On the Security of Polling Protocols in Peer-to-Peer Systems

Bartlomiej Sieka*
Computer Science Dept.
Univ. of Illinois at Chicago
Chicago, IL 60607-7053
bsieka@cs.uic.edu

Ajay D. Kshemkalyani*
Computer Science Dept.
Univ. of Illinois at Chicago
Chicago, IL 60607-7053
ajayk@cs.uic.edu

Mukesh Singhal†
Dept. of Computer Science
The Univ. of Kentucky
Lexington, KY 40506-0046
singhal@cs.uky.edu

Abstract

The peer-to-peer (P2P) network model differs from the well established client-server model in that all members of the network are assigned an equal role. P2P networks are recently gaining increasing popularity. Providing security in distributed content sharing in P2P networks is an important challenge. This paper identifies security vulnerabilities in the protocols for sharing servents' reputations in the Gnutella P2P system, proposed recently. It demonstrates attacks on the protocols that allow an attacker to alter the results of the voting procedure. The paper then presents a protocol that is resilient to the attacks described. In the proposed protocol, enhanced security against various attacks is achieved using smart design and a combination of various techniques such as the use of digital signatures for message integrity and random numbers for message freshness.

1. Introduction

Peer-to-peer (P2P) network systems are an important and rapidly growing paradigm for Internet communication. A P2P system is an alternative solution to the client-server model to establish communication between nodes in a network, where all the participating computers/nodes are able to interact directly with each other. Any node can act as a server to others and at the same time, it can play the role of a client (hence the name used to describe a node of a P2P network: *servent*). Communication and exchange of information is performed directly between the participating peers. P2P networks exhibit a high level of self-organization and are able to perform very efficiently despite the lack of any prior infrastructure or authority. The philosophy of this model requires that "if you want to enjoy the services which other nodes provide, you should provide service to other nodes". That is to say, under P2P computing model, the relationships among the nodes in the network are equal. Examples of P2P systems include Napster [15], Gnutella [10, 11], Freenet [4], Pastry [17], Tapstry [23], Chord [20], P2PWNC [9], Proem [14], and CAN [16].

P2P file sharing networks are an important and rapidly growing part of the Internet communication. Millions of people are using P2P networks today to share text, software, audio, and video files stored on their computers [7]. By helping users communicate directly with minimal (or in some cases without) central coordination, P2P networks can allow people to share data with far greater freedom and flexibility than any existing internetworking technology. P2P file sharing networks differ from other Internet applications in that they tend to share data from a large number of end user computers rather than from the more central computers we generally think of as Web servers. There are several well known commercial products available that permit P2P file-sharing.

However, the use of P2P file-sharing software can raise serious security issues [3, 6, 13, 19], as often sensitive personal information can be at risk due to improper usage by users. At the same time, file-sharing technology is largely user controlled, which is sometimes beneficial but hard to regulate. In addition, to fully realize the potential of the P2P paradigm, such systems must be able to support an open environment where mutually distrusting parties with conflicting interests are allowed to join. Even in a closed system of a sufficiently large scale, it may be impossible to assume that no participating nodes have been compromised by attackers. In such environments, where there are many diverse parties without a pre-existing trust relationship, the security is particularly important and nontrivial. However, most P2P systems do not take security into consideration by design. A number of security attacks on P2P systems are possible including denial-of-service attacks, replay attacks, collaborated or un-collaborated attacks by malicious nodes, incorrect routing updates, black hole attacks (modifying the

* This research was partially supported by NSF grant CCR-9875617.
† This research was partially supported by NSF grants IIS-0242384 and IIS-0324836.

routing message to say a node has the shortest path to some nodes) and worm hole attacks (collusion by two malicious nodes to make the packets they want flow through them). In addition, management of trust information of nodes and access control to resources are two important issues.

In this paper, we are interested in the security of the Gnutella system, specially the security of its polling protocols. Because of its trust in intermediate third parties, Gnutella is susceptible to malicious behavior of nodes [22]. However, if one considers the new paradigm of attacks where services are targeted as opposed to hosts, there has been little work done. In the Gnutella protocol, intermediate parties are able to see a significant share of the queries from all servers within their local sub-graph. What damage could be inflicted upon the network if a node misbehaves and uses query information? Every super node has the ability to see a large proportion of the queries. It may be unclear how much damage can be inflicted on the network by a node's ability to listen to much of the query and response traffic, but it is certainly clear that anonymity is compromised. Furthermore, even in the best case, this opens the P2P network to other attacks.

Background on the Gnutella P2P System: The Gnutella P2P system is fully decentralized without any central server and without any storage structure [10, 11]. The Gnutella is basically a resource search and discovery system. It helps a node in the system locate other nodes that have the resource/file that the former node is looking for. After a node with the needed resource is located, the former node can directly get the resource from this node. The Gnutella protocol uses five types of messages: Ping, Pong, Query, QueryHit, and Push. A node sends a Ping message to discover more nodes on the Gnutella network for future file searches. A Pong message is sent in response to a Ping message. A node sends a Query message to search resources in the Gnutella network. A QueryHit message is sent by a node that received the Query message and has the requested resource. The Push message is sent to download a file from a firewalled node. Ping and Query message are broadcast in the system. These messages have a time_to_live (TTL) field that controls the number of hops they will travel.

To search for a resource/file, a node broadcasts a Query message to its neighbors. When a neighbor receives this message, if it does not have the resource being searched, it decrements the TTL field and broadcasts it to its neighbors provided the TTL has not reached zero. A node that has the requested resource informs about it to the initiator of the Query message by sending a QueryHit message. A QueryHit message is unicast and is sent along the same path (in the reverse direction) on which the corresponding Query message arrived.

In the Gnutella protocol, messages are sent in plaintext, and there is no message encryption, authentication or integrity check. Therefore, the Gnutella P2P system is vulnerable to many security attacks like message flooding, denial of service attacks, message replay attacks, man-in-the-middle attacks, etc. In this paper, we address several security problems, such as those due to replay attacks and man-in-the-middle attacks, in the polling protocol used to collect the reputation of servents in the Gnutella P2P system.

The rest of the paper is organized as follows: Section 2 describes the basic and enhanced polling protocols for the Gnutella system proposed in [5]. Section 3 presents an attack on one of the crucial messages used in those protocols (namely the `PollReply` message). Sections 4.1 and 4.2 describe attacks on the original Basic Polling Protocol and Enhanced Polling Protocol, respectively. Section 5 gives the proposed solution that counters these attacks and then makes a correctness argument. Section 6 concludes the paper.

2. Polling Protocols of Gnutella

As the P2P networks' main area of application is distributed content sharing, an important issue is that of the peers security against malicious actions of others. The main problem is that such networks do not provide accountability mechanisms. Often a node finds itself in a situation that requires an interaction with a complete stranger. One of the ways of dealing with this problem is to employ reputation based techniques [1, 2, 12, 21]. In this approach, nodes maintain information about peers' behavior and decide on their actions based on this information. Nodes can gather reputation from their own past experiences with other nodes, but also can rely on the reputation as reported by other members of the network. Usually a combination of these two mechanisms is used. To address the above issues, authors in [5] propose two protocols (Basic Polling Protocol and Enhanced Polling Protocol) that allow servents to gather information about other nodes' behavior and use it to make informed decisions on whom to trust. They propose a voting procedure by which servents can learn and also express their view of peers' reputations. The additional phases of polling and vote evaluation are introduced to the original P2P protocol and a verification procedure is added before downloading. The reputations' sharing method proposed is designed to work with Gnutella. Note that the polling protocol we describe in this paper can be applied to any P2P network and thus it can serve as a building block for a broader spectrum of reputation management mechanisms.

The protocols proposed in [5] consist of four phases.

Phase 1 (Resource Searching): In this phase, the initiator broadcasts a request to peers indicating that he wants to download a certain file. Peers that have the requested file and wish to allow the download respond to the initiator. This phase is unchanged from the Gnutella pro-

tocol and is also the same in both Basic and Enhanced Polling Protocols.

Phase 2 (Polling): This is a new phase introduced by the authors of [5]. The initiator solicits other peers' opinions (votes) about a given set of offerers (i.e., servents that sent their replies in phase 1). He broadcasts the `Poll` message and receives `PollReply` messages. In the Basic Polling Protocol, votes are cast anonymously and the integrity of the `PollReply` messages is provided by the use of a hash function. In the Enhanced Polling Protocol, voters do disclose their $servent\_id$ when voting and digital signatures are used to guarantee the integrity of `PollReply` messages. A formal description of the phase 2 steps for both proposed protocols is given in Figure 1.

Phase 3 (Vote Evaluation): This is also a phase added to the Gnutella protocol. It is used to perform integrity check of the votes and authentication of the votes. In both protocols, this phase is carried out via a direct secure communication channel that is outside of the P2P network. The basic protocol verifies the integrity of votes and their authenticity. The enhanced protocol only authenticates the servent (vote integrity is provided by means of the digital signature in phase 2). A formal description of the phase 3 steps for both proposed protocols is given in Figure 2.

Phase 4 (Resource Downloading): This phase is enhanced from its original Gnutella version by checking if the source of the download is really a node with a given $servent\_id$. This is designed to thwart the man-in-the-middle attack during the download (i.e., impersonating the source of the download).

Refer to Figure 3 for an example message flow for the basic variant of the protocol (due to space limitation we omit the example of the Enhanced Protocol).

The described protocols address the very important issue of security in P2P networks and aim at providing an effective solution. As it turns out, however, the approach is vulnerable to malicious actions by peers. Authors in [5] propose to use a hash function to ensure message integrity. In the next section, we demonstrate how this integrity check can be compromised after circumventing confidentiality of the `PollReply` message. Furthermore, even though the use of direct transmissions outside of the P2P network and the use of public key encryption for integrity and confidentiality are assumed in [5], it is still possible to perform a successful attack on the proposed voting protocol. We present ways of affecting servents' reputations in an unauthorized way. We then demonstrate how the security flaws can be overcome by proposing a modified protocol.

3. Attack on the Confidentiality of the `PollReply` Message

Before concentrating on the confidentiality of the `PollReply` message, let us first briefly discuss what kind of information might be worth keeping secret during a voting procedure. Assuming that Alice is one of the peers in the network, we may want to ensure the following properties.

1. Alice does not know what other peers' opinions about her are.
2. Assuming that Alice knows the others peers' opinions, she does not know who expressed which opinion.
3. Even though Alice forwards a `PollReply` message, she does not know the opinions contained in it.
4. Even though Alice forwards a `PollReply` message and knows the opinions, she does not know who expresses which opinion.

The first two properties are difficult to achieve due to the nature of the P2P network and the anonymity of the communication. In particular, since `Poll` messages are anonymous, any servent can broadcast such a message inquiring about any set of servants (including himself) and gain the knowledge about how others vote.

The aim of the `PollReply` message being confidential is to ensure the last two of the above properties. As it turns out, hiding the contents of this message is not an easy task. The use of the public key encryption for the `PollReply` message does not ensure its confidentiality. Since the public key to be used for encryption is simply broadcast over the network, the following man-in-the-middle attack is possible.

1. Alice broadcasts `Poll`(T, PK_{poll}).
2. Mallory generates a (public, private) key pair (PK_{fake}, SK_{fake}) and sends `Poll`(T, PK_{fake}).
3. Mallory receives `PollReply`$(E_{PK_{fake}}(contents))$. He then decrypts the `PollReply` message with SK_{fake}, encrypts it again with the original poll key PK_{poll}, and forwards it to Alice.

This attack involves steps 2.3 and 2.4 of the original approach. Lack of `PollReply` confidentiality allows an attacker to selectively discard votes. An attacker can easily make the message fail the integrity check performed by the recipient, thus canceling all the votes contained in it. This is successful, since the poll originator is supposed to discard votes that do not pass the integrity check. Since votes are effectively not confidential, an attacker can drop a message that contains votes unfavorable for him or others that

Basic Polling Protocol	Enhanced Polling Protocol
2.1 Select list of offerers	2.1 Select list of offerers
2.2 Generate public-secret keys (PK_{poll}, SK_{poll})	2.2 Generate public-secret keys (PK_{poll}, SK_{poll})
2.3 Poll peers about set of offerers T: `Poll`(T, PK_{poll})	2.3 Poll peers about set of offerers T: `Poll`(T, PK_{poll})
2.4 Receive from servent i, $\forall i \in$ {peers that reply to the poll}	2.4 Receive from servent i, $\forall i \in$ {peers that reply to the poll}
`PollReply`$(E_{PK_{poll}}(IP_i, port_i, Votes_i, h(IP_i, port_i, Votes_i)))$	`PollReply`$(E_{PK_{poll}}(IP_i, port_i, Votes_i, servent\_id_i, S_{SK_i}(IP_i, port_i, Votes_i, servent\_id_i), PK_i))$

Figure 1. Step 2 of the polling protocols

Basic Polling Protocol	Enhanced Polling Protocol
3.1 Remove suspicious voters	3.1 Remove suspicious voters
3.2 Verify votes by sending `TrueVote`$(Votes_i)$ to servent i, $\forall i \in$ {peers that require verification}	3.2 Check identity by sending `AreYou`$(servent\_id_i)$ to servent i, $\forall i \in$ {peers that require verification}
3.3 Receive `TrueVoteReply`$(response_i)$ from servent i	3.3 Receive `AreYouReply`$(response_i)$ from servent i

Figure 2. Step 3 of the polling protocols

he colludes with. Also, he can decide to tamper with messages carrying positive votes of servents that he wishes to harm.

As with any man-in-the-middle attack, this one also requires the attacker to be "close" to the target in order to be able to intercept and inject messages (i.e., to be in the "middle"). Assuming that nodes in a P2P network are not mobile, it might at first seem that such an attack is difficult to mount, since it is not easy to get close to the target. If the target is fixed, i.e., when a malicious node wants to attack a particular peer, getting "close" to the target might indeed pose some challenge. Note however, that there exists a reversed scenario, in which the attacker does not have to get into the victim's proximity. A malicious node can simply decide to attack only those nodes, that are sufficiently "close" to perform the attack. So even if we assume that the man-in-the-middle attack can be successfully mounted only against the neighboring nodes, it can still pose a serious threat to the P2P network. As we will show in the following sections, a malicious peer can influence the results of the polling procedure. Given the fact that the polling results are used by nodes to form opinions about others, and the fact that opinions can be propagated through the network, it can be seen that the consequences of a localized man-in-the-middle attack can in fact be network-wide.

As already mentioned, assuring the confidentiality of communication in a P2P setting is not trivial. Moreover, it is our conjecture that in the absence of a Certificate Authority or a Trusted Third Party, a secure and anonymous communication between two entities not sharing a previous common context is not possible. A possible way of overcoming this limitation is to allow for an initialization phase, in which cryptographic material can be preloaded into nodes. Alas, this approach does not seem to be very well suited to the P2P scenario.

Note that one must carefully consider the tradeoffs between the cost of public key encryption and the level of security it provides. As it was demonstrated, the confidentiality of the `PollReply` message can be broken. The attack requires some effort though, and there may exist circumstances where the limited security provided by encrypting the message is worthwhile. This however, comes at the cost of using public key encryption and also requires a key pair generation for every poll, both operations being computationally intensive.

4. Attacks on the Polling Protocols

In this section, we present two attacks targeted at influencing the voting procedure. We start with an attack on the basic version of the protocol, and then describe an attack on the enhanced version.

4.1. Attack on the Basic Polling Protocol

This attack is based on the fact that the $servent\_id$ is not bound with the $(IP, port)$ pair during the polling and vote evaluation phases. The association between them is established during the download phase, which allows an attacker

1. Alice broadcasts a `Query` message and based on received `QueryHit` messages, decides on a set of servents that she wants to inquire about. Let T denote that set.
2. Alice broadcasts `Poll`(T, PK_{poll}).
3. Bob receives the `Poll` message and decides to express his votes. He replies with `PollReply`$(E_{PK_{poll}}(IP_{Bob}, port_{Bob}, Votes_{Bob}, h(IP_{Bob}, port_{Bob}, Votes_{Bob})))$.
   Celine also receives the `Poll` message and wants to express her opinions, too. She replies with `PollReply`$(E_{PK_{poll}}(IP_{Celine}, port_{Celine}, Votes_{Celine}, h(IP_{Celine}, port_{Celine}, Votes_{Celine})))$
4. Alice receives `PollReply` messages from Bob and Celine. She now verifies the votes received by sending `TrueVote`$(Votes_{Bob})$ and `TrueVote`$(Votes_{Celine})$ to Bob and Celine, respectively.
5. Upon receiving `TrueVoteReply`$(response_{Bob})$ and `TrueVoteReply`$(response_{Celine})$ and verifying the validity of the $response$ field, Alice can use the voting results to make a decision about the source of the download.
6. Alice checks the $servent\_id$ for the source of the download and proceeds appropriately.

Figure 3. Basic Polling Protocol Example

to boost his reputation without disclosing his $servent\_id$, as long as he is not the source of a download. It requires the attacker to disclose his IP address, but this does not really matter, since it is the $servent\_id$ that the reputation and credibility are associated with, and since the IP address is volatile (e.g., dial-up connections or machines behind gateways using the Network Address Translation (NAT) protocol). Below are the details of the attack (corresponding steps of the Basic Polling Protocol are step 2.3 and step 2.4).

1. Alice broadcasts `Poll`(T, PK_{poll}).
2. Mallory forges suitable votes and using his IP and port, sends a `PollReply` message to Alice `PollReply`$(E_{PK_{poll}}(IP, port, Votes, h(IP, port, -Votes))$.
   Note that Mallory can compute the correct digest, since the hash function is public.
3. Alice receives `PollReply` sent by Mallory. If in the further steps of the protocol, she wishes to check Mallory's votes, she contacts him using the IP and port and verifies the votes. Mallory then can disconnect and dial up again, thus obtaining a new IP address, whereas his $servent\_id$ remains undisclosed.

Note that it is irrelevant if Mallory is a member of the originally broadcast set of servents T.

Observe also, that it is the attacker that is being contacted in the vote evaluation phase (Phase 3). Since the votes were actually expressed by him, he is also able to send a valid `TrueVoteReply` message; the $response_i$ must depend only on votes expressed in the previously sent `PollReply` message and the $(IP, port)$ information.

The attack can be countered by dropping the anonymity of the votes (refer to Section 5 for our proposed solution). This is in accordance with common sense, since expressing opinions about others should be coupled with taking the responsibility for one's opinions.

4.2. Attack on the Enhanced Polling Protocol

It is possible to perform a man-in-the-middle attack allowing the attacker to forge the votes of a servent replying to the `Poll` message. The attacker can modify the set of voters that expressed their opinions, although he can't alter the votes themselves. Nevertheless, it allows the attacker to gain an unfair advantage in the following ways.

1. He can remove himself from the votes expressed if a servent voted not in his favor.
2. He can remove a vote that is unfavorable for a servant that he is in collusion with.
3. He can remove a vote that is favorable for a servant that he wishes to harm.

Let T denote the set of servents that the originator wishes to inquire about. An attacker can forge the `PollReply` so that it contains votes for any set N such that $N \subseteq T$. The details of the attack follow.

1. Alice broadcasts `Poll`(T, PK_{poll}).
2. Mallory generates a key pair (PK_{fake}, SK_{fake}) and sends `Poll`(N, PK_{fake}).
3. Mallory receives `PollReply`$(E_{PK_{fake}}(IP, port, Votes, servent\_id_i, S_{SK_i}(IP, port, Votes, servent\_id_i), PK_i))$. Note that $Votes$ contains only votes for servents from the set N that was selected by Mallory. Mallory decrypts the `PollReply` message with SK_{fake} and now he can check if the votes are to his liking. If he decides that the votes suit his needs, he encrypts the `PollReply` message and forwards if to Alice. If Mallory prefers that Alice does not see the votes contained in the `PollReply`, he simply discards the message.

4. Alice receives `PollReply` sent by Mallory and makes her decisions based on the altered (N was substituted for T by Mallory) list of votes. In particular, Mallory can drop votes unfavorable for him.

This attack involves execution of steps 2.3 and 2.4 of the Enhanced Polling Protocol. It also relies on the consistency of votes between the moment when Mallory learned them and the moment when a given voter expressed his votes in response to Alice's poll. However, Mallory can refrain from activities hurting his reputation for this period or decide to just remove himself from the list T. Of course, Alice could monitor the network for `Poll` messages that have the list of requested votes differing by one servent and assume that the removed servent is the attacker, but this can be easily foiled by Mallory broadcasting a number of `Poll` messages with different servents removed from the list.

Note that a man-in-the-middle attack is also considered in [5] and measures to counter it are presented in that paper. However, the objective of that attack is to trick a servent into downloading malicious content. Our attack, on the other hand, aims at falsely influencing servents' reputations and – as shown above – is not deterred by the challenge-response mechanism used in the download phase.

Let us now return to the feasibility of the man-in-the-middle attack. As mentioned before, it is particularly easy to mount against neighboring peers. We still consider it a serious threat - since peers' reputations can be propagated across the network, and influencing them only for a small set of neighbors can still have network-wide consequences.

5. An Improved Security Polling Protocol

In this section, we first propose an improved protocol that aims at overcoming the security flaws leading to the attacks described in Sections 4.1 and 4.2. Then we discuss the security of the proposed solution.

5.1. Proposed solution

We assume that the $servent\_id$ is the hash of the public key of the servent. This quite common approach solves the important problem of binding the public key with the $servent\_id$. This use of public key cryptography far outweighs any of its drawbacks, e.g., the private key compromise renders the corresponding $servent\_id$ unusable and the – hopefully good – reputation associated with it is lost. Note that in any use of public key cryptography, the private key compromise yields non-trivial problems, e.g., invalidating the corresponding public key. We also propose that the hash function used is publicly known. This is necessary for other servents to be able to verify that the given public key belongs to a given $servent\_id$.

The $servent\_id$ serves as a permanent identity for a node. That is what is used by other nodes during polling. The reputation of each node is also associated with the $servent\_id$. We also assume that the $(IP, port)$ information is volatile, i.e., the mapping from $servent\_id$ to $(IP, port)$ can change over time. The rationale behind this assumption is the widespread use of dial-up connections and dynamic-IP DSL connections. Let us observe that nothing prohibits nodes from assuming multiple identities (by generating multiple (private, public) key pairs and thus obtaining multiple $servent\_ids$). One can think of how this fact makes the system prone to the Sybil attack [8]. Let us postpone this discussion till after the solution is fully described.

Let us now focus on the `PollReply` message. In our solution, we propose two options regarding the issue of encrypting its contents.

1. For reasons of performance, one may choose not to encrypt the contents of the `PollReply` message. This saves the time spent on encryption, as well as the time spent on generating a new key pair for each poll round. Let us observe though, that sending this message in cleartext makes it easier for a malicious node to inspect its contents and drop it, if that suits that node's needs. On the other hand, we have shown in Section 3 that adding encryption does not provide secrecy.
2. If the additional level of security provided by encrypting the contents of the `PollReply` message is desired in a particular implementation, and the cost of the asymmetric cryptography is acceptable, one can use the encryption. In this case (also assumed in [5]) the `Poll` message carries additional information about the public key to be used for the given poll round.

The final decision on the use of encryption for the `PollReply` message depends on the particular setting in which the protocol is to be used. We believe that one must carefully balance the tradeoffs involved and make the decision based on the operating environment (hostile vs. friendly) and intended application.

We decide to employ a vote verification procedure for selected peers. Also, to prevent a possible replay attack, we propose the use of random numbers as poll identifiers (this ensures freshness of messages). Instead of using a publicly known hash function, we use digital signatures for message integrity checking. Details of the protocol are given in Figure 4, with the boldface font used to distinguish newly added steps.

Let us now describe how the proposed solution works. The resource searching phase is the same as in [5]. The initiating servent sends out a `Query` message soliciting downloads and receives a `QueryHit` message from possibly multiple offerers. The initiator then selects a set of offerers about which he wants to learns others' opinion. Then the

Data definitions:

- R: **a randomly generated integer number**.
- $Votes$: **a vector of pairs of the form** $\langle (servent_1, vote_1), (servent_2, vote_2), \ldots, (servent_N, vote_N \rangle$. N **denotes the number of servents polled in Phase 2. Note that** $vote_i$ **can have a special value** *unknown*.

Phase 1: Resource searching

1.1 Start a search request by broadcasting a `Query` message
`Query`$(min\_speed, search\_string)$.

1.2 Receive a set offers from offerers O
`QueryHit`$(num_{hits}, port, IP, speed, Result, trailer, servent\_id)$.

Phase 2: Polling

2.1 Select top list $T \subseteq O$ of offerers.

Option A (`PollReply` not encrypted)

2.2A Generate a random number R**.**

2.3A Poll peers about the reputations of offerers T
`Poll`(T, R)**.**

2.4A Receive a set of votes from voters V (here shown a reply from servent $i, i \in V$)
`PollReply`$(R, IP_i, port_i, Votes_i, PK_i, S_{SK_i}(R, IP_i, port_i, Votes_i, PK_i))$.

Option B (`PollReply` encrypted)

2.2B Generate a random number R **and a key pair** (PK_{poll}, SK_{poll})**.**

2.3B Poll peers about the reputations of offerers T
`Poll`(T, R, PK_{poll})**.**

2.4B Receive a set of votes from voters V (here shown a reply from servent $i, i \in V$
`PollReply`$(E_{PK_{poll}}(R, IP_i, port_i, Votes_i, PK_i, S_{SK_i}(R, IP_i, port_i, Votes_i, PK_i)))$.

Phase 3: Vote evaluation

3.1 Remove from V, voters that appear suspicious.

3.2 Select a random set $V' \subseteq V$ **of voters and for each** $i \in V'$**, perform the vote integrity check by sending a** `Verify` **message**
`Verify`$(R, Votes_i)$**.**

3.3 Expect back confirmation messages from each selected voter
`VerifyReply`$(R, Votes_i, S_{SK_i}(R, Votes_i))$**.**

Phase 4: Resource downloading. This phase is the same as in the original paper.

Figure 4. Proposed solution

polling phase begins. The servent generates a random number and broadcasts a `Poll` message asking its peers to share information about reputations of servents in T. The random number R serves as an identifier for a given polling round. In case when encryption is used for the `PollReply` message, a per-poll key pair should be generated at this point and the public key should be included in the `Poll` message.

Peers wishing to share their opinions on servents from T send a `PollReply` with the same random number R, their votes, public key, and the $(IP, port)$ information that will be used later during the verification procedure. If encryption is being used, then the contents of the `PollReply` message is encrypted using the per-poll public key sent in the `Poll` message. Note that the message is sent over the Gnutella network to assure the poll initiator anonymity and that its integrity is provided by means of a digital signature [18]. Observe also,

that votes are not anonymous since the $servent\_id$ can be obtained by hashing the public key. Votes must be made identifiable so that servents can be held accountable for how they vote. The vector of votes $Votes$ consists of pairs, each of which expresses the sender's opinion about a given peer. It is required that this vector have a pair for every peer in the T set. If the voter does not have an opinion about a given peer, the special variable *unknown* is to be used in the corresponding entry in the $Votes$ vector. For example, if $T = \langle Bob, Celine, Tom\rangle$ and voter i wants to express the opinion only about Bob and Tom, then his $Votes$ should have the form of $\langle(Bob, vote_i^{Bob}), (Celine, unknown), (Tom, vote_i^{Tom})\rangle$.

The vote verification phase begins after poll results are received. First, `PollReply` messages that do not pass the integrity check are discarded. The integrity check consists of verifying the digital signature and checking if the $Votes$ vector has an entry for every peer in T for the corresponding poll. Then a set of voters is selected and they are directly contacted to confirm their votes. Direct communication is achieved by means of a secure channel outside of the P2P network (e.g., SSL) and the provided $(IP, port)$ pair - as suggested in the original paper. Vote verification consists of digitally signing (i) the votes, and (ii) the random number identifying a given poll. The digital signing is performed by a directly contacted voter. Since the voter's private key is required to sign the contents of the `VerifyReply` message, the confirmed vote can be associated with a given servent. The remaining steps of the protocol remain as in the original approach.

5.2. Security of the Proposed Solution

The proposed solution is resilient to both the attacks described in Sections 4.1 and 4.2.

The essence of the attack on the Basic Polling Protocol is the possibility for an attacker to cast votes without any accountability. The solution requires that the `PollReply` message contain the $(IP, port)$ pair as well as the public key PK_i. Since the $servent\_id_i$ is a hash of the public key PK_i, the identity of a particular voter is known to the initiator. This way, a rouge node can not get away with malicious voting and maintain its reputation unaffected. In particular, voting for oneself can be easily detected by simply checking the $Votes$ vector.

The attack on the Enhanced Polling Protocol is based on the ambiguity of the contents of the `PollReply` message. Let us focus on the scenario in which a node wants to cast votes only for a subset of the servents contained in the `Poll` message. Assume that we have two polls. Assume also, that the set of offerers in one poll (i.e., servents whose reputations are being polled) is denoted by A and the set of offerers of the other poll is denoted by B, where $B \subset A$. Now there is a servent that wants to express his opinions, but only about offerers in some set C, such that $C \subset B \subset A$. It is easy to see, that a `PollReply` message that he sends can not be unambiguously attributed to any of the two polls. This, coupled with the lack of a mechanism ensuring message freshness, leads to the possibility of a man-in-middle attack that allows the manipulation of the voting procedure results. However, the attack can be thwarted. The solution employs a random number as a poll identifier and also requires that the `PollReply` field $Votes$ contain an entry for every servent in the set T in the `Poll` message. The initiator needs to keep track of poll identifiers and the corresponding set of servents, and using that information, he can rule out poll replies that have been tampered with.

To sum up: the security of our solution in achieved thanks to three factors.

- The use of a random number to identify the poll. This makes the protocol resilient to replay attacks. It also helps the initiator correctly correlate votes and vote verification messages, and helps to thwart the attack on the Enhanced Polling Protocol.
- Integrity of the `PollReply` message is achieved by digitally signing its contents. It prevents attackers from tampering with the votes expressed by others and fixes the vulnerabilities that lead to the attack on the Basic Polling Protocol.
- The use of a direct secure channel that is outside of the P2P network is crucial for the verification phase. This allows for impersonation attempts to be detected.

Let us now return to the issue of the Sybil attack mentioned in Section 5.1. The possibility of this kind of attack can be reduced by following some precautions by the poll initiator. Namely, if there exists an one-to-one mapping between the $(IP, port)$ pairs and $servent\_ids$ for a given poll, we can assume with high probability that no node has assumed multiple identities during the voting. If such a mapping does not exist, then there are at least two `PollReply` messages received by the initiator that contain the same $(IP, port)$ pair, but different $servent\_ids$. This can be interpreted as an attempt to mount a Sybil attack, and the votes carried by such messages can be, for example, discarded. We believe however, that the Sybil attack can not be prevented entirely. We attempt to thwart it by checking if two votes do not purport to come from two different $servent\_ids$ associated with the same $(IP, port)$ pair. However, if the attacker has control of two different machines (possibly on different continents) with two different IP addresses, then there is not much that can be done to prevent him from circumventing the polling scheme.

6. Discussion

This paper addressed various issues in providing enhanced security for the P2P protocol Gnutella [10, 11]. We described security flaws in the Basic and Enhanced Polling Protocols proposed to maintain servents' reputations in P2P Gnutella networks [5]. First, we demonstrated how the confidentiality of a crucial message of the protocol can be circumvented, which in turn renders the integrity mechanism of the Basic Protocol ineffective. Then we showed how both the Basic Protocol and the Enhanced Protocol can be attacked, allowing an attacker to affect the votes expressed by others. We fixed the security problems in the original design described in Section 4 and proposed a correct protocol. We employed digital signatures, random numbers for message freshness and - optionally - public key cryptography. The security of both our and the original protocols depends on the existence of a secure communications channel (used for vote verification) operating outside of the P2P network. This assumption is a bit restrictive and exploring ways to overcome it seems like an interesting and promising future research direction. Although the focus of the paper was on Gnutella, the generalized results can more broadly be applied to other P2P protocols.

References

[1] K. Aberer and Z. Despotovic, Managing Trust in a Peer-To-Peer Information System, *In Proceedings of the 10th International Conference on Information and Knowledge Management*, Atlanta, Georgia, USA, November 2001, pp. 310-317.

[2] S. Braynov and T. Sandholm, Incentive Compatible Mechanism for Trust Revelation, *In Proceedings of the 1st International Conference on Autonomous Agents and Multiagent Systems*, Bologna, Italy, July 2002, pp. 310-311.

[3] M. Castro et al., Secure Routing for Structured P2P Overlay Networks, *In Proceedings of the 5th Usenix Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Boston, Massachusetts, USA, December 2002.

[4] I. Clarke et al., Freenet: A Distributed Anonymous Information Storage and Retrieval System. *In Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, California, USA, July 2000, pp. 46-66.

[5] E. Damiani et al., Managing and Sharing Servents' Reputations in P2P Systems, *IEEE Transactions on Knowledge and Data Engineering, 15(4)*, July/August 2003, pp. 840-853.

[6] N. Daswani and H. Garcia-Molina, Query-Flood DoS Attacks in Gnutella, *In Proceedings of the 9th ACM Conference on Computer and Communications Security*, Washington DC, USA, November 2002, pp. 181-192.

[7] A. Davidson, P2P File Sharing Privacy and Security, *Testimony before the House Committee on Government Reform Center for Democracy and Technology*, May 2003.

[8] J. R. Douceur, The Sybil Attack, *In Proceedings of the IPTPS02 Workshop*, Cambridge, Massachusetts, USA, March 2002, pp. 251-260.

[9] E. C. Efstathiou1 and G. C. Polyzos, Designing a Peer-to-Peer Wireless Network Confederation, *In Proceedings of the 3rd International Workshop on Wireless Local Networks (WLN 2003)*, Bonn/Konigswinter, Germany, October 2003, pp. 774-775.

[10] Gnutella, `http://www.gnutella.com/`.

[11] The Gnutella protocol specifi cation, `http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf`.

[12] M. Gupta, P. Judge and M. Ammar, A Reputation System for Peer-to-Peer Networks, *In Proceedings of the 13th International Workshop on Network and Operating Systems Support for Digital Audio and Video, (ACM Press)*, Monterey, California, USA, June 2003, pp. 144-152.

[13] W. Kim, S. Graupner, and A. Sahai, A Secure Platform for Peer-to-Peer Computing in the Internet. *In Proceedings of the 35th Annual Hawaii International Conference on System Sciences* Hawaii, USA, January 2002, p. 304.

[14] G. Kortuem, Proem: A Peer-to-Peer Computing Platform for Mobile Peer-to-Peer Computing. *ACM Mobile Computing and Communications Review, 6(2)*, 2002.

[15] Napster, `http://www.napster.com/`.

[16] S. Ratnasamy et al., A Scalable Content-Addressable Network, *In Proceedings of the ACM SIGCOMM'01*, San Diego, California, USA, August 2001, pp. 161-172.

[17] A. Rowstron and P. Druschel, Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems, *In Proceedings of the IFIP/ACM Middleware 2001*, Heidelberg, Germany, November 2001, pp. 329-350.

[18] B. Schneier, *Applied Cryptography. Protocols, Algorithms and Source Code in C*, Wiley and Sons, 1996.

[19] E. Sit and R. Morris, Security Considerations for Peer to Peer Distributed Hash Tables, *In Proceedings ofr the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02), LNCS 2429, Springer*, Cambridge, Massachusetts, USA, March 2002, pp. 261-269.

[20] I. Stoica et al., Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications, *In Proceedings of the ACM SIGCOMM'01*, San Diego, California, USA, August 2001, pp. 149-160.

[21] B. Yu, M. Singh, An Evidential Model of Distributed Reputation Management, *In Proceedings of the 1st International Conference on Autonomous Agents and Multiagent Systems*, Bologna, Italy, July 2002, pp. 294-301.

[22] D. Zeinalipour-Yazti, Exploiting the Security Weaknesses of the Gnutella Protocol, *Course project for "Seminar in Computer Security" at the University of California Riverside, Computer Science*, March, 2002.

[23] B. Y. Zhao, J. D. Kubiatowicz and A. D. Joseph, Tapestry: An Infrastructure for Fault-Resilient Wide-Area Location and Routing, *Technical Report UCB//CSD-01-1141, U. C. Berkeley*, Berkeley, California, USA, April 2001.