Blacklisting Malicious Websites using Peer-to-Peer Technology

Jan Göbel Ben Stock Philipp Trinius Felix C. Freiling
Laboratory for Dependable Distributed Systems
University of Mannheim, Germany

ABSTRACT

The misuse of websites to serve exploit code to compromise hosts on the Internet has increased drastically in the recent years. With new methods like Fast- or Domain Fluxing the attackers have found ways to generate thousands of links leading to malicious webservers in a very short time. With the help of the distributed blacklist solution we propose in this paper we are able to quickly respond to new threats and have the ability to involve different sources to collect information about malicious websites. It is therefore possible to protect networks from threats that they have not even been targeted for yet, by sharing attack information globally.

1. INTRODUCTION

1.1 Malicious Websites and Current Countermeasures

With the increase of security in server based applications, attackers have started to target client side applications, such as the web browsers or document readers. As these applications are installed on almost every host they make a valuable target for an attacker. In order to get people to visit specially prepared websites that exploit current web browser vulnerabilities, links are advertised using email SPAM. Other methods include blog comments, guestbook entries, twitter, or messages distributed across social networks as done by the Koobface worm [1].

One (still rather successful) solution to this problem is aggressive filtering of email SPAM. But SPAM filters only tackle the distribution of malicious URLs through email and not the other distribution paths. The economic damage caused by successful exploits has spawned new commercial security services that offer *URL blacklists*, i.e., lists of URLs that are classified as malicious [14]. Browsers, for example, can be configured to automatically query these lists before visiting a URL. This is the idea behind technologies such as Google Safe Browsing [6], Microsoft SmartScreen [10] and Web of Trust [15]. However, all these systems have problems.

EUROSEC 2010 April 13, 2010, Paris, France.

Google Safe Browsing [6] and Microsoft SmartScreen [10] use a *closed* blacklist, i.e., the way in which their blacklist is managed is rather intransparent which does not contribute to security. In our view, a blacklisting service should be *transparent*, i.e., the way in which the blacklist is built and managed should be clearly documented and accepted by the security community. An implication of transparency is that the blacklist should be *accurate*, meaning that a URL appears on the list if and only if there is evidence that it is malicious.

In contrast to the solutions by Google [6] and Microsoft [10] the *Web of Trust* (WOT) [15] is rather transparent, based (partly) on a community effort that relies on a public voting system to determine if a certain website is to be considered malicious. Any participating user can rate the sites visited in different categories like trustworthiness, vendor reliability, privacy, and child safety. A web browser plugin retrieves the information about the websites while surfing the Internet and displays the information to the user. While this system has a rather high latency to flag a website as malicious, it also relies on the fact that more honest users rate a site than dishonest ones. Furthermore, *all* currently existing solutions follow a centralized approach to manage the blacklist, which does not scale and makes the service vulnerable to denial-of-service attacks.

The problems of current solutions are amplified by the tremendous mass of links leading to exploit sites that are distributed across the Internet. With techniques like Fast- or Domain Fluxing [8] the attackers have great tools at hand to facilitate the process of link generation. As a result, the links to one malicious website, which are received at diverse locations or sensors, may differ. Therefore, localized blacklist services and services that build upon majority voting also have limitations. A blacklisting service should therefore be *transparent*, *accurate* and have a *low latency*.

1.2 Distributed Infrastructures: P2P

The straightforward solution to the requirement of low latency is to use a distributed storage and retrieval infrastructure for the blacklisting service. In particular, peer-to-peer (P2P) technology has proven to be able to serve millions of users at a high speed. The worst case lookup time with P2P protocols like Chord [13], Pastry [12], Kademlia [9], or Tapestry [16] is $O(logN)$, where N is the number of participating nodes. More recent distributed hash table (DHT) algorithms, like Kelips [7], or extensions like Beehive [11] can achieve a lookup performance of $O(1)$, thus, performance issues are negligible. Another advantage of P2P technologies

is that the blacklist can also be maintained in a distributed fashion, i.e., many users can contribute information in a scalable manner.

We are aware of other work that has used P2P techniques to combat email SPAM. Berkes [3] developed the idea of a P2P-based SPAM blocklist. The system can be queried by email servers to filter out SPAM. Technically, the system is implemented using CGI scripts on webservers that form the distributed network. Clients can therefore use regular HTTP requests to retrieve the blocklist entries. The DHT entries contain IP addresses of hosts that are known to send email SPAM. However, with the rise of SPAM botnets blacklisting IP addresses has become less effective.

Another distributed system to fight SPAM is proposed by Brodsky et al. [4]. The prototype is called Trinity and uses Chord as underlying P2P protocol. The main focus is on SPAM generated by botnets. Again this service is meant to be used by email servers, thus it is implemented as a plugin to SpamAssassin. To the best of our knowledge, P2P technologies have not yet been applied to the field of URL blacklisting.

1.3 Contributions

We propose the concept of a globally distributed URL blacklist service built on top of P2P technology. Every time a client of a participating network wants to visit a website it queries the nearest P2P node for information about the domain name of the website in question. The answer includes all information about malicious pages hosted on this domain. Clients that query the blacklist are called *consumers* in this paper.

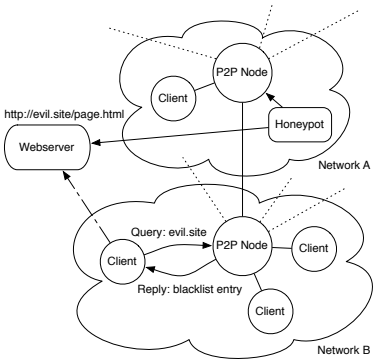

Figure 1: Basic design of the distributed blacklist service.

We explicitly allow different parties to contribute information about malicious websites collected at their location. We call nodes that contribute such information *contributors*. This information can for example be collected using client honeypots that constantly crawl the Internet or use links extracted from SPAM emails as input vector. Allowing many parties to contribute information has many advantages, as can be seen from the following example (illustrated in Figure 1). Consider two networks A and B. Both participate in the distributed blacklist service, i.e., each network runs a node that is connected to the P2P network. Additionally, network A runs a client honeypot system, that visits all URLs found in email SPAM at the local email server. The result of each visit is sent to the local P2P node, which in turn stores the information in the P2P network. The webclients of each network query the blacklist service before visiting any website. If both networks A and B are targeted by the same SPAM campaign with links to malicious webservers that are detected by the honeyclient system of network A all hosts of network B are protected, even if the SPAM filter mechanisms of network B fail.

Without sharing the security relevant information between the networks, the protection of a single network is just as strong as the security mechanisms installed in this network. Of course, information should only be contributed by *trustworthy* sites that also should provide evidence of what they detected. This evidence can be checked, for example, by own client honeypots and can contribute to other forms of trust management and (write) access control to the blacklist. Overall, the more institutions participate in the blacklist service the better is the overall protection for each user querying the service.

The remainder of this paper gives more details of our approach:

- We present an overview of the design of our system in Section 2.
- We explain the P2P protocol that forms the basis of the distributed blacklisting service in Section 3.
- We describe how contributors store their information in the blacklist and how consumers can access this information in Section 4.
- We argue that the system is secure by performing a brief threat analysis in Section 5.

Since the system is still in development, we cannot provide performance measurements or a detailed security analysis. This will be provided in an extended version of this paper.

2. DESIGN

The globally distributed blacklist service has four main components: (1) a protocol that allows interconnection of several thousand nodes plus the ability to provide access to stored information for millions of consumers, (2) an interface for both contributors and consumers, (3) the possibility to remove entries that have been expired, and (4) a method to verify the source that originally provided the information about a malicious website.

The first component is based on current P2P protocols like Kademlia. As these protocols are developed for information exchange between millions of hosts with low insertion times and fast lookup performance they are an excellent choice for this kind of task. However, from a security point of view a few enhancement need to be made.

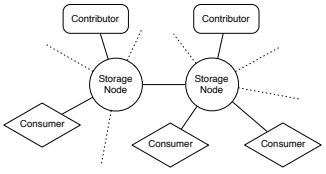

Figure 2: Schematic overview of the different node types.

We chose to add the concept of different types of nodes. Figure 2 illustrates the three basis node types and how they are interconnected. The classical *storage nodes* are responsible to store the information about malicious websites. These nodes make up the core P2P network and provide the information for all other nodes using the blacklist service.

2.1 Data Organization

Besides a method for data insertion we also need an update mechanism to add new data to an existing entry. All entries in the blacklist are stored with the hashed *top-* and *second*-level *domain* name of a website as the key. In the following we refer to the combination of top- and second-level domain as *tsDom*. Since more than just one page of a particular domain can contain malicious content, each entry may consist of a list of hashed subpages. For example, if we want to visit `sample.com/siteX/page1.html` we query the blacklist for the tsDom key generated from `sample.com`. In return we receive a list of hashed subpages that were classified as malicious. We then check whether one of the received entries matches the (hashed) subpage we are trying to visit and act accordingly. If one of the contributors determines that another website hosted at `sample.com` is malicious it needs to perform an update to add this new subpage to the list of existing ones. To avoid inconsistency, storage nodes need to check whether they already have an entry for a particular tsDom key, which needs to be updated, or if a new one needs to be created.

2.2 (Trusted) Contributors

Next to the storage nodes we need *contributors*, i.e., nodes that are able and allowed to store information in the P2P network. These nodes can be any kind of analysis system, e.g. a client honeypot. The P2P protocol must therefore be enhanced by an authentication mechanism. We chose standard public key cryptography for this task. Each network that runs contributors is required to have a key that authorizes the contributor nodes to perform store and update operations on the blacklist service. We therefore rely on a centralized trusted authority (CTA) that issues certificates to trusted members of the set of contributors. The storage nodes only accept signed data from contributors. To identify valid contributors the trusted authority frequently publishes a signed list of valid certificates across the P2P network. Storage Nodes and Consumers can verify the published list with the certificate of the central authority which is integrated into the consumer software. See Section 5 for details.

2.3 Consumers

Finally, we have the *consumers*, i.e., the nodes that represent users that surf the Internet. The most obvious implementation of consumers are plugins for common web browsers, like Mozilla Firefox or Microsoft Internet Explorer. Any end-user that wants to use the blacklist service needs to install such a plugin and configure it to use available storage nodes. A different approach is to configure web proxies to act as both storage nodes and consumers.

As consumers run on end-user computers they are the most untrusted part in the whole blacklisting approach and therefore are only allowed to query the blacklist. That means, these nodes can neither store nor update data in the blacklist.

2.4 Expiry Mechanism

Most malicious websites are online for a rather short period of time. To avoid the delivery of outdated data each blacklist entry contains an expiry date. The storage nodes need to iterate over all their stored entries autonomously and remove the ones that have expired at least once a day. Checking the expiry date of a blacklist entry can also occur upon operations in which it is involved, like query or update operations.

3. P2P PROTOCOL

For a prototype we propose the implementation of a protocol similar to Kademlia [9]. Each storage node is identified by a unique identifier called *nodeID* and every entry that is stored in the distributed hash table (DHT) requires a unique key called *objectKey*. The nodeID is generated from the IP address, the service port, and the MAC address of a storage node by using a function f, for example SHA-1 (secure hash algorithm 1):

$$nodeID = f(ipaddress||serviceport||macaddress).$$

The objectKey equates to the use of function g, e.g. SHA-1, on the top- and second-level-domain name (tsDom) of a URL that is to be blacklisted:

$$objectKey = g(tsDom).$$

```
<dht timestamp = '2012/01/02'>
  <entry objectKey = sha1('sample.com') timestamp = '2012/01/02'>
    <subpage page_hash = sha1('www.|/evil.html')
             sensor_sign = 123
             expire_date     = '2012/01/02'/>
           .....
    <subpage page_hash = sha1('gallery.|/index.html')
             sensor_sign = 402
             expire_date     = '2012/01/01'/>
  </entry>
...
</dht>
```

Figure 3: Example DHT entry of the storage nodes

The DHT entries contain the following information: the objectKey, a list of hashed URLs without the tsDom together with the signature of the honeypot that provided the data, and an expiry date, as displayed in Figure 3. Since all information contained in a DHT entry is hashed, we also preserve privacy for any querying host, as it is not possible without brute forcing to determine the complete website a user is visiting.

The objectKey is also used to determine the storage nodes that store the blacklist information for a certain tsDom. Just as Kademlia we can determine the distance of the nodeID and the objectKey and choose the n closest nodes that store the data. The higher the value n, the more offline nodes are tolerated by the system and the lower the lookup time. There is a fundamental tradeoff between replication and resource consumption: the number of copies of a DHT entry generally improves the lookup performance at the cost of resources.

To spare resources and to consider that malicious websites can turn benign again it is necessary to have an expiry

date (e.g. two days) for each blacklist entry stored in the DHT. That way URLs that are only short lived are removed rather quickly from the list. URLs with a longer lifetime will eventually be advertised more often and therefore will be inserted by contributors more than once. This results in frequently advertised links to stay longer on the blacklist.

Another important aspect is storage node synchronisation. If a storage node is turned off , e.g. to update the system, or loses connection for any reason, no updates to stored blacklist entries can be received. Without synchronisation consumers or other storage nodes querying information from this storage node receive old data. In order to handle synchronisation we favour the following solution: We perform no synchronisation and make the consumers wait for at least n replies in order to compensate $n-1$ storage nodes with old information. Additionally, we make storage nodes forget about all their stored data, and reestablish the data over time. As data is stored at several storage nodes, at least one, which received all recent updates will answer to consumer queries. The low expiry date and the fact that active malicious URLs are added as long as they are frequently advertised, support this solution and allow the network to resynchronise shortly.

To set up the initial P2P network we need at least one running storage node, that others can use to join the network. During the bootstrap process (the initial connection to the network) participating nodes (storage nodes, consumers, and contributors) exchange their knowledge about neighbours, valid contributor certificates, and, in case of storage nodes, DHT entries.

4. INTERFACES

4.1 Consumer Interface

The consumer interface on the client side is realized as a plugin for the web browser. As all common browser applications support third party plugins this is the most easy, secure, and comfortable way of allowing users to take advantage of the blacklist service. Upon the start of the browser the plugin connects to the P2P network using one of several configured storage nodes. Note that a consumer that is connected to a storage node does not become part of the P2P network. Client computers are not reliable enough to serve as storage nodes as they are offline rather often, which would result in too much fluctuation of data in the P2P network.

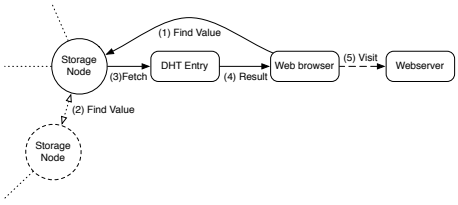

Figure 4: Sequence of a consumer fetching a blacklist entry.

The initial connection to the P2P network is called the bootstrap process. During this process the plugin receives information about other storage nodes and the signed list of currently valid contributor certificates. The certificate of the central trusted authority is also already contained in the plugin to verify contributor certificates. To release the initial storage nodes, consumers randomly choose from the list of received ones for future connections.

Once connected the plugin sends a request to the blacklist service for every URL that is visited. As plugins have full access to the browser controls this can be easily achieved without any additional user interaction. Upon request the storage node either returns the DHT entry for the tsDom, that no entry exists, or that a timeout occured. The basic sequence of a client fetching information for an URL is displayed in Figure 4.

In the first case the plugin extracts the list of subpage hashes and verifies if the URL that the user is trying to visit is contained. Possible parameters within the subpage are cropped before the hash calculation. Although the contributors visit URLs with parameters we crop this information before creating a blacklist entry. We therefore consider a subpage as malicious regardless of the parameters that are provided. Together with the signature of the contributor we can also verify that the data has not been tampered with.

If a website a user is trying to visit is blacklisted, the plugin disables any active content to make it impossible to be harmed by code normally executed on the malicious website. Otherwise the website is loaded and presented to the user.

In the second case, where no DHT entry is found, the website is loaded in a normal fashion.

For the third case (timeout), we suggest to act like the website is blacklisted, otherwise the user might get a false sense of security.

4.2 Contributor Interface

The contributor interface allows the submission of DHT entries for the distributed blacklist. We propose the usage of client honeypots as main contributors, but other systems are also possible.

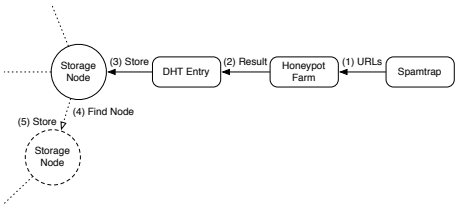

Figure 5: Sequence of blacklist entry contribution.

Contributors connect – just like consumers – to a storage node of the P2P network and follow the same bootstrap procedure.

Whenever a malicious website is detected the information is stored at the blacklist using the store command of the P2P protocol. Figure 5 displays the basic sequence of blacklist entries being added to the DHT. We propose the use of SPAMtraps to collect email SPAM at many different locations, to increase the diversity of received messages, and extract all contained URLs. The extracted URLs serve as input for so called Honeypot Farms, i.e. a network of client honeypots running in parallel to investigate as much URLs as possible.

In case a malicious URL is found the results are stored in a DHT entry and sent to the nearest known storage nodes.

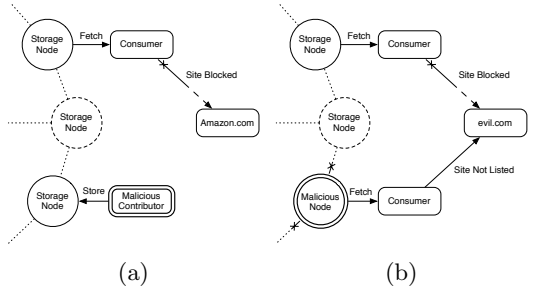

Figure 6: (a) Malicious honeypot contributes false data. (b) Malicious storage node withholds blacklist data.

From this point on the P2P protocol takes care of distributing the information in the network.

5. BRIEF SECURITY ANALYSIS

In this section we discuss four attacks that aim at the most valuable targets of the blacklist service. The carefully selected attacks cover the three main aspects of security analysis, namely: Confidentiality, Integrity, and Availability (CIA).

The first presented attack refers to an adversary that has compromised a single (or few) contributor(s) and is able to store false data in the blacklist, i.e. benign websites are marked as being malicious. The motive of such an attacker is to hinder users to visit popular public webservices, for example *Amazon.com*, to blackmail the owner of that service. The attack is similar to the classical denial of service attack against websites. Figure 6a illustrates the described attack scenario.

To defend against such adversaries, we introduce a public key infrastructure (PKI) for the contributors. Every running contributor needs a certificate that is manually verified and signed by a central trusted authority. We refer to this authority as *CTA* in the following. This certificate must be used to sign any data submitted to the blacklist service to ensure non-disputibility. The central authority is therefore involved in any new contributor joining the service and is also responsible for the distribution of all valid certificates in the P2P network. A list of all currently signed certificates is sent to m storage nodes known to the CTA. This list contains a timestamp and is signed with the CTA's public key. The initial m storage nodes autonomously distribute the list by sending it to all neighbouring nodes. As the list is signed and the CTA's signature can be verified, manipulation is not possible. If a contributor submits blacklist entries, the corresponding storage nodes check the signature against the list of known signed certificates. In case it is invalid, the submitted blacklist entry is dropped. On every update of the signed certificate list storage nodes check their DHT entries to remove entries that are no longer a valid. As a result, it is not possible to contribute data without a valid certificate and in case false data is found, the corresponding contributor can be disabled by removing the corresponding certificate from the CTA's list.

Of course we cannot assure that all the storage nodes can be trusted and follow the procedure of certificate validation

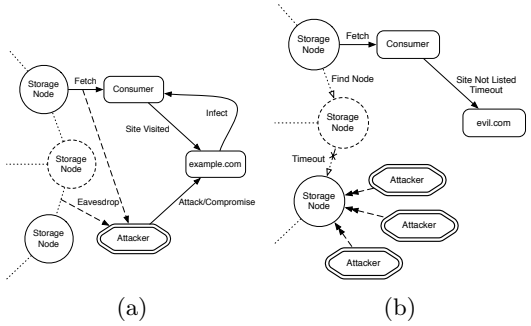

Figure 7: (a) Attacker eavesdrops on connections to initiate targeted attack. (b) A group of attackers performs a denial of service attack on a storage node.

as we intended. Therefore, the consumers also need to have access to the signed list of valid certificates. Everytime a consumer performs the bootstrap process, the latest list of valid certificates is received.

The consumer is therefore able to verify the data returned from the blacklist and can ignore revoked entries without the need to trust the storage nodes.

The attack depicted in Figure 6b describes an attacker that controls one or many storage nodes. The attack can be splitted into two cases: First, the attacker is able to withhold blacklist data from any consumer or storage node that connects to the storage nodes under control of the adversary (*Sybil Attack*). Second, an adversary controlling nodes in the network can also refuse to relay any requests of adjacent nodes and thus cut off part of stored information from the rest of the nodes (*Eclipse Attack*). The motive here is to prevent certain websites from being listed as malicious.

Douceur [5] showed that in a publicly accessible P2P network, the Sybil attack cannot be prevented, thus we can only make it harder. As we do not allow storage nodes to freely choose their nodeID, an adversary cannot assure to be the only one controlling storage nodes that contain DHT entries for certain domains. Thus, the bigger the P2P network grows, the harder it is to perfom this attack. For the case that a consumer connects to a storage node under control of an adversary, we propose the usage of multiple entry points, that are obtained during the bootstrapping process. However, the list of entry points can be forged by a malicious storage node as well, thus we cannot completely defend against this kind of attack.

For the second case, Baumgart and Mies [2] developed an advanced form of the Kademlia protocol – *S/Kademlia*. Their protocol uses parallel lookups over multiple disjoint paths and therefore removes the possibility to disrupt the network by cutting the link.

An Attacker that is able to eavesdrop on connections either between a client and a storage node or between two storage nodes at any point of the P2P network targets the privacy of clients. The motive here is to determine the surf behaviour of a victim to launch targeted attacks. As illustrated in Figure 7a an attacker can learn what sites are visited by a consumer regularly an try to compromise the

webserver in order to get hold of the consumer.

We mitigate this threat by hashing the tsDom and the combination of subdomain and path, i.e. DHT entries do not contain any clear text information about the URLs they represent. As a result, an attacker can not easily determine what websites a consumer visited. However, the hash for prominent domains can be calculated in advance, so that an attacker that eavesdrops between the consumer and a storage node can gain insight knowledge on what top- and second- level domains, i.e. *google.com*, are visited by the consumer. For this kind of attack to work an attacker needs fundamental access to the network infrastructure, which results in more easier ways to determine a consumers surf behaviour that would even reveal the complete URL.

Finally, we describe an attacker that performs denial of service (DoS) against storage nodes of the distributed blacklist. In this case the motive is to take down some storage nodes or the complete blacklist service in order to prevent malicious sites from being listed. Depending on the size of the P2P network and the replication factor n of stored data, this attack is usually uneffective, as the network is designed to tolerate the failure of a certain number of nodes. Figure 7b illustrates the case where several attackers perform a denial of service attack on one of the storage nodes to enforce a timeout to other nodes querying blacklist entries for domains hosted at this node.

6. CONCLUSION AND FUTURE WORK

We described a distributed approach to share information about malicious websites using current P2P technology. The proposed setup consists of three participating parties, namely: storage nodes, contributors, and consumers. The data about malicious websites is stored by contributors in the DHT and retrieved by the consumers prior to visiting a website with the help of a browser plugin. The storage nodes host the DHT and serve as entry points for the consumers. To ensure security aspects, a central trusted entity was introduced, that verifies the contributors. As each contributor signs the data it contributes, consumers can verify it upon retrieval. We presented four of the most threating attacks and showed how to defend against them. However, as with all security mechanisms, there exists no single fail-safe solution, thus additional protection mechanisms are still needed.

The main advantages of our proposed concept are: we are able to share information regarding malicious websites between different networks to improve the overall security and we are much more resilient to attacks than traditional client-server approaches. After all our concept is not limited to blacklisting malicious websites, but can be extended to other aspects of network security as well.

For future work we plan to implement a prototype system in order to measure the effectiveness of our approach.

Acknowledgments

We would like to thank Zinaida Benenson who provided valuable feedback on a previous version of this paper.

7. REFERENCES

[1] J. Baltazar, J. Costoya, and R. Flores. The Real Face of KOOBFACE: The Largest Web 2.0 Botnet Explained. Technical report, Trend Micro, 2009.

[2] I. Baumgart and S. Mies. S/kademlia: A practicable approach towards secure key-based routing. In *ICPADS '07: Proceedings of the 13th International Conference on Parallel and Distributed Systems*, pages 1–8, Washington, DC, USA, 2007. IEEE Computer Society.

[3] J. E. Berkes. Design of a DDoS Attack-Resistant Distributed Spam Blocklist. In *CCN '04: 2nd International Conference on Communication and Computer Networks*, 2004.

[4] A. Brodsky and D. Brodsky. A Distributed Content Independent Method for Spam Detection. In *HotBots'07: Proceedings of the First Workshop on Hot Topics in Understanding Botnets*, page 3, Berkeley, CA, USA, 2007. USENIX Association.

[5] J. Douceur. *The Sybil Attack*, volume 2429/2002 of *Lecture Notes in Computer Science*, pages 251–260. Springer Berlin / Heidelberg, 2002.

[6] Google Inc. Google safe browsing. Internet: `http://code.google.com/apis/safebrowsing/`, Feb. 2010.

[7] I. Gupta, K. Birman, P. Linga, A. Demers, and R. van Renesse. Kelips: Building an Efficient and Stable P2P DHT Through Increased Memory and Background Overhead. In *IPTPS'03: 2nd International Workshop on Peer-to-Peer Systems*, 2003.

[8] T. Holz, C. Gorecki, K. Rieck, and F. C. Freiling. Measuring and detecting fast-flux service networks. In *NDSS*. The Internet Society, 2008.

[9] P. Maymounkov and D. Mazières. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Peer-to-Peer Systems*, pages 53–65. 2002.

[10] Microsoft Inc. Microsoft smartscreen. Internet: `http://blogs.msdn.com/ie/archive/2008/07/02/ie8-security-part-iii-smartscreen-filter.aspx`, Feb. 2010.

[11] V. Ramasubramanian and E. G. Sirer. Beehive: O(1) Lookup Performance for Power-Law Query Distributions in Peer-to-Peer Overlays. In *NSDI '04: 1st Symposium on Networked Systems Design and Implementation*, pages 99–112, 2004.

[12] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.

[13] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, volume 31, pages 149–160, New York, NY, USA, October 2001. ACM.

[14] urlblacklist.com. Urlblacklist.com website. Internet: `www.urlblacklist.com`, Feb. 2010.

[15] WOT Services. Web of trust (wot). Internet: `http://www.mywot.com/`, Feb. 2010.

[16] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: a resilient global-scale overlay for service deployment. *Selected Areas in Communications, IEEE Journal on*, 22(1):41–53, 2004.