Securing Pseudo Identities in an Anonymous Peer-to-Peer File-Sharing Network

Tom Chothia
CWI, Kruislaan 413, 1098 SJ, Amsterdam, The Netherlands.
Email: Tom.Chothia@cwi.nl

***Abstract*— MUTE is an anonymous peer-to-peer network that is used by hundreds of thousands of people to share files. Peers in this network are identified by randomly chosen pseudo identities and a probabilistic time-to-live counter is used to stop an attacker from being able to tell how far a search has come or has to go. The aim of the system is to hide the IP addresses of the file-sharers from an attacker that is acting as one or more peers inside the network. This paper describes the MUTE system, and then goes on to outline an attack on the anonymity of a peer based on "stealing" pseudo identities. We then show how using an authentication key as a pseudo identity can stop this attack and we describe the implementation of this solution as part of the MUTE system.**

I. Introduction

MUTE is an anonymous peer-to-peer file-sharing system; with over 900,000 downloads[1] it is one of the most popular anonymous file-sharing systems and has served as the inspiration for a number of similar systems [1], [4] and academic papers [2], [4], [5], [8].

Peers using MUTE will connect to a small number of other, known peers; only the direct neighbours of a peer know its IP address. Communication with remote peers is provided by sending messages hop-by-hop across this overlay network. Routing messages in this way allows MUTE to trade efficient routing for anonymity.

There is no way to find the IP address of a remote peer, and direct neighbours can achieve a level of anonymity by claiming that they are just forwarding requests and files for other peers. In this way peers in the network keep their file-sharing activity secret from an attacker who may be acting as one or more peers in the network. There is no anonymity from an attacker who can monitor the entire network or be sure that they have a peer completely surrounded.

Every peer picks a random pseudo ID that it uses to identify itself. When a peer receives a new message it records the connection over which the message was received as a possible route to the pseudo ID that the message was from. In this way the peer builds a routing table for each pseudo ID. There is a danger that an attacker may be able to link the pseudo identity and the IP address of its direct neighbours, and thus find out which files the neighbours are requesting and offering.

An attacker, acting as a peer in the network, can "steal" a pseudo identity by sending fake messages using the target identity as the "from ID". If it sends enough messages then its neighbouring peers will forward messages addressed to the target identity over their connection with the attacker. The one exception to this is if the identity the attacker is trying to steal belongs to the neighbour, as the neighbour will never forward messages addressed to itself. Therefore the attacker can use this method of stealing identities to test any identity it sees; if the identity cannot be stolen then it belongs to the neighbour.

We solve this problem by stopping the attacker from being able to forge messages. We do this by having all peers start by generating an authentication and signature key. The peers can then use the authentication keys as their pseudo identities. These authentication keys would be used in exactly the same way as the peers' identities. However, each peer would also sign the message ID. When any peer receives a message, it can check the signed message ID using the "from ID" authentication key. As the attacker cannot correctly sign the message ID it can no longer forge messages.

We first discovered the existence of an attack by modelling MUTE in the pi-calculus [5]. The contribution of this paper is a technical description of MUTE and the attack, the fix for that attack and the description of how this fix was added to MUTE. The use of a public key as an identity has been proposed as a way to stop denial of service attacks in IP6 [10]. There are a number of other systems and theoretical designs for anonymous file-sharing, for full details we refer the reader to our previous survey paper [6].

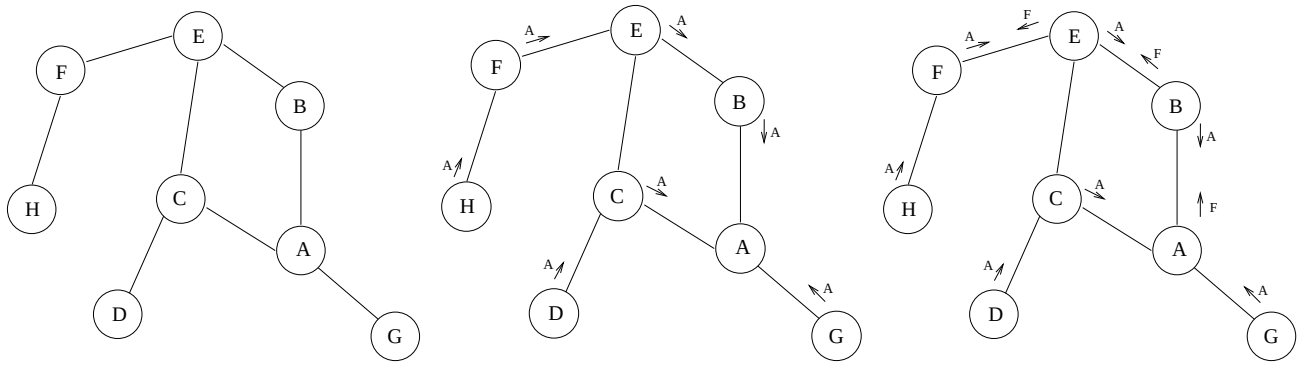
In the next section we describe the MUTE routing protocol, and then in Section III we outline our attack on MUTE. Section IV shows how these kinds of attacks can be stopped and describes the fixes made to the MUTE system.

II. The MUTE Peer-to-Peer File-Sharing System

The MUTE system is based on the Ant Colony Optimisation algorithm [7], which is in turn based on the way ants use pheromones when looking for food [3]. The peers in a MUTE network form a fixed overlay network, as illustrated in Figure (a), each peer connects to a number of other peers (its neighbours). The only way in which new connections can be made is if an IP address is manually entered into a peer[2]. This means that there is no way for a peer in the network to learn the IP address of a remote peer. Communication between any two peers is encrypted using a AES symmetric

[1] source: `http://sourceforge.net/projects/mute-net`

[2] A "Blender" service offers new peers a limited number of randomly chosen peer IP addresses, so that they may join the network in a safe way



(a) A network of peers (b) routing tables after a search by A ... (c) ... and after a reply by F

key and communication between remote peers is possible only by sending messages hop-by-hop across this overlay network, encrypting and decrypting the message on each hop.

In order to build up an "ad-hoc" routing table for the overlay network of peers, each peer picks a random 160 bit number as a "pseudo ID". This ID is used as an address for the peer, and any message that originates from that peer is labelled with it. The connection that brings messages from a pseudo ID is also a possible connection over which to send messages to that pseudo ID therefore each peer can dynamically construct a routing table by recording the connections on which the messages arrive as the route to that ID. When multiple connection carry messages from the same ID, the fastest, most used connection will usually be the best.

In order to search the network, a peer broadcasts a search message with its own pseudo ID, a unique message identifier and a probabilistic time-to-live counter. The search message is sent to the peer's neighbours, which in turn send the message to all of their neighbours until the time-to-live counter runs out. Upon receiving a message a peer first checks the message identity and discards any repeated messages, it then records the connection on which the message was received and the pseudo ID of the sender. The results of peer A sending a search message are illustrated in Figure (b). As the search message fans out each peer records the direction it came from. Peer E will receive two copies of the message, one from B and another from C, here we assume the connection A, B, E is faster than A, C, E, and so the peer E discards the message from C and adds the connection to B to its routing table.

If the peer F, for instance, wants to reply to the search it forms a reply message addressed to A and marked as coming from F. Peer F does not know the location of A but it does know which of its connections is part of the fastest route to A, so it sends the message over that connection. Each peer does the same and the reply makes its way to E, then to B and finally to A. At the same time each of these peers record which direction the message from F came from in its routing table, as illustrated in Figure (c).

We note that the routing table tells the peers what direction to send messages addressed to a given pseudo ID, but not which peer has that ID. This inability to link pseudo IDs and IP addresses is the basis of the anonymity offered by MUTE. For instance, in the exchange above, no peer can be sure that their neighbour was the up-loader or the down-loader.

To stop peers using the message's time-to-live counter to work out where a search originates or ends a three phase probabilistic "Utility Counter" is used. The first phase, is equivalent to the originator of the message picking 1, with probability $\frac{2}{3}$, or 2 with probability $\frac{2}{9}$, or n with probability $\frac{2}{3^n}$. This value is then counted down by 1 each hop. When it reaches 0 the peer moves to the second phase, which forwards the message for 5 hops. A message with an expired time-to-live counter will be dropped with probability $\frac{3}{4}$, or forwarded to n randomly chosen neighbours with probability $\frac{1}{2^{n+2}}$.

We now describe the details of the MUTE routing protocol:

- On start up the MUTE application generates a pseudo ID and a 1024-bit RSA public/private key pair. It also fixes the values of the utility counter that it will use in all of its searches (hence avoiding a statistical attack).
- The peer then attempts to make connections to its list of other known peers and uses its public key to set up a AES 128-bit symmetric session key for each connection.
- After establishing its connections it selects a subset of these peers to which it will forward messages with expired utility counters, as described above, and creates an empty list of "seen" message IDs and an empty routing table.

The peer is then ready to start sending and receiving messages. These messages have the following format:

```
MessageID: 6 digit number & a time stamp
From ID:   The pseudo ID of the sender
To ID:     The pseudo ID of the recipient
             or "ALL" for searches
Flags:     The phase of the counter and
             maybe ROUTE_ONLY | FRESH_ROUTE
UtilityCounter: base-10 ASCII counter value
Length:    The length of the body
Body:      AES encrypted message
```

The flags allow for a fine-grained control over the routing: The ROUTE_ONLY flag indicates that the peer should only forward the message if it has the "To ID" in its routing table.

The FRESH_ROUTE flag indicates that the peer should delete both the sender and the receiver from its routing table.

Search messages are addressed to "ALL" and are sent to all of the peer's neighbours with the utility counter value that was chosen when the peer started up. When a peer receives a message it processes it in the following way:

- If the message ID is in the list of seen message IDs or the time-stamp[3] is old the message is discarded. Otherwise the message ID is added to the list of seen messages.
- If there is a list of channels for the "From ID" in the routing table then the peer adds the channel on which the message was received (dropping the oldest channel if there are now more than 50). If the "From ID" is not in the routing table it is added along with the channel it arrived on. The oldest "From ID" is dropped if the routing table now contains more than 50 IDs.
- If the "To ID" is ALL, or the "To ID" is unknown, and the utility counter has not expired the peer reduces the time-to-live and forwards the message to all of its neighbours, decrypting and re-encrypting the body of the message with the appropriate AES key. If the utility counter has expired then the message is only forwarded to the subset of peers selected at start up, if any.
- If the local routing table includes a list of channels for the "to ID" then the peer selects one of these channels at random. As repeat messages do not add entries to the routing table, selecting randomly will return one of the fastest, most used channels. The peer forwards the message on this channel, encrypted with the AES key for that channel.

The peer can then process and reply to the message, if necessary.

III. Description of the Attack on MUTE

The attacker can "steal" an ID by sending fake messages using the ID it wants to steal as the "from ID". Any peer that receives such faked messages will incorrectly add entries to its routing table indicating that it should send messages addressed to the ID along its connection to the attacker, rather than the connection that leads to the original owner of the ID.

If the attacker sends enough messages then it can be sure that its neighbours will send all messages addressed to the stolen ID in its direction. The only exception to this is if the ID the attacker is trying to steal belongs to the neighbour, as the neighbour will never forward messages addressed to itself. Therefore the attacker can use this method of stealing IDs to test any IDs it sees; if an ID cannot be stolen then the ID belongs to the neighbour.

We saw in Section II that MUTE looks at the 50 most recent messages when deciding where to route a message. This means that if the attacker can send 50 messages with the target ID to its neighbour, without any messages with that ID coming from anywhere else, then the attacker knows that it must receive any messages sent to that ID via the target peer, unless the ID belongs to the target peer. But how can the attacker be sure that the target peer's routing table is not also being affected by real messages from the target ID, which are not past onto the attacker? If the attacker receives a search message from a neighbour once, it knows that all other messages that the neighbour receives in the same way will also be forwarded to the attacker. However it is possible that the neighbouring peer could receive messages from the target ID via two different routes, one of which is dropped before being passed onto the attacker. This would pose a problem as if the attacker cannot detect all of the messages coming into the neighbouring peer then it cannot be sure that it has successfully stolen the ID.

Luckily, for the attacker, only IDs that are seen on search messages with phase-1 counters are possibilities for the neighbours ID and only search messages with phase-3 counters can be dropped. If the attacker sees some messages with a phase-1 counter and others reach the neighbour with a phase-3 counter and are dropped, we know that the messages that are dropped must be slower. These slower messages will be discarded as duplicates by the target peer, therefore they will not affect the routing table.

There is still a small possibility that the neighbour is receiving or forwarding a file from the real owner of the ID, in which case the large number of messages that the neighbour is receiving might mean the attacker fails to steal an address that does not belong to the target peer. To avoid this possibility the attack can be repeated at regular intervals. The attack on MUTE would run as follows:

1) The attacker uses MUTE's blender service to find the IP address of a peer, makes two connections to it, monitors these connections and selects the "from ID" with the highest, phase-1, utility counter.
2) The attacker forms new search messages using the selected ID as the "from ID" and repeatedly sends them to the neighbour until it has sent 50 messages without receiving any messages from the ID it is trying to steal.
3) The attacker then sends a reply message addressed to the selected ID along its other connection with the target peer.
4) If the attacker receives the message back then the selected ID does not belong to the target peer, so the attacker must select another ID and start again. Otherwise, with a high degree of probability, the attacker has found the neighbour's ID and can then find out what files the neighbour is sharing.

IV. Secure Pseudo Identities

The attack is made possible by the MUTE protocol's adaptive routing system and the fact that peers will never forward messages addressed to themselves. Key to the success of the attack is the attacker's ability to fake messages with another peer's ID.

We can solve this problem by stopping the attacker from being able to forge messages. We are only interested in stopping the creation of fake messages that use someone else's ID. We are not interested in the actual value of the pseudo ID and we wish to maintain MUTE decentralised nature. Our solution is to have all peers start by generating

[3]To avoid the exact time leaking any location information a Lamport style counter [9] is used

an authentication and signature key. The peers then use the authentication keys as their pseudo IDs. This authentication key would be used in exactly the same way as the peers' ID. However, each peer would also sign the message ID. When any peer receives a message, it can check the signed message ID using the "from ID" as the key. As the attacker cannot correctly sign the message ID it can no longer forge messages. This scheme is, in general, backwards compatible: older peers need not be aware that the ID is also an authentication key. The checking is also optional; peers may choose to only check messages if they spot suspicious activity.

The level of popularity enjoyed by any system that claims to offer anonymity to the user will be partly based on the level of trust potential users place in these claims. To maintain a level of trust in the MUTE system it was important to implement this fix before the flaw became widely known. So we contacted the developers of the MUTE system and suggested the fix described in this section. They where pleased to have the attack pointed out to them however they also had a strong desire to ensure that the fix was backwards compatible with the previous version of MUTE. This meant that the IDs could not be longer than 160 bits, which is too short for a RSA public key.

SHA1 hashes are exactly 160 bits so we use a SHA1 hash of the peer's 1024-bit RSA public key as the pseudo ID (the same public key that the peers use to exchange their symmetric channel keys). We sign the message ID along with the counter time-stamp and only forget message IDs that have a time-stamp that is too old to accept. A very patient attacker could wait for the counter to loop and then run a replay attack. So the peers record the value of the counter at which they picked their IDs and change to a new ID if they ever loop round.

The prefix PKH is added to the new pseudo IDs and two flags are added to the flags part of the message headers: "`PUBLIC_KEY`_key", where "key" is the hex-encoded RSA public key used to sign the message ID and "`SIGNED_ID`_sig", where "sig" is the message ID (including the time-stamp) signed with the private key. So the start of the message header changes from:

$$(MessageID, From\_ID, To\_ID, FLAGS : ...$$

to become:

$$(MessageID, \#(K_a), To\_ID, FLAGS : PUBLIC\_KEY\_K_a, SIGNED\_ID\_S_{K_s}(MessageID), ...$$

Where K_a is the public (authentication) key, $\#(K_a)$ is the SHA1 hash of K_a, which is being used as the "From ID", and $S_{K_s}(MessageID)$ is the message ID signed with the private (signature) key.

When one of the updated peers receives a message, after checking to see if the message ID is a repeat, it checks to see if the "from ID" starts with PKH, if so it checks that the hash of the key part of the `PUBLIC_KEY` flag matches the rest of the "from ID" and that the signature of the message ID matches the `SIGNED_ID` flag. If either of these fail to match, or the flags are not present, then the message is discarded. Older peers, oblivious of all this, will ignore the flags and process the message as before. These older peers are still at risk.

The use of a hash of the key is secure: in order to generate a legitimate message for a pseudo ID that is a hash of an authentication key the attacker would have to come up with an authentication key that matched the hash, a new message ID and the signature of that message ID that matched the authentication key. Assume for contradiction that the attacker can come up with these, if the authentication key in the message header is the same as the one given to the attacker then the attacker has broken the RSA signature scheme. Whereas, if the authentication key in the header is different, but still matches the hash, then the attacker has broken the SHA1 hash collision problem for a given hash. Both of these problems have been shown to be computationally hard.

Tests run by MUTE's developers found that the extra cryptographic operations did not increase download times noticeably. This was because the biggest time delay in the system is caused by the hop-by-hop routing, compared to which the extra cryptographic operations where insignificant.

This solution was added to the 0.5 release of MUTE, the C++ source code is available at `http://mute-net.sourceforge.net`. Since its release in June 2006 the code has been downloaded over 150,000 times.

V. Conclusion

The attack on MUTE we have described involves trying to force a neighbour to misroute messages addressed to an ID; this only succeeds if the ID does not belong to the neighbour. We fix MUTE by using the hash of an authentication key as the peers' pseudo ID. It may be possible to carry out similar attacks on other ad-hoc networks in which the nodes generate their own IDs, a similar defence may also be useful.

Acknowledgement: We would like to thank Jason Rohrer for his helpful comments on this work. This work was partly supported by the ITEA Trust4all and Credo EU STREP 033826 projects.

References

[1] Ants p2p, http://antsp2p.sourceforge.net/, 2003.

[2] Andres Aristizabal, Hugo Lopez, Camilo Rueda, and Frank D. Valencia. Formally reasoning about security issues in p2p protocols: A case study. In *Third Taiwanese-French Conference on Information Technology (TFIT)*, 2005.

[3] R. Beckers, J. L. Deneubourg, and S. Goss. Trails and u-turns in the selection of the shortest path by the ant lasius niger. *Journal of Theoretical Biology*, 159:397–415, 1992.

[4] Steve Bono, Christopher A, Soghoian, and Fabian Monrose. Mantis: A high-performance, anonymity preserving, p2p network, 2004. Johns Hopkins University, Tech. Rep. TR-2004-01-B-ISI-JHU.

[5] Tom Chothia. Analysing the mute anonymous file-sharing system using the pi-calculus. In *FORTE2006*, volume 4229 of *LNCS*, 2006.

[6] Tom Chothia and Konstantinos Chatzikokolakis. A survey of anonymous peer-to-peer file-sharing. In *EUC Workshops, LNCS*, volume 3823, 2005.

[7] Marco Dorigo and Gianni Di Caro. The ant colony optimization meta-heuristic. In David Corne, Marco Dorigo, and Fred Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw-Hill, London, 1999.

[8] Byung Ryong Kim, Ki Chang Kim, and Yoo Sung Kim. Securing anonymity in p2p network. In *Smart objects and ambient intelligence (sOc-EUSAI 05)*. ACM press, 2005.

[9] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 1978.

[10] Gabriel Montenegro and Claude Castelluccia. Statistically unique and cryptographically verifiable (sucv) identifiers and addresses. In *Symposium on Network and Distributed Systems (NDSS 2002)*, pages 87–99. Internet Society, 2002.