Compromising Anonymity Using Packet Spinning

Vasilis Pappas, Elias Athanasopoulos, Sotiris Ioannidis, and Evangelos P. Markatos

Institute of Computer Science (ICS)
Foundation for Research & Technology Hellas (FORTH)
{vpappas,elathan,sotiris,markatos}@ics.forth.gr

Abstract. We present a novel attack targeting anonymizing systems. The attack involves placing a malicious relay node inside an anonymizing system and keeping legitimate nodes *"busy."* We achieve this by creating circular circuits and injecting fraudulent packets, crafted in a way that will make them spin an arbitrary number of times inside our artificial loops. At the same time we inject a small number of malicious nodes that we control into the anonymizing system. By keeping a significant part of the anonymizing system busy spinning useless packets, we increase the probability of having our nodes selected in the creation of legitimate circuits, since we have more free capacity to route requests than the legitimate nodes. This technique may lead to the compromise of the anonymity of people using the system.
To evaluate our novel attack, we used a real-world anonymizing system, TOR. We show that an anonymizing system that is composed of a series of relay nodes which perform cryptographic operations is vulnerable to our packet spinning attack. Our evaluation focuses on determining the cost we can introduce to the legitimate nodes by injecting the fraudulent packets, and the time required for a malicious client to create *n-length* TOR circuits. Furthermore we prove that routers that are involved in packet spinning do not have the capacity to process requests for the creation of new circuits and thus users are forced to select our malicious nodes for routing their data streams.

1 Introduction

Anonymizing systems have been steadily becoming popular as network users that want to hide their identity are using them when accessing Internet services, like Web browsing and Instant Messaging. Anonymity in networks dates back to more than twenty years, when Chaum [7] introduced the concept of anonymous communications. Over the last ten years there have been a series of proposals for anonymizing systems for numerous services. We refer the reader to [18, 12, 19, 22, 8, 16, 20, 10] for some of the most popular anonymizing system proposals.

Following user needs, anonymizing systems have moved from being purely academic proposals and have been deployed as real-world infrastructures. One

of the most popular existing solutions for using the Internet in an anonymous fashion is the TOR system [11]. TOR has been specifically designed for providing anonymity for low-latency services such as the World Wide Web.

TOR is composed of a collection of routing nodes that are available for circuit creations between entities that want to communication in an anonymous fashion. For example, if Alice wants to surf on Bob's web site using TOR, she will have to pick three or more available TOR routers, create a circuit that end's at Bob's web site, and proceed to tunnel her requests over that circuit. Bob on his end will never come in direct contact with Alice but only with a TOR router. This way Alice's anonymity is preserved.

Theoretically, it is possible for an attacker to place malicious nodes inside an anonymity network that is circuit-based (like TOR) and manage to compromise Alice's anonymity, should those malicious nodes are selected in Alice's circuits. However, when such systems contain thousands of routers, the probability of being selected is relatively low, unless the attacker injects large numbers of malicious nodes. In this paper we present a novel attack in which a malicious user injects just a few nodes in system like the above in order to keep legitimate routers busy. At a later stage, the attacker can add malicious nodes that will be relatively idle, and in this way increasing the probability of having them selected.

The rest of this paper is organized as follows. We discuss and compare to prior work in Section 2. In Section 3 we give a detailed presentation of the basic concept of the packet spinning attack. The evaluations of the attack and its magnitude is presents in Section 4. Based on our experimental findings we show how an attacker can actually compromise anonymity in TOR in Section 5. In Section 6 we propose *Tree Based Circuits*, a countermeasure aimed at defeating packet spinning attacks. Finally we conclude and discuss future work in Section 7.

2 Related Work

There are numerous research papers identifying possible attacks against modern anonymizing systems. One fundamental attack against anonymizing systems is based on *traffic analysis* (see, e.g., [17, 3]). Traffic analysis, in the context of anonymizing, is the process of passively monitoring streams of an anonymizing system and trying to correlate them by identifying specific patterns, aiming to reveal the sender or the recipient of an anonymous communication. Traffic analysis has evolved [14, 9] to a practical way of breaking the anonymity provided by an anonymizing system. For example, Danezis *et al.* have shown how traffic analysis can successfully break the anonymity provided by TOR [15].

Apart from traffic analysis, there are other possible attacks against anonymizing systems, and our work here is more closely related to those, since we don't use any traffic analysis to carry out our attack. More precisely, previous work has presented a study on attacks against anonymizing systems which are based in open MIX routers [5]. With respect to TOR, and not considering traffic analysis, there are two major attacks on its anonymizing scheme. The first one suggests to inject malicious nodes in a TOR overlay that lie about their available bandwidth

and consequently are selected for TOR circuits with higher probability [4]. In the second one, Danezis *et al.* are taking advantage of the circuit creation process to compromise TOR [6]. Specifically, when a malicious node realizes that it is unable to compromise a TOR circuit, meaning it is not an entry or an exit node in a TOR circuit, it breaks the circuit and it forces the user to initiate a new one. It uses this technique repeatedly with the hope that the new circuit will contain the malicious node as an entry or exit node.

Borisov *et al.* recently proposed an opportunistic bandwidth measurement algorithm for TOR to replace self reported values [21]. This technique addresses attacks like the one described above [4] and also has a good impact on TOR's overall performance because it achieves better load balancing. Surprisingly, this technique is beneficial for our attack. In the current implementation of TOR the ORs advertise the same bandwidth both under normal conditions and under the LOOP phase (where our spinning cells consume most of their bandwidth). Using the Borisov's technique though, after the LOOP phase our idle malicious ORs will be more likely to be chosen, because the legitimate ones will advertise less available bandwidth.

3 Packet Spinning Attack

In this section we describe in details the packet spinning attack. First, we present the basic idea and then we focus on the parameters which are critical and can make the attack stronger. Even though the attack is feasible in any anonymizing system which uses intermediate relay nodes for hiding the identity of a packet sender, and each relay node is involved in some cryptographic operations, in this analysis we will focus on the TOR anonymizing system.

Overview. The TOR anonymizing system is composed of a collection of nodes[1] that relay traffic from a user's computer to a target service. These relay nodes, which act as packet forwarders are called Onion Routers (ORs), since Onion Routing[13] is used during the routing process. A user, who wants to utilize TOR, runs a TOR client on their computer, called TOR Proxy (TP). The TP contacts the TOR Directory Servers, which list all the available ORs, and then builds TOR circuits. Typically, a TOR circuit comprises of three ORs, the entry, the middle and the exit OR, but the system does not impose any constraints in the length of a TOR circuit. As long as the user transmits data, the TOR circuit remains functional. The information transmitted by the TP is encapsulated in TOR cells. A TOR cell is considered as the base information unit of transmission via TOR and it is 512 bytes long in size.

To provide stronger anonymity guarantees, the TOR system is designed so that any OR except the last one, is unable to identify its position in the TOR

[1] At the time of writing this paper the number had reached about 2500 nodes (`http://torstatus.kgprog.com/`).

circuit[2]. On the other hand, to avoid eavesdropping, each TOR cell is routed using Onion Routing. That is, each TOR cell is multiply encrypted using symmetric cryptography. Each OR can decrypt only a single layer of the cell using its shared session key. Thus, the TP must encrypt each cell with all the shared session keys of the ORs that compose the TOR circuit.

The Packet Spinning Attack relies in two fundamental principles:

- Since the complete circuit is not known by every OR, circular circuits are not detectable.
- A legitimate OR will always spend some time in cryptographic operations.

The attack consists of two two phases, (*i*) the LOOP phase and (*ii*) the COMPROMISE phase. We continue by describing each phase in detail.

LOOP Phase. During the LOOP phase, an adversary attempts to keep a significant amount of legitimate ORs busy in spinning fraudulent packets. For an adversary to launch the attack in its simplest form, they need a malicious TP and a malicious OR. The malicious TP colludes with the malicious OR. The TP creates a TOR circuit which starts and ends at the malicious OR. The TP creates a packet, which it encrypts layer by layer using the shared symmetric session keys of the legitimate ORs composing the initiated TOR circuit, and then forwards it to the malicious OR. The malicious OR does not decrypt the packet but instead it immediately forwards it to the next legitimate OR of the circuit. The OR decrypts a layer of the packet and forwards to the next one, and so on. Finally, the packet completes a cycle and reaches the malicious OR completely decrypted. Upon receipt, the malicious OR drops the packet and re-injects the initial, fully encrypted packet, back into the circuit. This marks an artificial spin of a packet inside a TOR circuit. The same operation can be repeated indefinitely.

A schematic representation of the LOOP phase can be seen in Figure 1. To further amplify the attack, the malicious TP can build a series of loops, in various combinations, always using a single malicious OR, who is responsible for maintaining the loop.

It is vital to observe, that one malicious OR can keep multiple legitimate ORs busy. This is achievable for two reasons. First, the malicious OR spends much less time in packet routing, since it is not involved in any cryptographic operation. We evaluate this further in Section 4. Second, the malicious OR can be part of a circular circuit of arbitrary length. The default length of a TOR circuit is three ORs, but the protocol specification does not impose any constraints in building larger circuits. The ability of building TOR circuits of arbitrary length is also further explored in Section 4.

In addition, as we experimentally observe in Section 4 the difference in routing effort between the malicious OR and the legitimate ones increases as the packet size (the number of cells composing the initial packet) grows.

[2] Although the first one is also able to know its position by checking whether the IP address of the node before it is in the set of the Tor nodes. This set is available through the Directory servers. [4]

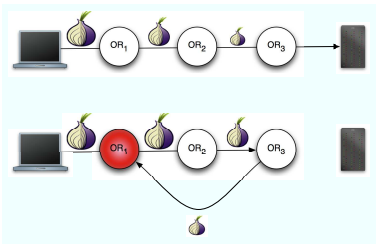

Fig. 1. Schematic representation of the packet spinning attack (LOOP phase). In the top part of the figure we depict the normal operation of a TOR circuit. Each OR decrypts a layer of the incoming packet and forwards it to the next one, until the packet reaches the final destination. In the lower part of the figure we depict the spinning packet attack. A malicious OR (solid cell), part of the circuit, injects again the initial packet in the circuit. The legitimate ORs continue to decrypt the packet in each spin, but the malicious one is not involved in any cryptographic operation.

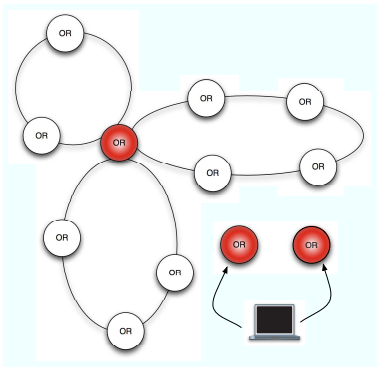

Fig. 2. Schematic representation of how an adversary can compromise the anonymity of a user by employing the packet spinning attack (COMPROMISE phase). In this figure, the adversary has injected three malicious ORs (solid cells) in the anonymizing system. One of them has built circular circuits with legitimate ORs in order to keep them busy, and two of them are free of resources in order to serve as routers for new TOR connections.

COMPROMISE Phase. When the attacker completes the LOOP phase, they are able to launch the COMPROMISE phase, in order to reveal anonymous communications. The adversary injects malicious ORs, which are not selected as part of any *"spinning circuit."* That is, the injected ORs are idle and therefore they can be selected for legitimate TOR circuits with greater probability. As we investigate further in Section 4, even if legitimate ORs, which are part of a spinning circuit, are selected in the creation of a new TOR circuit, most likely they will not be able to join it. That is, if a legitimate TP selects ORs, which are occupied in spinning circuits, there is a great probability for the TP to receive a *timeout* from the selected ORs and continue building the circuits with new ORs. Sooner, or later the legitimate TP will select idle ORs like the ones injected in the system by the adversary. We schematically present the COMPROMISE phase in Figure 2.

4 Attack Evaluation

In this section we estimate the firepower of the packet spinning attack. We experimentally evaluate the attack magnitude by placing a malicious OR in a TOR overlay on two fronts:

- The overhead for a malicious OR to forward spinning packets versus the overhead of a legitimate OR to perform the same operation.
- The time required for a malicious TOR client to build arbitrary length TOR circuits.

Routing Overhead. The spinning packet attack is based primarily on the fact that a malicious OR may route packets faster than a legitimate one, because it is not involved in any cryptographic operation. The malicious OR is positioned in a circular TOR circuit and it is continuously injecting TOR cells inside the circuit, without decrypting the cells it receives (see Figure 1).

To measure the effort spent by a malicious OR to conduct the attack, in contrast to a legitimate OR, we conducted the following experiment. We placed three ORs on three hosts. Each host was running a single OR and all the three hosts were isolated from any external network traffic. One of the ORs was modified to be malicious. The same host that runs the malicious OR is also running a modified TP to create the circular circuit, as well as another 20 TOR processes and the directory servers needed for each OR to resolve the other available ORs. That is, the host running the malicious OR was significantly more loaded than the ones running the legitimate ORs. We used the default configurations of all ORs and we used the latest version of TOR (0.1.2.18) at the time we conducted the experiments.

We conducted several experiments for various numbers of spinning cells. In each run we measured the time needed for an OR - legitimate or malicious - to route the incoming cell. The time measured was from the point that the OR received a packet, up to the point the OR had sent the packet to the next hop of the circuit. For each experiment, the TP sends to the artificially made circuit a packet of specific size in terms of number of cells. Recall from Section 3 that the base information unit in TOR is the cell, which is equal to 512 bytes. That is, the TP sends packets that are multiple of one cell in length. From the moment the TP sends spinning packet to the circuit, the packet starts spinning with the assistance of the malicious OR and the TP is never again involved in the process. In Figure 3 we present the cost of routing the spinning cells for the legitimate and the malicious ORs. Observe, that for each OR, malicious or legitimate, the effort grows linearly to the packet size. In addition, the effort of the malicious OR is significantly lower to the legitimate ORs, and the difference increases as the cell number increases. More precisely, the difference in the effort can grow from approximately 60% to nearly 85%. In Table 1 the relative percentage of the difference during the routing effort is listed for each experiment.

As far as the spins achieved by each spinning packet are concerned, we show the time required for a series of cell relays in Figure 4. Each OR can spin a cell at a maximum rate of 25 relays per second.

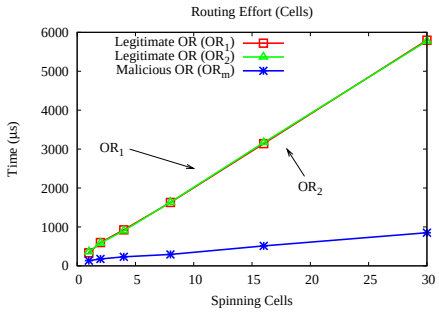

Fig. 3. The cost in terms of time for legitimate ORs to forward spinning packets in contrast with the effort of a malicious OR.

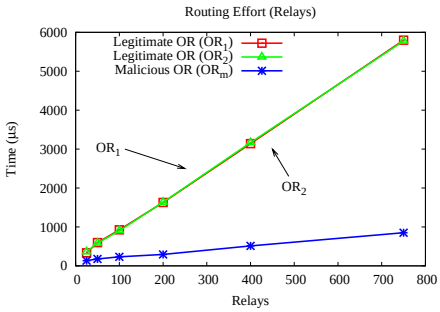

Fig. 4. The cost in terms of time for legitimate ORs to forward spinning packets in contrast with the effort of a malicious OR.

OR_1	OR_2	OR_m	$\frac{OR_1-OR_m}{OR_1}$	$\frac{OR_2-OR_m}{OR_2}$
336.199	361.403	133.136	60%	63%
598.276	584.898	176.997	70%	70%
926.444	901.521	232.435	75%	74%
1629.16	1635.22	294.156	82%	82%
3140.73	3167.97	513.536	84%	84%
5798.09	5777.1	850.997	85%	85%

Table 1. The percentage difference of the routing effort spend by the malicious OR (OR_m) relatively compared with the routing effort spend by the legitimate ORs, (OR_1, OR_2). Routing effort is recorded in μs, which is the period of time required for an OR to send the packet to the next hop in the circuit after it has successfully received it.

Circuit Building. One fundamental property of the packet spinning attack is that a malicious OR can occupy several legitimate ORs, by making circular circuits of arbitrary length. Recall that the default length of a TOR circuit length is three, but the protocol does not impose any constraints for larger circuits.

In Figure 5 we present real world experiments for the creation of long TOR circuits over time. Notice that, even though the time needed for the circuit creation increases exponentially in terms of the circuit length, we were able to create TOR circuits of more than 30 hops. This means that using one malicious OR we would be able to keep busy more than 30 legitimate ORs.[3]

5 Compromising Anonymity

Based on the results we highlighted in Section 4, we proceed to explore how an actual packet spinning attack can compromise the anonymity of users that utilize a real anonymizing system, namely TOR.

[3] In reality, we can do even better. We can use one malicious OR to issue circular circuits with several other ORs, as it is depicted in Figure 2.

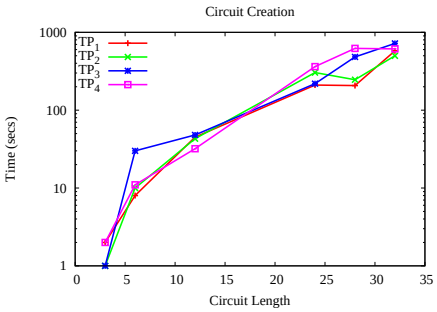

Fig. 5. The time needed for a malicious OR to issue TOR circuits of arbitrary length.

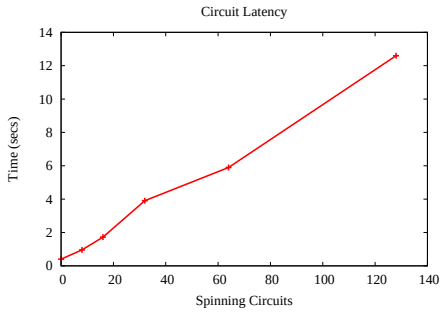

Fig. 6. Time latency for building additional circuits over an existing set of Tor nodes.

Experimental Setup. To conduct the experiments we used the `TorFlow` [2] package which is a complementary package of the TOR project [1]. `TorFlow` is written in `Python`, and it is composed of a series of scripts, which utilize the TOR control channel to communicate with active ORs. The TOR software supports a protocol for OR instrumenting. A client can connect to a specific port, bound to the control channel of an OR and give commands in a request-response fashion. Some operations that are allowed are: building circuits, attaching streams to circuits and querying an OR for various statistics.

All experiments were held in a private and isolated from the rest of the Internet TOR overlay. Using `TorFlow` we developed scripts that were creating a malicious OR which was part of the overlay. The rest of the ORs were kept intact in terms of software modifications. The only modification we did was the bandwidth constraints of the legitimate ORs. We set all legitimate ORs to be bounded to 1 Mbit per second. We did this to shorten the times it took to conduct our experiments. As we will discuss later, an adversary could launch a similar attack in a TOR overlay that is composed of ORs that experience greater bandwidth, by injecting more spinning cells in the artificial made loops.

Our strategy was the following. We were starting a TOR overlay of a variable number of ORs and we had a `TorFlow`-based Python script that was acting as a malicious TP, by instrumenting a malicious OR, and a legitimate TP that was trying to access the Internet using our isolated TOR. When our scripts weren't running, the legitimate TP could access the Internet using our TOR in a normal fashion. On the other hand, as we explore in detail in the rest of this section, when our scripts were running, the legitimate TP was experiencing side effects that could potentially lead to anonymity compromising.

Packet Spinning Effects. The first evident behavior experienced by the legitimate TP was the inability of circuit building in time. When a TP tries to access the Internet using TOR, it first selects three ORs and then tries to build the circuit telescopically. That is, it first contacts and establishes connection with the entry router, it then expands the circuit by tunneling the requests through

the entry router until the circuit is created. However, there is a timeout for the creation of circuits, which is set by default at 60 secs. In addition, ORs that were part of spinning circuits were unable to process the circuit creation requests fast enough and thus most of the circuit creation operations issued by the TP were failing.

In order to demonstrate this side effect more clearly, we contacted an experiment with a TOR overlay that was running on a single host. We eliminated out all network latencies since all communications between the TOR processes were local. Using our scripts we created an artificial loop of length five (the malicious OR was the first and last router and the legitimate ones were the three in the middle) and we started spinning cells inside. We then forced another script to create the same circuit. We proceeded in adding more spinning circuits and trying to build a new circuit over them. In Figure 6 we present our results.

Notice, that when there are no spinning circuits the circuit creation is almost spontaneous. However, as the spinning circuits increase, the circuit creation becomes a long process (recall that we have eliminated all network latencies, since the overlay runs in a single host) exceeding a period of 10 seconds.

6 Countermeasures

An anonymizing system that is based on a series of in-between relay nodes, and on TOR like circuits, is vulnerable to a packet spinning attack since it permits the creation of circular circuits. An adversary could utilize these circuits, having a malicious relay node that is not taking part in cryptographic operations and thus it has time to flood the circular circuit with fraudulent packets. We carried out all this analysis, using a real anonymizing system, TOR.

An obvious countermeasure would be to embed information of a circuit in all participating relayers. This would prevent the creation of circular circuits, but subsequently would decrease the anonymity level provided by the system. We are against any countermeasure that degrades the anonymity level of existing systems.

Instead of preventing circular circuits, our solution aims at reducing the effects of artificial loops in an anonymizing overlay. We propose existing anonymizing systems to employ *Tree Based Circuits (TBCs)*. More precisely, instead of having serial circuits, like the ones used in TOR, we propose that circuits will expand from the entry node in a tree fashion targeting the final destination. In Figure 7 we depict a TBC. The entry node issues two connections with two middle nodes and each of the middle nodes issues two connections with two exit nodes. In this example, one exit node is shared between the two middle nodes, but this is not obligatory. The dashed lines present OR connections that are ready to be utilized and the solid lines present an active TOR stream.

Introducing TBCs in an anonymizing system like TOR raises some important questions. *How fast a TBC can be constructed?* Recall, that TOR is used for low-latency communications. *Does it degrades the level of anonymity?* Notice, that by employing TBCs will be impossible to make circuits with many levels in terms

of hops, since trees grow exponentially. *Is a TBC vulnerable to packet spinning?*. In the rest of this section, we address each of this question in detail.

How fast a TBC can be constructed? Currently, TOR builds four circuits on startup. This is done for redundancy. Each OR maintains these four circuits and it is able to use any of them upon a circuit failure. Our proposal, follows the same logic, but instead of creating four distinct circuits, we create a TBC. A TBC has many circuits that can be used as alternatives upon a circuit failure. Moreover, the TBC can be created asynchronously; the client does not need to wait for the whole TBC creation in order to start transmitting information. The TP remains responsible for the circuit creation, but for each expand operation, the circuit is expanded towards multiple directions giving, schematically, the impression of a tree structure.

Does it degrades the level of anonymity? The ability of building large circuits in terms of hops gives the impression of higher anonymity, since the packet is relayed more times, although the latency, for the same reason, is increased. By employing TBCs the relay node number increases, but the latency does not, since the hops from the entry node to the exit one are kept at a low level (again there is no constraint for the depth of the TBC, but it is evident that large TBCs are an expensive operation, since trees grow exponentially). Essentially, the relay nodes involved in a transaction are of the same magnitude as of the relay nodes involved in current TOR circuit, but the relay nodes involved in the whole protocol negotiation are many more. The additive cost of a TBC in contrast with a plain TOR circuit is that each OR has to maintain some state in order to correctly route requests back to the TP.

Is a TBC vulnerable to packet spinning? A TBC can not contain loops, but again an adversary can place some malicious nodes in order to create TBCs that send requests back to the entry node and in this way create artificial loops. However, the adversary has to own more than one node in order to compromise a circuit, but, more of importantly, the users can escape more easily from loops by routing their requests using TBCs instead of plain circuits. A TBC gives the user more alternative circuits to the final destination, which in turn decrease the probability of encountering nodes that are overwhelmed by spinning packets in a circular circuit.

7 Conclusion and Future Work

In this paper we presented a novel attack against modern anonymizing systems, in which a series of relay nodes route cryptographically wrapped packets. The attack is based on inserting a malicious node in an anonymizing overlay, that is able to construct circular circuits and forces packets to spin indefinitely inside those loops. In this fashion an adversary can keep legitimate routers busy while at the same time they can inject their own, unloaded, malicious nodes. Since these nodes are not kept busy, they have a higher probability of being selected by users wanting to utilize the anonymizing system. This way, new circuits are

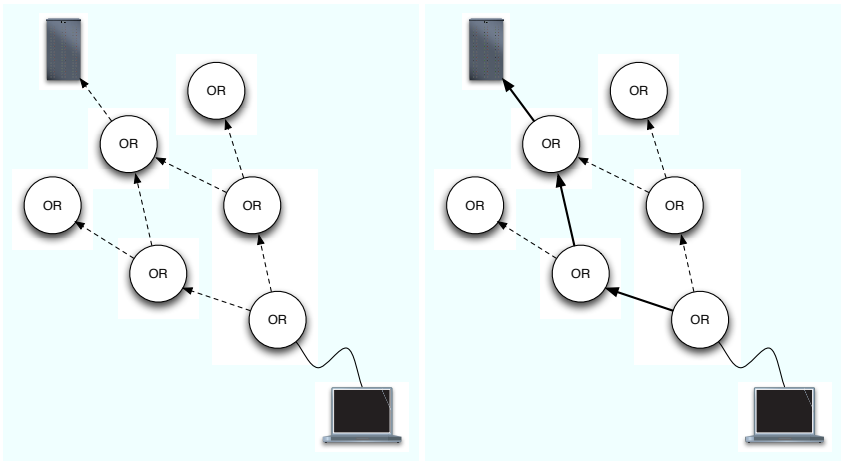

Fig. 7. Proposed solution. Tree based TOR circuits that are not vulnerable to circular circuits.

more likely to contain malicious nodes, and the anonymity of the user can be compromised.

We evaluated our attack using a real-world TOR system and our evaluation showed that such an attack is feasible. Finally, we came up with a method to counter packet spinning attacks and proposed Tree-Based Circuits. We showed that TBCs can be constructed relatively fast, they do not degrade the anonymity properties of the system and they are not vulnerable to packet spinning. Part of our future work is to implement TBCs in the TOR system and evaluate their performance, as well as further examining if a TBC is vulnerable to similar attacks like the one presented in this paper.

Acknowledgments

We thank the anonymous reviewers for their valuable comments. Vassilis Pappas, Elias Athanasopoulos, Sotiris Ioannidis, and Evangelos P. Markatos are also with the University of Crete. This work was supported by the Marie Curie Actions - Reintegration Grants project PASS. Elias Athanasopoulos is also funded from the PhD Scholarship Program of Microsoft Research Cambridge.

References

1. The TOR Project. `http://www.torproject.org/`.
2. TorFlow. `https://www.torproject.org/svn/torflow/README`.
3. Adam Back, Ulf Möller, and Anton Stiglic. Traffic analysis attacks and trade-offs in anonymity providing systems. In Ira S. Moskowitz, editor, *Proceedings of Information Hiding Workshop (IH 2001)*, pages 245–257. Springer-Verlag, LNCS 2137, April 2001.

4. Kevin Bauer, Damon McCoy, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Low-resource routing attacks against tor. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2007)*, Washington, DC, USA, October 2007.
5. Oliver Berthold, Andreas Pfitzmann, and Ronny Standtke. The disadvantages of free MIX routes and how to overcome them. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 30–45. Springer-Verlag, LNCS 2009, July 2000.
6. Nikita Borisov, George Danezis, Prateek Mittal, and Parisa Tabriz. Denial of service or denial of security? How attacks on reliability can compromise anonymity. In *Proceedings of CCS 2007*, October 2007.
7. David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 4(2), February 1981.
8. Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, July 2000.
9. George Danezis. The traffic analysis of continuous-time mixes. In *Proceedings of Privacy Enhancing Technologies workshop (PET 2004)*, volume 3424 of *LNCS*, pages 35–50, May 2004.
10. George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, pages 2–15, May 2003.
11. Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.
12. Michael J. Freedman and Robert Morris. Tarzan: A Peer-to-Peer Anonymizing Network Layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002)*, Washington, DC, November 2002.
13. David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. Hiding routing information. In *Information Hiding*, pages 137–150, 1996.
14. Nick Mathewson and Roger Dingledine. Practical traffic analysis: Extending and resisting statistical disclosure. In *Proceedings of Privacy Enhancing Technologies workshop (PET 2004)*, volume 3424 of *LNCS*, pages 17–34, May 2004.
15. Steven J. Murdoch and George Danezis. Low-cost traffic analysis of Tor. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*. IEEE CS, May 2005.
16. Arjun Nambiar and Matthew Wright. Salsa: A Structured Approach to Large-Scale Anonymity. In *Proceedings of CCS 2006*, October 2006.
17. Jean-François Raymond. Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 10–29. Springer-Verlag, LNCS 2009, July 2000.
18. Michael Reiter and Aviel Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1), June 1998.
19. Marc Rennhard and Bernhard Plattner. Introducing MorphMix: Peer-to-Peer based Anonymous Internet Usage with Collusion Detection. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2002)*, Washington, DC, USA, November 2002.

20. Rob Sherwood, Bobby Bhattacharjee, and Aravind Srinivasan. P5: A protocol for scalable anonymous communication. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, May 2002.
21. Robin Snader and Nikita Borisov. A tune-up for Tor: Improving security and performance in the Tor network. In *Proceedings of the Network and Distributed Security Symposium - NDSS '08*. Internet Society, February 2008.
22. Li Zhuang, Feng Zhou, Ben Y. Zhao, and Antony Rowstron. Cashmere: Resilient Anonymous Routing. In *Proc. of NSDI*, Boston, MA, May 2005. ACM/USENIX.

Appendix: Technical Discussion

Routing in Tor

Each Tor node has a routing table that contains entries in the form: (source connection, source circuit id) - (destination connection, destination circuit id)[4]. Upon a circuit establishment, each node adds an entry in its routing table to forward cells for that circuit. In order for the last node (exit node) to know its position, the routing entry for the specified circuit does not have a destination connection (is has a NULL value of destination connection). After the circuit establishment, Tor nodes are able to route cells in a circuit. The pseudo-code of the function executed upon receiving a new cell is shown in Figure 8.

A simple way to make a cell spin in a circular circuit is to change the last node's routing entry's destination connection from NULL to the connection with the second node in the circuit. That way whenever the last node receives a packet it will forward it again to the second node in the circuit.

```
function receive_cell(cell c)

  decrypt_one_layer(c)

  if (is_recognized(c)
    //do exit node stuff
    //...
  else

    next_conn, next_circ_id = get_route_info(c)

    if (next_conn)
      c.circ_id = next_circ_id
         send_cell(c, next_conn)
    else
      //circuit stops here bu the cell wasn't recognized
      drop_cell(c)
```

Fig. 8. The receive cell Tor's function pseudocode

Spin in Tor implementation

Keeping the previous in mind, we altered the source code of Tor in order to implement the cell spin. The procedure to make a cell spin in in a Tor circuit using a colluding TP and an colluding OR is comprised by the following steps:

[4] Connection denotes a TLS connections with other Tor nodes. Many Tor circuits can be multiplexed in a single connection.

1. `Establish a circular circuit.` Although Tor's node selection algorithm never selects a node twice (as an entry and as en exit) we used the control component of Tor (through `TorFlow` [2]) to explicitly select the nodes for a circuit. That way, we create a circular circuit with out colluding OR placed at entry and exit positions.
2. `Inform the colluding OR.` In order to get the cell spin we have to inform the colluding OR to change its routing table. So, the first thing to do is to get the destination connection and destination circuit id pair that forwards cells to the second OR. This is done by sending a cell (containing a special message) encrypted only with the entry node's key to the circuit. That way our colluding OR (as an entry node) recognizes it, keeps the destination connection and circuit id and forwards it down the circuit.
3. `Change OR's routing table.` The previous cell that our colluding OR forwarded will end up again to it but this time it wont be recognizable. So, our colluding node will get an unrecognizable cell that stops there. That time it will suppose that this cell is the one it forwarded before and will set the destination connection and circuit id of the current routing entry to the values kept at step 2.