Short Paper: Challenges in protecting Tor hidden services from botnet abuse

Nicholas Hopper[⋆]

University of Minnesota, Minneapolis, MN USA
hopper@cs.umn.edu

1 Introduction

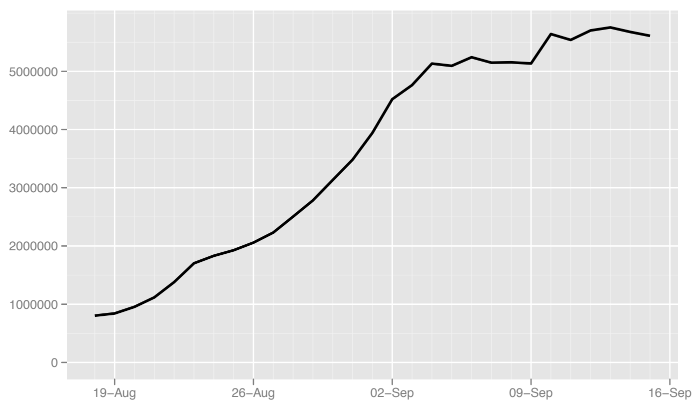
In August, 2013 the Tor anonymity network saw a rapid spike in the number of directly connecting users, due to the large "mevade" click-fraud botnet running its command and control (C&C) as a Tor Hidden Service. Figure 1(a) shows that estimated daily clients increased from under 1 million to nearly 6 million in three weeks. Figure 1(b) shows the effects on performance: measured downloading times for a 50 KiB file doubled, from 1.5 seconds to 3.0 seconds.

However, the amount of traffic being carried by the network did not change dramatically. The primary cause of the problems seems to be the increased processing load on Tor relays caused by the large increase in key exchanges required to build anonymous encrypted tunnels, or *circuits*. When a Tor client connects to the network, it sends a CREATE cell to a Tor node, called a *guard*, which contains the first message g^a in a Diffie-Hellman key exchange, called an "onion skin"; the node receiving the create cell computes the shared key g^{ab} and replies with the second message g^b, creating a 1-hop circuit. After this, the client iteratively sends onion skins in EXTEND cells to the end of the circuit, which extracts the onion skins and sends them in CREATE cells to the next relay, until all three hops have exchanged keys.

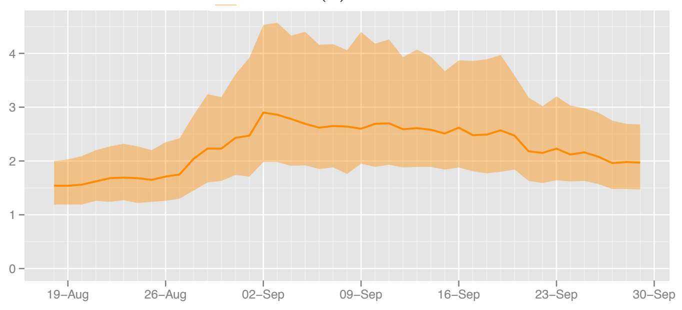
– Extending a circuit – decrypting an "onion skin" and participating in a Diffie-Hellman key exchange – is sufficiently compute expensive that busy relays can become CPU-bound.
– The hidden service protocol – explained in section 2 – causes at least three circuits to be built every time a bot connects.
– When onion skins exceed the processing capacity of a relay, they wait in decryption queues, causing circuit building latencies to increase.
– Queued onion skins eventually time out either at the relay or the client, causing the entire partial circuit to fail, causing more onion skins to be injected to the network.

In response to this, the Tor Project released a modified version (0.2.4.17-rc) that prioritizes processing of onionskins using the more efficient ntor [8] key exchange protocol. Adoption of this release has helped the situation: as Figure 1 shows, measured download 50 KiB times as of late September decreased

[⋆] Work done while on sabbatical with the Tor Project



(a)



(b)

Fig. 1. (a) Estimated daily Tor users, and (b) Measured 50 KiB download times, in seconds, 18 August to 13 September 2013

to roughly 2.0 seconds. Figure 2 shows that failed circuit extensions using tor version 0.2.4.17-rc range between 5% and 15%, while circuit extensions using the stable release, version 0.2.3.25, range between 5% and 30%.

In this paper, we consider long-term strategies to ease the load on the network and reduce the impact on clients, and describe the challenges in evaluating and deploying these schemes. We assess these strategies with the security goal of ensuring the availability of Tor under the threat of a botnet that uses hidden services as its primary C&C channel, keeping in mind that a "long-term" strategy must contend with a botnet that could be deployed in response to these strategies, where the behavior of both the botnet and the Tor software can change adaptively to circumvent mitigation mechanisms.

2 Background: Tor Hidden Services

The Tor network provides a mechanism for clients to anonymously provide services (e.g., websites) that can be accessed by other users through Tor. We briefly review the protocol for this mechanism:

1. The hidden service (HS) picks a public "*identity key*" PK_S and associated secret key SK_S. The HS then computes an "*onion identifier*" $o_S = H(PK_S)$

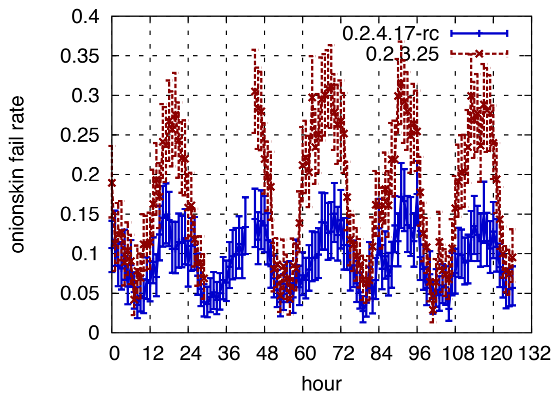


Fig. 2. Hourly measured failure rates, starting 27 Sept. 2013, of EXTEND cells

using a cryptographic hash function H. Currently, the hash function H is the output of SHA1, truncated to 80 bits. This 10-byte identifier is base32-encoded to produce a 16-byte `.onion` address that Tor users can use to connect to HS, such as `3g2upl4pq6kufc4m.onion`.

2. The HS constructs circuits terminating in at least three different relays, and requests these relays to act as its *introduction points* (IPs).
3. The HS then produces a "*descriptor*," signed using the SK_S, that lists PK_S and its IPs. This descriptor is published through a distributed hash ring of Tor relays, using o_S and a time period τ as an index.
4. A client connects to the HS by retrieving the descriptor using o_S and τ, and building two circuits: one circuit terminates at an IP and the other terminates at a randomly-selected relay referred to as the *rendezvous point* (RP). The client asks the IP to send the identity of the RP to the HS.
5. The HS then builds a circuit to the RP, which connects the client and HS.

Since lookups to the distributed hash ring are performed through circuits as well, and each descriptor has three redundant copies, a client connecting to a hidden service could require building up to 6 circuits; to reduce this load, clients cache descriptors and reuse rendezvous circuits any time a request is made less than ten minutes after the previous connection.

3 Can we throttle by cost?

Since the primary concern from the point of view of the other users of Tor is the *rate* at which botnet nodes consume the collective computing resources of the relays, one set of potential solutions is to attempt to throttle or otherwise limit the rate of requests from the botnet. Two key points to recall in evaluating solutions from this class are that (i) in many ways the botnet has more resources available than the set of all regular Tor clients and (ii) neither bots nor the C&C server are constrained to follow the standard Tor algorithms, although the current implementations may do so.

One way to control the rate at which circuit building requests enter the network is by making it costly to send them. Tor could do this by requiring proof of the expenditure of a scarce resource, for example, human attention, processor time, bitcoins, and so on. If the cost to build a circuit or connect to a hidden service can be correctly allocated it could be the case that ordinary users and services can easily afford the cost while the price for a botnet becomes prohibitive. Depending on the resource used, correctly allocating the cost is an important research question; we consider the problem for Proof of Work (CPU-based) and CAPTCHA (human attention-based) systems below.

Besides the cost allocation problem, another technical challenge is ensuring that resources can't be double-spent, so that each resource expenditure in a given time period only authorizes a single circuit or hidden service connection. Several approaches exist, but each would require further investigation:

– Make the unit of pay cover a single circuit extension and have one of the relays extending the circuit issue a challenge back to the client, which then must be answered before the CREATE (or EXTEND) cell is processed, similar to the scheme described by Barbera *et al.* [3]. This has the unfortunate side effect of adding an extra round-trip time to every circuit-building request. Finding a way to hide this extra round-trip time could make it a viable alternative, for some resources.
– Relay descriptors could include "puzzle specifications" that describe what the challenge will be for a given time period, requiring a method to prevent "precomputing" a batch of payments before the time period; how to solve this problem is an open question.
– Another method would use an extra trusted server that verifies resource expenditures and issues relay- and time period-specific signed tokens, similar to *ripcoins* [12] or the tokens in *BRAIDS* [10]. Using blinded tokens would limit the trust required in the server so that it can't compromise anonymity, and relay-specificity would allow each relay to verify that tokens aren't double-spent. However, this adds an extra signature-verification to the task of onion-skin processing and another server and key that must be maintained.

Proof of work (proves once more not to work?) When the resource in question is processor time and challenges are, e.g. hashcash [2] targets, the cost allocation strategy should dictate that the hidden service must pay a cost for each connection, since bots clients and normal hidden service clients will have essentially identical use profiles (from the point of view of relays) and computational resources. On the other hand, the C&C hidden server(s) will collectively initiate many more circuits than any single "normal" hidden server.

The key security challenge when considering an adaptive botmaster's response to this approach is the "chain-proving" attack (by analogy to chain voting [11]). In this attack, the C&C server solves the first challenge it receives when a bot contacts the hidden service, but then on each additional challenge, the *previous bot* is asked to solve the puzzle in time to allow the *next bot* to connect. In principle the difference in latencies (caused by the need to pass a puzzle to the bot through Tor) could potentially be detected, but an adaptive botmaster could well build shorter circuits, and employ multiple bots in an effort to reduce the time needed to solve a "proof of work" puzzle.

CAPTCHAs If CAPTCHAs are used to verify expenditure of human attention, the relative cost allocation should change to favor the client: clients of most hidden services will have human users, while hidden servers will not. This raises additional technical problems, such as how CAPTCHAs can be served through Tor without a GUI interface, how a user's solution can be transferred to the hidden service without violating privacy or allowing overspending, and how to deal with the needs of completely headless services where neither the HS client nor the HS server have a user's attention to give.

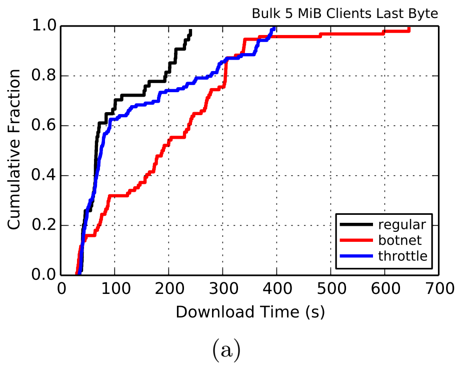
Another technical challenge to deploying CAPTCHAs is how to defeat computationally expensive automated solvers. Most commercially-deployed CAPTCHAs can be solved with success rates on the order of 10% per challenge [7, 1], and the typical service mitigates this by temporarily blacklisting an IP address after a small number of attempts. With anonymous users, this becomes a more challenging problem to solve; without blacklisting a bot can simply attempt as many CAPTCHAs as necessary to obtain an automated solution.

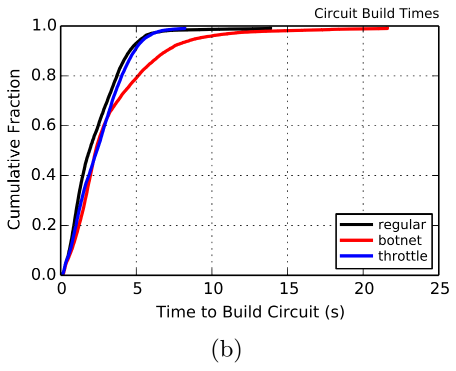
4 Can we throttle at the entry guard?

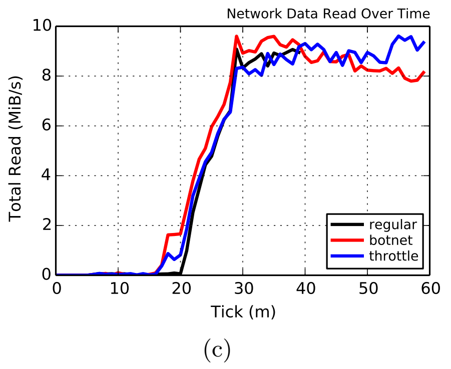
A more direct approach would be to simply have guard nodes rate-limit the number of EXTEND cells they will process from a given client. If the entry guard won't process the EXTEND cell needed to build a circuit, the hidden server can't flood the network with onion-skins. Notice that this measure won't prevent *bots* from flooding the network with circuit requests; it simply makes the network ineffective from the botmaster's standpoint and thus, motivates botmasters to find some other C&C channel that causes less stress on the Tor network.

Effective circuit throttling at the guard node faces a number of challenges, however. Biryukov *et al* [4] found that the most popular hidden services see over 1000 requests per hour; if we assume that these hidden services won't modify Tor's default behavior, then guard nodes need to allow each client to extend over 300 circuits per hour; but since there are currently over 1200 relays acting as guards, a single C&C server run by an adaptive botmaster could build 360 000 circuits per hour at this rate. We could decrease the cap and try to make it easier for busy hidden servers to increase their guard count, but this significantly increases the chance that a hidden server chooses a compromised guard and can be deanonymized.

One possibilty would be to use *assigned guards.* In this approach, ordinary clients would pick guards as usual, and guards would enforce a low rate-limit







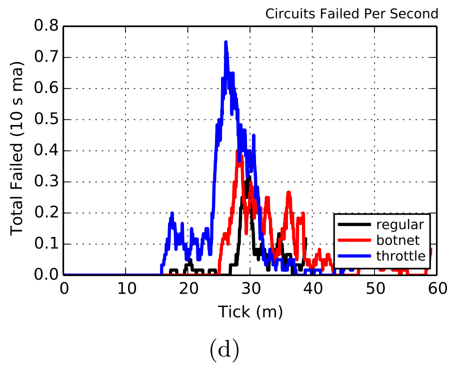


Fig. 3. Results of guard throttling: 20 relays, 200 clients, 500 bots. (a) 5MiB download times, (b) Circuit build times, (c) Total bytes read (d) circuit failures

r_{default} on circuit extensions, for example 30 circuits per hour. [1] Clients that need to build circuits at a higher rate r_{server} – say, 2000 per hour – could follow a cryptographic protocol that would result in a verifiable token that assigns a deterministic, but unpredictable, guard node for the OP when running on a given IP address. These OPs could then show this token to the assigned guard and receive a level of service sufficient for a busy hidden server, but not for a flood of circuit extensions. An example of this type of protocol appears as Protocol 3 (section 3.3) in the *BRAIDS* design by Jansen *et al.* [10]. The rates r_{default} and r_{server} could appear in the network consensus, to allow adjustments for the volume of traffic in the network. Figure 3 shows the result of simulating this strategy with $r_{\text{default}} = 10$ and $r_{\text{server}} = 2000$ using the *shadow* simulator [9]; despite nearly identical bandwidth usage, the throttled simulation has performance characteristics similar to the simulation with no botnet.

An additional technical challenge associated with guard throttling is the need to enforce the use of entry guards when building circuits. If the C&C server joins the network as a relay, CREATE cells coming from the hidden service would be

[1] Naturally, finding the right number to use for this default rate is also an interesting research challenge: a very low rate-limit could prevent bots from flooding the network but might also disrupt legitimate hidden service clients

indistinguishable from CREATE cells coming from other circuits running through the relay, effectively circumventing the rate limit. In principle this could be detected by a distributed monitoring protocol, but designing secure protocols of this type that avoid adversarial manipulation has proven to be a difficult challenge.

5 Can we reuse failed partial circuits?

Part of the problem caused by the heavy circuit-building load is that when a circuit times out, the entire circuit is destroyed. This means that for every failed CREATE, at least three new CREATE cells will be added to the network's load. If we model the entire Tor network as having probability p of having a CREATE cell timeout, then the expected number of CREATE cells needed to successfully build a circuit will be the X_0 satisfying the linear system:

$$\begin{aligned} X_0 &= pX_0 + (1-p)X_1 && +1 \\ X_1 &= pX_0 \quad + (1-p)X_2 && +1 \\ X_2 &= pX_0 && +1 \ , \end{aligned}$$

where X_i is the expected number of cells to complete a partial circuit with i hops. This gives us $X_0 = \frac{p^2-3p+3}{(1-p)^3}$.

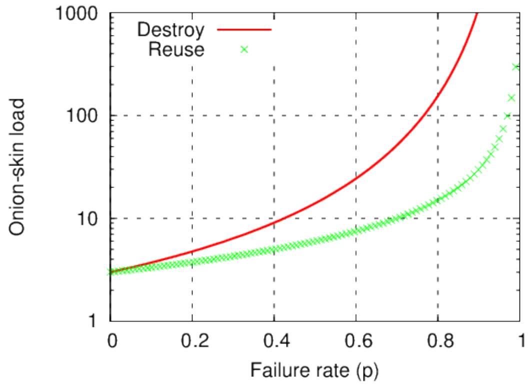


Fig. 4. Expected onion-skin load per circuit created, for failure rate p

Conceptually, we can reduce this load by re-using a partially-built circuit, e.g. when a timeout occurs, we truncate the circuit and attempt to extend from the current endpoint. In this case, the expected number of CREATE cells needed to build a circuit will be simply $X_0' = \frac{3}{1-p}$. Figure 4 shows plots of both functions. We can see that for high enough failure rates, this change causes a substantial reduction in load for the network. Figure 2 shows typical failure rates for a stable (TAP) and release candidate (ntor) roughly one month after the beginning of the botnet event; we can see that at the observed failure rates ranging from 10%-25%, reusing partial circuits would reduce the load on the network by 10-30%.

Of course, this model ignores the fact that failure probabilities are neither static nor uniform across the entire Tor network, and the fact that many nodes use "create fast" cells to exchange a first-hop key without using Diffie-Hellman key exchange. Reducing the load introduced by failures will also reduce the rate of circuit failures overall, but since CPU capacities vary widely across the Tor network (and load balancing is by the essentially uncorrelated bandwidth of nodes) the size of the actual effect due to this change is difficult to predict. Further evaluation will be needed. Additionally, this change would also somewhat increase the power of selective denial of service attacks [6], although such attacks typically only become noticeably effective in situations where we would already consider Tor to be compromised.

6 Can we isolate Hidden Service circuits?

Another approach to protect the regular users of the Tor network from resource depletion by a hidden-service botnet would be to isolate hidden service onion-skin processing from ordinary processing. By introducing a mechanism that allows relays to recognize that an EXTEND or CREATE cell is likely to carry hidden service traffic, we could provide a means to protect the rest of the system from the effects of this traffic, by scheduling priority or simple isolation.

An example of how this might work in practice is to introduce new NOHS-EXTEND/NOHS-CREATE cell types with the rule that a circuit that is created with an NOHS-CREATE cell will silently drop a normal EXTEND cell, or any of the cell types associated with hidden services. If relays also silently drop NOHS-EXTEND cells on circuits created with ordinary CREATE cells, then NOHS-CREATE circuits are guaranteed not to carry hidden service traffic. Updated clients would then create all circuits with NOHS-CREATE unless connecting to a hidden service. When a sufficient number of clients and relays update their Tor version, a consensus flag could be used to signal relays to begin isolating processing of ordinary CREATE cells. For example, these cells might only be processed in the last 20ms of each 100ms period, leaving 80% of processing capacity available for regular traffic. The flag could be triggered when hidden service circuits exceed a significant fraction of all circuits in the network.[2]

This solution protects the network and typical users from a massive botnet hidden service, but would, unfortunately, intensify the effect on users of legitimate hidden services in time periods when an attack was detected. As with guard throttling, the intended effect would thus be to encourage botmasters to develop C&C channels that do not stress the Tor hidden service ecosystem, while providing stronger protection against botnet clients flooding the network.

One privacy concern related to this approach is that as the network upgrades to versions of Tor supporting NOHS-CREATE, identification of hidden-service traffic approaches deterministic certainty. By contrast, current hidden service circuits follow traffic patterns that allow them to be identified with high statistical

[2] Detecting this condition in a privacy-preserving manner represents another technical challenge requiring further research.

confidence [5] only. Because (excluding botnet traffic) the base rates of hidden service traffic compared to all other traffic are low, this will also decrease the privacy of hidden service users. One potential mitigation mechanism would be to have clients only use NOHS-CREATE when the consensus flag for hidden service isolation is activated, which would indicate that hidden service clients would already have a large anonymity set.

7 Conclusion

Although this document has described several possibilities that either limit the attractiveness of Tor Hidden Services as a mechanism for C&C communication or limit the impact of these services on other users of Tor, all of the approaches present research challenges for the security community in some way. We hope that this short paper will encourage new research in this direction.

8 Acknowledgements

Thanks to Mike Perry, Ian Goldberg, and Roger Dingledine for helpful comments about the problems discussed in this paper. This work was supported by the U.S. National Science Foundation under grants 1111734 and 1314637 and DARPA.

References

1. Ahmad, A.S.E., Yan, J., Tayara, M.: The robustness of google CAPTCHAs. Technical Report Computing Science Technical Report CS-TR-1278, Newcastle University (2011)
2. Back, A., et al.: Hashcash-a denial of service counter-measure (2002)
3. Barbera, M.V., Kemerlis, V.P., Pappas, V., Keromytis, A.: CellFlood: Attacking Tor onion routers on the cheap. In: Proceedings of ESORICS 2013. (September 2013)
4. Biryukov, A., Pustogarov, I., Weinmann, R.P.: Content and popularity analysis of tor hidden services. arXiv [cs.CR] (August 2013)
5. Biryukov, A., Pustogarov, I., Weinmann, R.P.: Trawling for tor hidden services: Detection, measurement, deanonymization. In: Proceedings of the 2013 IEEE Symposium on Security and Privacy. (May 2013)
6. Borisov, N., Danezis, G., Mittal, P., Tabriz, P.: Denial of service or denial of security? How attacks on reliability can compromise anonymity. In: Proceedings of CCS 2007. (October 2007)
7. Bursztein, E., Martin, M., Mitchell, J.: Text-based CAPTCHA strengths and weaknesses. In: Proceedings of the 18th ACM Conference on Computer and Communications Security. CCS '11, New York, NY, USA, ACM (2011) 125–138
8. Goldberg, I., Stebila, D., Ustaoglu, B.: Anonymity and one-way authentication in key exchange protocols. Designs, Codes and Cryptography **67**(2) (May 2013) 245–269
9. Jansen, R., Hopper, N.: Shadow: Running Tor in a Box for Accurate and Efficient Experimentation. In: Proceedings of the Network and Distributed System Security Symposium - NDSS'12, Internet Society (February 2012)

10. Jansen, R., Hopper, N., Kim, Y.: Recruiting new Tor relays with BRAIDS. In Keromytis, A.D., Shmatikov, V., eds.: Proceedings of the 2010 ACM Conference on Computer and Communications Security (CCS 2010), ACM (October 2010)
11. Jones, D.W.: Chain voting. In: Workshop on Developing an Analysis of Threats to Voting Systems, National Institute of Standards and Technology. (2005)
12. Reiter, M.K., Wang, X., Wright, M.: Building reliable mix networks with fair exchange. In: Applied Cryptography and Network Security, Springer (2005) 378–392