion, and thus we can reset the honeypot once we have observed each URI several times. We have implemented several heuristics to decide if we have already collected enough useful emails and still evaluate which technique works best in practice.

Furthermore, we have implemented a priority queue that enables us to periodically monitor a given spam botnet: Once we have collected enough spam messages from a given binary and thus do not obtain any new information, we revert the honeypot back to a clean state and start the next bot. However, we enqueue the binary for another analysis run since the spam campaign could change and we could observe new spam mails when we execute the bot again later on.

4.2 Towards Proactive Spam Filtering

The spam mails collected with the help of the system proposed in the previous section can be used to generate spam filtering rules in a *proactive* way: since we observe a live spambot, we can be sure that we observe *only* spam messages. Based on the collected information, we can generate detection rules. The easiest rule is an URI blacklist based on the advertised URIs we observe in the collected messages. We can also analyze the spam message in more detail: quite often, the spam engine of a bot contains unique artifacts that can be used to identify the spambot and classify a given message as spam. These artifacts could be specific header fields or the arrangement of certain header fields or body text. Finally, we can generate a model describing the overall structure of the spam message. This model would include the complete template we extracted by analyzing several spam samples. An incoming mail message could then be checked against this model and if it matches, the mail would be classified as spam.

We analyze all collected spam mails and extract common parts to generate a model that matches the spam emails collected from a single spambot. For extraction we use a variation of the longest common substring (LCS) algorithm. We generate all common substrings of all emails contained in a single observed spam run and fill the gaps with placeholders to form a raw template. More specifically, we use the following algorithm to compute a single raw template:

1. First, we read all emails belonging to a single observed spam run and sort them according to their text length. Thus, longer emails are processed first.
2. In the second step, we take the first email from the sorted list and consider it as our first raw template named α.
3. Then, we take the next email from the list and merge it with α to form a second raw template named β, which is then one step more specific than the previous template α.
4. Now, we compare the two raw templates α and β and determine the amount of text that was replaced by placeholders. If the percentage of removed text is below a predefined threshold θ, β becomes our new α, the email used to form β is removed from the list and we continue with step three. That way only emails that do not modify α too much are added to the final template of this run. If the percentage of changed text is above θ, the current β is too generic and is therefore discarded. The email that was used to generate β is moved back to the list of emails and we take the next email from the list to create a new β, as described in step three.
5. As long as emails are in the list we continue with step three. At some point no more emails are left for processing or only those emails are left that did not fulfill the threshold criteria θ. That means that α becomes static. At this point, the raw template is finished and a new raw template generation process begins, with all emails that are left from the previous run.

After the template generation process we end up with a certain number of raw templates, and the number of email that were used to form each. The generated raw templates do not contain any regular expressions yet, but only placeholders. Thus, we generate from each email that was used to form the template a regular expression to replace the placeholders. To achieve this we analyze the variable parts of the emails, i.e., the parts of the email that are replaced by placeholders in the raw template, and store the length of the longest and shortest variable part. Furthermore, we analyze the characters of the variable part to decide if the resulting regular expression should contain digits, characters, or special characters, or all together.

Once all placeholders are replaced with a regular expression, we have a precise template in form of a single regular expression matching all emails of the

```
Subject: ([\:\'\,\?\w]){3,11} ([\,\'\?\s\w]){14,35}
X-Mailer: Microsoft Outlook Express 6.00.2900.2180

Body:
Lose Weight -Burn Fat -Look great and feel=20
great -
&nbsp;
http://([A-Za-z]){9,14}.chat.ru
&nbsp;
And do all this with a miracle weight loss fruit &#8211; ACAI BERRY &#8211;=
Best=20
of all &#8211; it&#8217;s completely FREE for a limited time! Click here to=
 receive your=20
completely free bottle of ACAI BERRY supplemnt!
```

Fig. 2: Final spam template extracted from analyzing over one thousand emails received by a single spambot.

particular spam run captured in our monitoring environment. Note, that in case a spambot sends messages with permuted words, sentences, or various synonym words, templates can either become too generic or several templates are generated for each message with a permuted sentence for example.

Figure 2 shows the results of such a template generation process. The template was generated from 431 spam emails sent by a single spambot. During this spam run only a single campaign was advertised. The figure shows that the template used by the spambot contains very little variable parts. Most of the text is completely fixed, leaving only the subject, and the advertised URL as variable parts within the generated template. Note that the URL is only partially variable, the domain name of the URL is fixed.

In the current template generation process, we only consider the subject, x-mailer, and the complete body of spam messages. Any other information including attachments are stripped from the emails, as most of the header fields like “From:” and “To:” contain only variable parts which do not add more information to the conciseness of the template.

5 Preliminary Results

We obtain malware samples with the help of different honeypot solutions. For example, we use different honeypots to collect samples of autonomously spreading bots and worms, and honeyclients to collect samples of malware that is installed via malicious websites. Furthermore, suspicious samples can be manually submitted to our analysis environment. In total, we receive between 400 and 1,500 unique samples per day using this setup. All samples are analyzed with the help of an automated tool called CWSandbox [19], and a sample that shows signs of spam behavior can then be analyzed in the honeypot environment described above. We keep the samples in a queue and periodically execute them again, such that we can observe changes in the spam runs of a given spam botnet and obtain the latest set of spam messages.

Executing a spambot allows us to efficiently collect spam messages. In the current setup, we use a time-out of 30 minutes after which we reset the honeypot and execute the next malware sample. During this period, the bot typically sends a few hundred up to a few thousand spam messages, we even observed several bots sending more than 50,000 spam messages in this short amount of time.

So far we have executed 40 different spambots in the analysis environment. This resulted in a total of 100.977 spam emails to use for template generation. First checks against a local spam folder with 20.290 spam emails revealed that 30% of the spam email belong to spam campaigns we were able to collect in our observation environment. Some of these spam emails are even more than one year old and still use the same template. As this first test was rather quick and did not consider the full templates we generate, we guess that the detection rate is even higher. The reason why we were not able to use full templates yet is that for the spam campaigns found in the local spam folder we do not have enough emails that were also sent by the bots that ran in our analysis environment. As the detection rate of the spam emails with the help of templates highly depends on the diversity of the emails used to generate the template, a slight variation in the spam campaign can render a template useless.

The following example describes the problem. We used 493 emails from a Casino advertising spam campaign collected during June and November 2008 to evaluate the template generation process. We collected 71 emails in the analysis environment advertising the same campaign on November 10th, 2008. From these 71 emails we generated a single template and tested it against the 493 emails from the spam folder. Only 26 emails matched the template (5% detection rate). The reason for this low detection rate is the low diversity of spam emails collected in the analysis environment during the 30 minutes of execution time: they are all the same. If we take a single slightly different email from the 493 emails and add it to the template generation process, the detection rate rises to 26%. If we add another email to the template generation process, we even achieve 99% detection and all that changes within the template is the advertised URL, the text message is untouched. As a result, the number of emails needed to form a good template depends on the diversity of the spam campaign and the time span the template should be used without update. For the example above it suffices to use three emails for template generation to obtain a 99% detection rate.

6 Conclusion

In this paper, we introduced a technique to learn more about current spambots. We execute a spambot in a controlled environment and collect all mails sent from this bot by emulating mail servers. The technique is proactive in the sense that we do not wait for a message to arrive at an end-users mailbox before using it for classification, but we directly interact with the spambots and botnet controllers. This allows us to obtain information about new spam runs earlier than with current approaches. Furthermore, since we only collect spam messages, we can use them to generate filtering rules. We presented a simple approach to enhance current URIBLs, but the collected information can also be used to extract the message template used by the bot. This information can then be used to examine an incoming message, and if it matches the template, it is highly likely spam.

References

1. M. Andreolini, A. Bulgarelli, M. Colajanni, and F. Mazzoni. HoneySpam: Honeypots Fighting Spam at the Source. In *Proceedings of the SRUTI'05*, 2005.
2. I. Androutsopoulos, J. Koutsias, K. V. Chandrinos, G. Paliouras, and C. D. Spyropoulos. An Evaluation of Naive Bayesian Anti-Spam Filtering. In *Workshop on Machine Learning in the New Information Age*, 2000.
3. H. Drucker, D. Wu, and V. Vapnik. Support vector machines for spam categorization. *IEEE Transactions on Neural Networks*, 10(5):1048–1054, 1999.
4. Honeynet Project. Know Your Enemy Lite: Proxy Threats – Port v666, 2008. URL: `http://honeynet.org/papers/proxy/index.html`.
5. J. P. John, A. Moshchuk, S. D. Gribble, and A. Krishnamurthy. Studying Spamming Botnets Using Botlab. In *Proceedings of NSDI'09*, 2009.
6. J. Jung and E. Sit. An Empirical Study of Spam Traffic and the Use of DNS Black Lists. In *Proceedings of the 4th ACM Conference on Internet Measurement*, 2004.
7. J. Kim, K. Chung, and K. Choi. Spam Filtering With Dynamically Updated URL Statistics. *IEEE Security and Privacy*, 5(4), 2007.
8. C. Kreibich, C. Kanich, K. Levchenko, B. Enright, G. M. Voelker, V. Paxson, and S. Savage. On the spam campaign trail. In *Proceedings of LEET'08*, 2008.
9. R. Lemos. McColo Takedown Nets Massive Drop in Spam, 2008. URL: `http://www.securityfocus.com/brief/855`.
10. A. Pathak, Y. C. Hu, and Z. M. Mao. Peeking into Spammer Behavior from a Unique Vantage Point. In *Proceedings of LEET'08*, 2008.
11. N. Provos, D. McNamee, P. Mavrommatis, K. Wang, and N. Modadugu. The Ghost in the Browser Analysis of Web-based Malware. In *Proceedings of HotBots'07*, 2007.
12. A. Ramachandran and N. Feamster. Understanding the network-level behavior of spammers. *SIGCOMM Comput. Commun. Rev.*, 36(4):291–302, 2006.
13. A. Ramachandran, N. Feamster, and D. Dagon. Revealing Botnet Membership Using DNSBL Counter-Intelligence. In *Proceedings of the SRUTI'06*, 2006.
14. M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A Bayesian Approach to Filtering Junk E-Mail. In *Learning for Text Categorization.* AAAI Technical Report WS-98-05, 1998.
15. J. Stewart. Top Spam Botnets Exposed, April 2008. URL: `http://secureworks.com/research/threats/topbotnets/`.
16. J. Stewart. Spam Botnets to Watch in 2009, January 2009. URL: `http://secureworks.com/research/threats/botnets2009/`.
17. S. Venkataraman, S. Sen, O. Spatscheck, P. Haffner, and D. Song. Exploiting Network Structure for Proactive Spam Mitigation. In *Proceedings of 16th USENIX Security Symposium*, 2007.
18. Y.-M. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, and S. T. King. Automated Web Patrol with Strider HoneyMonkeys: Finding Web Sites That Exploit Browser Vulnerabilities. In *Proceedings of NDSS'06*, 2006.

19. C. Willems, T. Holz, and F. Freiling. CWSandbox: Towards Automated Dynamic Binary Analysis. *IEEE Security and Privacy*, 5(2), 2007.
20. Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten, and I. Osipkov. Spamming Botnets: Signatures and Characteristics. In *Proceedings of SIGCOMM'08*, 2008.