Analyzing the Pynchon Gate Protocol

Fred Wulff

March 21, 2008

1 Abstract

I analyzed the Pynchon Gate Protocol, a pseudonymous mail reply handling protocol introduced by Sassaman, Coen, and Mathewson, and in particular BitTorrent's suitability for it's purpose within that protocol. After an unfulfilling attempt at using PRISM, I wrote a BitTorrent simulator and used that to verify the feasibility of an attack delaying the distribution of the mail and possibly removing anonymity. I also detail an attack I found during the process of codifying the protocol for PRISM centered on spamming.

2 The Protocol

The basic aim of the Pynchon Gate Protocol(introduced in Sassaman, et al.[1]) is to provide distribution of mail sent to pseudonymous addresses with information-theoretic protection of the identity of the recipients and relatively small bandwidth costs. My aim in studying it is to find issues with the protocol using computational and advise how these issues may be fixed or worked around.

The protocol itself proceeds in rounds (the paper suggests that each round might take 24 hours, but doesn't elaborate on a rationale). Between each round, a relatively trusted pseudonym server collects email responses for the pseudonyms. At the beginning of a round, the nym server collates the received messages into a hash table, and distributes this table to a group of untrusted distribution servers via BitTorrent. A client wishing to retrieve their mail then queries the servers by sending them a bit vector corresponding to buckets in the table. The server xor's the corresponding buckets together. The client constructs its bit vectors such that all of the buckets cancel out expect the desired one. Since all of the bit vectors are needed to determine the desired bucket, the client is protected unless all n distribution servers collude.

[1]Sassaman, Len, et al., "The Pynchon Gate." WPES 2005. Accessed at http://www.abditum.com/pynchon/sassaman-wpes2005.pdf, March 13 2007

3 Methodology

I began my analysis of the Pynchon gate protocol by attempting to model the network with PRISM, but I found that (perhaps due to my limited knowledge of the PRISM model description language) I was unable to model the network to a sufficient depth to be useful - in particular, midway through the process I became interested in modelling the BitTorrent distribution aspects of the network, and was unable to find a good way to represent the flow of a large, variable number of buckets being sent around the network. This is probably my fault and could be resolved with more work on PRISM and/or more thinking about good ways to approximate it.

Anyways, I ended up doing most of my work in and on a homemade BitTorrent simulator, written in Python.

My tentative attacker model is: someone knows a pseudonym (why else would they care about identities on the server?), and would like to break the pseudonymity of the correspondent or, failing that, prevent the person from effectively using the pseudonym. The attacker has access to a number of compromised computers, but not a sufficient number to have it be likely that a client will pick only those computers to be the distributors that it draws from.

4 Results

I was able to come up with two attacks that I think have a reasonable enough chance of working to be worrisome.

4.1 The Mountain of Spam Attack

As anybody with an inbox knows, spamming is well within the reach of many malcontents. However, it poses particular risks to a client of the Pynchon Gate system. Suppose an attacker sends a large number of messages to the victim. Obviously, if a client then downloads a significantly larger than usual number of buckets, then an opportunity for traffic analysis presents itself. The Sassaman paper presents two suggestions on this front: in footnote 3, they suggest that the nym server build a summary message of messages received, and that the client then send a control message out of band to the nym server indicating priorities for messages. In footnote 5, they suggest that the bucket size be increased for larger volume recipients. However, neither of these fixes really address the problem. The latter is inflexible and wasteful in the non-DOS case (since bucket sizes must remain constant to prevent traffic analysis). The former is more promising, but has a couple of crucial flaws.

First, by requiring the response, message bodies are delayed for a cycle. This means that an attacker who can elicit speedy responses (e.g. by engaging in an ongoing flame-war) can determine bucket sizes by modulating the amount of spam and looking for a delay in responses. Since the spam can be sent surreptiously, the experiment can be repeated. Since bucket size is a measure that's apparent to the untrusted distributors, depending on variation in bucket size, this may noticeably diminish the number of possible correspondents.

Second, this requires direct communication with the nym server. Although the authors don't mention it, one of the strengths of this system is that the forward channel doesn't have to go through the nym server after the initial account creation when using an unauthenticated protocol such as SMTP. This makes it harder to attempt analysis based on the forward channel since the attacker would have to look for all possible anonymous remailers. The proposed update removes this benefit, and, worse, removes it only for a small selection of participants, possibly significantly harming anonymity.

4.2 Bittorrent (or similar) vulnerabilities

The Sassaman paper devotes only one sentence to the actual distribution from nym server to the distribution servers: "This data should be transmitted using a bandwidth-sparing protocol such as BitTorrent so that the collator does not need to send the entire pool to each distributor." Given the pedigree of the authors, it seems reasonable to assume that BitTorrent is, in fact, the intended bandwidth-sparing protocol, and the draft protocol specification explicitly includes BitTorrent. In an attempt to investigate how functional this is, I implemented a simulator of BitTorrent (simplifying assumptions are listed in comments to the code, but I believe they're all reasonable) based on the protocol specifications and analysis of the mainline client's code. The paper considers the protocol as an atomic step before distribution, but this assumption doesn't hold up to scrutiny. With the untrusted distributor model, we'd generally like to accept wide-ranging distributors, so as to prevent legal or organizational strong-arming. But what happens if a user on dial-up decides to run a distributor? In the worst case it might take more than a cycle to download a cycle's worth of data. We obviously can't wait that long. This suggests an attack if we can ensure that our peers get the data before the good peers, and maintain that exclusivity while clients query us.

So, what does it take to delay the other BitTorrent peers? Trying the simulator for 1024 blocks with a ratio of 1 bad peer to 4 good peers with the bad peers sharing information but not playing any other dirty tricks gives the bad peers the full data at about round 120, whereas the first good client does obtain the full data until around round 190. Obviously it depends on the volume of data and bandwidth involved, but since we can increase the data by simply spamming the server, it doesn't seem unreasonable that we can obtain a significant amount of time at this ratio. If we intentionally slow down transfers by a factor of 10 from the bad peers, the ratio raises to 120:300. There have also been attacks reported in which a block of bad data can be sent to a peer with no ill effect. Giving our bad peer this capability (but a normal rate) results in a ratio of 120:340. Additionally, It's relatively well-documented that a number of current BitTorrent clients, including the mainline client, are vulnerable to the same machine masquerading as hundreds of clients, resulting in disproportionate bandwidth allocation[2]. In any case, by combining these techniques, we should be able to get a fair chunk of time in which our clients are the only game in town. The buckets are still encrypted, but we've at the very least removed the information theoretic security advantage, and there are all kinds of timing and tagging tricks that might be possible given sole control over the distributors.

All in all, BitTorrent appears to be a very poor security fit for this protocol.

4.3 Possible Fixes

In order to avoid the mountain of spam attack, I suggest a modification of the correspondence with the nym server strategy. First, I propose that all communication be delayed one cycle. If necessary, the cycle length can be shortened and the bucket size made smaller so that bandwidth remains unchanged. Further, the bucket size should be fixed to the same size for all participants under the theory that any differentiation is just inviting attack. There are a couple of ways that the raw spam problem can be avoided. If the pseudonym is going to be used only for correspondence with a limited number of people, a number of one time keys can be included in the message and checked for on receipt of replies. Otherwise, perhaps encode a set of predetermined control messages (agreed upon ataccount creation, presumably before the attacker was watching the pseudonym) as part of a private signature in an email for which one expects a response. When the nym server picks up the response, it can enable one of a variety of spam fighting strategies. Presumably the responses would be idempotent and plausibly deniable as control structures, so the attacker would be unable to mount a replay attack and hopefully unaware of what said attack would consist of.

The BitTorrent attack is somewhat more troubling. The BitTorrent protocol simply doesn't seem to be very well suited to security contexts. I suggest replacing it with a simple SFTP-based tree-base protocol, with the nym server randomly picking the elements of the tree. It might not be the most bandwidth efficient protocol, but it means that Nym server is in charge of distribution, rather than possibly malicious peers. There could also be some variety of statistical enforcement of fairness over time to ensure that the same servers don't start serving each time.

5 Conclusion

The Pynchon Gate protocol appearss to need more evaluation before it's ready for production use. However, it is an interesting protocol and, I believe, is worth examining further. In particular, I would be interested in seeing an analysis of the Byzantine postman recovery procedure suggested by the followup. There are

[2]Crosby, Scott A and Dan S Wallach. "An Analysis of Two BitTorrent Distributed Trackers." South Central Information Security Symposium 2006. http://cops.csci.unt.edu/sciss/2006/23/index.html

also plenty of possibilities for replacing BitTorrent as the distribution mechanism or even just firming up BitTorrent's security.

6 Resources

- BitTorrent Specification - http://wiki.theory.org/BitTorrentSpecification
- Pynchon Gate Homepage - http://www.abditum.com/pynchon/
- Pynchon Draft Specification - http://freehaven.net/pynchon/doc/pynchon-spec.txt
- Mainline BitTorrent source - http://download.bittorrent.com/dl/BitTorrent-5.2.0.tar.gz