Lightweight Hidden Services

Andriy Panchenko*, Otto Spaniol†, Andre Egners†, and Thomas Engel*
† Computer Science department, RWTH Aachen University, Germany
E-mail: {surname}@cs.rwth-aachen.de
* Interdisciplinary Centre for Security, Reliability and Trust
University of Luxembourg, Luxembourg
E-mail: {firstname.lastname}@uni.lu
http://lorre.uni.lu/~andriy/

***Abstract*— Hidden services (HS) are mechanisms designed to provide network services while preserving anonymity for the identity of the server. Besides protecting the identity of the server, hidden services help to resist censorship, are resistant against distributed DoS attacks, and allow server functionality even if the service provider does not own a public IP address. Currently, only the Tor network offers this feature in full functionality. However, the HS concept in Tor is complex and provides poor performance. According to recent studies, average contact time for a hidden service is 24s which is far beyond what an average user is willing to wait. In this paper we introduce a novel approach for hidden services that achieves similar functionality as HS in Tor but does so in a simple and lightweight way with the goal to improve performance and usability.**

Additionally, contrary to Tor, in our approach clients are not required to install any specific software for accessing hidden services. This increases usability of our approach. Simplicity makes our approach easier to understand for normal users, eases protocol reviews, and increases chances of having several implementations of the protocol available. Moreover, simpler solutions are easier to analyze and they are naturally less prone to implementation failures rather than complex protocols. In this paper, we describe our approach and provide performance as well as anonymity analysis of resulting properties of the protocol.

I. Introduction

The primary goal of location hidden services is to preserve the anonymity of the service provider (i.e., responder anonymity) and to resist censorship. A hidden service can be, for example, a news website that is declared as illegal or is censored in a totalitarian regime. To the best of our knowledge, Tor is a single deployed anonymization network that provides state-of-the-art hidden services functionality to its users (see Section VIII for a discussion of other approaches and their inapplicability in our scenario). However, due to a complex design the performance of hidden services leaves much to be desired: it takes 24 seconds on average to contact a service which is offered anonymously in the Tor network [13]. Additionally, having only a single possibility to run the hidden services creates the risk of so-called *software monocultures*. In this case, failures in the single protocol or implementation can paralyze the whole network, possibly compromising the overall security.

Recently, several new approaches for anonymization have been proposed. One of them is Shalon [18], which is a lightweight anonymization protocol based on open standards. It aims to reduce complexity and delivers high bandwidth. The most significant advantage compared to other approaches is that Shalon makes use of out-of-the-box nested TLS connections to achieve a simple and elegant version of onion routing (onion routing is the state-of-the-art approach to achieve low latency anonymization on the Internet). The key feature of Shalon is the buildup of anonymous communication on top of the HTTP/SSL protocol suite. However, providing hidden services is not possible in the current design of Shalon. Due to its simplicity and high performance we have decided to build our HS design based on the Shalon concept.

The first and primary goal of hidden services is to have server resistance to distributed DoS (DDoS) attacks, i.e., it is required to mount a DDoS attack on the entire Shalon network to attack a hidden service. Secondly, hidden services are resistant to physical attacks as the location of the server is not hidden. Thirdly, as a side effect, hidden services can be offered even if the service provider does not own a public IP address or is located behind a firewall. Lastly, for convenience reasons we want Shalon to provide similar functionality as Tor – the state-of-the-art anonymization network used today.

The paper is structured as follows. Section II gives background information on Tor – the state-of-the-art approach for low-latency anonymization. In Section III we describe the building blocks that we used to build our approach. This is followed by a protocol description in Section IV and the evaluation of the protocol in terms of performance (Section VI) and security (Section VII). Section VIII summarizes related work in the paper's context. Finally Section X concludes and discusses the contributions of this paper.

II. Background on Tor

Tor is a low-latency overlay network deployed in late 2003 with the goal to provide protection against a *non-global adversary* [3]. It consists of servers that are called *onion routers* (ORs). Currently the network is comprised of about 2,000 ORs[1] [21], [15] that are running more or less permanently by volunteers scattered around the globe. Each OR runs on an Internet end-host and maintains TLS connections to many other ORs at any time.

To anonymize Internet communications, end-users run an *onion proxy* (OP) on their computer that is listening locally

[1] as in June 2011

for incoming connections and redirects TCP-streams through the Tor network. Thus, it makes use of *source routing*, which means that the client determines the whole path through the overlay network. When sending out the redirected TCP-streams, the 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). A circuit length of three constitutes a reasonable trade-off between security and performance, where the role of the middle OR is to hinder the last OR in the circuit (the *exit node*) to learn the identity of the first OR (the *entry node*). If the latter two cooperate, users can be deanonymized [3].

During circuit creation in Tor, the circuit initiator (OP) 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 reassambles the TCP packets and delivers them to the final destination. The Diffie-Hellman key exchange is used in order to prevent replay attacks and to provide perfect forward secrecy.

Once a circuit in the Tor network is established, the user's OP can use it as a tunnel for arbitrary TCP connections. To increase the protection against profiling attacks, a circuit is only used for a limited amount of time (or until a threshold of data volume has been transferred). In the current implementation, after approximately ten minutes a circuit is discarded and a new one is put into use. 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.

III. Fundamentals of Hidden Services in Shalon

Even though we aim to achieve a similar functionality to hidden services in Tor, contrary to their approach, we want our design to be based on the philosophy of Shalon: simple design and, if possible, reuse of existing standards and software libraries. Hence, similarly to Shalon, the basis for our approach is an SSL capable HTTP proxy. HTTP is a simple, standardized, and popular protocol used on the Internet. SSL is also a popular and standardized protocol to encrypt data in communication channels. However, in order to offer hidden services, there is a need to open a listening port on a remote machine (which we further call a *contact point*) and to accept multiple connections over this port. The connections should be forwarded back to the service provider. Since this forwarding must be performed efficiently (e.g., it would be inadequate to use a separate tunnel for every new connection), there is a need for a multiplexing protocol between the service and the contact point. In order to reach these needs, we modified the freely available proxy server Muffin [1]. We preferred this proxy server written entirely in Java because it is simple (contrary to Squid[2]) and does not have licence limitations (contrary to DeleGate[3]). The Muffin proxy was extended to support SSL, opening of a listening port, and multiplexing for handling multiple connections.

The contact point is the heart of the HS concept in Shalon. It acts as a gateway between the hidden service and its clients. We added a BIND command to the HTTP proxy. It is similar to the BIND command of the SOCKS protocol which is used to instruct the proxy server to open a listening socket on a specific port and to transparently forward data between the socket and the anonymization tunnel. If the proposed port is occupied, the contact point selects a random available port. Finally, the contact point informs the hidden service about the allocated port. This port is afterwards specified in the descriptor[4] of a HS.

After a successful port allocation for serving as a contact port for the hidden service, both ends of the connection – the hidden service initiator and the contact point – start the multiplexing protocol in order to handle simultaneous connections from clients in parallel. This is done transparently to the clients. For multiplexing connections in Shalon we selected the *WebMUX protocol*. WebMUX [5], originally developed by the W3C HTTP-NG group, is a protocol for multiplexing several streams over a single TCP connection. It is a further development of the *Session Control Protocol* (SCP) [19], [4] which provides lightweight multiplexing of data streams on top of a reliable stream oriented transport. WebMUX is intended for, but in no way restricted to, transport of web related protocols. Its original goal was to support simultaneous rendering of embedded objects in HTML documents in an efficient way. According to the authors, the protocol is simple, has low overhead, high performance, is deadlock-free (using credit based flow control), and allows multiple application layer protocol connections to be multiplexed over the same TCP connection. An implementation of the protocol adds special header fields to each application layer message, additionally there are control messages not containing any payload. Those control messages are, amongst other things, used for a credit based control flow scheme.

We selected WebMUX because it is a well documented standard with an availability of several reference implementations both in C and in Java programming languages. With the help of the WebMUX protocol it became possible in Shalon to multiplex many connections between the clients and the HS over a single tunnel to a contact point. We extended the HTTP proxy Muffin to include the *MUX* command. It instructs the proxy to run a WebMUX protocol on top of the existing connection, i.e., to alter the connection into a multiplexed connection. Therewith, several parallel streams can share the same anonymization tunnel. Incoming requests are managed

[2] `http://www.squid-cache.org/`

[3] `http://www.delegate.org/`

[4] Descriptor is a piece of data containing, among other things, contact information of a service.

internally, i.e., they are transparently delivered to the proxy server and treated as regular requests. Stream multiplexing allows to decrease the number of tunnels used in Shalon. This decreases the time needed for tunnel establishment and makes it difficult for an attacker to distinguish different streams. Hence, stream multiplexing increases the protection against several attacks such as, e.g., the website fingerprinting [20].

IV. Protocol for Hidden Services in Shalon

We propose to perform communication between a hidden service and a client as shown in Figure 1. Every anonymization tunnel is a normal onion encrypted Shalon tunnel consisting of at least two intermediate nodes.

First, the hidden server creates an anonymization tunnel and connects to one or several random nodes with hidden service functionality (i.e., nodes that support multiplexing and opening of listening ports). The HS asks for opening a listening port (1). If the node accepts the request, the connection is kept open until one of the nodes decides to tear it down; otherwise the hidden service tries other nodes until it succeeds. Next, the hidden service creates an anonymizing tunnel ending at the directory service (2) and asks it to publish the contact information of its hidden service. After this, the hidden service is ready to receive requests from clients.

We distinguish two types of clients: *anonymous* and *non-anonymous*. The difference between these two types of clients is that anonymous clients perform all their request through an anonymization tunnel whereas non-anonymous clients contact hidden services directly at contact points. Knowing the ID of the HS, clients connect to the directory service (3) and ask for the contact information of the identified service. This information contains, among other things, the addresses of contact points. There can be multiple contact points per hidden service. The clients select a random contact point and, finally, connect to the specified port of that contact point (4). Now the contact point starts to pass the data between the open tunnel from the hidden server and the client's connection.

As mentioned above, our design supports two types of clients: anonymous and non-anonymous. Non-anonymous clients can access hidden services completely without the Shalon software. To this end, we extended the proxy server Muffin and added a web interface with a simple query front-end for a distributed hash table (DHT) (network information in Shalon is stored in a DHT [17]). The front-end allows descriptor retrieval of both anonymization nodes and hidden services. Two formats are supported: *plain* and *formatted*. Plain descriptors are served in a machine-readable XML format and used for Shalon internal lookups. Formatted descriptors are human-readable and tailored for retrieval via a web browser. They are presented in the form of a web page with the contact information in the form of clickable links pointing to the contact points.

Additionally, we integrated the DHT used as a directory for hidden services in Muffin and extended it with a filter for detecting directory requests (i.e., telling them apart from anonymization requests) and redirecting them to the DHT.

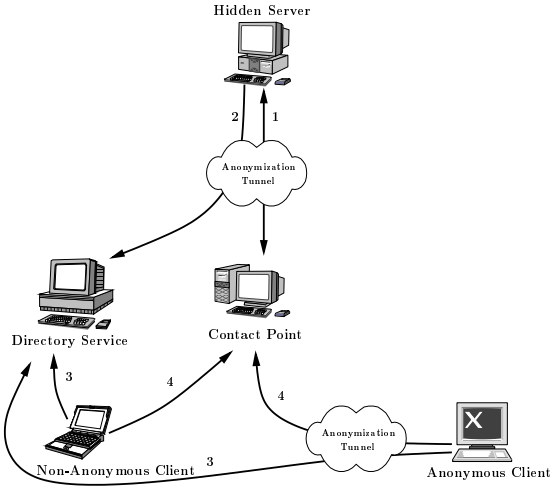

Fig. 1. Use of hidden services in Shalon

Therewith it is possible to transfer directory information *in-band*, i.e., on the same port as the anonymized communication. As a side effect, this feature makes it difficult to distinguish what kind of information is transferred over a tunnel (e.g., whether it is a DHT lookup or anonymization data), allows to make anonymous and encrypted DHT queries, and eases configurations to bypass firewalls (since only a single port has to be unblocked).

There are two modes of operation for hidden services in Shalon: (i) *TLS gateway mode* and (ii) *open/public mode*. In the former mode, the HS starts a TLS gateway at the hidden server node between the service and the tunnel to the contact point. The gateway is used to authenticate the service to the clients and to provide end-to-end encryption. To authenticate the service, the certificate of the gateway is signed with the private key of the service. As clients know the public key of the service from the descriptor, the signature of the certificate can be checked during the connection establishment. Herewith the clients can make sure that they communicate with the intended service and protect themselves from the man-in-the-middle (MITM) attack. Additionally, in this case the connection between the client and the service is end-to-end encrypted. In the open/public mode, on contrary, no end-to-end encryption is applied. Hence, the contact point can see the content of communication and, additionally, mount the MITM attack.

V. Design Implications

With our extension, Shalon is able to provide the hidden service functionality to any TCP based service. To this end, the Shalon client has to be configured to offer the HS by specifying the IP address and port (usually at the local host) where the service is running. Additionally, it is possible to specify the number of contact points. The ID of the service is the SHA-1 hash (160 bit) of the public key of the service. The URLs of hidden web servers consist of an ID (free of colons) ending with *.shalon* extension. The Shalon proxy (i.e.,

anonymization node) is capable to filter out these URLs, look up the contact information, and establish a connection to the HS. Recall that the descriptor of HS is stored under its ID in the DHT. Similarly to node descriptors in Shalon, descriptors for hidden services are stored and processed in the XML format. The descriptor includes addresses and ports of contact points, public key of the service, operation modes, etc. The descriptor is signed with the private key of the service. Hence, it is possible to check its integrity. Contrary to lookups for random nodes, IDs of hidden services are known in advance, thus, DHT lookups in the ID space of hidden services are not subject to attacks such as route capture and information leakage [17].

As mentioned above, our design supports two types of clients: anonymous and non-anonymous clients. Additionally, hidden services support TLS gateway mode and/or open mode. In case of an anonymous connection, the client creates a standard three-hop circuit to the contact point. If the HS is operated in the TLS gateway mode, the three-hop circuit can be used to directly connect to the contact point and to initiate the TLS handshake. Further communication with the HS is encrypted. In an open mode, it is recommended to extend the circuit to the contact point before connecting to the HS port. Otherwise, the last hop in the circuit and nodes on the path between the last hop and the contact point are able to see the content of the communication. To support the extension of the tunnel to the contact point, the descriptor of the HS includes the ID of the contact point so that the clients can find out the descriptor of the node acting as a contact point for the service.

Knowing the IP address of the contact point and the corresponding port, the hidden service can be contacted directly if it is offered in open mode. In case of TLS gateway mode, a TLS connection first has to be established. In case of a web server, simply the use of the HTTPS protocol allows to securely connect to the hidden service. Otherwise, a TLS wrapper is required before the clients can proceed with the application layer data. The wrapper communicates with the TLS gateway at the server side which transparently forwards data to the hidden service.

VI. Performance Evaluation

We evaluated the performance of our approach in a local network under controlled conditions as well as in the PlanetLab testbed [2] "in the wild". Furthermore, we compared the performance of our approach to provide hidden services to the approach of Tor. Every measurement was repeated 20 times to calculate the average value together with the 95% confidence intervals. For every throughput measurement, we transferred 10 MByte of data. To enable a fair comparison, we used the standard tunnel/circuit path length of three hops. Figures 2 – 5 shows the results of the evaluation and the comparison. The PlanetLab setup consisted of 400 nodes distributed all over the world. To measure the performance of Tor, we used the Puppettor Framework[5] that interacts with the Tor process with the help of the Tor Control Protocol. It allows to start several local Tor processes and to configure them "on the fly", e.g., to connect to a private or the real Tor network.

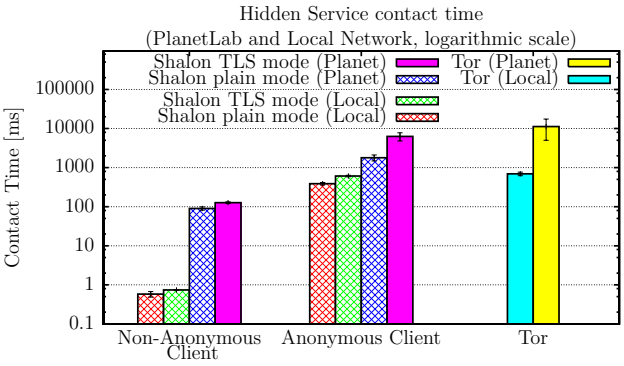

Fig. 2. Connection time in local network and in PlanetLab

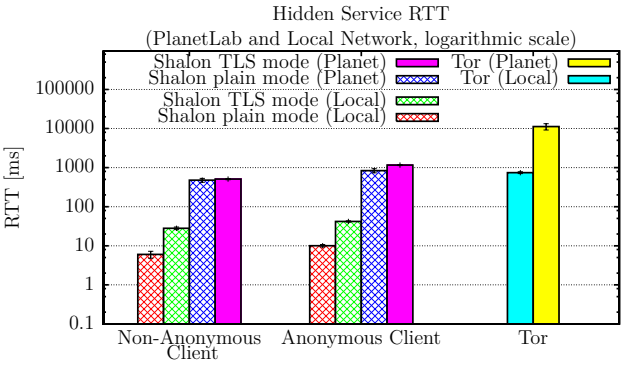

Fig. 3. RTT in local network and in PlanetLab

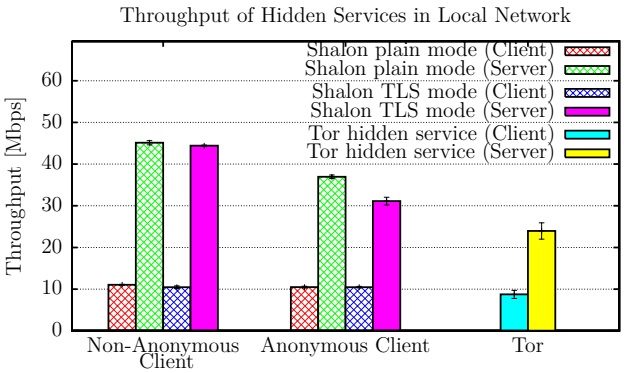

Fig. 4. Throughput in local network

The connection establishment time and round-trip times (RTTs) are plotted in a logarithmic scale. The same representation form is used for the throughput measurements in PlanetLab. The contact time is the time needed to establish a connection to a contact point. In case of a TLS gateway mode, this time also includes a TLS connection establishment. The throughput evaluation includes the throughput of a client as well as the throughput of a server. Since a client is the

[5]Available on GIT: git://git.torproject.org/git/puppetor

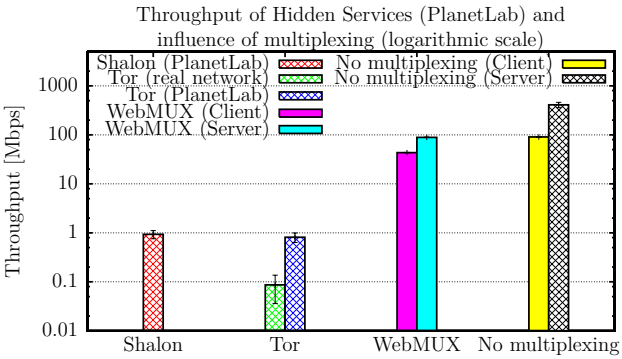

Fig. 5. Throughput in PlanetLab and influence of multiplexing

bottleneck in a throughput measurement, the performance of a server is significantly higher than that of single clients. In general, the results justify our expectations: a simpler protocol yields higher performance, i.e., significantly lower RTT, higher throughput, and lower connection establishment times. An interesting finding is that even in PlanetLab, hidden services in Shalon provide lower RTT than Tor in a local network.

Besides the throughput of hidden services in PlanetLab, Figure 5 shows the influence of multiplexing on the throughput. The measurement is performed between two nodes (one-hop tunnel) connected in a local Gigabit Ethernet network: directly and through the WebMUX server and the WebMUX client. As expected, the multiplexing negatively influences the throughput. The RTT of both direct and multiplexed connections was similar (about 4.8 ms on average).

We also evaluated the lookup times for HS descriptors. To this end we used the Mojito[6] implementation of Kademlia. The average time to find and fetch one descriptor in PlanetLab is about 8 seconds.

VII. Security Considerations

As all modern low-latency anonymization systems, Shalon and hidden services in Shalon do not have the goal to provide protection against a global attacker. This is made intentionally since, firstly, a global attacker is rather unrealistic and, secondly, every added piece of protection significantly reduces performance. We consider the same attacker model as Tor. Despite the simplicity of our approach, we claim that it provides a similar protection as the concept of hidden services in Tor – the state-of-the-art anonymization network used today. There are two major attacks to consider regarding hidden services: deanonymization of the service and the Denial of Service attack.

Deanonymization of hidden services is possible in similar situations as in Tor, i.e., when the adversary controls the first and the last node in the tunnel. In terms of hidden services, the last node simply corresponds to a client connecting to the HS. The first node in the tunnel may find out that it is serving a HS tunnel (e.g., by analyzing traffic patterns). But in order to reveal the service, it needs to go through all available hidden services and to establish a test connection to them (in order to match it to the pattern observed in the tunnel). However, since we apply a modified DHT where it is possible to lookup the value if and only if the exact ID is known, there is no efficient way to find out all existing hidden services in the system. Additionally, since we use contact points, there is no way for a client to force a HS to build new circuits. Contrary to this, every new connection from a client in Tor forces a HS to establish a circuit to a rendezvous point. Even though the exposure to this attack is already mitigated by the design of hidden services in Shalon, it can be completely eliminated when using guard nodes (cf. [14]).

To mount a DoS attack on hidden services, the attacker needs to attack the contact points. However, hidden services may react on failures of contact points and adapt to the situation, e.g., by selecting new contact points and/or by increasing their number. In case of a failure at a contact point, the hidden service first tries to reestablish the connection to the failed contact point. This prevents hidden services from modifying their descriptors because of contact points updates. To differentiate between the quality of service offered by different contact points, we propose to collect the statistics about the ratio of failures per time interval. This ratio is further used when deciding which contact point to select. Regardless of whether the list of contact points is updated or changed, hidden services are continuously available under the same ID.

Depending on the local legislation, operation of contact points can be a legal risk in some countries. Because the contact points are the public nodes for contacting hidden services, for an external observer it looks like these nodes themselves are offering the service. In order not to obligate all Shalon nodes to offer contact point functionality, we made this as a feature which can be specified by node operators. The nodes supporting contact point functionality are marked as such in their node descriptors. To this end we extended the format of Shalon descriptors. Moreover, the operators of contact points may specify that they support only hidden services offered in a TLS gateway mode. In this mode the operators of contact points do not necessarily know what content they are serving unless they explicitly connect to the service they are offering. Still, the situation can be additionally improved by obscuring the HS offered by contact points: it is possible to put a magic string (similar to cookies in HTTP) in descriptors of hidden services. Hidden services allow connections only from clients knowing the magic string. As contact points neither know the magic string nor the ID of the service they are offering in order to look up the descriptor and find out the magic string, they are not in the possession of information needed to connect to the HS. Hence, hidden services in Shalon can be offered in a mode that provides a plausible deniability for operators of contact points: in this operation mode contact points can in no trivial way find out which content they are serving (recall that without knowing the ID of the service it is not possible to efficiently find its descriptor).

[6]See `http://wiki.limewire.org/index.php?title=Mojito`

VIII. Related Works

Hidden services in the Tor network were deployed in 2004 [3]. They rely on rendezvous points that connect anonymous circuits originated 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 his 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. Contrary to Tor, our approach abandons the separation into introduction and rendezvous points in favor of contact points that provide dual functionality. This simplifies the design and contributes to a significantly higher performance. While Tor encrypts data on both the link layer and the circuit layer, Shalon encrypts on the tunnel layer (equivalent to circuit layer in Tor) alone. This leads to a significant performance increase as shown by the measurements [18]. According to Loesing et al. [13], the average time before sending out the request and getting the answer from a hidden service in Tor is 24 seconds. Moreover, the study revealed that most of the time is spent on connection establishment to rendezvous and introduction points. Clearly, most users do not tolerate such a high latency. According to a study of Wendolsky et al. [22], most users are eager to wait up to 4 seconds while requesting a website. Practical proposals to decrease the latency include cannibalization of existing circuits (extending them to introduction/rendezvous points instead of creating new circuits from scratch) and connecting to several introduction points in parallel and retaining the first successful connection while tearing down the other connections.

Øverlier and Syverson [9] propose an approach to increase the resistance of the hidden service design in Tor against DDoS attacks and to hide the existence of the hidden services from everyone (in particular from directory service) except the users knowing its service address. Unfortunately, this design leads to further performance penalties. E.g., to protect introduction points, authors suggest to use the so-called *valet nodes* between the introduction points and the clients. This leads to an even higher number of nodes in the tunnel to the service. Additionally, the authors propose the use of cryptographic tickets to authenticate clients. The improved protocol also allows differentiation of clients, e.g., for different quality of service classes.

In their follow-up publication [10], Øverlier and Syverson propose a simplified design for hidden services. The idea is to combine the introduction and rendezvous point in a single node and therewith to reduce the required number of nodes. The client connects either directly through a valet node to the hidden service, or the hidden service creates a tunnel to the node before the valet node in the client's tunnel which is then used as a rendezvous point. The latter is done to ease the load on the circuits from hidden services to valet nodes.

The probably most famous work in the area of hidden services is also done by Øverlier and Syverson. It is about locating hidden services in Tor [14]. Even though the idea is based on the well known fact that controlling the first and last node of a circuit leads to deanonymization, the authors show the effectiveness of this attack in the scope of hidden services. A client initiates as many new connections to a hidden service as needed until on one of the circuits from the hidden service to a rendezvous point a malicious node is selected as the first (entry) node. The attack can be eliminated using guard nodes.

Besides hidden services in Tor, there are similar concepts in other networks, e.g., the *eepSite* concept in I2P [6] and *Freesites* in the Freenet network [7]. 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.

Secure Overlay Services [8] is an architecture that specifically designed to resist DoS attacks. However, this approach is applicable for classes of communication where both participants are known to each other (i.e., have some form of pre-established trust relationship). This clearly does not apply to our scenario where the identity of communicating parties has to be hidden.

IX. Discussion

One of the main differences of our approach to hidden services in Tor is that we abandon the separation into introduction and rendezvous points in favor of contact points that unify these two functionalities. This makes attacks like [14] not possible in Shalon. However, the security and performance implications (because of possible workload on the contact points) of this approach have to be studied more deeply. An additional downside of our approach is that in the current version the HS cannot perform access control directly on the contact point.

Contrary to Tor, we also refrain from the use of centralized directories in favor of modified distributed hash tables. Hence, it is not possible to efficiently list all the hidden services. The attacker cannot easily find out the contact points responsible for a HS if the ID of the HS is not known. This is done in a natural way without using additional cryptographic techniques as proposed in [9].

Another major difference to hidden services in Tor is that we support two types of clients: anonymous and non-anonymous. Non-anonymous clients do not have to install anonymization or any additional software to access the hidden services. However, they still need to find out the addresses of contact points. To this end we provide a web-based DHT front-end at every Shalon node. Connecting to Muffin on the Shalon port with a browser, users knowing the ID of a hidden service can request its contact information through a web form. The IDs of hidden services can be distributed in a variety of ways: e.g., through a hidden wiki as it is done in the Tor network, through mailing lists, etc. Also private IDs are possible, e.g., via encrypted e-mails or through an offline distribution.

Similarly to Tor, our approach provides the application transparency, i.e., there is no need to modify services itself, any TCP-based service can be offered as a Shalon HS.

In terms of scalability, we estimate our design regarding the process of HS traffic anonymization to be in the same scale as in Tor. However, similarly as in Shalon [18], our approach for HS requires less cryptographic operations (recall that Tor encrypts everything twice: there is a TLS layer between the nodes as well as cryptography on the circuit layer between the client and corresponding ORs). Therefore, we expect HS in Shalon to be able to serve more clients with the same amount of CPU power. Alternatively, given the same number of users, more CPU power per user should be available for HS in Shalon (the CPU saturation is suspected to be the limiting factor for most Tor nodes [16]).

We also considered other alternatives as an underlying tunneling protocol for Shalon. One of them was SOCKS. SOCKS is a byte level protocol with an attractive feature – the BIND command which opens a listening port on a remote machine. This can be useful to provide support for hidden services. However, only a single connection can be accepted on the opened port. Furthermore, according to the SOCKS specification [12], [11], an opened port should only be used for connections from the hosts a client already has an existing connection to (e.g., a second stream for a FTP connection). Thus, misusing this feature for hidden services would not be conform to the standard usage of SOCKS. Another possibility would be to use *SSH (Secure SHell)* tunneling and multiplexing. SSH is an application layer protocol for a secure login to a remote computer [23]. It supports the so-called *port forwarding*, which allows to open a listening port on a local machine and to forward all connections to this port via the SSH connection to a server side and then to a specified host and port. This forwarding supports multiplexing of multiple connections via a single TCP connection using the SSH integrated session management. The integration of this protocol in Shalon as well as its adaption for our scenario (e.g., disabling of authentication and encryption) would again require modification of a complex protocol in order to make it suitable for our needs. Beyond doubt these are not the only alternative protocols Shalon potentially could use.

A similar situation arises with the multiplexing protocol. To provide hidden services efficiently, Shalon needs a protocol that allows for tunneling multiple concurrent, interleaved streams over a single TCP connection. We applied the WebMUX protocol to fulfill this goal. However, possibly in the future the SPDY[7] protocol would be a better alternative for this goal. The SPDY protocol is an alternative method for transferring web content over TCP, designed to improve efficiency and performance. It is a transport layer for HTTP with multiplexing functionalities. As most desired kind of traffic in anonymization networks is HTTP, we find it challenging to use the multiplexing protocol designed to improve efficiency and performance of HTTP traffic.

Because of the integration of the Muffin proxy and DHT in Shalon, there is no need for any external software that needs to be configured and started separately. Additionally, it allows for "in-band" network information distribution and multiplexing of multiple streams over a single TCP connection. Non-anonymous clients of hidden services do not need any additional software to access the services. To the best of our knowledge, this is the only existing approach for hidden services that offers this feature. As our performance evaluations show, our approach outperforms Tor hidden services regarding all the studied performance metrics. For instance, the HS setup and contact time is by the factor of 2 faster than in Tor, the RTT is up to the factor 18 lower than in Tor.

Our future work will include design and evaluation of possibilities to offer several descriptors for the same HS (e.g., using the conjunction of IDs with salt values) with different contact points and quality of service to serve different groups of users. Additionally, we plan to provide tunnel multiplexing between servers. So far only streams of a single client are multiplexed in a single TCP connection. Of a great interest would be to multiplex connections from different clients within a single TCP connection between two nodes. This would reduce the number of required file descriptors and obscure client tunnels from an external observer.

X. Summary and Conclusion

In this paper, we proposed how Shalon – a lightweight anonymization protocol based on open standards – can be extended to offer hidden service functionality to its users. Hidden services preserve anonymity for a service provider, can be used to protect against distributed DoS attacks, and help to overcome censorship. All in all, we achieve similar properties as hidden services in Tor, but in a simpler and more elegant way, reusing existing protocols and software. The evaluations show that our design provides similar level of protection as hidden services in Tor but at the same time offers significantly higher performance, e.g., the RTT of our scheme is up to the factor 18 lower than in Tor. The key feature of our approach is the fact that hidden services in Shalon can be accessed without installing any additional software, which is not possible when using hidden services in Tor.

Acknowledgements

Parts of this work have been funded by the National Research Found (FNR) of Luxembourg within the CORE grant project MOVE and by the EU FP7 IP OUTSMART.

References

[1] Muffin: An open-source cross-platform filtering Web proxy server written in Java. `http://muffin.doit.org`.

[2] 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.*, 33(3):3–12, 2003.

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

[4] K. Evans, J. Klein, J. Lyon, and S. Spero. *Session Control Protocol V2.0*.

[5] J. Gettys and H. F. Nielsen. The WebMUX Protocol. Internet Engineering Task Force: Internet-Draft, Aug. 1998.

[6] I2P project. http://www.i2p.net/, 2009. Visited May 2009.

[7] See `http://dev.chromium.org/spdy/`

[7] 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*, pages 46–66, July 2000.

[8] A. D. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure Overlay Services. In *In Proceedings of ACM SIGCOMM*, pages 61–72, 2002.

[9] Lasse Øverlier and P. Syverson. Valet Services: Improving Hidden Servers with a Personal Touch. In G. Danezis and P. Golle, editors, *Proceedings of the Sixth Workshop on Privacy Enhancing Technologies (PET 2006)*, pages 223–244, Cambridge, UK, June 2006. Springer.

[10] Lasse Øverlier and P. Syverson. Improving Efficiency and Simplicity of Tor circuit establishment and hidden services. In N. Borisov and P. Golle, editors, *Proceedings of the Seventh Workshop on Privacy Enhancing Technologies (PET 2007)*, Ottawa, Canada, June 2007. Springer.

[11] Y.-D. Lee. *SOCKS 4A: A Simple Extension to SOCKS 4 Protocol*.

[12] M. Leech, M. Ganis, and Y.-D. Lee. SOCKS Protocol Version 5. RFC 1928, Network Working Group, March 1996.

[13] 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, July 2008. IEEE CS Press.

[14] L. Øverlier and P. Syverson. Locating Hidden Servers. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*. IEEE CS, May 2006.

[15] P. Palfrader. Number of Running Tor Routers. http://www.noreply.org/tor-running-routers/.

[16] 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, March 2008. IEEE Computer Society Press.

[17] A. Panchenko, S. Richter, and A. Rache. NISAN: Network Information Service for Anonymization Networks. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (ACM CCS 2009)*, Chicago, Illinois, USA, November 2009. ACM Press.

[18] 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.

[19] S. Spero. *SCP - Session Control Protocol V1.1*.

[20] Q. Sun, D. R. Simon, Y.-M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu. Statistical Identification of Encrypted Web Browsing Traffic. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy (IEEE S&P)*, pages 19–30. IEEE Computer Society Press, May 2002.

[21] Tor Network Status. https://torstatus.kgprog.com/.

[22] R. Wendolsky, D. Herrmann, and H. Federrath. Performance Comparison of low-latency Anonymisation Services from a User Perspective. In N. Borisov and P. Golle, editors, *Proceedings of the Seventh Workshop on Privacy Enhancing Technologies (PET 2007)*, volume 4776 of *Lecture Notes in Computer Science*, pages 233–253. Springer-Verlag, June 2007.

[23] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Connection Protocol. Internet Engineering Task Force: RFC 4254, Jan. 2006.