Final thesis

Analysis and Simulation of Threats in an Open, Decentralized, Distributed Spam Filtering System

by

Gabriel Jägenstedt

LIU-IDA-EX-G-12/008--SE

2012-09-04
Final thesis

Analysis and Simulation of Threats in an Open, Decentralized, Distributed Spam Filtering System

by

Gabriel Jägenstedt

LITH-IDA-EX-G-12/008--SE

Examiner: Professor Nahid Shahmehri
Dept. of Computer and Information Science
at Linköpings universitet
Abstract

The existence of spam email has gone from a fairly small amounts of a few hundred in the late 1970’s to several billions per day in 2010. This continually growing problem is of great concern to both businesses and users alike.

One attempt to combat this problem comes with a spam filtering tool called TRAP. The primary design goal of TRAP is to enable tracking of the reputation of mail senders in a decentralized and distributed fashion. In order for the tool to be useful, it is important that it does not have any security issues that will let a spammer bypass the protocol or gain a reputation that it should not have.

As a piece of this puzzle, this thesis makes an analysis of TRAP’s protocol and design in order to find threats and vulnerabilities capable of bypassing the protocol safeguards. Based on these threats we also evaluate possible mitigations both by analysis and simulation. We have found that although the protocol was not designed with regards to certain attacks on the system itself most of the attacks can be fairly easily stopped.

The analysis shows that by adding cryptographic defenses to the protocol a lot of the threats would be mitigated. In those cases where cryptography would not suffice it is generally down to sane design choices in the implementation as well as not always trusting that a node is being truthful and following protocol.

**Keywords**: spam; filter; electronic mail; trust; threat; mitigation; trap
Acknowledgements

I wish to thank my examiner and supervisor Nahid Shahmehri, for having the patience that enabled me to finish this thesis at my own pace. On top of that thank you for giving me a goal to strive for and the confidence to try. I would also like to thank Maria for pushing me to finish when I was going too slow. Finally a big thank you to Jonatan and Rahul for invaluable comments on the thesis.
## Contents

1 Introduction ........................................... 1  
   1.1 Spam .............................................. 2  
   1.2 TRAP ............................................... 3  
   1.3 Purpose .......................................... 4  
   1.4 Question ......................................... 5  
   1.5 Method ............................................ 5  
      1.5.1 Threat Analysis ............................... 6  
      1.5.2 Simulator .................................... 6  
      1.5.3 Simulating attacks ......................... 7  
      1.5.4 Analyse simulations ....................... 8  
   1.6 Sources .......................................... 8  

2 TRAP explained ...................................... 9  
   2.1 Trust Metric ..................................... 9  
   2.2 Nodes ........................................... 10  
   2.3 Routing .......................................... 10  

3 STRIDE Analysis .................................... 11  
   3.1 Data Flow Diagram ............................... 11  
   3.2 Threat Matrix ................................... 12  
   3.3 Choosing Threats ............................... 13  

4 Simulator Redesign ................................ 14  
   4.1 Overview ......................................... 14
<table>
<thead>
<tr>
<th>Section</th>
<th>Start Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2 Design Choices</td>
<td>15</td>
</tr>
<tr>
<td>4.3 Message Handling</td>
<td>16</td>
</tr>
<tr>
<td>4.4 Simulator Input</td>
<td>16</td>
</tr>
<tr>
<td>4.5 Problematic Threats</td>
<td>16</td>
</tr>
<tr>
<td>5 Simulation</td>
<td>18</td>
</tr>
<tr>
<td>5.1 Baseline</td>
<td>18</td>
</tr>
<tr>
<td>5.2 Reporters</td>
<td>19</td>
</tr>
<tr>
<td>5.3 Holders</td>
<td>19</td>
</tr>
<tr>
<td>5.4 Setting up a Simulation</td>
<td>19</td>
</tr>
<tr>
<td>5.5 Output</td>
<td>20</td>
</tr>
<tr>
<td>6 Results</td>
<td>21</td>
</tr>
<tr>
<td>6.1 Baseline</td>
<td>21</td>
</tr>
<tr>
<td>6.2 Fake Reports</td>
<td>22</td>
</tr>
<tr>
<td>6.3 Fake Responses</td>
<td>23</td>
</tr>
<tr>
<td>6.4 Duplicate Responses</td>
<td>25</td>
</tr>
<tr>
<td>7 Discussion</td>
<td>29</td>
</tr>
<tr>
<td>7.1 Analysis</td>
<td>29</td>
</tr>
<tr>
<td>7.1.1 Holder</td>
<td>30</td>
</tr>
<tr>
<td>7.1.2 Reporter</td>
<td>31</td>
</tr>
<tr>
<td>7.2 Conclusions</td>
<td>31</td>
</tr>
<tr>
<td>7.2.1 Fake Reports</td>
<td>31</td>
</tr>
<tr>
<td>7.2.2 Fake Responses</td>
<td>32</td>
</tr>
<tr>
<td>7.2.3 Duplicate Responses</td>
<td>32</td>
</tr>
<tr>
<td>7.3 Future Work</td>
<td>32</td>
</tr>
<tr>
<td>A STRIDE Analysis</td>
<td>34</td>
</tr>
<tr>
<td>A.1 External Entities</td>
<td>34</td>
</tr>
<tr>
<td>A.1.1 Sender</td>
<td>34</td>
</tr>
<tr>
<td>A.2 Data Flows</td>
<td>35</td>
</tr>
<tr>
<td>A.2.1 Sender to Receiver</td>
<td>36</td>
</tr>
<tr>
<td>A.2.2 Receiver to Requester</td>
<td>37</td>
</tr>
<tr>
<td>A.2.3 Requester to Holder</td>
<td>39</td>
</tr>
<tr>
<td>A.2.4 Holder to Requester</td>
<td>39</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

In the year 1971 the first email was sent in ARPANET [1], it was not THE beginning, as some form of mail had been in existence for several years [2]. But this was the year when a computer and user were first distinguished [1], and as such it was most certainly some form of beginning. The mail systems of today do not necessarily have much in common with those systems of old but the basics remain the same; a means of sending a message from one computer to another.

The Simple Mail Transfer Protocol (SMTP) which is used in modern email applications is described in RFC 5321 [3]. In short SMTP is built on a client-server model where a user uses an SMTP client and sends mail to an SMTP server. However a server can also act as a client and relay received messages to another SMTP server.

In Figure 1.1 we show how mail traffic moves over the network. Let us assume that Alice wants to send an email to Bob. Starting at the computer of Alice the mail is sent to an SMTP server, this server does a Domain Name System (DNS) lookup to find the address for the SMTP server in charge of Bob’s computer. Finally the message is sent to the server which promptly delivers it to Bob.

Here the SMTP servers can be considered to be an Mail Transfer Agent (MTA) and the computers of Alice and Bob are both an Mail User Agent (MUA). An MTA uses SMTP to communicate while an MUA uses a fetch
1.1 Spam

In 1978, seven years after the first email, what is considered to be the first spam email ever was sent [4]. This mail was sent by a marketer at Digital Equipment Corporation (DEC) to several hundred recipients. This may not sound like a lot but since the ARPANET on the west coast, where the spam was sent out, had a grand total of 1200 possible targets, the action was quite grievous indeed. The marketing ploy got a lot of attention and most of it negative, much like reactions to spam today, although now the number of spams are closing on 200 billion [5].

It does not take much imagination to realize that given the SMTP network described in Figure 1.1 a lot of traffic will be generated to send emails that no one wants to receive. Every spam that is sent through the system uses resources that could be better utilized elsewhere or for other purposes. For a single mail this might not be a big problem but considering the sheer amount of spam in circu-
Figure 1.2: The TRAP Protocol

In short a great deal of computer resources get used to filter out unsolicited mail. This problem is only related to the actual resources used by the spam and does not take into account malware or scams that might come with the spam. In short, just as in 1978, spam is still very much unwanted and unwelcome.

1.2 TRAP

The TRAP protocol was designed as a way to determine how likely it is that a specific sender is sending spam. This information can be used by a spam filter to decide how harsh it should be in its filtering against the sender.

We accomplish this by having a peer-to-peer network keep track of the reputation of a certain sender, this knowledge is called trust. The protocol is extensively described in Shahmehri et al. [6] but we will briefly mention the integral parts here.

In short TRAP consists of a set of nodes that have different roles in the TRAP network. These nodes are as shown in figure 1.2:

- Senders, who send mail to some recipient that is using the network. These are not strictly considered a part of TRAP but we have to take them into account in the protocol due to the fact that the protocol is designed to reason about how trusted they are. This would be difficult if they had no representation.

- Holders, sometimes called a Trust Holder, keep track of the how
trusted another node is, known or previously unknown to the network. This is the node which implements the Trust Metric described by Shahmehri et al. [7]. The metric is designed to run in a peer-to-peer network and as such can handle trust calculations even when it is impossible to know if one or several participants are playing fair.

- Receivers, who receive messages from Senders and either request to know how trusted the Sender is or report that a message has been received or both.

- Reporter, the part of a receiver that sends reports about experiences. This node may be considered a part of a receiver but is often represented as a separate node to simplify the protocol as well as the analysis.

- Requester, a node which asks for trust values of a Sender. This node is generally implemented on a Receiver but does not require it.

Important messages in TRAP are requests, reports and responses.

- Requests are sent to Holders when a Receiver wants to know the trust of a Sender.

- Responses contain a Holders trust value of a Sender.

- Reports contain an experience of an interaction with a Sender, this experience is used to update the trust currently stored about a Sender.

1.3 Purpose

This thesis aims to determine the effectiveness of certain threats, attacks and mitigations within the TRAP protocol. We want to find out if a threat can be turned into an attack and if such an attack where to occur, whether or not the proposed mitigation is effective.
1.4 Question

It is impossible to simulate all the threats of the analysis. In part this is due to time constraints but it is also caused by the fact that some threats cannot be simulated reliably. This led us to choose threats which are fairly easy to simulate. We also tried to choose threats which either seemed likely they would be simple for an attacker to abuse or that are particularly effective. Finally it is important to note that the threats were only chosen to the best of our knowledge and there may have been better choices. However, even if better choices exist the main purpose of the simulations is to provide a proof of concept that shows that the simulator can be reliably used to test attacks and mitigations. We have chosen a set of threats that leads us to the following questions.

- How does sending fake reports affect trust?
- How can we defend against fake reports?
- How does sending fake responses affect trust?
- How can we defend against fake responses?
- How does sending duplicated responses affect trust?
- How can we defend against multiple responses?

1.5 Method

To answer these questions we will be using simulations to emulate the behaviour of the protocol. In order to write the simulator we need to know what requirement we have for it. The requirement process will start with a threat analysis of TRAP and selecting what threats to analyse further. Knowing what threats we will simulate gives us an idea of what requirements we have of the simulator. We were given a working simulator for the trap protocol. It was originally used in Shahmehri et al. [6] however it was not designed to allow for threat modelling. In order to comply with the requirements we will be modifying the simulator to better suit our needs.
Finally, with simulator finished we will write tests for the chosen threats and run the simulations. The effect of the simulations will be shown in graphs that visualize how trust is affected by attacks and mitigations.

1.5.1 Threat Analysis

In order to understand what parts of the protocol could be in danger we need to first analyse the existing protocol outline [6]. For this we employ a method developed by Microsoft called Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service and Elevation of Privilege (STRIDE) [8]. Using this method we enumerate the threats against TRAP without taking into regard what is improbable or even impossible due to protocol constraints. It is from this set of threats that we have chosen the threats that we can simulate and analyse further.

A short explanation of the threats follow.

**Spoofing** When an attacker attempts to claim it is someone else. Mostly used to impersonate nodes in the network which have privilege of some form.

**Tampering** A type of attack where an attacker tries to change some form of value. This could be a return address or a trust value for example.

**Repudiation** The attacker claims that it has not sent a certain message. By trusting an attacker attack venues such as replay attacks are opened.

**Information Disclosure** An attacker gets access to information that they should not be able to access.

**Denial of Service** A Denial of Service (DoS) attack makes sure that some form of service is unavailable to other nodes. This is typically done by overloading the service.

**Elevation of Privilege** The threat of an attacker gaining privileges that it should not have.

1.5.2 Simulator

The simulator that we have chosen to use is designed to emulate only the existence of spam and how this would affect the trust values of a given
holder and is limited in scope. The main purpose of the pre-existing simulator appears to be to test the possible parameters of the metric and to observe how it reacts in a complex network. This leads to the simulator not implementing certain features that we need:

- It does not provide a way to add functionality to nodes, this makes it impossible to create a simple change in behaviour without duplicating a lot of code.

- It does not separate trust values, meaning that all holders store their calculated trust in the same variable. This makes it impossible to implement malicious holders.

- It does not have the debugging facilities or output handling that would be desirable when working with a large number of tests. That is it lacks a framework since it was designed to run only a single simulation.

- It does not implement all the messages in the TRAP Protocol, most notably it does not implement responses, something that will be very interesting to simulate for us.

On the positive side the simulator has a good amount of functionality and has an extensive config where a lot of settings can be modified. By extending on this config and as well as writing a system that can add functionality on demand we hope to create a more robust simulator that will be able to run a lot more types of tests than previously possible.

1.5.3 Simulating attacks

The simulator is implemented on top of Pastry [9], a routing layer protocol. The actual simulation is twofold, first designing tests and second running them in the simulator. We want the simulator to be able to simulate the behaviour of the TRAP protocol as closely as possible while still making it possible for nodes to subvert the protocol and act maliciously.

Tests or simulations will primarily be implemented by changing the interface of nodes so that they handle messages in a different way than originally intended.
1.6. Sources

There are some limitations on what can not be tested. We will not be able to spoof who sent messages and thereby test a system for signatures. This is due to the fact that Pastry does not allow a message to have a different sender than the endpoint it is sent from. This may be a reasonable limitation but makes spoofing hard to monitor.

1.5.4 Analyse simulations

After we have run all the simulations the generated data must be analysed. That is the final part of the thesis but some ground work will have to be done where each test output needs to be sorted and converted into graphs. Using these graphs it will be a lot easier to see how trust behaves under different settings and node behaviours.

1.6 Sources

This report builds in large on the existing protocol description of TRAP and the metrics it uses [6][7].
We use the simulator designed by Shahmehri et al. [6] as a basis for our own.
Furthermore it draws from an analysis made of the protocol. This analysis has not previously been published but will be included in the appendix.
Chapter 2

TRAP explained

TRAP was designed to help in the ongoing battle against spam. It attempts to provide information about email by monitoring the original senders. In essence it keeps track of how much spam has been encountered previously from the sender, this information can be used by a spam filter to determine what defence procedures to take when handling the mail [6].

2.1 Trust Metric

The most important mechanic of TRAP is the trust metric. The metric is the mathematical formula for deciding whether a sender is trustworthy or not. Since TRAP is intended to run in peer-to-peer systems the metric is designed to be dynamic and tamper resistant [6].

With this in mind there are two types of trust factors in TRAP; short term trust which reacts quickly to changes in behaviour of the node and a long term trust factor which remembers how the node has acted over time. The combination of these two factors helps metric adapt to the ever changing landscape of a peer-to-peer network. In this thesis we will only be looking at short term trust.
2.2 Nodes

All nodes in TRAP get assigned a unique ID calculated from their Internet Protocol (IP) address by hashing the IP with SHA256. This means that the only way to get access to a new ID is by changing your IP. Furthermore this means that it must be considered hard to choose your ID. This notion is very important to TRAP and is one of the reasons that targeted attacks in TRAP can be considered hard [6].

Trust is stored in a holder that gets assigned to the ID. This is done by hashing the ID once again to find the node that should be responsible for storing its trust. This is done several times so that each Sender has a set of Holders that handles its trust. This ensures that no single Holder will become too important in managing trust. At best a Holder could slightly lower or raise the trust value calculated at a Requester.

2.3 Routing

Message Routing in the system is implemented in Pastry, a peer-to-peer system which also handles data lookups. The nodes that need to participate in the Pastry layer of TRAP are Reporters, Requesters and Holders.
Chapter 3

STRIDE Analysis

The threat analysis of TRAP was made against the protocol described in Shahmehri et al. [6]. It was run as a STRIDE analysis and as such is mostly concerned with the security of the protocol itself. It is designed to look at messages and nodes that send and receive messages. This means that there are areas that the analysis was not able to cover. For example, it does not differentiate between a malicious node sending random data or a malfunctioning one. It can only be used to show that some node is sending data that is incorrect. This simplification results in a model which is a lot easier to analyse but that may ignore parts of the protocol. However we believe that the model is good enough and shows an accurate view of the actual protocol.

3.1 Data Flow Diagram

The first step of the analysis was to create a Data Flow Diagram (DFD) seen here in figure 3.1. The DFD has a number of dotted lines which show trust boundaries. These trust boundaries show where the protocol has high risk of being tampered with in some way. More specifically, it shows data channels that pass between nodes such that it is impossible to know that the other party can
3.2 Threat Matrix

Table 3.1 adapted from Hernan et al. [8] shows what STRIDE threats can affect which elements of the protocol as modelled in the DFD. This tells us that processes in a DFD can be considered most vulnerable to attack. They might not be the easiest to attack but certainly are vulnerable.
to the largest amount of attack venues.
Since a process is basically input and output we are able to emulate their
behaviour by deciding what messages they receive and send. We will be
using this knowledge when we write our simulator later.

3.3 Choosing Threats

Although a full analysis of the protocol was done this section focuses on the
threats named in 1.4, fake responses, fake reports and duplicate responses.
The effects of positive and negative responses can be observed by looking
at the requester in figure 3.1. The same line of reasoning can be used in
determining the effects of both fake reports and duplicate responses.
While a process may be vulnerable to several different threats we are only
looking to model the actual attack and its mitigation. This means that an
attack could just as well happen in transit between nodes. Thus a model
will not show one single threat but several and any mitigation should be
equally effective against them all.
This led us to a solution where we wrote tests that react on a node getting
a message in the normal fashion of the protocol and observing how trust
is calculated when that attack is active. We where mostly concerned with
how reporters and holders could attack the protocol.
The full STRIDE analysis can be seen in appendix A.
Chapter 4

Simulator Redesign

The existing simulator for TRAP that was used as the base in [6] has a lot of the functionality needed for our thesis. We will here try to describe the simulator and what has been done with it to make it run the simulations we want.

4.1 Overview

The simulator consists of several parts. The underlying layer to TRAP in this case is Pastry [9]. Pastry handles the message passing and base functionality of TRAP.

TRAP uses a Peer To Peer (P2P) protocol design to facilitate storing knowledge about trust in the network. The main simulator is setup in a few steps. First it loads the configuration which defines the boundaries and parameters of each test that should be run. For each test the simulator will overwrite the base configuration in order to run a specific test.

When a simulation starts it initializes an environment as well as the configuration settings needed to run a simulation. Certain settings are initialised by a specific instance that is passed to all nodes adding a type of shared memory. It keeps track of the nodes in the network in order to be able to lookup information that would otherwise be hidden from all nodes, such as
total number of messages processed in the simulation.

All nodes of the network are created and assigned the services they should perform. Without a service pack nodes pretty much just forward messages. Once all nodes have been created they are booted into the network. This will result in nodes sending mails if they are senders or initialising essential values otherwise.

After this the network starts running and keeps sending messages and receiving them until it has finished the test.

At this point a final write to the test log will occur and then the next test will start.

### 4.2 Design Choices

TRAP would ideally assign holders to a sender dynamically. However the simulator will only assure that a certain amount of holders is assigned. The simulator will not choose these according to the methods defined in the TRAP paper [6]. Instead the simulator adds the exact amount of holders into a node array. This method works well enough for our simulations.

The glaring problem that faced us was the fact that there is no way to differentiate between different nodes. This indicated that we had to redesign the simulator to allow more freedom when assigning node functionality. Since we want to assign functionality to nodes depending on tests, we found it unfeasible to use inheritance. Instead we decided to use a method called dispatching in order to enable a dynamic test framework. This is described in section 4.3.

Another issue we faced was that there was functionality missing from the simulator, such as messages never being sent in the network. This was the case with responses not being sent to requesters, something that is needed in order to implement fake responses and duplicate responses as well as defense against these attacks.

Also the simulator was designed such that it would only run one test and then quit. Finally there was no way to differentiate between different tests and parameters as there was no way to handle output from more than one run.
4.3 Message Handling

The most noticeable change in the simulator is how message passing is handled or more precisely message receiving is handled. This is done by implementing what we call a dispatcher. A dispatcher in this case is an instance that keeps track of what functions to call when a message is received by a node such as a holder or receiver. Each instance of the dispatcher can add functionality to the node. It will then call those functions when the right type of message is received. The dispatcher is not capable of removing functionality which may make it a bit harder to implement nodes that change their behaviour over time. However this functionality should be fairly trivial to add for someone who wants to run such tests.

All nodes in the simulator have handlers attached for handling messages. By overwriting those handlers it is possible to introduce new behaviour into the network.

4.4 Simulator Input

In order to make testing dynamic there had to be certain changes to input and output of the simulator. On top of the original configuration system we added parameters depending on tests that overwrite the default settings in a modular way.

The simulator is now capable of simulating a set of tests with different spam values. It can now simulate any type of misbehaving node that sends and receives messages.

4.5 Problematic Threats

While a lot of threats can be simulated a few are quite difficult. The most obvious problematic threats are threats that take up fairly small amounts of resources for individual attackers but need a lot of attackers to function thus resulting in a simulation needing vast resources of memory or storage to run.
Another problematic attack is sender spoofing. Pastry does not allow the messages sent via an entry point to have a different sender than is named in the actual message. This may well be possible to get around in some manner but we deem it outside the scope and time frame of the thesis.
Chapter 5

Simulation

The first step in the simulation part of the thesis was to select a set of threats to model and test in the simulator. Since the original threat analysis was quite extensive we tried to limit the selection considerably. Apart from the limits of the simulator itself we wanted a proof of concept that would show that the simulator does what it should and that the tests are realistic. The most obvious attacks are those performed by a reporter or by a holder. This is due to the very nature of TRAP, a requester for instance is rarely able to send messages that can subvert the protocol. This is why we chose to simulate the types of attacks that would be common on those type of nodes in the first case.

5.1 Baseline

In order to be able to compare the tests against something we created a baseline set. This is comprised of a test at all the spam levels that we want to look at. In this case we have one test for no spam coming from the sender and one test where all the mail is spam.
5.2 Reporters

For reporters one would typically be concerned with those malicious nodes that send fake reports, that is either sending reports about messages that have never been received or modify actual experiences. A slight variation of this is reporters that send more than one report of each experience. In general we can compound these two slightly different attacks into one attack that sends multiple fake reports. In short the later attack is merely a more powerful version of the previous one.

5.3 Holders

When it comes to holders there are two obvious threats to the system. One is fake responses, the other is duplicate responses.
Fake responses can in theory be used in TRAP if you own a majority of the holders. It is far easier to slander a node then it is to give it positive ratings since TRAP is more prone to listen to negative feedback than positive.
Another attack by a holder node that the protocol had no defense against in its original implementation is when a holder decides to send duplicate responses to a request and thus giving greater weight to the malicious nodes responses than the nodes who only sent a single responses.

5.4 Setting up a Simulation

Creating the simulations was fairly straight forward. A lot of effort was put into making sure that adding a test will be easy.

1. Firstly we need to decide what type of malicious nodes should be active in the test. The malicious nodes are written as classes implementing malicious behaviour and added to the base node as modules.

2. Secondly we need to decide what parameters the test should change. We do this by creating a config file where we can set number of total nodes, number of malicious node per type as well as a test name.
3. Thirdly the handlers have to be imported into the main simulator, this is currently done manually but could be made automatic with a little work.

Ideally we would have a system where we put the handlers and properties file in a folder and they are then imported and assigned dynamically, however we had to limit the time spent on this part of the project so that it did not get out of hand.

5.5 Output

There was a lot of data generated from these simulations. The data of interest is how trust values change at the holders and what trust value gets reported to the requesters.
Chapter 6

Results

The following is the results gathered from the simulations. In all the graphs you will see two sets of values. The actual trust as well as perceived trust. The second value is the trust value that the requester calculates from the information they receive from the holders.

By observing received trust and determining how it differs from actual trust, we are able to see how efficient attacks and mitigations against the protocol are. Seeing a big difference indicates that the requester has been tricked into believing that the trust is something other than it really should be.

The chosen graphs are from one holder of five in each case. All other graphs where virtually equal since the attacks we test do not really change any values on any single holder, even malicious holders keep the knowledge of the actual trust.

6.1 Baseline

The positive and negative baselines give a pretty consistent behaviour showing that without outside factors trust behaves just as we would expect. The baseline is shown in figures 6.1 and 6.2.
6.2 Fake Reports

For the sake of the tests being as obvious as possible, all malicious reporters send three reports per mail received. A fake report can be explained as a reporter claiming that something has happened when it has not. In the case of fake reports actual trust and perceived trust has the same value. The reason for this is that perceived trust is calculated at the requester and never differs from actual trust when reports change value.

Figure 6.3 shows how trust behaves when a reporter claims that mails are spam even though they are not. By comparing this to 6.1 it becomes obvious that negative feedback has a huge impact on the trust levels. Negative feedback greatly lowers trust fairly quickly. The trust does not converge as fast as in other tests however.

Figure 6.4 shows the trust levels when the mail is spam and the reports
say that it is not. As in the previous test we can see a difference compared to 6.2. Note that the difference here is only slight, the convergence comes only a few packages later. Depending on the amount of active reporters it might be possible to converge to zero even later.

6.3 Fake Responses

A malicious holder is capable of sending any trust it wants to a requester. This is pretty straight forward and there are no implemented defenses against this. These tests exist mainly to show how important a real solution to misbehaving nodes is for TRAP.

Figure 6.5 shows a holder attempting to give beneficial feedback about a node that is spaming. This figure shows that the difference between the
6.3. Fake Responses

Figure 6.3: Reports claim spam when none exists

Figure 6.4: Reports claim no spam when spam exists
The next level of malicious holder is one that duplicates responses. This means that for each request they respond several times leading to their trust values being weighed higher by the requester. This attack is essentially the
same thing as fake responses where an attacker was able to take over several holders.

Figures 6.7 and 6.8 show the effects of duplicate responses with and without enforcing unique responses. The difference between these two graphs is that in 6.7 the perceived trust is obviously lowered while enforcing unique responses shows that only the effects of one fake response is prevalent as in 6.3.

Figures 6.9 and 6.10 show the same but with the network sending spam and just as before, by comparing these two graphs with 6.4 one can see that unique responses make this attack behave like fake reports.
Results

Figure 6.7: Duplicate negative responses, no spam

Figure 6.8: Duplicate negative responses, unique messages, no spam
6.4. Duplicate Responses

Figure 6.9: Duplicate positive responses, spam

Figure 6.10: Duplicate positive responses, unique messages, spam
Chapter 7

Discussion

In this section we analyse the graphs presented in chapter 6. We also present our conclusions and finally some ideas for future work in this area.

7.1 Analysis

The analysis is split into attacks against the holders and attacks against the reporters. In order to have something to compare the tests against we ran baseline tests. These tests did not have any added behaviour or defensive measures.

The baseline was applied with the simulation running at different spam levels. The spam level is defined between zero and we decided to create baselines for both maximum and minimum spam. We chose the extremes since all values between would only test the attacks at a smaller degree.

It is obvious from Figures 6.1 and 6.2 that under small levels of spam trust values rise and under large amounts of spam the trust values converge to zero.
7.1.1 Holder

It is interesting to see how trust levels are affected by a Holder sending fake responses to the Requester. This can be seen in Figures 6.5 and 6.6. When there is no spam and the holder is sending negative feedback the results are slightly worse than during the baseline. The same goes for positive feedback during high spam. This is due to the fact that the metric for combining trust does not take into account the behaviour of the holders and trusts them all equally. Ideally TRAP should distinguish between wellbehaved and misbehaving nodes, however this is outside the scope of this thesis.

Sending duplicate responses is something that is not covered in the original protocol[6]. However it is quite obvious that by sending duplicate responses and having the Requester accept them as correct, a Holder could potentially gather a greater weight in the trust calculation acting as though it has a majority of the votes on the subject.

The effects of such an attack are identical to getting control of more holders but a lot simpler to accomplish. In our test our malicious holder sends five responses thus getting five votes of eight. By looking at the baseline 6.1 and comparing to our attack 6.7 we can see that compared to the baseline this attack manages to smear the reputation of the sender without much problems.

While this may seem like a big issue it should be rendered impossible if duplicate responses are not allowed. This can be done by assigning a nonce to each request, unique for every holder and enforce signatures to accept a response. In this way only one response ever gets used from each holder and none is able to get more votes. The results of this defensive measure can be seen in 6.8. By simply ensuring that a holder can only send one report or rather only allow one report to be counted we notice that this defense makes the attack behave like a normal fake response attack. This allows the same defensive precautions used against fake responses to be used against duplicate responses as well. In the same way Figures 6.2, 6.9 and 6.10 show both an effective attack and mitigation to achieving a higher trust than the metric would normally allow for.
7.1.2 Reporter

Due to the nature of TRAP there can be no simple way to determine whether a report is authentic or not. This leads to attackers being able to affect the trust metric under the correct circumstances. By comparing the baseline of no spam 6.1 with a malicious reporter sending fake reports 6.3 we can see that with these settings of TRAP it is fairly easy to provide negative feedback on a sender and lowering its trust significantly. Actual trust in this case seems to be very susceptible to malicious reports. As soon as the negative reports stop it will eventually fall back to normal values. This attack does appear to provide a very simplistic denial of service attack.

We do not have any working defenses against this attack. Possible solutions to the attacks are designing some form of system for verification and authentication of reports. Another solution might be to ensure that only specifically trusted nodes are allowed to report experience. In the current state, where anyone can be a reporter issues like this will turn up.

In Figures 6.2 and 6.4 we can see that the effect of positive reports is not nearly as effective and actually TRAP seems to handle that situation well.

7.2 Conclusions

In this thesis we have investigated attacks on TRAP as well as their mitigations. The attacks are discussed below.

7.2.1 Fake Reports

We started by evaluating how fake reports affect the system. We have noticed that fake reports are more effective for lowering trust than gaining it. This makes the attack effective for slander but not as effective for getting spam past the system. Since the primary concern of TRAP is to hinder spam false positives is a lot less problematic than false negatives.

With this said we believe that although fake reports can be effective it is not something that will hinder TRAP from working well. It would certainly be good for TRAP to add some form of authentication for reports anyway, if for no other reason than to minimize the traffic.
7.2.2 Fake Responses

One of the most basic attacks against trust is to take control of a holder node and modify its return values. Although this might seem like a very easy attack it is far from simple to find out what node or nodes to attack. This is primarily due to the way that TRAP assigns holders. In IPv4 an attack such as this might be feasible since creating a rainbow table of IP->ID is simple with a reasonable amount of work, however if TRAP is implemented on IPv6 it is not really a reasonable attack. See Appendix B for details.

Now if one was able to get total control of a set of holders an attack would be possible. Attacks on this level will be fairly effective if the weight of all holders opinions are counted as equal. This type of attack will be mitigated efficiently if some form of trust value where to be implemented for holders as well.

7.2.3 Duplicate Responses

In the case of duplicate fake responses the same as above applies with the added issue of a single holder being able to fake that they are more important than other holders. Though this on the surface looks like a very big problem it has a fairly simple solution which puts it on the same level as any node sending just one fake response.

In order for this attack to be possible we had to trick the simulator into allowing more than one report. By making sure that all requests are signed and have a unique id as well as an id for the overall request only one response will ever be counted. In order to allow duplicate responses it is necessary to use information that would not normally be known to a requester, more precisely how many fake responses would get sent so that the average could be calculated reliably.

7.3 Future Work

From the analysis made of TRAP and the simulations done here we think that the most important venue of further work on TRAP is to introduce
some form of trust values for holders. This would greatly benefit the protocol and make it a lot more stable. We also think that by using some form of cryptographic techniques, signed messages and authenticated reports would become more feasible. This would be beneficial for TRAP in general since a lot of the threats against the protocol can be mitigated with authentication.
Appendix A

STRIDE Analysis

In this section we present a listing of threats to specific parts of TRAP. Each section lists what could happen if one of the six attacks spoofing, tampering, repudiation, information disclosure, denial of service and elevation of privilege where to be successful. Refer to figure 3.1 for a model of the protocol.

A.1 External Entities

Only one important external entity has been observed in the system. There is no way for external entities to directly influence TRAP as they are only subject to Spoofing and Repudiation.

A.1.1 Sender

**Spoofing**

_Threat_
The initial contact in the system is made by the sender. If the IP can be spoofed in this contact the rest of the system will be bypassed. This means that TRAP relies on IP being hard to spoof. It is hard to determine how TRAP would be affected by anonymisers such as
The Onion Router (TOR) or remailers.

**Mitigation**
TRAP must be implemented on a protocol which makes sure that spoofing is not possible or at the very least is hard.

**Manifestation**
If TRAP is badly implemented spoofing will be possible, there is nothing in TRAP that prevents sender spoofing since there is no authentication, in other words if the system is deployed on a system that allows sender anonymity or claiming other names spoofing will be possible.

**Repudiation**

**Threat**
If it was possible for a sender to send spam and claim to have not sent them they would be in a very good position. One way to do this is by spoofing the sender.

**Mitigation**
TRAP should be implemented on a platform that does not allow spoofing. It should also validate all messages that come from a sender. This validation can be used for experience reports.

**Manifestation**
A repudiation attack could manifest through spoofing of the sender. If the validation step is not done right it might be possible to claim that reports are made in error.

### A.2 Data Flows

According to Fig 1 there are 8 data flows that have to be analysed. It is important to note that although 2-3, 4-5 and 6-7 are depicted as different flows it is hard to distinguish between them, this distinction has been made due to the fact that sometimes a receiver acts as a requester/reporter and sometimes not. If they are separate nodes the flow will be important
and small variations might occur. Data flows are affected by Tampering, Information Disclosure and Denial of Service.

A.2.1 Sender to Receiver

Tampering

Threat
If it was possible for an attacker to tamper with mails sent by trusted senders they could send spam. However this will not stop TRAP from eventually branding the malfunctioning sender as malicious.

Mitigation
The trust metric used in TRAP should be able to detect and compensate when a single node starts behaving badly and sending spam. It might become an issue if a spammer is able to do this in bulk and tamper with large amounts of flows. Since this flow is unsecure mitigation should happen in the metric.

Manifestation
Can be achieved by spoofing the sender. Other ways to tamper with the flow requires violating the integrity of the channel and the message itself. Since mail has no real integrity protection the protection lands on the channel. There seems to be quite a good amount of possibilities with subverting channels such as performing a Man in the Middle (MITM) on a WiFi router. TRAP will report and adapt to messages from a sender since the channel can not be trusted.

Information Disclosure

Threat
Mail is by design unable to protect information sent. This is outside the scope of TRAP and will not affect the inner workings of the Trust metric.

Mitigation
The only way to ensure privacy of email is by encryption. TRAP does not attempt to protect information received via this flow since the information is not secret.
Manifestation
Information disclosure can manifest through either a side channel attack or by observing the channel and message. Since there is no confidentiality in mail and it is easy to mount a MITM, TRAP is vulnerable to this attack.

Denial of Service

Threat
By aiming a Denial of Service at the connection between sender and receiver a mail will not arrive at its destination, but this should not affect the metric.

Mitigation
No mitigation possible or needed during a direct attack. If the attacker is attacking the destination it might be possible to prevent attacks.

Manifestation
A denial of service can manifest in many ways. The only real ways that can be dealt with by TRAP is if the receiver has been tampered with, spoofed or is under a DoS attack itself. All direct attacks on the data flow are outside the scope of TRAP.

A.2.2 Receiver to Requester

Tampering

Threat
If an intruder can access the contact sent and change the ID to a known trusted ID, the returned value will be modified. Even if you can not change an ID you could possibly change bits of a message and thereby make it impossible to actually do the check. This could lead to the message being ignored by the requester.

Mitigation
It is important that requests are authenticated in some way, for example by signing the messages. Since the information in a message
could be wrong it is very important with decent validation and input sanitization. There are also timestamps that are added to messages. To address MITM it is important to secure messages in the network by forcing sender and receiver authentication.

Manifestation
The integrity of a request is non-existent if the receiver can be spoofed. Depending on the hash used to protect against replay attacks it might be possible to create messages that give the same hash but have different plain texts. Attacks on the channel that would allow messages tampering to occur are in general either dependent on MITM or some form of routing attack on Pastry.

Information Disclosure

Threat
By observing what IDs pass through the system checks it might be possible to create a map of what IDs are frequently passing by a node. If it is possible to turn the ID into an IP it could be possible to (via other means of cracking) get access to nodes that pass messages through your node and abuse this knowledge. It might be possible to use rainbow tables to find the IP of an ID. See B.

Mitigation
If this information is to be unknown it is vital to implement TRAP ontop of a protocol that has more IP that IPv4.

Manifestation
A logical attack would be to use rainbow tables to find the real IPs if TRAP does not give them up of its own free will. Note that this will work with IPv4 but not IPv6. This can only happen if there is a way to make observations in the channel. A node inserting itself in the route path can obviously read the messages.

Denial of Service

Threat
If an attacker does not want IDs to be checked they could attempt
to bring down the chain that propagates the messages. This means having to take down a lot of nodes who most likely are highly diversified in nature and location. It appears unlikely that it could be done in reasonable time.

Mitigation
By deploying defences against node failures and having recovery systems in place it is possible to ensure that it will be hard to stop the transfer of messages.

Manifestation
An attack against one of the endpoints might lead to a DoS. It might also be possible to saturate the link by flooding it with more messages than can be handled by the system. An attacker could corrupt messages in such a way that no messages will be accepted by the requester leading to dropped messages.

A.2.3 Requester to Holder
See A.2.2

A.2.4 Holder to Requester

Tampering
Threat
If an attacker is able to access and change values in these messages it could lead to control of what trust the requester thinks the sender has.

Mitigation
There should be some system for ensuring that messages and values are authentic. For example Trust values (or entire responses) should be signed. Also TRAP mitigates this threat by having several holders all with their own paths making it hard to change values in all of these responses.

Manifestation
The integrity of a request is non-existent if the holder can be spoofed.
This should be mitigated by signing. Only the real holders would be able to create valid responses. Depending on the hash used to protect against replay attacks it might be possible to create messages that give the same hash but have different plain texts. It is important that the encryption of signatures is strong. Attacks on the channel that would allow attacks on messages to occur are in general either dependent on a highly unpredictable man in the middle or some form of routing attack on Pastry.

**Information Disclosure**

**Threat**
If an attacker can access this data flow it could be able to build a table of what nodes have what trust and possibly use this for their gain. But since any node can request this information it should not be a problem if such information does not give an attacker the advantage.

**Mitigation**
It might be beneficial to ensure that any intermediate node will not be able to read the trust values in transit. This should be needed only if it can be considered a problem that nodes on a path know what trust values are propagated along that path. It could potentially be worse than being able to find trust for nodes you choose to lookup via requests. If it is considered a problem some form of cryptographic solution should be able to solve it.

**Manifestation**
If there is no encryption information can be disclosed. If information should not be disclosed to an attacker there has to be a working encryption of data as well as making sure that the algorithm is strong. It is important that the encryption process is not easily ignored by the use of side channel attacks.

**Denial of Service**

**Threat**
If it was possible to stop the messages containing responses it would
be possible to bypass the checks on an IP and force the receiver to assign default trust in senders. It might primarily be used to remove bad trust from a sender however it could also be used to reduce the trust of other parties to default forcing a lot more work for the spam filters. If it is possible to force other trusted nodes to get default trust it might be possible to create a bottle neck which could lead to a DoS at another point in the network or deny good service to nodes that should have it.

Mitigation

It is very important that default trust is not beneficial to an attacker, if it is not it will be of no use to attack this flow. Since TRAP is dynamic and distributed there should be no single route that could be DoSed which would lead to a full compromise of the system.

Manifestation

Can be realised through tampering width the data flow. Even if it is impossible to falsify messages it could be possible to change them in such a way that the intended information can not get to the receiver. It would also be possible to realise a DoS by incapacitating endpoints in some way, making sure the message either is not sent or is not received. An attacker could attempt to send a response before the real response is sent. This can work if there is no sender authentication or if there is some weakness in the authentication. It should be a fairly simple process to corrupt a message if it passes a malicious node. However if encryption is utilized it would be impossible to know what messages are about what senders. It might also be possible to incapacitate the channel by consuming essential resources for the network such as returning huge amounts of fake responses. However this should be mitigated quite well by the fact that TRAP is decentralized, there is no single point of failure.

A.2.5 Requester to Receiver

See A.2.4
A.2.6 Receiver to Reporter

**Tampering**

**Threat**
If an attacker can alter reports in transit they can change the claimed experience.

**Mitigation**
Reports should not be modifiable after creation. Enforce signing or encryption.

**Manifestation**
An attacker could pose as a receiver sending fake responses. Signatures mitigate this fairly well. Only the holders would be able to create valid responses. Depending on the hash used to protect against replay attacks it might be possible to create messages that give the same hash but have different plain texts. It is important that the encryption of signatures is strong. Attacks on the channel that would allow attacks on messages to occur are in general either dependent on a highly unpredictable man in the middle or some form of routing attack on Pastry.

**Information Disclosure**

**Threat**
If an attacker can observe reports they will learn what senders are known to the TRAP network already. This knowledge could be used to track down nodes that already have a good trust. In turn if those nodes where possible to take control of an attacker could abuse the networks trust in that node.

**Mitigation**
Either TRAP should not leak information or ensure that that information is not detrimental to the functionality of TRAP. This could be done by deploying encryption.

**Manifestation**
If there is no encryption in place information will be able to be disclosed by sheer luck of being in the right place. If information should
not be disclosed to an attacker there has to be a working encryption of data as well as making sure that the algorithm is strong. It is important that the encryption process is not easily ignored by the use of side channel attacks.

**Denial of Service**

**Threat**

When an attacker is able to stop the traffic of reports they could make it impossible for a certain node to report their experience, however for this to be efficient in the entire system it would be necessary to stop reports from a multitude of reporters. It is very hard for an attacker to do this.

**Mitigation**

TRAP tries to work around Denial of Service by being dynamic and distributed. It should be able to recover from attacks of this nature.

**Manifestation**

Can be realised through being able to tamper with the data flow. Even if it is impossible to falsify messages it could be possible to change them in such a way that the intended information can not get to the receiver. It would also be possible to realise a DoS by incapacitating endpoints in some way, making sure the message either is not sent or is not received. An attacker could attempt to send a response before the real response is sent. This can work if there is no sender authentication or if there is some weakness in the authentication. It should be a fairly simple process to corrupt a message if it passes a malicious node. However if encryption is utilized it would be impossible to know what messages are about what senders. It might also be possible to incapacitate the channel by consuming essential resources for the network such as returning huge amounts of fake responses. However this should be mitigated quite well by the fact that TRAP is decentralized, there is no single point of failure.

**A.2.7 Reporter to Holder**

See A.2.6
A.2.8 TRAP to DHT

Tampering

Threat
By changing data in transit you could redirect any attempts to call other nodes to a node of your choice. For example all calls to holders go to the attacker or redirect all traffic to one node in order to cause a denial of service. This will of course need the traffic to always pass over the same node.

Mitigation
The mitigations for spoofing of certain nodes should make this attack fruitless as long as the signing metrics does not get compromised say by rerouting a key exchange and mounting a man in the middle attack.

Manifestation
If an attacker can get into the channel sending control messages it might be possible to modify these to the attackers benefit. It might also be possible to violate the integrity of the messages by cracking the crypto used to encrypt them. It is unlikely but possible depending on protection measures.

Information Disclosure

Threat
Once again, by observing the traffic passing through a node a user could gain information about the network.

Mitigation
TRAP must be aware that information about routing is not safe and should not rely on its secrecy.

Manifestation
Aside from the issues with tampering side channel attacks might be able to gather information on what nodes forward messages where.

Denial of Service
STRIDE Analysis

Threat
If it were possible to aim a denial of service at the communications with the DHT in TRAP it would be impossible to send any data in the network. This would lead to the whole system failing. Alternatively it might be possible to target certain specific messages.

Mitigation
It should be virtually impossible to stop all communication within a distributed system.

Manifestation
By targeting DHT control messages and refusing to send these or any other messages it would be impossible for Pastry to know how to route messages. It might also be possible to saturate the pipeline by sending lots of DHT messages.

A.3 Data Stores
A data store in the DFD is persistent data storage.

A.3.1 Holders

Tampering

Threat
By tampering with a Holder an attacker can modify trust values.

Mitigation
It should be hard for a sender to choose its own holder/holders. By storing trust values on multiple holders and using all of them to calculate trust it is highly unlikely that compromises of a single node will be an issue.

Manifestation
Since any node in the system can in essence be a holder it will be very hard to define a set of possible attacks that could lead to tampering. TRAP cannot impose defences against all possible tampering threats. However this would only become an issue if several of the holders
assigned to a source or even all are compromised. Although some may have low security others may have very high.

**Information Disclosure**

**Threat**

It seems impossible to stop the spreading of information since any node could send false requests asking for trust values. However by accessing the information directly from the holder you might get a better mapping of the network around the holder.

**Mitigation**

It should be hard to observe the traffic around a holder. Furthermore, the holder should not reveal information unless there is a valid, provable mail.

**Manifestation**

If an attacker can bypass the protection scheme of a single holder and data is stored unencrypted or otherwise easily intelligible they could read out trust values. Another venue of attack is trying to employ side channels. Depending on how storage is managed it could be possible to read backups instead of original data. There may also be remnants of data before and after the data is used.

**Denial of Service**

**Threat**

By setting a denial of service attack on a holder an attacker could change how the final trust is calculated. In order to seriously impact the network it would be necessary to aim the DoS at several holders. It might in theory be possible to DoS all holders containing trust for a certain node at once forcing new Holders to be assigned, if these holders could be influenced to give good trust to some sender an attacker might be able to gain control of the network.

**Mitigation**

Pastry should be able to fix nodes that drop from DoS just as it can fix normal failure. The diversity of the nodes assigned as holders
should make it very hard to bring down the set of holders in a short amount of time giving the network time to repair itself.

**Manifestation**

TRAP does not monitor data in storage. There is no way for TRAP to know that data has not been tampered with. By destroying data all trust in the node will be void. This problem is in a way mitigated by the decentralized structure of the system however. In theory it could be possible to fill the data base with redundant trust values but in practice it would be very hard to choose enough IPs that with a limited amount of hash passes would evaluate to a specific holder. Even if a DoS would succeed there are still more holders so it is a futile attack if it only targets one node.

### A.3.2 DHT

Although the Pastry DHT is a distributed data store it still contains a lot of information that could be abused in the wrong hands. Since all nodes contain part of the DHT it must be resilient to attacks. No single node should be able to compromise the system.

**Tampering**

**Threat**

It would be very simple for any node in the network to refuse to forward a routing request or reroute it to another node. If a node could fail to forward messages or send them to other nodes than wanted and not have it be known to the network there might be possibilities of attack.

**Mitigation**

If TRAP can ensure that all messages received are handled by a verified user so that any sub rings will not be able to validate messages then sub ring routing attacks become useless.

**Manifestation**

An attacker might be able to insert nodes into the network that reroute calls into sub rings controlled by the attacker. To make the
attack efficient in some cases you might need a lot of compromised
nodes, it is hard to know how feasible this is. Furthermore most
attack venues presented but the work flow between TRAP and the
DHT apply here.

Information Disclosure

Threat
The information stored in a DHT should not be secret. Since there
is no way to ensure that all nodes play fair the full knowledge of the
contents of a DHT should give no advantage to an attacker.

Mitigation
Place no inherent trust in holders; no input from the DHT should
be inherently trusted. Also having full knowledge of what node is in
which nodes table should not give any benefits to an attacker.

Manifestation
If an attacker could gather every piece of information in the DHT
they would have total knowledge of the network. However the actual
attacks can pretty much only target a small part of a DHT at a time.

Denial of Service

Threat
Dropping all messages or some messages could quite efficiently create
a DoS on parts of the system.

Mitigation
Since the network is built to survive nodes joining and leaving it
seems difficult to do any lasting harm with this attack.

Manifestation
A node can attempt to drop messages, or many nodes could. It might
also be possible for an attacker to create large amounts of routing
messages to DoS select nodes.
A.4 Processes

A.4.1 Receiver

**Spoofing**

**Threat**
If an attacker can pose as a receiver they are able to generate fake traffic in the form of requests and reports. They can claim that they have received mail that does not exist or they could collude with a partner that actually sends the mail.

**Mitigation**
TRAP would benefit from a mechanism to protect against fake reports. The only protection that can be ensured is that the original experience of the report actually has happened. What happens before that it is impossible to verify since it is outside the bounds of TRAP.

**Manifestation**
Since there is no authentication needed to join the network any node can spoof a receiver. By sending fake emails a receiver could generate a lot of reports quickly. Additionally, if it is possible for a receiver to reliably spoof reports they could potentially lower or increase trust of sources. A receiver could also create requests if they want to learn what node holds trust for a certain address.

**Tampering**

**Threat**
If a receiver could be tampered with it would enable the bypassing of TRAP. This might be done by installing some form of malware on the receiver for example.

**Mitigation**
Since the receiver is the first contact with TRAP they have to be very resilient to attack and tampering. The protocol itself can not in any way ensure that tampering can not happen. In many ways the receivers are the first line of defence, they should be rigorously checked.
for vulnerabilities. Receivers should implement input validation and other checks to ensure that there are no obvious holes.

**Manifestation**
A common attack comes from putting a process into a corrupt state. This can likely be done by sending input to the receiver.

**Repudiation**

**Threat**
A receiver could potentially fake not having received responses on their messages and force Pastry to reroute messages. They could potentially claim to have sent messages they have not sent as well. By doing this they can attempt to discredit other nodes. It is also important to not implement protocol calls that could create huge amounts of traffic with a small amount of work from the attacker such as creating many messages by sending few.

**Mitigation**
The same mechanism that provides signing can also provide verification of identity. The effects of claiming to not have received messages as well as claiming to have sent messages that they have not should only lead to standard checks. TRAP must verify that a node really has failed. False reports of failure should be reprimanded. TRAP should not send enough messages to DoS the target node.

**Manifestation**
If the system has weak signatures and/or easily broken replay countermeasures it might be possible to repudiate an event. If one node keeps claiming that another node is not responding it might be possible to overload that target node with pings from its neighbors trying to check if it exists, coupled with additional messages sent to the node it might be able to DoS the node.

**Information Disclosure**

**Threat**
If an attacker can trick the receiver into revealing information they
could get access to sender information as well as some trust values. It could also learn what reports are sent and what requests are made in the system.

**Mitigation**
By storing values of importance only in memory the risk of disclosure is greatly reduced. A requester need not store any values. It could also be possible to encrypt traffic coming in and out from the node to ensure a higher level of secrecy. If this is a good idea remains to be seen.

**Manifestation**
Tampering threats against the node could just as easily lead to information disclosure. There is also the possibility of side channel attacks.

**Denial of Service**

**Threat**
It would certainly be possible to leverage a DoS attack on a receiver; however it is unclear what good this would do for an attacker. Generally if the receiver is down spam will not be able to enter the system. The main difference between receivers is likely how zealous they are with their filtering which might lead to different receivers lowering or raising trust at different paces. If it is possible to DoS the service without touching the actual MTA however it might become an issue.

**Mitigation**
Employ safe default behaviour. If a DoS happens against the receiver drop incoming mail rather than skipping checks since it might be an attempt to bypass the filtering. If it is possible to filter but not to check trust, filter with default values. This may lead to degraded service for trusted sources as such it may be possible to claim that it is better to allow a few spam pass the filter since an abuse of this mechanic could create a DoS.

**Manifestation**
An obvious way to attempt DoS is if the resources to receive mail
are larger than the resources to check them. If the queue is filling up but the filtering and trust checks are taking longer eventually either mail would need to be dropped or checks would need to be skipped. A general DoS on fundamental resources can be done but seems unfruitful.

**Elevation of Privilege**

**Threat**
The nodes in the network do not have any special level of privilege. But if it is possible to by some method gain rights to the node in question it might be possible to override safety procedures put in place. If the attacker is able to artificially raise the trust the network has in a node this could also be abused.

**Mitigation**
TRAP has to protect storage that contains trust in nodes. It is also important that all input to the process is validated.

**Manifestation**
Tampering with the data store that handles trust could lead to unintended trust in certain nodes.

### A.4.2 Requester

**Spoofing**

**Threat**
An impostor posing as a requester could send fake requests as well as fake responses to a receiver.

**Mitigation**
There needs to be a system in place to prevent fake responses. There has to be some cryptographic solution to ensure that a response is a real response from the holder and not made up by the some other party, this should be possible to do via signatures and timestamps. No one apart from the intended recipient should be able to answer a request.
**Manifestation**

There is no authentication of requesters. This single fact makes it easy to spoof one. However if the sender has to sign requests it will not be possible to spoof a specific requester.

**Tampering**

**Threat**

By tampering with an existing requester that gets requests from a receiver that the attacker is interested in they may be able to divert the traffic and remain unchecked if there is no way to know that trust is gathered are from a certain source. The same issues apply as if you would spoof a requester. The big difference is initial trust in the system.

**Mitigation**

Do input validation on responses, error reports and mail to ensure that no malicious code can be run. Work in protected memory.

**Manifestation**

Requesters are vulnerable to malicious input from either a sender or holder. If the requester and returner or requester and receiver are implemented on the same system it might be possible for one of the sister processes to get access to memory from the requester and tamper with it.

**Repudiation**

**Threat**

If a requester can reliably lie about not having sent requests or not having received responses they could cause other parts of the network to act, such as resending messages or probing for failure. This reaction could potentially cause denial of service through forcing a node out of the network or by sending more probes than it can handle. If one node on a path does not forward messages to the endpoint TRAP might need to reroute messages, this could lead to extensive forced traffic on the network.
Mitigation
Enforce identity verification and authenticity checks on messages to ensure that the requester does not send lies about what it has and has not done. Also force signing of requests and checking of signatures on the receiving end.

Manifestation
By sending requests on behalf of another node they could cause a lot of traffic towards said node. This should be handled by the spoofed node. The receiver could also claim to have not received responses.

Information Disclosure

Threat
If an attacker can trick the requester into revealing information they could get access to receiver information as well as some trust values.

Mitigation
By storing values of importance only in memory the risk of disclosure is greatly reduced. A reporter need not store any values. It could also be possible to encrypt traffic coming in and out from the node to ensure a higher level of secrecy. If this is a good idea remains to be known.

Manifestation
Side channel attacks might be possible and should be addressed. By claiming to be a receiver a requester could be asked to fetch information about nodes. This is intended and the information should be totally safe. If input validation fails an attacker could in theory get access to information about what requests and responses are parsed by the node.

Denial of Service

Threat
By removing the possibility to do requests a holder could force a receiver to treat all messages as being at default trust value. This could also be considered an attack designed to either elevate privileges or lowering the trust of otherwise trusted sources.
Mitigation
TRAP should ensure that the default trust is not a value that can be of any benefit to the attacker. This can be done by having the highest level of checks of sources that are considered to at default trust. The receiver should be resilient to standard DoS attacks.

Manifestation
By flooding the requester with fake error reports and responses they might be forced to handle more traffic than expected or possible resulting in a downtime, either by filling up some queue bottle neck or by consuming computational power in the process. If input to the node is not validated the attacker could potentially execute some malicious code causing failure.

Elevation of Privilege

Threat
The nodes in the network do not have any special level of privilege. But if it is possible to by some method gain rights to the node in question it might be possible to override safety procedures put in place. If the attacker is able to artificially raise the trust the network has in a node this could also be abused.

Mitigation
TRAP has to protect storage that contains trust in nodes. It is also important that all input to the process is validated.

Manifestation
If input is not properly validated an attacker could potentially get unintended rights on the requester.

A.4.3 Reporter

Spoofing

Threat
An impostor posing as a reporter could generate false reports.
Mitigation

By validating reports spoofing should not be possible. This would force an attacker to have a real experience to create a report which in turn means that it will be very hard to slander other sources.

Manifestation

An attacker could attempt to create fake reports of experiences but if TRAP enforces signatures these would quickly be discarded. If there is a weakness in any of the signing protocols or hashing algorithms it might be possible to still claim being a certain node. If the keys are not stored in a secure location and transferred securely they could be used by an attacker gaining access to them. There is no real authentication system so in theory anyone could impersonate any reporter.

Tampering

Threat

By tampering with an existing reporter an attacker could do pretty much anything a spoofed reporter could but with a different level of trust.

Mitigation

Tampering is only possible if there are weaknesses in the software, by applying checks and testing on input and memory handling it should be possible to rule out the most obvious problems.

Manifestation

Reporters can receive input from a receiver, this input should be validated. If a reporter accepts other input such as errors these too have to be checked.

Repudiation

Threat

If an attacker is able to send reports and claim that they have not sent them we might have a problem. This is an issue that is prevalent through the entire message passing system.
Mitigation
This is countered by message validation/signing which acts as verification and non-repudiation, messages should be time stamped. Experience reports need to be provable.

Manifestation
An attacker could attempt to send the same report several times in order to faster gain trust, or lower trust. This is in part fixed by timestamps where a message can not be processed twice. By sending the same report several times with different timestamps this might be bypassed. However with decent experience report verification this should be much harder.

Information Disclosure

Threat
A reporter could potentially divulge information about how often and how strictly reports are created by the receiver. By observing the traffic when sending reports to a reporter it might be possible to learn what nodes are trust holders for the original sender.

Mitigation
By not storing information in the reporter the only information disclosed is the traffic going to and from the node.

Manifestation
Side channel attacks might be possible and should be addressed. If input validation fails an attacker could in theory get access to information about what reports and experiences are parsed by the node. Since reports are stored in holders it is possible to attack them instead as well.

Denial of Service

Threat
By taking a reporter offline an attacker could make sure that bad reports are never sent.
Mitigation
Since the TRAP network is distributed it should be able to replace failing reporters without much problem.

Manifestation
This can manifest either by another process that causes a DoS (such as the receiver), by consumption of essential resources or by malicious input. A reporter needs to accept experiences from a receiver and if these come in large amounts it might be unable to process them all before having to drop them. If the input is not properly validated any amount of bad things could happen.

**Elevation of Privilege**

Threat
The nodes in the network do not have any special level of privilege. But if it is possible to by some method gain rights to the node in question it might be possible to override safety procedures put in place. If the attacker is able to artificially raise the trust the network has in a node this could also be abused.

Mitigation
TRAP has to protect storage that contains trust in nodes. It is also important that all input to the process is validated.

Manifestation
If input is not properly validated an attacker could potentially get unintended rights on the requester.
Appendix B

Rainbow Attack on IPv4 based ID

In TRAP nodes are assigned an ID based on their IP. The IP is hashed with SHA256 leading to a unique ID. [6]
Since hashes are required to always evaluate to the same value given the same input it is possible to calculate all hashes in a system if the number of possible inputs is not too big. This leads to SHA256 not being secure against rainbow table attacks on the ID of TRAP nodes. Naturally this is not the fault of SHA256 but is rather an effect of IDs needing to be unique and never change no matter how many times a node joins a network.
Given a search space of $2^{32}$ and the not far fetched possibility of generating 50000 hashes per second, one would need roughly $\frac{2^{32}}{50000 \times 3600} \approx 23$ hours to generate all possible hashes in the system and it would take somewhere in the order of 70GB of storage.
This can not be considered overly hard and leads to any attacker being able to have full knowledge of what ID corresponds to which IP. However by upgrading the network to IPv6 the situation is changed greatly. The same numbers in IPv6 is that you would need $5444517870735015415$ Yottabyte of storage and $215805661416120283779410$ millennia for the calculation, neither of which is likely to happen anytime soon.
This leads us to think that TRAP should preferably be implemented on top of IPv6 if possible and if it is not there should be some safeguard to ensure that it is hard to calculate the IDs, such as salts, however this in itself leads to a slew of new problems that we do not intend to go into further. The only thing that is important to remember is that by allowing salting of IDs we effectively make it possible to generate different IDs for the same IP so it is very dangerous grounds to walk on, albeit interesting.
Glossary

**ARPANET** A military owned network, often considered the original internet. 1, 2

**DEC** Digital Equipment Corporation. 2

**DFD** Data Flow Diagram. 11, 12

**DHT** Distributed Hash Table. ix, 44, 45, 47, 48

**DNS** Domain Name System. 1

**DoS** Denial of Service. 6, 37, 39, 41, 43, 46–48, 50–52, 55, 58

**IP** Internet Protocol. 10, 32, 34, 38, 41, 47, 59, 60

**MITM** Man in the Middle. 36–38

**MTA** Mail Transfer Agent. 1

**MUA** Mail User Agent. 1

**P2P** Peer To Peer. 14

**SMTP** Simple Mail Transfer Protocol. 1, 2

**STRIDE** Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service and Elevation of Privilege. 6, 11–13

**TOR** The Onion Router. 35
Bibliography


2012.


[6] Nahid Shahmehri, David Byers, and Rahul Hiran. Trap: Open de-
centralized distributed spam filtering. Technical report, Department of 
Computer and Information Science at Linkoping University, 2011.

[7] Nahid Shahmehri, Claudiu Duna, and Germano Caromni. Dynamic 
trust metrics for peer-to-peer systems. In Proceedings of the 16th In-
ternational Workshop on Database and Expert Systems Applications 
(DEXA’05), 2005.

[8] Shawn Hernan, Scott Lambert, Tomasz Ostwald, and Adam 
Shostack. Uncover security design flaws using the stride approach. 
us/magazine/cc163519.aspx.
The existence of spam email has gone from a fairly small amounts of a few hundred in the late 1970's to several billions per day in 2010. This continually growing problem is of great concern to both businesses and users alike.

One attempt to combat this problem comes with a spam filtering tool called TRAP. The primary design goal of TRAP is to enable tracking of the reputation of mail senders in a decentralized and distributed fashion. In order for the tool to be useful, it is important that it does not have any security issues that will let a spammer bypass the protocol or gain a reputation that it should not have.

As a piece of this puzzle, this thesis makes an analysis of TRAP’s protocol and design in order to find threats and vulnerabilities capable of bypassing the protocol safeguards. Based on these threats we also evaluate possible mitigations both by analysis and simulation. We have found that although the protocol was not designed with regards to certain attacks on the system itself most of the attacks can be fairly easily stopped.

The analysis shows that by adding cryptographic defenses to the protocol a lot of the threats would be mitigated. In those cases where cryptography would not suffice it is generally down to sane design choices in the implementation as well as not always trusting that a node is being truthful and following protocol.

<table>
<thead>
<tr>
<th>Nyckelord</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>spam; filter; electronic mail; trust; threat; mitigation; trap</td>
<td></td>
</tr>
</tbody>
</table>
Copyright

Svenska
Detta dokument hålls tillgängligt på Internet - eller dess framtida ersättare - under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår. Tillgång till dokumentet innehåller tillstånd för var och en att läsa, ladda ner, skriva ut enskilda kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.
Upphovsmannens ideella rätt innefattar rätt att bli nämhnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.
För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

English
The publishers will keep this document online on the Internet - or its possible replacement - for a period of 25 years from the date of publication barring exceptional circumstances.
The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.
According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.
For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Gabriel Jägenstedt
Linköping, September 5, 2012