Allan Friedman, Harvard University
L Jean Camp, Harvard University

Keywords: Peer-to-peer, P2P, distributed systems, decentralization, anonymity, distributed hash tables, groupware, reputation

Peer-to-peer networks break the dominant network paradigm of client-server relationships in information exchange by allowing heterogeneous multipurpose machines to interact and share functionality. Like any architecture, different systems have different purposes and necessitate different security protections. This chapter will identify several key features of peer-to-peer networks and the security needs they raise, and some of the solutions proposed by existing systems.

INTRODUCTION

Peer-to-peer systems (P2P) have grown in importance over the last 5 years as an attractive way to mobilize the resources of Internet users. As more and more users have powerful processors, large storage spaces and fast network connections, more actors seek to coordinate these resources for common goals. Because of their unique decentralized nature, security in these systems is both critical and an interesting problem. How do you secure a dynamic system without central coordination? Good security on P2P systems must reflect the design goals of the system itself. In this chapter, we introduce some basic concepts for P2P systems and their security demands, and then discuss several case studies to highlight issues of robustness, privacy, and trust. We analyze systems that offer secure distributed routing, privacy-enhancing and censorship-resistance publishing, shared storage and decentralized collaborative work spaces. We conclude by examining how accountability and trust mechanisms must be built into a secure system.

ABOUT PEER-TO-PEER SYSTEMS

What is P2P

Peer-to-peer systems have two dominant features that distinguish them from a more standard client-server model of information distribution: they are overlay networks that have unique namespaces. P2P systems link different, possibly heterogeneous systems as 'peers,' and allow them to interact on top of existing network configurations. It does this by defining relationships unique to that system, usually in the form of a topology by which systems are linked. Occasionally a system will piggyback atop an existing namespace, such as the IP/port labeling, but still treats these separately from the Internet-layer protocols. The combined effect of independent systems interacting through a unique namespace is *decentralization*. Not every system generally considered a P2P system is strictly decentralized in management—the infamous Napster actually ran through a centralized server—but the matter of decentralization brings out important concerns for system security.

Although they have won the most attention for their roles as (often illicit) file-swapping networks, peer-to-peer systems can serve many functions, and design considerations must reflect these. One proposed use of a P2P system was that of shared expertise. A Gnutella pioneer has suggested that the system could be used to propagate queries of any sort through a network until a peer felt that it could respond to the query, anything from a lexical look-up to a specialized calculation (Kan, 2000). In a network with varied resource availability, P2P systems have been used to distribute those resources to those who need or want them the most. The shared resources in question are usually computation or storage. This can create efficient resource usage,

given a low marginal cost of using a resource not otherwise needed. Beyond sharing computation power for enormous tasks, P2P networks have been proposed as an escrow system for digital rights management, or for distributing searches across a wide range of other peers. A new class of business software systems known as 'groupware' uses many P2P principles. Groupware networks are closed networks that support collaborative work, such as the Groove network discussed below. Finally, the decentralized nature of peer systems offers many positive features for publishing and distribution systems, not least of which is their resilience to legal and physical attacks and censorship. P2P architectures have many different functions, and different functions lead to different design conclusions, with respect to overall system structure, and specifically to security issues.

A Sample Architecture: Gnutella

Gnutella can illustrate the role of security in the design of a P2P system. Developed in 2000, this system was one of the first widely-deployed truly decentralized P2P networks, and was used largely for file-swapping. Nodes keep a list of other nearby nodes in the network. When they have a query, a node will send the message to all neighbors, with a time-to-live counter measured in hops. Nodes will pass the query along until the counter runs to zero; any node that can positively respond to the query will contact the original node directly, using network location information embedded in the original query. Nodes discover each other through IP/port announcements, which are unauthenticated, opening the way for flooding or denial of service attacks. Peers are not required to authenticate either the query or the response. An attacker can return an erroneous response, directing the querying node to download files from the target node: if enough seekers are duped, the target node cannot identify or defend itself against the Denial of Service attack. Moreover, since a node, its query and its target are not secured with information-hiding techniques, information about all can be exposed. This has been exploited by vigilantes to publicly expose the machines of those who sought child pornography, and by the recording industry to identify holders of illegally obtained music files. Concerns about security for P2P systems have very real ramifications.

Before we can examine security in a P2P system, it pays to have a brief look at the components of a P2P system for construction. Like all complex systems, designing peer-to-peer networks is a matter of trade-offs. Because they are defined by their heterogeneous environment, it is impossible to optimize for everything. Once distributed, implementing additional control structures is much more difficult, so the original design must at least consider the following issues. The system consists of clients running on hosts. Hosts must be made aware of each other and have a means to communicate, forming nodes in a network. Nodes that wish to communicate to other nodes without a direct connection need a routing mechanism to go through the network. A routing mechanism may or may not need special consideration for the return path. Routing is distinct from the default network-level routing used by the host, and rides above that layer. Instead, the namespace of the P2P system is used to find appropriate peers. Routing for specific queries can take different forms, including a broadcast to all known nodes, a multicast through neighbors with a search horizon, or an interconnection through well-placed "super-peers" at key junctions in the network. If attention is not paid to how nodes are distributed throughout the network, there can be load-balancing issues resulting in traffic problems. If we assume a dynamic Internet, then nodes need to know how to enter the network, and how to leave gracefully. The existing network must be able to integrate new nodes, and recover after a node failure. As more nodes join, the network must scale well to maintain functionality. Many standard design trade-offs, such as redundancy versus efficiency, are made all the more salient by the heterogeneous environment we assume in P2P.

P2P and security

Security expert Bruce Schneier is often quoted as claiming that "Security is a process." As such, it matters very much whether security administration is centralized or decentralized. The basic model of the commercial Internet is the client-server relationship. As the network user base grew, less and less responsibility for administration was placed on the edges of the network, and more was concentrated in smart 'servers.' This model is most evident on the World Wide Web, but file servers, mail servers and centralized security administration mechanisms such as Virtual Private Networks have all grown apace. In a centralized system, security policy can be dictated from a single location, and a 'that which is not permitted is forbidden' policy can be enforced with firewalls, monitoring and intervention.

On the other hand, centralization offers a single point of failure. Both direct malicious attacks and lax or negligent administration at the heart of a centralized network can undermine security for an entire system. With a single breach, outgoing or incoming content can be corrupted or intercepted, or the system can be rendered inoperable with a denial of service attack. Critical infrastructure systems such as the Domain Name System (DNS) have redundant implementation exactly because a single point of control is vulnerability in and of itself. In a decentralized P2P system, bad behavior has a locality impediment. Malicious attacks need to occur at or near every part of the system it wishes to affect.

P2P systems lack the tools available to a centralized administrator, so it can be much more difficult to implement security protections on a deployed P2P system. Moreover, the absence of a defensible border of a system means that it is hard to know friend from foe. Malicious users can actively play the role of insiders—i.e., they can run peers themselves, and often a great number of them. Doceur (2002) notes that this is incredibly hard for any open system to defend itself against this sort of

attack, known as a Sybil attack (named after a famous Multiple-Personality Disorder case study). Several solutions are discussed below. As such, many systems are designed with strong security considerations in mind, depending on their function, from cover traffic to prevent monitoring of file transfer to a reputation system that can discourage single actors from controlling too many trusted nodes. As above, the wide range of security issues makes optimizing for all not a feasible option. This will become evident in our discussions of actual P2P systems below, but first we present some of the more common security demands.

One way to think about P2P is whether it applies at the network level, the application level or the user level. At the network level, an adversary may try to break the routing system, or block access to information by impeding queries, or partitioning the network. At the application level, an adversary can attempt to corrupt or delete data stored in the system or in transit. Finally, the users themselves can be the subject of attack if an adversary goes after anonymity and privacy protections.

Camp (2003) offers an operational definition of trust based on the risk that a system will fail to perform as specified. This is a broader view of security, but one that suits our purposes. In a P2P system, we are interested in not only how one aspect of the system behavior withstands attack, but how the entire system is defended. Since policy is not set dynamically, but is the sum of all behavior in the network, security also includes the expected behavior of other peers, and what their incentives are. Layers of protection interact. These levels translate into more specific security goals for system to operate reliably.

In the examples that follow, we highlight security features of these systems, and the design choices made to offer optimal security for a given task. Discussions of cryptanalysis and other field-specific security issues employed by P2P systems are outside the scope of this chapter, as are more complete descriptions of these systems and their performance. Instead, these descriptions are intended to illustrate security mechanisms rather than fully describe all nuances of any P2P system.

The case studies below highlight some of these goals. Secure distributed transport mechanisms like Tarzan ensure that the identity of the user and the content of the message remain a secret while secure overlay mechanisms are designed to ensure that the message gets through and gets there efficiently. Freenet and Free Haven ensure adequate file maintenance, so that users can reliably access any file they want, or any file that have been published, respectively. Groove allows networks of peers to manage themselves. Distributed storage mechanisms can create conditions for cooperative behavior and coordinated dependencies. Finally, we discuss accountability devices protect against free riding and nodes that are known to misbehave.

IMPLEMENTING SECURE PEER-TO-PEER SYSTEMS

Anonymous Transport: Tarzan

Peer-to-peer systems can serve disparate functions, and while many of them primarily act as applications or services, they can also serve as transport-layer anonymizers. The Tarzan system is a decentralized, distributed Chaumian mix system that enables client applications to seamlessly direct traffic through an anonymous network at the transport layer. Unlike a centralized server-based anonymous traffic system like Anonymizer, Tarzan's P2P nature ensures that no one actor needs to be trusted for anonymous communication at the IP level. Users can direct their traffic into a tunnel through the network of peers, so the packets exiting the tunnel cannot be traced back to the original sender, even in the face of substantial network-wide traffic analysis.

Tarzan, as described by Freedman and Morris (2002), works as follows. Each node selects, from what it can see of the network, a set of peers to act as mimics. Initial node discovery and subsequent network maintenance is based on a gossip model, where nodes share their view of the network. Mimics are selected randomly from the available nodes for security and balance, in some verifiable manner. Each node exchanges a constant rate of cover traffic of fixed size packets with its mimics using symmetric encryption. Symmetric keys are distributed using the relays' public keys. Actual data can now be interwoven into the cover traffic without an observer detecting where a message originates.

To build a path, the sending node randomly selects a given number of mimics and wraps the message in an "onion" of symmetric keys (Syverson, Goldschlag and Reed, 1997) from each node on the path. The sender passes the packet—outwardly indistinguishable from cover traffic—to the first node in the chain, which removes the outermost wrapper with its private key, and then sends it along to the next node. With the exception of the last node, each node in the chain is aware of the node before and after it in the chain, but has no way of telling where it is in the chain itself. That is, the node cannot tell if it is the first hop or the penultimate hop. The final node in the chain of mimics acts as the Network Address Translator for the transport layer, and sends the packet to its final destination through the Internet. This final node must know the content and destination, but has no information about the sender.

Nodes store a record for the return path, so a reply from the Web host contacted can be received by the final node in the chain, rewrapped with its private key, and sent back to the penultimate hop. The message is then passed back through the chain, with each node adding another layer of encryption. The originating node can use the public keys of each node to unwrap the layers and read the message. Since it is the only node to know the public keys of each hop along the path, the content is secure.

The P2P nature of this system makes a network decentralized, and thus highly scalable. The presence of cover traffic and the fact that all nodes are peers means that no actor inside the system or out can readily identify the originator of the message. Unlike onion-routing, the sender does not have to trust the first hop, and the list of available nodes can be dynamically maintained for participant flexibility. Moreover, the majority of cryptographic work is performed by the original sender, which must decrypt n public keys to read the reply through and n-node chain. Each node in the chain only performs one encryption or decryption, and implementation results have shown that the computational time is dominated by the expected latency of the underlying Internet.

Efficient Robust Routing: Secure Overlays

Like Tarzan, routing overlays also seek to provide a secure layer of communication, but they differ in their security objectives. Rather than protecting the anonymity and content of communications, routing overlays such as Chord, CAN and Pastry try to provide mechanisms for nodes to access objects in a fashion resistant to malicious interference. Overlays use their namespace structure to build robust routing mechanisms, so that an adversary with a significant number of nodes in the system could not cripple communication and access throughout the system. How the nodes and objects manifest themselves at the application layer is not directly relevant to the overlay design: these systems can be used for storage, publication, or even multi-casting.

The salient feature of all routing overlay models is a large ID space, from which node identifiers and object keys are defined, forming a large, distributed hash table. Each object key is mapped through the overlay to a unique participating node, called the object's *root*. The protocol routes messages with a given object's key to the object's root. An object is replicated throughout the system to ensure greater robustness; each protocol uses a replica function to map an object's key to its replica's key, which can then be used to find the copy of the object stored at the replica's root. Like any hash tables, lookup can be very efficient, but distributed hash tables pose unique security questions.

Each overlay system is somewhat different, of course, but most follow a given structure. The generic system here is based on the discussion in Castro, Druschel, Ganesh, Rowstron and Wallach (2002) and the Pastry system (Rowstron and Druschel, 2001). Nodes maintain two lists of other nodes, a routing table of other nodes throughout the ID space, and a list of "neighbors" or nodes that are in close proximity inside the number space. Each entry in the table consists of an identifier in the namespace, and the IP address of the node belonging to that identifier. To route a message, a node consults these tables, and attempts to forward the message closer to the identifier keyed in the message. Routing tables are organized by key prefix, such that a host will try to find the node with an identifier closer in the namespace to the query key. Failing that, it will pass the message along to its neighbors in an attempt to find the proper location in the namespace. If the key is not found in the routing table or the neighborhood, the query is dropped.

This system is effective, but is not in and of itself secure, since there is no guarantee that an object will be placed on legitimate, trusted replica nodes. Nodes must regularly maintain their routing tables for arrivals and departures of other nodes. Such operations are difficult, especially in the face of high node churn, but are generally feasible in well-behaved networks. More importantly, malicious nodes or collections of nodes can seek to undermine routing efforts, or even censor documents. For instance, an attacker could surround an object's root with collaborating neighbors, and not pass on any message to that object's root. To prevent this, secure overlay systems implement secure routing primitives.

First, node identifiers must be guaranteed to be randomly assigned to every node. If this is not the case, nodes under malicious control can simply choose their IDs to ensure that all the keys of a given object will point to hostile nodes. This randomization is not difficult if there is a centralized server solely responsible for handing out identifiers on joining the network; proof-of-work techniques or micropayments can be used to prevent Sybil attacks or pseudospoofing. Of course, P2P systems are often designed to explicitly avoid a centralized server of any sort, even one used only to assign names. Wallach (2002) acknowledges that, without a central authority, current means of assigning random identifiers can impose additional security risks. Routing table maintenance must also be secure against attack, so that the chances of having a corrupt node in ones routing table will not be greater than the percentage of corrupt nodes in the system.

Secure overlay models can provide for message delivery to as many non-corrupt nodes as possible. Much of this is predicated on the fact that one's ID space neighbors are very likely to be geographically disparate, and thus less likely to be under the control of a single adversary. If we to sacrifice some performance by filling the routing table based on diversity rather than low latency, security increases. In Pastry, when less than 30% of nodes are not trustworthy, there is a 99.9% probability that at least one valid replica will receive the message (Castro et al, 2002). This particular defense may decline in utility if the attack is coordinated through a large set of geographically dispersed peers, such as an array of PCs under the control of "Trojan Horse" malware.

One criticism of routing through the overlay model, rather than a higher application level, is that it is harder to eject a misbehaving node. While nodes may recognize that other nodes are not behaving as they should, proving systematic misbehavior is difficult at the routing layer. Apparently malicious behavior could be ascribed to the underlying network, and vice versa. Thus, any accusation mechanism could be used to attack legitimate nodes and the system itself.

Secure overlay mechanisms and Tarzan both focus on message delivery, but with different security objectives. Tarzan focuses on protecting the end parties, their identities and the content of the message. A secure overlay like Pastry is designed to protect the traffic flow itself. Both can be implemented in such a fashion to aid in the others goal. Tarzan, for instance, can add on control mechanisms to guarantee flow control, and pastry can add layers of encryption, and even use a multi-hop system to replicate a mix-net. Neither of these solutions, however, would be as secure in their objective as the other system. Security objectives of P2P systems need to be designed in from the beginning.

Censorship-resistant Anonymous Publishing: Freenet and Free Haven

P2P systems can also be application-specific. One of the more common applications of P2P systems is publication and storage of documents. Again, the security design of these systems is largely based on the purposes of the system and the security goals of the designers. Several systems have been designed for the express purpose of censorship-resistant publishing, to make real the rhetoric of the freedom of ideas that the Internet was supposed to enable. However, a single server can be blocked for an entire nation (Zittrain and Edelman, 2002) and open documents can be identified and blocked. Secure P2P systems can be much harder to block, since they lack a single point of attack to block or destroy documents. We present, for comparison, two such systems, Freenet (Clarke, Sandberg, Wiley, and Hong, 2000) and Free Haven (Dingledine, Freedman, and Molnar, 2000b).

Freenet is designed to acknowledge that censorship can take the form of eliminating sought-after documents, or punishing those who can be observed reading those documents. It consists of a network of peers that host encrypted documents. Peers use keys to request documents. Its two unique features are the structure of its storage space, and the system of using keys to access content.

The storage mechanism of Freenet can be thought of as a nested caching structure, in that data is stored as it is used. A peer request propagates through the network, with each node knowing only the node that passed the request along. A node that can positively respond to the request will pass the document back through the chain, one node at a time. Each node may keep a local copy of the document, in addition to sending it along towards its destination, so that it can respond to future requests itself, rather than passing the query to network neighbors again. This means that copies of the document will be duplicated as they are requested. It also prevents a single hosting node from being overwhelmed by requests for a popular file. Finally, this local caching policy causes a locality of resources, where documents tend to be hosted close to regions of the network where the document is heavily demanded. As nodes fill their storage space with documents that they pass along to others, they will replace older and larger documents first. This leads to a more efficient use of network space, where peers host documents that are highly demanded and small enough not to be a burden on network resources, as well as creating an incentive to minimize file size of documents stored on the network.

As mentioned above, peers use keys to locate and access the encrypted documents on the network, and there are four key styles that can be used. Keys are used to identify the document sought, and decrypt it once it is obtained. The simplest is just a key derived from a hash of the document itself. This insures that the key is unique, and the document is tamperproof. However, they can be difficult to remember, and how does one obtain the key of a document one doesn't know about ahead of time? Moreover, the fact that the documents are tamperproof with respect to the key means that they cannot be edited (capabilities that allow you to trust the integrity of an editable document require an infrastructure like that of Groove, described below). For easier searching, Freenet allows keys to be derived from keyword strings. A one-way function is used to convert the keywords into a key, which is also used to access the document as above. This method has two flaws, however. First, popular key words can be vulnerable to dictionary attacks, where malicious nodes will attempt to use the client's transformation function to find the key value of specific search strings. With this, they can censor specific search strings while still appearing, to an outside observer, to be a fairly well-behaved node. Moreover, there is no guarantee that multiple separate documents will not be signed with the same key word generated keys. Imagine, for example, the number of distinct documents that could be "The Bible." These two flaws make false replies fairly simple.

To allow for document editing, Freenet offers signature verification keys, where document keys are generated along with a public/private key pair. This allows the author of the document to resign the key with every edit, enabling a persistent pseudonym in the form of a public key. Users can now better trust the authenticity of a document as associated with a specific online identity. Finally, redirects allow queries for specific key words to return the content hash key, rather than the document itself, and the hash key can obtain a verifiable copy of the original document.

All documents in the system are encrypted with some type of key, so the host node may not know what documents it is serving at any point. This also allows plausible deniability should any one legally or physically attack the owner of a node. The creators of Freenet acknowledge that if a document is notorious enough, its digital signature will be recognized, and hosts can purge it from their system and refuse to process requests. However, should some one wish to have it in the system, it is a simple matter to extract it, change a single bit to generate a completely new digital signature, and reinsert the document into the network. Actively trying to remove a document from inside the system is difficult.

Freenet's P2P structure ensures that documents spread in a "sticky" fashion to resist attempts to stamp out popular information, while still protecting those who would either access or host that information.

Free Haven, like Freenet, is designed to be a censorship-resistant publishing network, but places its design priorities on highly robust anonymity and document durability. Free Haven uses cryptographic techniques to protect the identity of the reader, the server, the author, the document and the query, preventing any party other than that which is explicitly authorized to read or link information. Unlike some of the other P2P systems described in this chapter, its routing structure is relatively simple but uses very sophisticated techniques to protect anonymity and robust storage in a hostile environment.

As usual, content is kept on some number of machines in the network. The user, either a server in the network, or an outside machine, broadcasts a request to as many servers as possible, who then reply directly to the querying machine. The initiator sends a request to a server that consists of the following: the initiator's public key (which can be temporary), a hash of the public key of the document sought, and a reply block. A reply block consists of an encrypted nested chain of addresses, so that a party can use it to pass a message to a destination through several intermediary parties which do not know either the originator or destination of the message, or the content. If any recipient server has a document whose hash matches the hash of

the given key, they encrypt that document with the requester's public key, and send it back using the reply block. Thus, while the client must know of the existence of a server, it has no information about which server offered the document; nor does the server knows anything of the client.

Free Haven's strength as a P2P publisher lies not just with its robust anonymity, but its document management which maintains the files in the system against strong attacks. Before inserting a document into the system, it is encrypted and broken up into a large number of shares, only some fraction of which are necessary to recreate the document. That is, a client can rebuild the document without finding every share. The parameters of the number of shares generated and the number required for regeneration are chosen to minimize the number of shares necessary for reassembly, but also minimize the size of each file that must be acquired. Each share has a duplicate "buddy" share, and the two contain location and storage information about each other for system robustness. Shares also have expiration dates, before which servers make an effort to keep shares alive.

Once on a server, the document shares do not necessarily rest on the same server, but rather they are traded between servers. Trading provides cover traffic to hide actual information queries, as well as allowing servers to find a home for their shares before exiting the system. The constant movement also makes it more difficult to locate and attack all the shares of a document. The buddy system creates some degree of accountability, as a share will notice if its buddy is no longer available from the server on which it was last residing. Each trade involves updating the buddies of the shares traded. Servers hosting machines with lost buddies can communicate to other servers their knowledge of the unreliability of the negligent (or malicious) nodes. We discuss how the system uses reputation to track good behavior below.

Free Haven was designed to be robust in the face of many different attacks. Reader and server anonymity are protected with the reply-block mechanism, but can always be subverted with embedded malicious code in the documents, or very extensive traffic analysis across the network. To attack the publishing system, an adversary would have to acquire many shares. Simply destroying servers will not help, if the number of required shares for reassembly is small enough. Conversely, increasing the number of shares in the entire network minimizes the chances of deleting both a share and its buddy to attack the accountability mechanism.

Neither Freenet nor Free Haven can be said to be truly superior, or even more secure, since they have different design objectives and thus, different security demands. The designers of Freenet did not worry about a network-wide traffic analysis attack, since they assumed any adversary with that many resources could mount a more direct legal or physical attack on users (Langley 2000). The designers of Free Haven chose to take advantage of recent cryptographic developments to build a system that could stand up to a powerful cryptanalytic attack. At the same time, Free Haven was concerned with protecting *all* documents in the system, while Freenet was more concerned with guaranteeing access to those highly demanded at any given time. Dingledine et al (2000b) suggest that their system might be used to protect a phone book from a war zone, while Freenet designers envisioned spreading a current issue of a dissident newspaper. P2P systems must be built for specific goals, and their security should reflect this.

Secure Groupware: Groove

Groupware is a designed forum and set of tools for sharing information and communication channels, to enable collaborative work. Designing a secure P2P workspace is every bit as challenging as the systems described above for two reasons. First, a workspace has multiple internal applications, and the system must securely provide them all. Second, businesses that design workspaces must make them accessible to the average corporate computer user; user intervention as complex as assembling a reply block, or even verifying a digital signature can be too complex for a product to be widely attractive. Ray Ozzie's Groove Network is a collaborative digital space that is built to be secure, but designed to have as much security as possible completely transparent to the user (Groove Networks 2002). At the same time, security mechanisms need to be self-organizing from the client level, rather than dictated by a policy administrator. The proprietary system was built to be used in a business environment, which means not only the potential of a highly-motivated adversary, but the likelihood of a highly heterogeneous environment. Groove is designed to ensure a robust collaborative work environment even with some parties behind firewalls, others on low-bandwidth channels, and still more with intermittent connections. Moreover, everyone in the group may not expressly trust the other actors *inside* the system either. Protection is offered at a finer level than the enterprise. The system employs fairly established cryptographic techniques, but uses them in innovative ways.

Groupware requires the negotiation of trust between appropriate parties while excluding those who do not belong. Each user in Groove has an account resident on each machine, and one or more identities associated with each account. This enables users to use the same identity from multiple devices, as well as allowing a single user to have multiple identities in different contexts. A software engineer can use Groove to collaborate with his firm's research group, as well as work with several friends to develop a startup business without having to worry about information leaking between the two projects, or even allowing either project too know about the other. Identities are manifested in shared spaces, which can either allow for mutual trust or mutual suspicion. In the former, users are presumed to trust each other's identities, so verifiability of authorship is relaxed in favor of efficiency. In a mutually suspicious space, Groove enables tools to verify authorship of statements. We describe how this works below.

Whoever decides to initiate a shared space decides whether to declare it a trusted or suspicious space, and can specify several security parameters if desired. All data is encrypted, both on the wire and on disk. Messages are reliable, and even in a trusting mode, Groove can offer confidentiality and integrity of messages. Privileges can also be revoked.

All applications running on the Groove network use "delta" messages, or document updates that communicate changes of state to other members of the network, whether in a collaborative file or in a chat session. Each delta is encrypted using the group's symmetric key for fast encryption and decryption. The group has a separate shared key for authentication, which is used to sign the digest to ensure message integrity; both keys are reestablished every time some one leaves the group. Local files log all changes initiated by any actor.

When users might have reason to be suspicious, however, shared keys are not sufficient to instill trust in the integrity of authorship. After the author signs the delta message with the group encryption key, he signs a separate hash of the message body using his private key and the public key of the recipient. As implemented, Groove actually precomputes distinct pairwise keys from the Diffie-Hellman public keys of each group member and the sender's private key, which results in a smaller key and a less expensive verification procedure. Thus, appended to each message there will be a signed hash for all recipients in the group, who can use their own pairwise key of the sender and their private key to verify authenticity. The sender can even sign the message with her own key to verify its integrity on another device.

Messages do not have to broadcast to all nodes, but can use their local knowledge of the network structure to pass the delta messages to key central nodes. Moreover, messages are sequenced, so if a recipient misses a message for some reason, such as being offline, she can fetch another copy from the sender, or from any other member of the group. Other members of the group leave the pairwise keys intact, allowing the node to be confident of the authenticity of the message and its sender.

Of course, all this depends on entry into the groupspace, trusted or untrusted. Untrusted here does not mean a purely malicious actor, since all members of any group will have the ability to read messages sent through the group. Rather, it refers to a competitive situation, where users may need to know for certain who sent which message. If this is not the case, significant encryption overhead can be saved. Still, how does a user join a trusted group?

Suppose Alice wants to invite Bob into a Groove shared space. Bob must have a copy of the client software, or course. The most important aspect is for both Alice and Bob to trust that the identities "Alice" and "Bob" in the system match their belief of who the individuals actually are. This requires some out-of-band communication. That is, Bob must verify that the information being sent actually belongs to Alice, and vice versa. This process can be as simple as a phone call, where both trust that the phone number leads to the other person, and they can recognize the other's voice. Using some trusted non-Groove channel, they can verify the authenticators supplied by Groove (a public key) are valid, and Bob can join the group. There is no need to change the existing group keys, since Bob may need access to existing files. Should the existing group want to deny Bob access to certain information in the group, they must form a new shared space for Bob's involvement that does not involve the protected documents.

Groove has a fairly rigid set of security policies, but the P2P nature of the system protects system security, and increases system usability. Lacking a central point of failure, or a nexus where permissions can be negligently or maliciously opened, Groove's decentralization improves robustness. Anyone can start a group, but little active administration is required: all encryption after the initial trust phase described above is transparent to the user. This openness means that the parties do not need to rely on a third party to negotiate trust among group members.

Secure Distributed Storage

Using a P2P system for simple storage, rather than interactive or static publishing, poses a different set of security requirements. The encryption used for document security and integrity does not have to be as complex, since only the original owner need access the document itself: a strong symmetric cipher is completely adequate. Storage is a problem of coordination. A user may seek to store files on a network to as a standard data back-up measure, or to hide a document for later retrieval. Keeping data on a range of nodes offers strong fault tolerance and survivability, compared to the single point of failure in a solitary remote storage site. The security problem to be solved by a P2P storage system is instead a coordination problem.

It is obvious that the amount of storage consumed by a given node must be in proportion to the amount of storage space they will give the network. Otherwise, free riding will result in too much data for the storage space available. There are a number of fairly straightforward ways for a centralized administrator to coordinate storage. A third party can give each client a quota, and refuse access to the system for any part breaking their quota. This third party must be trusted and reliable, however. Some sort of currency can be used for storing clients to pay warehousing clients, but this requires a trusted currency and clearing mechanism. Alternatively, the trust can be placed in hardware, with a smart card containing a secure internal counter and an endorsement of the right to store information on the network. Of course, this only removes the need for centralization by one degree, since a trusted party must issue the cards, sign attestations about a right to store and verify that storage has been reclaimed before certifying a new counter value.

Centralized authority is not necessary if storage is distributed symmetrically. For ephemeral resources, such as downloading a single file, a tit-for-tat mechanism works well. The BitTorrent file distribution system lets peers download segments of large files from other clients also trying to access the same file. Clients essentially swap the segments that they have not yet downloaded. Each client uploads to others based on how much they have downloaded from those clients. Thus, every client has an incentive to upload what it has to other machines, so it can continue to download. This works well, but is not designed for long-term storage. A reduction in demand for a file will limit the number of BitTorrent clients that will swap segments to complete the download.

Long term storage necessitates accountability. Of course, a symmetric configuration constrains how storage arrangements work out, if each actor must trace through a network of obligations before finding someone who is owed storage space. If we use a simple barter system for the entire network, where a given actor can simply claim they have a right to store data on any computer in the network based on their own local storage, some certification is still needed for that claim. A trusted group of quota managers can prevent free riding by verifying those claims, but there is no real incentive for these nodes to take responsibility for auditing an entire network.

The storage system presented here by Ngan, Wallach and Druschel (2003) offers a mechanism for the publication and auditing of storage records for an entire network without relying on a single trusted party. Each node publishes a record of their storage activities, and a series of random, self-interested auditing is enough to encourage honesty.

Each node in the network advertises its total local storage capacity, the list of others' files stored locally, and its own files stored remotely. The first list can be thought of as the 'credits' to each node and the second as 'debits' to the system. Each record in this usage file contains an identifier and size information for the file, as well as an identifier and location information for the node. When node A seeks to store a file on another machine, the potential host examines the balance sheet of the would-be storer. If A is not consuming more storage than its capacity, the remote machine accepts storage if it has room itself. Each agent that is storing a file for node A finds that it is in its self interest to check on A to confirm that A does indeed have the hosted file in its debits file. Otherwise, the node in question can delete A's stored file and free up disk space for others to acquire most system 'capital.' As long as A does not know when any node will audit its published file, it will keep its debit file accurate for fear of losing stored data. This auditing mechanism prevents a node from hiding its own use of the storage network in order to store more than its share. It is important to note that the auditing communication must be anonymous and randomly timed, so that A cannot detect which creditor is auditing it and coordinate its response accordingly.

This standard auditing does not prevent a greedy node from claiming that it is storing files it actually is not. That is, A could claim a huge capacity, but then falsely assert that it is being used by some other node. Thus, A would not have to share capacity, but could use the capacity of the network. Fully auditing all outgoing obligations would be very expensive. The file I claim I store should be on some node's debit list, but a chain of obligations could go on for some time. Eventually, there must be an "anchor" that claims it is storing a file that is on no node's debit list. Finding this anchor, however, involves searching through deep trees of dependency. If all nodes are honest, there is no effective way to bound this search. The system requires that all nodes occasionally pick a node at random and examines its outgoing list. While this is not as effective as a full recursive audit, Ngan et al show that it is much cheaper and shows that perpetrators can still be caught with high probability even in the face of infrequent audits. Moreover, since the usage file has been digitally signed, it provides hard evidence of malfeasance, and the node can be ejected.

This model of a P2P storage architecture focuses on security against node collusion to free-ride on the system. Nodes can encrypt data for storage to protect it, and knowledge of some identifying information about the other node is necessary to coordinate storage. Like other P2P systems, it could be further enhanced by a reputation mechanism. If nothing else, an incentive to acquire a good reputation serves as a disincentive to get thrown out of the system once a good reputation is obtained. This is further discussed in the next session. Distributed storage can be an effective way to securely back up data, but attention must be paid to a secure alignment of incentives.

REPUTATION AND ACCOUNTABILITY

Peer-to-peer systems require the effective cooperation of disparate parties. Unlike a centralized client-server model, where system satisfaction rests only with the successful performance of one party, many different peers must all work together within the confines of the system. Yet systems must reflect the fact that there may well be actors not eager to cooperate, that may seek to free-ride on the resources of others, or actively subvert the system. Fortunately, the initial solution is trivial—avoid interactions with nodes that do not behave. Unfortunately, this is much more complex in practice. A simple list of who not to trust requires good information collection and distribution, and does not scale well. Use of reputation can make this easier.

A full discussion of reputation--or even an attempt at a formal definition—is outside the scope of this chapter. However, reputation presents several interesting security issues, as a solution and a new problem. For our purposes, reputation can be thought of as information about an actor that can aid in the prediction of future behavior. It serves two complementary purposes. First, the reputation of other actors can guide a decision maker's choice in selecting transaction partners. Second, it can act as an incentive for good behavior for those who fear acquiring bad reputations. A bad reputation often involves some sort of punishment or reduced privileges, so a good reputation can be a motivating factor for good behavior.

Reputation systems have been a popular topic in recent information technology literature, and they have blossomed in practice as well. The Search for Extraterrestrial Intelligence (SETI) is a P2P system that uses spare processing power on users' PCs to examine radio telescope data for patterns that might lead to the discovery of life beyond our solar system. It uses an advertised positive reputation system with group identities to encourage people to share computer resources. The desire to acquire a good reputation for social reasons grew so strong that people began to cheat, undermining the entire signal processing project. SETI was forced to implement an accountability mechanism that replicates all computations for quality control. This security control reduced the effectiveness of the system by half.

In general, reputation requires an enduring identifier of some sort, and the ability to map information about past transactions to that user. If a user wishes to base its opinion on anything other than her own experiences, she can solicit

reputation information from others, but now must have some metric to trust the information of others. This need for credibility adds an additional layer of complexity to the system. There are several ways to apply reputation to a P2P system.

First, one can ignore reputation within the system, and rely on users or applications to obtain and use the information they need. The P2P system can make this easier with good security. For instance, a Groove workspace offers no aid to a participant in evaluating the quality or trustworthiness of a document or fellow participant. Groove can, however, use cryptographic techniques to enable the user to know exactly who sent what information, so that an out-of-band reputation system based on external identities can be mapped into a Groove-supported work environment. It is also possible to imagine a reputation system layered on top of a P2P system, where users can define their own parameters of trust, and simply rely on the underlying P2P layer to provide information, which is evaluated at the reputation layer. This information can then be passed back to the P2P layer for a decision about storage, transmission, or whatever function the P2P system provides. By layering them in this fashion, the reputation system can be adjusted independently of the P2P system, following good security procedures of separating functionality whenever possible.

Dingledine, Mathewson and Syverson (2003) take the opposite approach, arguing that bolting security into an existing system is the incorrect solution. Their Free Haven system uses published reputations to encourage servers to keep shares they have promised other servers they will keep. Recall that documents are kept alive by servers trading them. Any server finding its trading contract broken can broadcast proof malfeasance with the signed contract and certified proof of failure to deliver. Verifying disputed claims grow to be complex very quickly, however, especially if the document in question was traded to another party. Free Haven uses credibility metrics to evaluate contested claims, which are in turn built from reputation records over time. The additional computation required for secure attestations places a fair bit of overhead on the system, without full incentives for any one node to follow through on the veracity of others' claims. The authors acknowledge that "designing the original system without a clear idea of the reputation and verifiability requirements made Free Haven's design complex and brittle."

The proprietary file swapping software Kazaa also built in some degree of reputation to prevent free riding. If more than one user requests a given document from a server, the requesters are queued, and order in the queue is determined by reputation, rather than order of arrival. Positive reputation is accrued by sharing files with others. This mechanism was not designed with any security protections, however, so when the source code was reverse engineered, competing vendors distributed a client which could set its own reputation. Since all reputation information is kept locally by the client itself, there is no structural way to prevent this. Where the reputation mechanism is integral to the design of a P2P system, the reputation mechanism must itself be secure. Otherwise, it becomes the weakest link in a system that might otherwise be secure.

A final aspect of the reputation mechanism is the cost of acquiring a new identity. If a fresh reputation is easily acquired, then anyone with a bad reputation will handily jettison their bad name for a new one. This creates a world where bad reputations simply won't exist. Unfortunately, an actor choosing by reputation must now assume the strong possibility that anyone with a new (neutral) reputation is untrustworthy. This puts honest newcomers at a disadvantage. One way around this is to require some form of payment to obtain a new identity in the system. Of course, this also imposes a penalty on newcomers. Friedman and Resnick (2001) have shown that any identity-bound reputation system with non-permanent identities must impose some cost on newcomers, either as an outright entry fee, or with a below-neutral reputation.

A final way to enforce good behavior is through a transitive reputation tokens, or a micro-currency. Rather than fixing reputation to an identity, micro-currency can be thought of as signaling information that can be passed from one person to another. There are many security issues with micropayment and electronic currency schemes that have been discussed elsewhere. (See e.g. Camp 2000) One system that implemented a successful micro-payment scheme is Mojo Nation, a P2P file-sharing system that used tokens known as "mojo" to encourage good behavior (Wilcox-O'Hearn, 2002). Every transaction required an exchange of mojo, so the more altruistic actor received something for giving up something of value. Thus, each actor had an incentive to offer files to the network, in the form of the ability to barter for more files from others. Mojo Nation avoided some of the cryptographic hurdles of a distributed system by relying on a centralized token-server for market clearing. However, the market was structured in such a way that trading could continue even if the token server was temporarily unavailable.

One definition of a secure system is one in which a user can have confidence of appropriate functionality. Peer-to-peer systems need security to defend proper operation of the underlying mechanism, but may also need additional layers of security to encourage proper operation of other peers. Some of the systems described in this chapter, such as the secure overlay, are designed to require as little reputation information as possible. Others, such as the secure storage system, are predicated on their ability to overcome the free rider problem without a centralized coordinator. Reputation and accountability mechanisms can play a key role in the security and functionality of P2P systems, but they must be designed and implemented as an integral part of the system.

CONCLUSIONS

Peer-to-Peer systems are incredibly flexible and can be used for a wide range of functions, often more effectively than their client-server analogs. They can be used to anonymously or robustly route through a network of peers, protect content from censorship, enhance and protect collaborative processes and coordinate use of diverse services. Security could be a concern to any of these functions, and should be treated accordingly. In the systems discussed in this chapter, the security is always on: no administrator can accidentally or maliciously turn it off. This constant presence and absence of centralized support means that

good security practices must be built into the protocols of the system. Each system has different design goals, and just as the security should reflect the purposes of the system, system design should reflect security exigencies.

GLOSSARY

Anonymity Not associated with a personal identifier. In the context of information networks, it is the ability to keep identifying information about the user from being known. There are different degrees of anonymity based on the set of possible identities, and the strength of the attacker determined to ascertain a given identity

Censorship Attack An attempt to keep the content from being accessible. It can manifest itself as an attempt to prevent content distribution, to prevent the content from being located, to pollute the content beyond use, or to expunge the content completely from the system.

Delta Message A message in groupware that contains only the changes to a given document. Delta messages often contain auditable information to create a history of the document, and security information to maintain document integrity.

Distributed Hash Table (DHT) A hash lookup system that routes queries through nodes, where the local routing table of each node is used to forward the query to the node that is ultimately responsible for the key.

Centralized Describes a system with a single point of control, with policy emanating from that one source. Such systems can be easier to directly administer, are useful for coordinating different tasks that need common information, such as sequential ID assignment, and can offer great economies of scale. However, the users must trust the centralized server; they are also particularly vulnerable to targeted attacks such as Distributed Denial of Service Attacks.

Decentralized Describes a system designed and implemented without a single administrative point of control. Decentralized systems can be more fault tolerant if properly designed, but it can be harder to coordinate behavior within the system.

Groupware Software that enables shared, closed environments for an identified set of users, particularly for collaborative work. It can allow users in different environments to share common tools and documents.

Locality A node's place in the network topology, either in the overlay network or the underlying internet architecture. Systems can be designed to capitalize on locality, such as Freenet's ability to "push" content closest to where has been recently demanded, or ignore locality, such as a secure overlay's tendency to map close internet addresses to random parts of the namespace.

Namespace A context formed by a set of unique identifiers that allows each object referenced to have a mapping from identifier to object such that there is no ambiguity to which object a name points to. The Domain Naming System (DNS) is an example of a namespace where every internet object is referenced by at least one URL, and every URL references at most one object.

Node A point on a network that is capable of sending or receiving information, either for itself or to relay a message to or from others.

Onion Routing A system for anonymous network communication designed to be resistant to both eavesdropping and traffic analysis from threats inside and outside the system. Messages are passed wrapped in layers (like an onion) of encryption, each corresponding to the key of one of the routing nodes, so that only the destination node can decrypt the message, or even determine the route of the message.

Overlay network A communication network that functions above the transport layer. A virtual network that is built on top of the basic internet architecture and that provides additional functionality such as searching, routing, encryption, etc.

Out-of-band A message that is not passed across the network in question, but through some other channel. Out-of-band messages can be used to help secure the identity of a party where different channels can reinforce beliefs the end parties have about their respective identities. Out-of-band messages can be passed on the same physical network but using a secure overlay network, as with signaling the telephone network.

Peer-to-Peer (P2P) A style of network in which nodes communicate to other nodes via ordinary intermediate network nodes, rather than a centralized server with full knowledge of the network so that each node performs client and server operations. Most P2P systems are overlay networks with unique namespaces.

Reputation Information about a actor's past action that is believed to be is correlated future behavior.

Reply Block A list of anonymous remailers with the recipients address at the end, used to direct a message to the final account without disclosing the address to the sender, or any intermediary. Reply blocks are used to allow anonymous communication between two parties, where the sender might want to receive a reply from the recipient, but not disclose her identity.

Shared Space In a groupware environment, the shared space is the common namespace of specific set of nodes in a single zone of administrative control. Common tools and documents can be available to all users while the groupspace exists.

Signature Verification Key The public half of a public/private key pair, used to verify that a document was signed by the private key. In the Freenet P2P system, a signature verification keys allow a document to be tracked by a pseudonymous author's signature verification key, rather than a key based on a hash of the document, or specific key words.

Survivability The ability to survive while under attack by exhibiting gradual degradation rather than failing entirely. Characterized by graceful degradation rather than catastrophic failure.

Sybil Attack Also known as pseudospoofing, an attacker acquires multiple identifiers in the system to undermine some function of the system, such as attempting to control a large portion of the key space in a distributed hash table. These attacks have proven hard to defend against in open, anonymous systems, and often inspire the use of reputation or cost driven overlays.

CROSS REFERENCES

REFERENCES

Camp L.J. (2000) *Trust and Risk in Internet Commerce*. Cambridge MA: MIT Press.

Camp L.J. (2003) Design for Trust. In R Falcone (Ed.) *Trust, Reputation and Security: Theories and Practice* (pp. 15-29) Berlin: Springer-Verlang.

Castro M., Druschel P., Ganesh A., Rowstron A., & Wallach D.S., (2002) Security for Peer-to-Peer Routing Overlays. *Fifth Symposium on Operating Systems Design and Implementation (OSDI '02)*, Boston, MA.

Clarke I., Sandberg O., Wiley B., Hong T.W. (2000) Freenet: A Distributed Anonymous Information Storage and Retrieval System. *Workshop on Design Issues in Anonymity and Unobservability (PET2000).* Berkely, CA

Dingledine R., Freedman M.J., Molnar D. (2000a) Accountability Measures for Peer-to-Peer Systems. In A Oram (Ed) *Peer-to-Peer: Harnessing the Power of Disruptive Technologies.* Sebastopol, California: O'Reilly.

Dingledine R., Freedman M.J., Molnar D. (2000b) The Free Haven Project: Distributed Anonymous Storage Service. *Workshop on Design Issues in Anonymity and Unobservability (PET2000).* Berkely, CA.

Dingledine R., Mathewson N., Syverson P. (2003) Reputation in P2P Anonymity Systems. *Workshop on Economics of P2P Systems*, Berkeley, CA.

Douceur, J.R. (2002) The Sybil Attack. *1st International Workshop on Peer-to-Peer Systems (IPTPS '02).* Cambridge, MA.

Freedman M.J., & Morris R. (2002) Tarzan: A Peer-to-Peer Anonymizing Network Layer. *ACM Conference on Computer and Communications Security (CCS 9).* Washington, D.C.

Friedman, E. & P. Resnick (2001). The Social Cost of Cheap Pseudonyms. *Journal of Economics and Management Strategy* 10(2): 173-199.

Groove Networks (2003) "Groove Security Architecture" White Paper Available at : http://www.Groove.net/pdf/security.pdf (Date of Access: February 25 2004).

Kan, G. (2000) Gnutella. In A Oram (Ed) *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. Sebastopol, California: O'Reilly.

Langley, A. Freenet. In A. Oram (Ed) *Peer-to-Peer: Harnessing the Power of Disruptive Technologies.* Sebastopol, California: O'Reilly.

Ngan, T.W., Wallach, D.S., & Druschel, P. (2003) Enforcing Fair Sharing of Peer-to-Peer Resources. *2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)* Berkeley, CA.

Rowstron A. & P. Druschel. (2001) Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. *IFIP/ACM International Conference on Distributed Systems Platforms*, Heidelberg, Germany.

Syverson, P.F., Goldschlag D.M., & Reed M.G. (1997) Anonymous Connections and Onion Routing *IEEE Symposium on Security and Privacy* Oakland, CA.

Wallach, D.S. (2002) A Survey of Peer-to-Peer Security Issues. *International Symposium on Software Security* Tokyo, Japan..

Wilcox-O'Hearn, B. (2002) Experiences Deploying a Large-Scale Emergent Network. *1st International Workshop on Peer-to-Peer Systems (IPTPS '02)* Cambridge, MA

Zittrain J and Edelman B, (2003) Internet Filtering in China *IEEE Internet Computing* 7(2): 70-77.