Introducing SOR: SSH-based Onion Routing

André Egners[†], Dominic Gatzen[†], Andriy Panchenko[*], and Ulrike Meyer[†]
[†] *Research Group IT Security, UMIC Research Center*
RWTH Aachen University, Germany
E-mail: {surname}@umic.rwth-aachen.de
[*] *Interdisciplinary Centre for Security, Reliability and Trust*
University of Luxembourg, Luxembourg
E-mail: {firstname.lastname}@uni.lu

***Abstract*—Traditional low-latency anonymization techniques apply non-standardized, complex, and often even proprietary protocols. Apart from poor performance, the high development effort leads to the existence of at most one single implementation. This in turn increases the risk of creating so-called *software monocultures*, where failures in the single implementation can compromise the overall security.**

In this paper we introduce SOR – a novel approach for anonymization that is completely based on standardized, well tested, and performance-tuned protocols. It utilizes out-of-the-box nested SSH connections to achieve an exhaustive state-of-the-art anonymization system based on onion routing. Our approach supports both sender and receiver anonymity. Besides of many audited implementations, the SSH protocol itself is mature and has been thoroughly analyzed with respect to security issues. The results of our evaluation show that our approach outperforms existing alternatives by a factor of up to nine without sacrificing the degree of anonymity. Moreover, SOR can be easily implemented which increases the chances of having many alternative clients available.

***Keywords*-Privacy, Anonymity, SSH, Onion Routing**

I. Introduction

With the growth of the information exchange on the Internet privacy issues attracted a significant importance and became a concern. While cryptography can be used to protect the content of communication, it is still visible who communicates with whom. Anonymous communication deals with hiding relationships between communicating parties. Without this protection an attacker is able to deduce information about the network addresses of involved senders and recipients. Often this is sufficient to uniquely identify a person and link this person to the content he accesses or disseminates.

Many approaches have been proposed to provide anonymity on the network layer. Still, only a few of them have reached wide-scale deployment, e.g., JAP [1] and Tor [2]. All of these approaches rely on non-standardized, complex, and often even proprietary protocols. This has several severe disadvantages. First, due to a high development effort there exists at most one single implementation. This inherits the risk of creating so-called software monocultures where a failure in the single implementation can paralyze the whole network, possibly compromising the anonymity of many users. Second, if the underlying protocol specifications are changed or updated often, as for example in Tor (e.g., the existence of already the 3rd generation of the directory protocol, which is definitely not the final one [3]), additional implementations are even more difficult to maintain. Third, the increase of multimedia content transmitted over the Internet leads to a drastic increase of bandwidth requirements. While the backbones and access networks of the Internet could be adjusted to the increased needs, current anonymizers have not been able to meet these demands, thus leading to a decrease in network performance. Practical usage of the state-of-the-art anonymization networks such as Tor and JAP usually leads to delays which are not tolerated by the average end-user [4]. This, in return, discourages many of them from the use of anonymization and hence indirectly lowers the protection of remaining users since the anonymity proportionally depends on the size of the user base.

The implications of a poor performance have been shown in [5]: the user base of a network drops linearly with increasing latency. Several papers have dealt with the performance analysis of anonymization systems [4], [6] and have shown the need and possibilities for better quality of service and increased performance [7], [8], [9].

In this paper we address the aforementioned problems by introducing SOR, a novel efficient anonymization technique purely based on SSH (Secure SHell) – a well-known, established, well-tested, and standardized protocol. Even though the primary intention of this protocol is to access shell accounts on Unix based systems, this protocol out-of-the-box provides all the features required to build a comprehensive anonymization approach. We show how SSH can be used as a building block to provide state-of-the-art anonymization based on onion routing. For this, merely a controller logic has to be implemented. We then provide a performance evaluation and discuss design implications of SOR. Furthermore, we compare our approach to Tor – the most popular low-latency anonymizer.

II. Related work

The typical approach for anonymization is to send messages not directly to the recipient, but rather on a detour through several *middle nodes*. This can be done in a way such that the middle nodes cannot determine for certain whether the relayed data originates from the predecessor or is forwarded on behalf of other users. The objective of this procedure is to hide the sender of a message, the receiver of a message, or the relationship between the sender and receiver.

In this paper, we focus on low-latency anonymization networks. These are designed for real-time communication, such as web-browsing or instant messaging. The oldest representative in this category is the *single hop proxy* approach, e.g., *anonymizer.com*. Single hop proxies, however, provide a single point of failure and trust. Therefore, they are inapplicable for users with a high demand for anonymity.

The most popular and widespread anonymizer today is Tor. Tor is a low-latency overlay network deployed in late 2003 with the goal to provide protection against a *non-global adversary* [2]. It consists of servers that are called *onion routers* (ORs). Currently the network is comprised of about 2,000 ORs[1] [10], [11] that are running more or less permanently by volunteers scattered around the globe.

To anonymize Internet communications, clients' software, the *onion proxy* (OP), constructs *circuits* of encrypted connections through a path of randomly chosen ORs. A Tor circuit, by default, consists of three ORs, where each OR only knows *(i)* which peer has sent him data (the predecessor) and *(ii)* to which peer he is relaying data (the successor).

During circuit creation in Tor, the circuit initiator uses Diffie-Hellman key exchanges to establish shared symmetric session keys with each OR in the circuit. The user's OP encrypts all traffic before it is sent over the circuit using these keys in reverse order, starting with the key of the last OR. Upon receiving traffic, each OR on the circuit removes (or adds, depending on the direction) one layer of encryption while relaying the data to the next OR, so only the last OR (the exit node) knows the actual destination of a TCP stream. The last node in a circuit reassembles the TCP packets and delivers them to the final destination.

Application data is generally transferred unencrypted between the exit node and the recipient on the Internet, unless the user and the recipient are using end-to-end encryption, e.g., TLS/SSL protocols.

Receiver anonymity in Tor is supported by so-called *hidden services*. They rely on rendezvous points that connect anonymous circuits originating from a client and a hidden service. A client selects a random Tor node as a rendezvous point, creates a circuit to it, and informs the hidden service through one of the introduction points about its willingness to communicate over a specified rendezvous point. Each of the principals relies on himself to build a secure circuit to both the introduction and the rendezvous point. According to Loesing et al. [12], the average time before sending out the request and getting the answer from a hidden service in Tor is 24 seconds, which is far more than the average user is willing to wait (this value is estimated to be about 4 seconds as stated by Wendolsky et al. [4]).

Besides hidden services in Tor, there are similar concepts in other networks, e.g., the *eepSite* concept in I2P [13] and *Freesites* in the Freenet network [14]. While eepSites are specially tailored for web services used in I2P, Freenet uses distributed redundant storage of data which is accessible only within the network.

AN.ON [1] (also known as JAP or Jondonym) is based on onion routing as well. One of the main differences to Tor is that users cannot choose the circuit freely between the relays. In AN.ON normally three relays are forming a predefined path, denoted as *cascade*. The user only has the possibility to choose one of the predefined cascades. Moreover, AN.ON does not provide forward secrecy[2].

Tarzan and MorphMix are two peer-to-peer (P2P) based anonymization techniques for implementing onion routing. Unlike the earlier approaches, a MorphMix node does not have to have knowledge about all other MorphMix nodes in the network. For the circuit setup, so-called *witness nodes* are used to facilitate the selection of nodes for circuit extension. A probabilistic checking is applied to detect collusion. However, this protection scheme has been shown to be broken. In Tarzan every node has a set of peer nodes for exchanging cover traffic which are called *mimics nodes*. Nodes select their mimics in a pseudo-random universally verifiable way. Each Tarzan node needs to know all other nodes in the Tarzan network. Circuits are built only through nodes that exchange cover traffic between themselves. Neither Tarzan nor MorphMix are in active use today.

The Invisible Internet Project (I2P) system[3] is an approach that makes use of so-called *garlic encryption*. This is a variant of onion encryption where multiple messages wrapped into a single "garlic message", encrypted with a particular public key. Intermediary nodes cannot determine how many messages are in the garlic message and where they are destined. The paths in I2P are not necessarily symmetric, as different circuits can be used for in and outgoing traffic. Problems with I2P include missing transparency for their network layer protocol and a complete lack of academic coverage.

Crowds [15] is an alternative to the techniques described above. It is based on a simple randomized routing protocol,

[1]as in September 2011

[2]Forward secrecy ensures that a session key created from a set of long-term asymmetric keys will not be compromised if the long-term private key is compromised in the future.

[3]See http://www.i2p2.de/ for more information.

where all participants forward messages on behalf of other users, as well as their own. The main idea of Crowds is to hide the identity of the sender during communication by routing the sender's messages randomly within a group of similar users ("blending into a crowd") before the message is sent to the final recipient. The GNUNet system [16] also makes use of a randomized routing protocol, where the forwarding of messages on behalf of the other nodes (here denoted the *"indirection"*) depends, among other things, on the load in the network. Crowds never left the stage of a research implementation, and further Crowds and GNUNet offer fairly weak protection against strong attackers [17], [18].

Another class of anonymizers rely on multi- or broadcast channels, e.g., [19], [20], [21]. However, due to poor scalability all of these networks are either not used anymore or do not have a widespread user base.

Shalon [22] is a recent proposal for anonymous communication that is fully based on standardized protocols and significantly outperforms Tor. The idea of Shalon is similar to our approach, however, Shalon is able to provide only sender anonymity. Even though there is an extension for Shalon for enabling hidden services [23], this extension uses modification of standardized protocols to achieve its functionality. Inspired by the idea of Shalon, we provide an exhaustive anonymizer that is able to provide both sender and receiver anonymity.

III. Fundamentals of SSH

This section briefly explains the key features of SSH that are leveraged in our design. For a comprehensive coverage on the SSH protocol an interested reader is referred to [24].

In short, SSH is a comprehensive protocol which takes care of the connection establishment, authentication, key exchange, encryption, as well as connection multiplexing and port forwarding. The SSH protocol allows three different modes of port forwarding: *local*, *dynamic*, and *remote*. Local port forwarding forwards traffic coming to a local port to a specified remote server and port via the SSH connection to a third node. For example, `ssh -g -L 4321:www.openssh.com:80 gate` creates an SSH connection to `gate`, listens locally on port 4321 for incoming connections, and redirects them all to port 80 of `www.openssh.com` via the SSH tunnel. Dynamic port forwarding (`-D` option) on the other hand allows the SSH client to dynamically chose to which host and port data will be forwarded via the SSH connection. This mode essentially represents a SOCKS proxy running on the localhost, but instead the traffic is routed via the SSH server. The reverse port forwarding (`-R` option) mode allows the SSH client to reserve a port on the SSH server. For example, `ssh -R 1234:intranet:80 gate` creates an SSH connection to `gate`, opens port 1234 on `gate`, and redirects all the incoming connections of this port to port 80 of `intranet` through the SSH tunnel.

IV. Our Approach

In this section we introduce our new anonymization technique called SOR. The key idea of our approach is to construct an anonymization overlay based on the SSH tunneling functionalities. We describe how to design a protocol that achieves both sender and receiver anonymity. To the best of our knowledge, this is the first proposal how standardized out-of-the-box protocol can be used to reach receiver anonymity. Our server is the standard SSH server without any modifications. Here, we describe the logic of the controller that has to be implemented at the client side.

A. Protocol for Sender Anonymity

A circuit in SOR is a chain of SSH connections which are telescoped into each other. To this end we utilize out-of-the-box features of the SSH protocol.

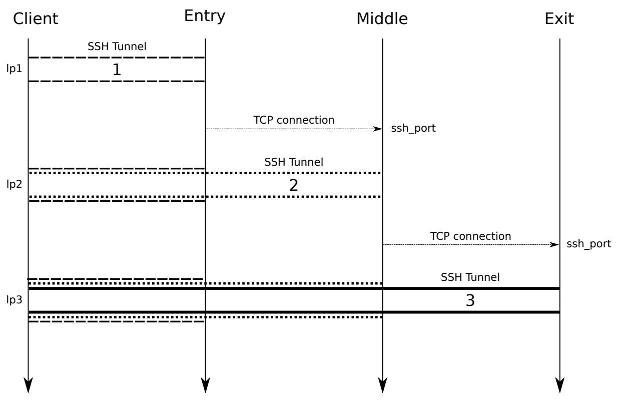

Figure 1. SOR-Circuit Construction

We make use of local and dynamic port forwarding of SSH as mentioned above. The goal is to use them to construct a tunnel of onion layered encryption. The construction (see Figure 1) works as follows:

(1) The client instructs SSH to create an encrypted connection between itself and the *Entry* node, and to perform local port forwarding from local port `lp1` via *Entry* node to the *Middle* node (`ssh -L lp1:Middle:22 Entry`).
(2) The client instructs SSH to create an encrypted connection between itself and the *Middle* node which is tunneled through the connection to the *Entry* node. Afterwards SSH performs local port forwarding from local port `lp2` via the established tunnel to *Middle* node to the *Exit* node (`ssh -L lp2:Exit:22 localhost -p lp1`).
(3) The client instructs SSH to create an encrypted connection between itself and the *Exit* node which is tunneled through the connection to the *Middle* node.

Afterwards SSH performs dynamic port forwarding on local port `lp3` (`ssh -D lp3 localhost -p lp2`).

The overall procedure yields a three hop circuit with three layers of onion encryption. Setting dynamic port forwarding on the last connection (3) to the exit node (besides creating an encryption layer between the client and the exit node) effectively enables the SOCKS proxy functionality of SSH which is made available to the client on local port `lp3`. Herewith the client is able to establish TCP connections to arbitrarily hosts through the tunnel to the exit node. Conveniently, multiplexing connection through a SOR circuit is automatically achieved by the multiplexing functionality of SSH.

Hence, we reached similar functionality of onion routing for client traffic anonymization as in other approaches, but in a simple and elegant way utilizing standardized protocols and their out-of-the-box implementations.

B. Protocol for Responder Anonymity

In the previous section we explained how to use SSH to construct the basic block of our approach, i.e., circuits for sender anonymization. Using this basic block, we are also able to produce a functionality similar to the hidden services in Tor, i.e., to achieve responder anonymity, hiding the IP address of the service from its clients and other network participants. We refrain from explaining the hidden service protocol in detail, as its event flow and naming of entities are very similar to that of the Tor hidden services [2].

In order to achieve responder anonymity, there is a need to have a possibility to open a port on remote machines (called Introduction Points in the terminology of hidden services in Tor). This is the fundamental difference to the tunnels needed for sender anonymity. The connections to the remote port have to be transparently multiplexed and forwarded to the tunnel initiator. To achieve this functionality, we apply reverse tunneling of SSH after connecting to the last hop (step (3) of the protocol in Section IV-A). Hence, one should perform `ssh -R rp1:localhost:lp3 localhost -p lp2` in order to open remote port `rp1` on the last hop (addressed by local port `lp2`) and redirect all the incoming connections to port `lp3` of `localhost`.

To connect to a hidden service the client needs to be in possession of a *descriptor* containing the necessary information, i.e., IP of the introduction points together with the port for the hidden service. For load distribution and omitting bottlenecks, similarly to Tor, the client in SOR randomly chooses a SOR node and declares it as a *rendezvous point*. To this end, the client creates a tunnel to the rendezvous point and sets local port forwarding to another port `rp1` on the rendezvous point (`ssh -L cp3:localhost:rp1 localhost -p cp2`). The client connects to a randomly chosen introduction point of the hidden service, informing about the proposed rendezvous point's address and port `rp1` which will be used to join the connections. The hidden service connects to the rendezvous point of the client using reverse port forwarding on the port `rp1`, (`ssh -R rp1:localhost:hp3 localhost -p hp2`). After the circuits have been joined, the client initiates an additional SSH handshake with the hidden service through the tunnel (`ssh -L cp4:localhost:hp4 localhost -p cp3`). The latter provides the end-to-end security for the connection (note that checking the certificate of the hidden service via SSH is only possible if the connection is originated from the client).

V. Evaluation

This section presents an evaluation of SOR with the most relevant performance measures, such as bandwidth, circuit buildup time, and latency. Our experiments have been carried out in a fully controlled lab network and in the Planetlab testbed [25]. In the following we compare our approach to Tor – the state-of-the-art anonymization network. In addition, we also evaluated the performance of the hidden service protocol of SOR and Tor. For this purpose a private Tor network has been set up and TorFlow [26] and TorCtl[4] frameworks were used to obtain the measurements.

A. Experiment Setup

The most relevant performance metrics for anonymization network are the RTT, circuit buildup time, and the achievable bandwidth. A circuit length of three is chosen for the evaluation. This is the standard length for Tor and other onion routing based approaches. To obtain stable results, the tests run were repeated 10,000 times.

1) Laboratory Experiment: The machines in the lab network were equipped with a 2.33 GHz Quad Core Xeon CPUs, 4GB of RAM, and a 1GBit Ethernet connection. Each node respectively runs a Tor or SOR process. An additional machine is used to perform the measurements. With respect to comparing two network anonymization protocols, choosing an environment which can be fully controlled and which does not exhibit any other resource consumption allows an evaluation in which close to no other effects can influence the protocol's performance.

Figure 2 (left) shows a direct median comparison (whisker bars represent the standard deviation from the median) of SOR and Tor in the local lab network with respect to sender anonymity. As it can be seen, the RTT in SOR is about a half of the RTT on Tor. Creating a SOR circuit (Buildup) takes longer than creating a Tor circuit. Tor outperforms SOR as it uses already established TCP/TLS connection between the nodes while creating a circuit. If one compares the circuit setup results in the *System-Ready* experiment, this

[4]PyTorCtl: https://gitweb.torproject.org/pytorctl.git

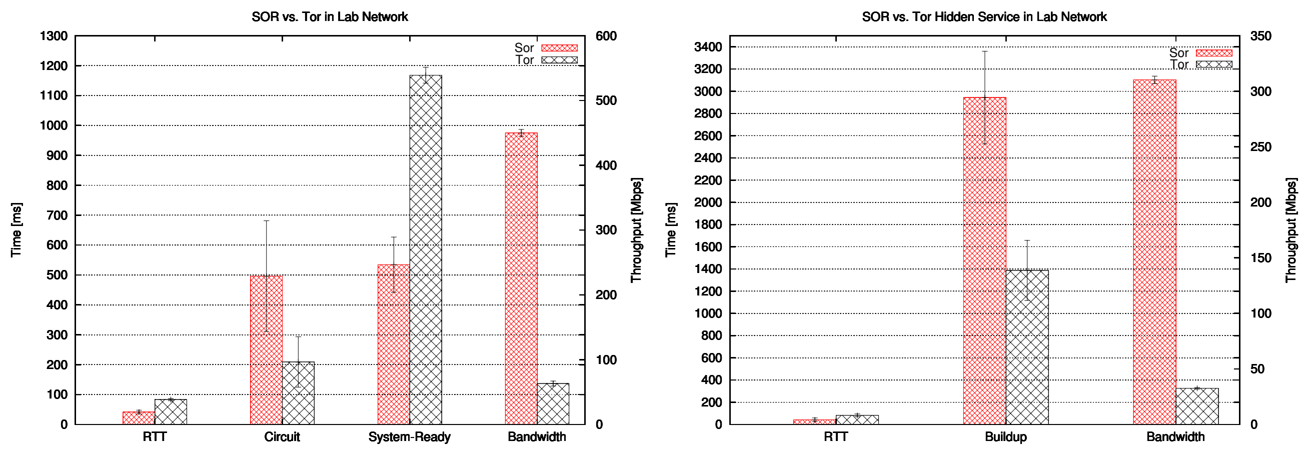

Figure 2. Left: SOR vs. Tor Performance (Median) / Right: SOR vs. Tor Hidden Service Performance (Median)

observation becomes obvious. System-Ready represents the duration until the anonymization system is ready to be used, i.e., circuits are available. It can be seen that Tor spends a significant amount of time establishing TLS connections to ORs in order to speedup the circuit setup afterwards. The most impressive result is that SOR outperforms Tor in terms of bandwidth by a factor of nine. We suppose that this difference is due to a mature and performance optimized implementation of SSH.

Figure 2 (right) shows the results of the hidden service performance of SOR and Tor as median (whisker bars represent the standard deviation from the median). The measurements have been obtained from a client connecting to a hidden service. It can be seen that the RTT is only slightly influenced by the increased circuit length from client to the hidden service. *Buildup* represents the time it takes for client so successfully establish a connection to a hidden service. Tor outperforms SOR by a factor of two. This observation can by explained by the fine tuned connection protocol of Tor. For example, one half of the Diffie-Hellman handshake between the client and the hidden service is already done when the clients request a connection via one of the introduction points. SOR has no such mechanism as it relies on SSH, i.e., the end-to-end encryption is only negotiated after the rendezvous point joined the circuits of the client and the hidden service. Also, Tor keeps known ORs cached in order to more quickly be able to create new circuits (SOR contacts each time the directory for fetching ORs information). Once the caching is disabled, establishing a connection to a hidden service in Tor takes significantly longer. With respect to the bandwidth SOR outperforms Tor significantly. Compared to the bandwidth achievable via regular circuits, the bandwidth achievable from a client to a hidden service is only slightly reduced.

SOR requires about 2.7 seconds to make a hidden service available to the network. This includes creating circuits to the introduction points and publishing the descriptor to the directory. Tor requires about 32 seconds until the hidden service is available to clients. The increase in time is possibly due to the more complex publishing mechanisms of the Tor directories.

2) Planetlab Experiment: Planetlab nodes[5] have to meet minimal requirements, i.e., have four cores CPU at 2.4GHz and 4GB of RAM. With respect to the network connection, nodes can be as slow as DSL lines. However, during our evaluation we observed a curiosity. The achievable bandwidth through a SOR-Circuit was in fact higher than the bandwidth to the node not using a SOR-Circuit. Since the Planetlab nodes are managed using SSH, we suspect that SSH is prioritized and gets higher amount of resources allocated to it. Moreover, it is questionable whether the results of network performance measurements in Planetlab can be compared. The resource usage, including CPU and networking, differs at any given time for any given node. This makes is hard to exactly reproduce the experiment setting when comparing the performance of two protocols. It has been shown that the resource consumption of nodes has great influence on the available performance of circuits in Planetlab [27]. Hence, we refrained from the comparison in PlanetLab. Hence, in our case measurements in Planetlab are less suited for a *fair* network performance comparison.

VI. Discussion

In this section we elaborate on a selection of challenges and security implications that we encountered during the design of SOR.

A. User Account Privileges

The primary use case of SSH is secure shell access on remote machines. As such, the notion of user accounts is deeply woven into the protocol and its implementations.

[5]http://www.planet-lab.org/node/225

One possibility to implement anonymization protocol using SSH on the server side is to allow empty passwords. This is however a critical security risk as it may allow logins for accounts accidentally set to an empty password. In order to make convenient usage of SSH user accounts in SOR, we propose to employ a single public/private key pair for the authentication of SOR users on all of the servers. The key can be distributed, e.g., together with the SOR software. Using only one single key pair for all the users is necessary since otherwise SOR users could be differentiated based on their personal public/private key.

Still, providing a user account to unknown and untrustworthy third-parties is a security risk. Therefore, we suggest to set *no-pty* for the users' public key in SSH's *authorized_keys* file and specify the shell to, e.g., */bin/false*. Setting no-pty itself does not prevent ssh command execution. The combination of both options, however, prevents the allocation of interactive sessions and command execution for the SOR users.

B. Exit Policies

In Section IV we discussed how SOR realizes the concept of exit nodes. The main purpose of exit nodes is to allow traffic to leave the anonymization overlay to any destination on the Internet. Therefore the operators of exit nodes are in the first instance responsible for any abuse that is done using their nodes, which can be a legal risk in some countries. Hence, it is important to restrict this type connections and traffic that may leave the network via an exit node. Tor has addressed this issue with exit policies that can be specified by the node owners.

Unfortunately, SSH itself does not provide filtering mechanisms for dynamic outgoing connections. Even separately disabling port forwarding modes (local, dynamic, remote) is not possible in the vanilla SSH implementation. Currently it is therefore only possible to restrict exits either by firewalling mechanisms (e.g., using user space applications such as *iptables*) or using a custom restricted user shell for filtering connections.

C. Host Key Checking

SSH provides a mechanism to authenticate the server to the client based on the server's host key. This is done by the SSH client when connecting to a SSH server in order to protect against man-in-the-middle attacks. If the key fingerprint of the server is known to the client, i.e., it has previously connected to this server or the fingerprint is inserted from the descriptor of the server, comparing the key fingerprint to the cached one will determine if the server is the authentic one. The host key checking also compares the IP address and port mapping to the host key. The SOR client has to intelligently manage this mapping as it will frequently change for middle and exit nodes. This is due to the fact that connections to middle and exit nodes are established through a port on local host in order to tunnel through previously established connections (see Section IV). Hence, our SOR client maintains a dynamic *known_hosts* file during the construction of circuits.

VII. Future Work

As stated in Section VI-A restricting the interactive access of the user to SOR routers is of great importance. In the future work we will explore whether developing a custom shell has signification benefits over using standard Linux tools to achieve a similar level of security. Moreover, we will investigate ways to implement the policy restrictions of exit nodes. In addition to jailing the user, the algorithms used by SSH need to be tightly controlled. Clients connecting to SOR routers using different cryptographic algorithms for authentication, encryption, and data integrity protection, are potentially vulnerable to being deanonymized based on these connection properties (e.g., clients having a unique configuration). For improving performance, we plan to investigate advanced methods for path selection in onion routing such as [9]. Furthermore, we plan to develop a SOR client in alternative programming languages. Currently the controller logic of SOR client is implemented using the Ruby programming language.

VIII. Conclusion

In this paper we introduced SOR – a novel approach for anonymization that is purely based on standardized, tested, and performance-tuned protocols. It utilizes out-of-the-box nested SSH connections to achieve state-of-the-art anonymization based on onion routing. Contrary to Shalon – another work that achieves anonymization using standardized protocols, our approach supports both sender and receiver anonymity. Besides of many audited implementations, the SSH protocol itself is mature and has been thoroughly analyzed with respect to security issues. The results of our evaluation show that our approach outperforms existing alternatives by a factor of up to nine without sacrificing the degree of anonymity.

Acknowledgments

This work has been supported by the UMIC Research Centre of RWTH Aachen University and the National Research Fund of Luxembourg within the CORE grant project MOVE and EU FP7 PPP project OUTSMART. Thanks to Mark Schlösser who sparked our interest in the initial idea of this work. Also thanks to Fabian Lanze who provided insight into the Tor control protocol.

References

[1] O. Berthold, H. Federrath, and S. Köpsell, "Web MIXes: A System for Anonymous and Unobservable Internet Access," in *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, ser. Lecture Notes in Computer Science, H. Federrath, Ed., vol. 2009. Springer-Verlag, Jul. 2000, pp. 115–129.

[2] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The Second-Generation Onion Router," in *Proceedings of the 13th USENIX Security Symposium*. USENIX Association, Aug. 2004, pp. 303–320.

[3] R. Dingledine and N. Mathewson, "Tor Directory Protocol Specification," https://www.torproject.org/svn/trunk/doc/spec/dir-spec.txt.

[4] R. Wendolsky, D. Herrmann, and H. Federrath, "Performance Comparison of low-latency Anonymisation Services from a User Perspective," in *Proceedings of the Seventh Workshop on Privacy Enhancing Technologies (PET 2007)*, ser. Lecture Notes in Computer Science, N. Borisov and P. Golle, Eds., vol. 4776. Springer-Verlag, Jun. 2007, pp. 233–253.

[5] S. Köpsell, "Low Latency Anonymous Communication – How Long Are Users Willing to Wait?" in *Emerging Trends in Information and Communication Security*, ser. Lecture Notes in Computer Science, G. Müller, Ed., vol. 3995. Springer-Verlag, Jun. 2006, pp. 221–237.

[6] A. Panchenko, L. Pimenidis, and J. Renner, "Performance Analysis of Anonymous Communication Channels Provided by Tor," in *Proceedings of the Third International Conference on Availability, Reliability and Security (ARES 2008)*. Barcelona, Spain: IEEE Computer Society Press, March 2008.

[7] R. Snader and N. 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.

[8] S. J. Murdoch and R. N. Watson, "Metrics for Security and Performance in Low-Latency Anonymity Systems," in *Proceedings of the Eighth International Symposium on Privacy Enhancing Technologies (PETS 2008)*, ser. Lecture Notes in Computer Science, N. Borisov and I. Goldberg, Eds., vol. 5134. Springer-Verlag, Jul. 2008, pp. 115–132.

[9] A. Panchenko and J. Renner, "Path Selection Metrics for Performance-Improved Onion Routing," in *Proceedings of the 9th IEEE/IPSJ Symposium on Applications and the Internet (IEEE SAINT 2009)*. Seattle, USA: IEEE Computer Society Press, July 2009.

[10] "Tor Network Status," https://torstatus.kgprog.com/.

[11] P. Palfrader, "Number of Running Tor Routers." [Online]. Available: http://torstatus.blutmagie.de/

[12] K. Loesing, W. Sandmann, C. Wilms, and G. Wirtz, "Performance Measurements and Statistics of Tor Hidden Services," in *Proceedings of the 2008 International Symposium on Applications and the Internet (SAINT)*. Turku, Finland: IEEE CS Press, July 2008.

[13] "I2P project," http://www.i2p.net/, 2009, visited May 2009.

[14] Ian Clarke, O. Sandberg, B. Wiley, and T. 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*, July 2000, pp. 46–66.

[15] M. K. Reiter and A. D. Rubin, "Crowds: Anonymity for Web Transactions," *ACM Transactions on Information and System Security*, pp. 66 – 92, April 1998.

[16] K. Bennett and C. Grothoff, "GAP – practical anonymous networking," in *Proceedings of Privacy Enhancing Technologies workshop (PET 2003)*, R. Dingledine, Ed. Springer-Verlag, LNCS 2760, March 2003, pp. 141–160.

[17] M. Wright, M. Adler, B. N. Levine, and C. Shields, "The Predecessor Attack: An Analysis of a Threat to Anonymous Communications Systems," in *ACM Transactions on Information and System Security TISSEC'04*, vol. 7 (4). ACM Press, November 2004, pp. 489 – 522.

[18] A. Panchenko and L. Pimenidis, "Crowds Revisited: Practically Effective Predecessor Attack," in *Proceedings of the 12th Nordic Workshop on Secure IT-Systems (NordSec 2007)*, Reykjavik, Iceland, October 2007.

[19] D. L. Chaum, "The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability," *Journal of Cryptology*, no. 1, pp. 65 – 75, 1988.

[20] R. Sherwood, B. Bhattacharjee, and A. Srinivasan, "P5: A Protocol for Scalable Anonymous Communication," in *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, May 2002.

[21] S. Goel, M. Robson, M. Polte, and E. G. Sirer, "Herbivore: A Scalable and Efficient Protocol for Anonymous Communication," Cornell University, Ithaca, NY, Tech. Rep. 2003-1890, February 2003.

[22] A. Panchenko, B. Westermann, L. Pimenidis, and C. Andersson, "Shalon: Lightweight anonymization based on open standards," in *Proceedings of the 18th International Conference on Computer Communications and Networks (IEEE ICCCN 2009)*. IEEE Computer Society Press, Aug. 2009.

[23] A. Panchenko, O. Spaniol, A. Egners, and T. Engel, "Lightweight hidden services," in *10th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (IEEE TrustCom 2011)*. Changsha, China: IEEE Computer Society Press, Nov. 2011.

[24] E. T. Ylonen, C. Lonvick, "The Secure Shell (SSH) Protocol Architecture," RFC, January 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4251.txt

[25] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "PlanetLab: An Overlay Testbed for Broad-coverage Services," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 3, pp. 3–12, 2003.

[26] M. Perry, "Torflow: Tor network analysis," http://fscked.org/talks/TorFlow-HotPETS-final.pdf.

[27] A. Panchenko, "Anonymous communication in the age of the internet," Ph.D. dissertation, Department of Computer Science, RWTH Aachen University, 2010, wissenschaftsverlag Mainz, Aachen.