Identifying Resource Authenticity in P2P Networks

Li Tang

Department of Automation, Tsinghua University, Beijing, China
tangli03@mails.tsinghua.edu.cn

Abstract. Peer-to-peer (P2P) networks have become immensely popular in recent years. Despite many other advantages, what attracts users most is their anonymity feature. However, anonymity also opens doors to various security threats and malicious behaviors that could severely harm and annoy the users. In this paper, major weaknesses of the present P2P file sharing systems are identified to prove the significance of improving the authenticity, confidentiality, and integrity of the resources to be downloaded. Several approaches are proposed to enhance the security of P2P file sharing: 1) Encrypting respondent messages to queries; 2) Authenticating servents and messages with digital signature; 3) Verifying integrity of resources with secure-hash digest; and 4) Filtering malicious resources before downloading by means of majority-rule-polling. The approaches are demonstrated, without losing universality, using the most popular P2P protocol Gnutella 0.4. Primary analysis demonstrated the effectiveness of the proposed schemes against most of the known network attacks.

1 Introduction

Peer-to-peer (P2P) networks became immensely popular in recent years, primarily because a series of P2P applications that enabled users to share resources without charge, e.g. Napster and various P2P clients based on Gnutella. Although a generalized definition of P2P includes both the client-sever mode and the directly exchanging mode amongst peers, today's main stream P2P technological development represents a paradigm shift apart from the traditional client-server architecture. P2P actually has become for the best example of the network protocols where the roles of all the nodes are equal, and there is no central host or process with specific responsibility to monitor and/or supervise the network behavior. Existing P2P applications can be categorized into four categories.

- *Information sharing*: P2P applications such as Napster, Clients based Gnutella, FreeNet, KaZaA and others that enable Internet users to share information resources.
- *Distributed computing*: P2P systems for distributed computing that share the computational power of participating nodes; a well-known example is SETI@Home.
- *Instant communication*: client programs such as ICQ, MSN Messenger and AOL Instant Messenger (AIM) that allows on-line chatting, voice messaging, and file sharing.
- *P2P collaboration*: human collaboration is intrinsically peer-to-peer, which is reflected by the P2P collaboration applications in software architectures; a famous example is Groove aiming at offering secure platform for hosting peer-to-peer applications.

This paper focuses on the P2P applications for file sharing, a particular type of the information sharing systems. P2P file sharing systems allow any user to share resources while maintaining the autonomy and independence from the centralized servers. Thus they usually have better availability and fault tolerance than the traditional client-server based systems. However, P2P file sharing systems introduce a whole new class of security threats, and they can be exploited to distribute malicious software, such as viruses and Trojan horses, even bypassing the protections of firewalls.

For example, it may be reasonable to trust a single centralized service, but obviously unwise to trust any multitude of anonymous resource-providers in the whole P2P network. Therefore, it is important to identify the authenticity of the resources offered by other peers. Otherwise, a malicious peer can easily deceive other peers, and hackers as well as worm virus can use spoofed identity to damage the whole P2P system.

Even if not taking the security threats into count, due to the lack of assured authenticity of the resources to be exchanged, the overall performance of P2P file sharing systems degrades severely for much time and bandwidth is wasted in downloading unwanted files. When a requestor peer attempts to search certain resources for which he or she does not have much information other than a few key words, the requestor may be confused by the list of results as it is hard to tell which labeled resource in the list is the targeted resource the requestor wanted, if any. In order not to miss some valuable information or absorbed extremely by some resource, e.g. a pop-music searched for a long time, the requestor would most likely download the file. After downloading the file to the local host, the requestor may find the file is actually a trash with an attractive name or description, but time, bandwidth, as well as temporary storage space have been wasted. Unfortunately, other users are likely to be blindfolded to the problem of this file, while the requestor may continually be fooled by many other files that have done this to a lot of other peer users.

In the typical Web environment, users use the perceived reputation of the source to estimate the quality and the level of risk associated to the resource. For example, users may only trust resources offered by servers with good reputation and avoid or refuse resources offered by unfamiliar servers. When they have no idea of the reputation to a server, their choice depends on the tradeoff of their interest level and their judgment of risk level of downloading the resource. However, as it is an anonymous environment, what is natural for the client-server model based Web becomes difficult to adapt in P2P systems where resource providers are identified by a pseudonym and/or an IP address.

Some argue that the most effective solution to such an identity problem within P2P environment is to use a central trusted server, which manages the validity of each peer and the shared resources. However, this brings several essential problems that deviates the spirit of P2P networks. Besides a high management cost, the central server becomes the performance and security bottleneck of the whole P2P networks, and the anonymity is destroyed to a certain extent for the peers' IP addresses are disclosed. Despite other approaches such as reputation models [1, 9] that are proposed to estimate the validity of the peers and resources, it is still far from reaching a fair, efficient and practical mechanism that is robust enough to resist cheat and attack.

To solve these problems, we propose several approaches using the digests of the resources based on secure hash function to provide authenticity; using encryption and digital signature to enhance confidentiality and integrity; using simple polling mechanism to help requestors in choosing resources as reliable as possible. In Section 2, we present the architectures of P2P file sharing applications and take Gnutella 0.4 as an example. In Section 3, the security risks for P2P are analyzed. Then, in Section 4, we introduce our novel approaches to P2P file sharing systems with Gnutella 0.4 as the example. In Section 5, we discuss the security improvements by using our enhancement to the P2P protocol. Finally, we conclude in Section 6 and discuss the future work.

2 P2P File Sharing System

In this section, we present the basic architecture on P2P file sharing applications, and take Gnutella that is one of the most popular P2P protocols as an example to describe the implementation of current P2P file sharing systems.

2.1 P2P File Sharing Applications

File sharing is one of the most successful applications of P2P systems, which has been testified by millions of users and formed a huge resource exchange networks. By September 2002, more than 100 million copies of the KaZaA file sharing application [2] have been downloaded with an increase rate of more than 3 million per week, and several billions of file-exchanges occur monthly on the P2P file sharing networks.

To identify the double responsibility of the nodes in the P2P systems, the concept of "neologism servent" is introduced. When offering its local resources, the servent acts as a server, while it can also act as a client when the servent issues a request and retrieves resources from others.

P2P file sharing applications involve two phases, search and download. The former one aims at looking for a servent that offers certain resource. In the latter one, requesting servent establishes a direct connection with the providing servent, and then starts to download the required resource. P2P file sharing applications generally adopt similar methods for downloading, either merely using the transfer protocols in the TCP/IP family (FTP, HTTP) or annexing some parallel sleight. P2P file sharing applications are mostly characterized by the different implementation in the search phase: pure P2P, centralized index, and distributed architecture with super-nodes. As it only affects the detailed implementation of our proposals, we will not concern the difference of the various implementation described above. In this paper, we focus our work on the ideal implementation that best represents the peer-to-peer spirit, pure P2P, where all nodes have equivalent roles. More specifically, we will refer to Gnutella 0.4 [3] protocol for its most popularity and widely implementation.

2.2 Gnutella 0.4

In Gnutella 0.4, each node participates in the P2P network by choosing a number of directly linked hosts, which constructs an overlay network based on the TCP/IP connections.

To search a file, the requesting servent sends a Query message containing the requested resource to every directly linked servents. Then the Query message continues to be forwarded by broadcast to further servents hop by hop. Upon receiving the Query message, each servent starts to look up their repository for the requested resource and reply with a QueryHit message if anything appropriate is found. The QueryHit message, containing the information needed for downloading the resource, is passed back along the same path as the Query message comes, but reverse direction. Even if the servent has found some results in its own repository, it will continue to broadcast the Query message to the neighboring servents except the provider of the Query message.

In other words, each Gnutella servent acts as a router for query forwarding. To avoid overloading the network, Time-To-Live (TTL) is used to limit the number of involved servents for each Query message. Every time after passing through a servent, the TTL value is decreased by one. The messages with its TTL reaching zero are dropped. As the TTL brings a limit on the servents'

reachability, each servant is actually able to interact with only a portion of all the servents in the overlay network. A servent's reachability also depends on the number of its connections with the neighbors (typical value is in the range of 2 to 6), which is chosen by considering the available bandwidth directly to the node and to the network infrastructure.

3 Security Threats to P2P File Sharing Applications

Although it is the anonymity feature that makes P2P such a great success, anonymity also opens doors to many possible misuses and abuses that exploit the network to spread malicious content. We take Gnutella 0.4 as an example to identify the disadvantages and security risks of the present P2P file sharing applications.

The 0.4 version of Gnutella protocol provides an almost ideal environment for the self-replicating malicious agents primarily due to two main features of P2P's design: the anonymity in the peer-to-peer communication and the variety of the shared files. The anonymity feature involves weakness because each individual servent for the combination of low accountability and over-trust. In a traditional Internet-based transaction, the administrator would be notified when discovering malicious content, so Napster with a centralized index server can disable a user's account if he or she was caught distributing malicious content. However in Gnutella 0.4, every servent with equal right can continuously provide spoofed content to any search request, which is difficult to prevent even by blacklisting the hostile IPs because of many servents using dynamic IP. Moreover, without trusted central supervisor in pure P2P networks, there is no mechanism to propagate the deceiving information to other servents.

Various attacks relying on the anonymity have been observed. For instance, the well-known VBS.Gnutella worm (often wrongly referred the Gnutella virus) spreads by making a copy of itself in the Gnutella program directory; then it modifies the Gnutella.ini file to allow sharing the .vbs format files in the Gnutella program folder. Another Gnutella worm called Mandragore [4] establishes a connection with local Gnutella client and pretends an active neighboring peer with local IP address. Then Mandragore will respond affirmatively to any intercepted request with a renamed copy of itself for the requesting peer to download.

The variety of file formats allowed by P2P file sharing applications, including executable binary code, is another favorable factor for spreading malicious resources. Furthermore, not only the risk presents when a user downloads certain executable content, but also audio and video files may harbor security threats as multimedia formats permit the introduction of links and active content that may be exploited to introduce unwanted software into a computer. For example, on April 2002 the BugTraq message revealed how an MP3 file played by WinAmp could be configured to execute arbitrary commands [5].

4 Enhanced Gnutella-Based Protocol

In this section, after giving the basic assumption, we present several novel approaches to increase the security of P2P file sharing applications. The approaches focus on enhancing the authenticity, confidentiality, and integrity of the results in the QueryHit messages before the requestor starts downloading. We will illustrate our improvements without losing universality based on Gnutella 0.4.

4.1 Basic Assumptions

Without affecting anonymity, each servent is required generate a pair of keys for asymmetric encryption. Our approaches encourage the use of fixed key pairs, but they are also allowed to be refreshed at each interaction. We use (PK_p,SK_p) to denote a pair of public and private keys associated with servent p, $\{M\}_K$ and $[M]_K$ to denote the encryption and signature of a message M by key K, respectively. The identifier of each servent, *servent_id*, is either the public key itself or a digest obtained with a secure-hash function [6]. Each resource is associated with an identifier, *resource_id*, computed as digest from the resource content with certain secure-hash function. *f(param1,param2,…)* denotes an algorithm that obtains main features such as the secure-hash functions have: one-way, weak collision resistance and strong collision resistance [7]. The detailed realization of *f(param1,param2,…)* is immaterial, but the following techniques may be worth considering: using a secure-hash function (maybe expensive) or using some other one-way algorithm computing on random selected parts of the resource (weaker but much cheaper [8]).

Each servent is required to maintain a local table to record its own attitude based on the exchange history for the resources in P2P networks, specifically, the reputation of the servents from which it has downloaded resources and quality of the resource itself. This table is organized with the keywords used in the query as index and the corresponding results (servent reputation and resource quality) as item value. The table is used to give requestors querying certain keywords some evaluations on the responding servents and their resources, if this servent queried the same keywords before.

To illustrating the communication procedures, we use p to denote the requestor servent, S to denote the set of servents connected to the P2P network at the time when p sends the query, R to denote the subset of S responding to the query (responders), E to denote the subset of S giving evaluation of resources or suggestion to p (estimators) and O to denote the subset of S claiming to offer resources to p (providers). Obviously, R is the union of E and O.

4.2 Enhanced Protocol

We improve the standard protocol in the following steps: *Query Message*, *QueryHit Message* and *Resource selection*.

- Step1: *Query Message*. On searching some resources, like in the standard Gnutella interchange, the requestor p broadcasts to all its neighbors a Query message containing the search keywords as well as p's public key PK_p which is used for the responders to encrypt the reply messages. Whether the parameters of *f(…)* are included is optional and depends on the detailed implementation of the algorithm for *f(…)*. Using additional parameters besides the resource file itself in *f(…)* prevents malicious peers from middleman attack by simply caching the value of *f(res)* (*res* denotes a resource file) that is much smaller instead of the real file *res* for spoofing. If some parameters offered by the requestor are sent together with the Query message, those that do not really possess *res* will not be able to figure out the correct result of *f(res,other_params)*. As the requestor's public key can appropriately play the role of *other_params*, here we assume no more parameters are offered by the requestor, for the sake of briefness in the following discussion.

- Step 2: *QueryHit Message*. There are two types of QueryHit messages in the enhanced protocol, namely *TypeO* and *TypeE*. Once receiving a Query message, each servent in O, which has the

resources matching the Query, responds with a *TypeO* QueryHit message. Compared with the standard QueryHit Message including *num_hits* the number of matching files, *ResultSet* a set of triples containing the files' names and related information, *speed* in Kb/second of the responder, *servent_id* of the responder, *<IP,port>* the pair useful for downloading the files, and *trailer* the field for carrying application-dependent information, the enhanced *TypeO* QueryHit additionally contains the *resource_id*, as well as the value of *f(resource,PKp)*, for each resource named in the *ResultSet*. Furthermore, the whole message is encrypted with the requestor's public key PK_p.

The servents in *E*, which offer evaluations according to their local tables, respond the requestor with a *TypeE* QueryHit message that is quite different from the standard and *TypeO* ones. First, the estimators find out, in their local table, the records indexed by some keywords similar to those in the recently received Query message. Then the records including *resource_id*, corresponding provider's *servent_id* and the estimator's evaluations are encrypted with the estimator's private key for signature. Finally, the cipher-text as well as the estimator's public key buildup the *TypeE* QueryHit message that is also encrypted with the requestor's public key PK_p.

Compared with the standard one in Gnutella 0.4, the formats and contents of the two types of enhanced QueryHit messages are shown as follow.

- *Standard QueryHit:*
  (*num_hits* || *ResultSet* (*<names,others>*) || *speed* || *servent_id* || *<IP,port>* || *trailer*)
- *Enhanced TypeO QueryHit:*
  ({*num_hits* || *ResultSet* (*<name,resource_id,f(resource,*PK_p*),others>*) || *speed* || *servent_id* || *<IP,port>* || *trailer* (*Enhanced O*)}$_{PKp}$)
- *Enhanced TypeE QueryHit:*
  ({ [*num_hits* || *ResultSet* (*<names,resource_id,offerer's servent_id,others>*) || *speed* (*NULL*) || *servent_id* || *<IP,port>* (*NULL*) || *trailer* (*Enhanced E*)]$_{SKestimator}$ || $PK_{estimator}$ }$_{PKp}$)

- Step 3: *Resource selection*. Upon receiving the QueryHit messages, the requestor *p* decrypts every one with its private key SK_p, and then continues to decrypt those of *TypeE* with the estimators' public key $PK_{estimator}$. Drop the invalid messages, and filter out the reduplicate ones. Afterwards, the remained *TypeO* QueryHit messages are clustered into several classes according to the *resource_id*. To prevent some malicious servents from storing only *resource_id* rather than actual resources for spoofing, we farther divide each class into several subsets by the value of *f(resource,*PK_p*)* and reserve only the largest one. After such processes, all the *TypeO* QueryHit messages in each left subset are considered to provide resources of the same content, so every subset can be represented by the *resource_id* in the *TypeO* QueryHit messages.

  The rating of the resources provided by each subset is calculated by polling. All the reserved subsets are candidates and the *TypeE* QueryHit messages are votes, of which each can evaluate one subset associated to certain *resource_id* only once. The resources corresponding to each subset are finally recommended to the requestor according to some well-chosen discriminant function that increase monotonically with the number of the praising (positive or "for" vote) *TypeE* QueryHit messages, $Num_{praising}$, and decrease monotonically with the number of the criticizing (negative or "against" vote) ones, $Num_{criticizing}$. For briefness, one may choose the discriminant function as $(Num_{praising} + 1) / (Num_{criticizing} + 1)$, using majority rule and every vote with equivalent weight in the poll.

  We have proposed a series of approaches for selecting reliable resources in P2P networks, in the three steps described above. Spending minor extra computing power of the involved servents, the approaches not only diminish the download risk but also prevent users from wasting time and

bandwidth on the spoofing resources. Better reputation models [1, 9] and even social voting theory can be used to improve the *Resource selection* method.

5 Security Improvements

The main purpose of our enhancement is to improve the overall security and performance of current P2P file sharing applications. Without losing universality, we continue to use Gnutella 0.4 as reference to analyze the effectiveness of the approaches against the ordinary network attacks. Throughout this section, we assume Alice to be a Gnutella requestor servent searching for a file, Bob to be a provider servent having what Alice wants, Carl to be an estimator servent giving Alice evaluation of some other servents and resources, and David to be a malicious servent.

5.1 Against Interruption, Interception, Modification and Fabrication

The enhancement to the standard protocol can easily avoid these most common security threats. As the Query of Alice is broadcasted to other peers, David is unable to interrupt all the messages along the overlay connections except the one passing through itself, which even doesn't affect Alice's searching at all.

It obtains no benefit for David to eavesdrop, modify or fabricate Alice's Query message, for he can obtain the same results by searching himself. Unless obtaining Alice's private key, David is unable to eavesdrop, modify or fabricate Bob's and Carl's QueryHit messages encrypted with Alice's public key. Whereas, the only way for David to achieve interception, modification or fabrication seems the Man in the Middle attach, which is also unavailing by our later analysis.

5.2 Against Distribution of Tampered Information

This attack makes use of the fact that before downloading some resources there is no way to verify their contents. A nasty attack is for David to simply respond with a tampered resource renamed as what Alice has requested. The actual file may contain a Trojan program, a virus (like VBS.Gnutella worm mentioned above) or a spam such as advertisement. Such attack is particularly familiar in the Gnutella networks as it virtually requires no hacking of the software client. One may argue that it seems unreasonable for David to sacrifice time, bandwidth and computing power to deceive Alice. What benefits will David get? The most possible motivation for David may be the same as those always sending spam emails, for either commercial or political purpose. Furthermore, those, whose intellectual property rights have been much violated by the exchange of copyright materials with P2P file sharing applications, seem to have more reason to hope the P2P networks crash.

Our approaches enable P2P file sharing applications to survive such attacks by means of the utilization of the one-way function $f(resource,PK_{Alice})$, where *resource* is what Alice wants and PK_{Alice} is Alice's public key, as well as the *Resource selection* phase. In order to deceive Alice to download his tampered file, David must assure his QueryHit message pass through Alice's *Resource selection* phase. Therefore, David has to possess a real copy of the most popular resources that will win in Alice's *Resource selection* phase, otherwise, David cannot figure out the correct value of $f(resource,PK_{Alice})$ and his QueryHit message will be invalid and dropped. However, as it is too expensive to store all the popular resources for each Alice's possible query, David has to find

other way to acquire the value of *f(resource,PK$_{Alice}$)*. Again because other provider servents' QueryHit messages are encrypted with Alice's public key, David has no choice expect the Man in the Middle attack.

5.3 Man in the Middle

As mentioned in the above two sections, Man in the Middle becomes the only effective way for David to carry out his attack. Man in the Middle attack takes advantage of the fact that David can be in the path between Alice and Bob or between Alice and Carl, for which P2P architecture provides more convenience than the traditional client-server model. As shown in Fig.1, David performs Man in the Middle attack between Alice and Bob as follow.

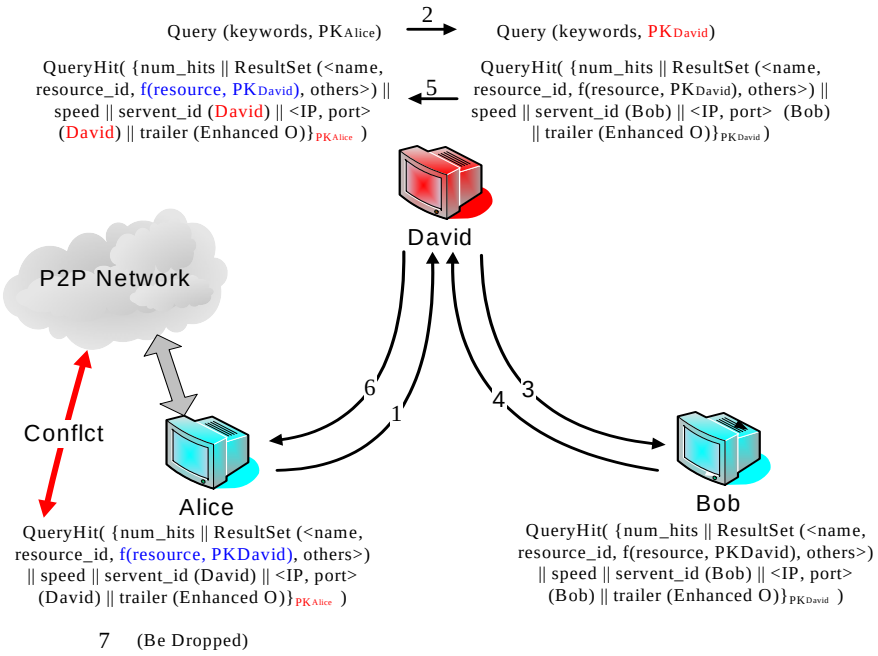

Fig. 1. Avoiding Man In the Middle attack

1. Alice broadcasts a Query.
2. David intercepts Alice's Query and replaces Alice's public key with his own public key.
3. David passes the changed Query to Bob and Bob responds as usual.
4. David decrypts Bob's *TypeO* QueryHit message and rewrites it with David's *IP*, *port*, *servent_id*, whereas David has no idea how to modify the value of *f(resource,PK$_{David}$)* to *f(resource,PK$_{Alice}$)*. Then David encrypts the changed QueryHit message with Alice's public key and passes it to Alice.
5. Alice receives David's QueryHit message and decrypts with her private key.

In the enhanced protocol, all of David's attempts are fruitless. David's QueryHit message will be dropped as invalid reply message during Alice's *Resource Selection* phase, because the message still contains *f(resource,PK$_{David}$)* calculated by Bob for David instead of *f(resource,PK$_{Alice}$)* that is

uniform in majority of the QueryHit messages returned by the natural provider servents. Further more, Alice discovers David's malicious behavior which is recorded in the local table and is able to remind other servents as estimators in future.

David's attacking between Alice and Carl is also impossible. Since Carl's *TypeE* QueryHit message is signed with Carl's private key, as far as David cannot figure out Carl's private key, he is unable to modify or fabricate Carl's evaluation by Man in the Middle attack.

6 Conclusions

In this paper we have analyzed the necessity to identify the resource authenticity in P2P networks account for both security and performance benefits. To achieve it, we proposed several approaches, such as encrypting respondent messages with the requestor's public key, authenticating servents and messages with digital signature, verifying integrity of resources with secure-hash digest, and filtering malicious resources before downloading by means of majority-rule-polling. It is demonstrated that such approaches obtain significant security improvements against various popular network attacks, including interruption, interception, modification and fabrication; the approaches are also very effective in restraining the malicious peers from distributing tampered information and performing Man in the Middle attacks. Though the approaches are mainly demonstrated using Gnutella 0.4, there is no obstacle to adopt them in other P2P file sharing applications.

The approaches are based on the assumption that the majority of servents in the P2P networks are friendly. In practical resource exchanges, the assumption is usually true but exception exists. If enough malicious servents complot together, they may have chances to dominant certain local areas or part of communicating procedures. In such cases, the mechanisms proposed in this paper are likely to be compromised. Fortunately, more robust and secure trust-models can be designed out by leveraging the well understood server-client security solutions, even as well as some social reputation and voting models.

Acknowledgements

The author would like to greatly thank Prof. Jun Li for numerous enlightenment and suggestions. The author would also like to thank Jin Zhou for helpful discussions. The author would finally like to thank the anonymous reviewers for their work.

Reference

1. E. Damiani, D. C. di Vimercati, S. Paraboschi, P. Samarati, and F. Violante. "A Reputation-Based Approach for Choosing Reliable Resources in Peer-to-Peer Networks". In *Proc. 9th ACM Conference on Computer and Communications Security*, pp. 207-216. ACM Press, 2002.
2. KaZaA. http://www.kazaa.com.
3. The Gnutella Protocol Specification v0.4 (Document Revision 1.2), June 2001. http://www9.limewire.com/developer/gnutella_protocol_04.pdf
4. Mandragore. http://www.openp2p.com/pub/a/p2p/2001/03/22/truelove.html?page=4.
5. BugTraq. http://online.securityfocus.com/archive/1/269724.

6. P.C. van Oorschot A.J. Menezes and S.A. Vanstone. “Handbook of Applied Cryptography”. CRC Press, 1996.
7. William Stallings. “Network Security Essentials: Applications and Standards (Second Edition)”. Upper Saddle River, New Jersey 07458, 2003.
8. P. Deutsch and J.L. Gailly. “ZLIB compressed data format specification version 3.3”. RFC 1950, IETF, May 1996.
9. F. Cornelli, E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. “Choosing Reputable Servents in a P2P network”. In *Proc. Of the Eleventh International World Wide Web Conference*, Honolulu, Hawaii, May 2002.
10. S. Marti, H. Carcia-Molina. “Identity Crisis: Anonymity vs. Reputation in P2P Systems”. *Proceedings of the Third International Conferences on Peer-to-Peer Computing*. IEEE, 2003.
11. G. Caronni, M. Waldvogel. “Establishing Trust in Distributed Storage Providers”. *Proceedings of the Third International Conferences on Peer-to-Peer Computing*. IEEE, 2003
12. D. S. Wallach. “A Survey of Peer-to-Peer Security Issues”. Rice University, Houston, TX77005, USA.
13. D. DeFigueiredo, A. Garcia, and B. Kramer. “Analysis of Peer-to-Peer Network Security using Gnutella”.