Protecting Privacy with Protocol Stack Virtualization

Janne Lindqvist and Juha-Matti Tapio

Helsinki University of Technology

Abstract. Existing approaches for providing privacy without infrastructure have proposed to use pseudorandom or disposable identifiers on some or all layers of the protocol stack. These approaches either require changes to all hosts participating in the communication or do not provide privacy for the whole protocol stack. An alternative approach for providing privacy could be to implement an operating system policy for controlling the privacy for all possible network applications, but this might take considerable effort, and cannot easily take into account different preferences of users. Building on previous work, we propose a relatively simple approach: *protocol stack virtualization*. For example, every application that is connected to the network receives a distinct identifier space from the available spaces on all layers of the protocol stack. This approach does not need any infrastructure support from the network and requires only minor changes to the single host that implements the privacy protection mechanism. We have implemented the approach as a user space daemon and have tested it with a number of legacy applications such as Skype, FTP, Firefox web browser and a video streaming application to prove that the approach does not require any changes to existing applications.

1 Introduction

In this paper, we consider the privacy of mobile computer users. Often, the most effective way to protect one's privacy is to remain anonymous, i.e., not to reveal one's name or other identifiers at all. When identifiers are absolutely necessary, such as in maintaining communications sessions and seamless mobility, pseudonyms can be used instead. This paper presents a new technique for implementing privacy for mobile Internet hosts: *protocol stack virtualization*.

Quite a bit of research has been done on anonymizing TCP/IP communication. The protection mechanisms vary greatly depending on the location and capabilities of the assumed adversary. There are, however, some common themes. One is that the defender avoids giving out identifiers, either explicit or implicit, which could be used to recognize it and to trace its actions over time and over different locations. The attacker, on the other hand, tries to use all available clues, such as forgotten identifier fields, protocol metadata, payload data and statistical profiling, to link together multiple appearances and actions of an individual user or computer. Another common theme is that the defender hides among other users, i.e., blends into a crowd. Indeed, anonymity is often measured as the size of the set of users among which one is indistinguishable (the anonymity set). This makes it, by definition, impossible to achieve anonymity alone.

Based on the above, it becomes clear that the complexity of modern software will cause problems for anonymity protection. The multitude of networking protocols and network-enabled applications found on a typical computer will make it easy to fingerprint the computer based on its individual software configuration and overall communication patterns, even if each protocol and application in itself has been carefully anonymized. Additionally, there will be conflicting requirements from the various protocols regarding the lifetime of pseudonyms and when they are replaced. Yet another issue is that wireless network interfaces will be continuously enabled, rather than switched on sporadically by the user, to meet the needs of the various applications. This may increase the number of occasions when there are not enough mobile hosts at the same network to provide the safety of a crowd.

Our main idea is to create a seemingly separate instance of the protocol stack for each application. The increased isolation between applications creates two kinds of benefits. First, each application will have a distinct set of identifiers for the link, network and transport layers. This enables us to deploy previously proposed privacy-protection mechanisms, such as address randomization, on per-application basis, without trying to configure one protocol stack that meets the peculiar requirements of all the applications. Second, the unique identity of the user (or computer) will be split so that it appears as a group of unrelated anonymous or pseudonymous entities, each of which is typically involved only in a single communication session with a single peer. This makes some of the profiling attacks more difficult to implement because, in order to profile the user or computer, the attacker needs to first establish which sessions belong to the same computer.

We have implemented the approach as a host-based network address translator (NAT) for Linux, and tested it with number of different real applications and low-end network infrastructure elements to show the feasibility and deployability of the approach. In the implementation, we borrow ideas from virtual NAT [34], a proposal to create a new IP address for every TCP connection [5] and transient addressing (TARP) [13].

The rest of the paper is organized as follows. In the next section we state the assumptions and requirements for the solution. Section 3 describes the design and implementation details and results are given in section 4. Section 5 gives related work. In section 6 we discuss the limitations and benefits of our approach and conclude the paper in section 7.

2 Problem Space, Assumptions and Requirements

Essentially, we do not attempt to provide anonymity, instead, we have taken a pessimistic view on the privacy problem. Our position is that eventually the user or the host operating system does something that will reveal the identity of the user and we desire to limit the consequences of this unavoidable circumstance. In other words the problem we address in this paper, is the preventing linkability of user actions, operating system and the applications. Even though some of the networked applications may take privacy into account, even *when* one application reveals the identity of the user, other network traffic originating from the same host can be linked to the user. Even though the other traffic may be encrypted (and in many cases it is not), a third party can gain useful information if it can know the identities of the communication parties even if the attacker cannot see the content of the communication.

We assume that the attacker can reside either in the local wireless or wired network or further away, one or more hops. The main focus of our work is to protect from a local attacker, for example, in a WLAN network, but at the same time, the approach protects the user from more distant attackers, too. The goal of the attacker is first to identify the user as distinctively as possible. The second goal is to gather other information related to the habits and network usage, for example, where does the user work, who he or she contacts.

Because privacy does not sell if it cannot be understood, a privacy protection mechanism should be very easy to deploy and use. Thus, we need to minimize the disruptions to the user experience as far as possible. To help deployability, we do not assume any infrastructure from the network to participate in the privacy protection mechanism. Also, for deployability reasons, we do not want to introduce a burden to the peers that a host communicates, since many service providers do not have incentives to protect the privacy of the user. Thus, the solution space is limited to approaches that provide unlinkability by modifying only a single host, the host that wishes to protect the privacy of the user.

Finally, one problem that is overlooked in the design of many privacy architectures: the desire to remain *reachable*. Although users appreciate the fact that their privacy is protected, they also desire to be reachable for example by their friends. These conflicting requirements need also be addressed by a privacy protection architecture. We summarize in the following list the requirements and assumptions for our approach.

- To provide *deployability* we
  - do *not assume* any infrastructure.
  - change *only* a *single operating system*.
- The approach should not change the *user experience*, for example,
  - require decision making
  - or learning from the user.
- The approach should allow the users to remain *reachable*.

3 Design and Implementation

The basis of the design is to use virtual randomized MAC addresses and new IP addresses for every flow or application. However, although the basis of the system already protects a lot regarding casual observers on the local link, we need more complexity to cover trivial fingerprinting attacks and avoid breaking interdependent protocols, such as ICMP error messages. The system consists of following:

- MAC address randomizer
- IP address autoconfiguration
- system introspection
- protocol helpers
- inbound packet filtering

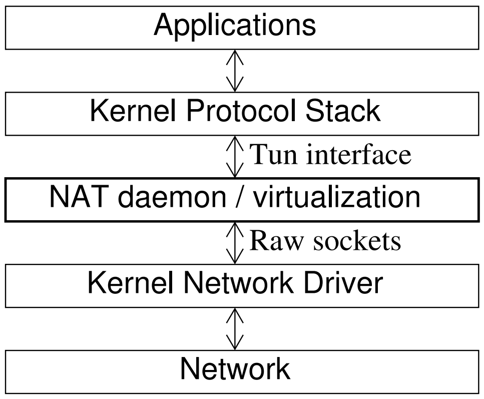

Fig. 1. Top-level implementation architecture

For every MAC address used, the system must obtain a new IP address. With IPv4, we can use DHCP, and with IPv6, the stateless address autoconfiguration [35] or DHCPv6. Although we later argue that given the scarcity of IPv4 address space the approach is more practical for wide deployment with IPv6, we implemented the protocol stack virtualization for IPv4 in order to test it with a number of legacy applications available for IPv4. Nevertheless, based on our preliminary IPv6 implementation work we do not perceive that extending the work for IPv6 could introduce problems with IPv6 legacy applications.

System introspection and protocol helpers are needed to guarantee the functions of some legacy applications. For example, ICMP error messages need to be mapped to the connections that are the reason for the messages. Further, to constrain the use of the address space, we need to map multiple different connections to a single application. We also need to filter some inbound connections to mitigate active attacks such as port scans that could reveal the use of protocol stack virtualization.

3.1 Implementation details

In essence, the implemented approach is a host-based network address translator (NAT) that instead of traditional NATs [33] provides a set of distinct identifiers for every network flow originating from the host. In other words, every instance of an application is given a new IP address and virtual MAC address, and flow identifiers in transport protocols.

To experiment with protocol stack virtualization we implemented a proof-of-concept daemon for Linux. The host-based NAT daemon is logically placed between the operating system's regular protocol stack and the network driver as shown in Figure 1. To simplify prototyping, we implemented the daemon with Python in user space. The daemon uses a tun-device to communicate with the protocol stack and raw sockets to communicate with the network driver.

The flow of the data packets within the daemon is described in Figure 2. All inbound and outbound packets are basically routed through the daemon. To pass packets for the protocol stack virtualization code in the daemon, a special tun-device is set up. The tun-device is configured with a special IP-address that is visible only in the operating system and the system's default route is set to that interface. Thus the kernel will by default route outgoing data packets to that interface and thus pass them to our userspace process. The NAT daemon reads those packets from the device and deconstructs them to modify the headers.

To map the outgoing packet to a new source IP address, we consider the IP protocol of the packet. For packets not associated with an already known connection we choose randomly the source IP address from an available pool and an unique MAC address for that IP address. In our tests we used a pool of 100 IP addresses that mapped one-to-one to MAC addresses chosen randomly from a pool of 2^{23} addresses.

Connection tracking is required for TCP and various other protocols and therefore the NAT daemon needs to use the same source address for packets related to the connection. A connection is mapped to a specific source address with a key formed by inspecting the packet. For TCP we use a triple ('tcp', destination address, destination port) as the key to identify a connection. We considered including source port in the key, but we observed that web browsers typically use multiple tcp connections from various source ports to download pages and images and using multiple source addresses for downloading content from one web server would be wasteful.

Because UDP applications typically use one UDP port to communicate with multiple destinations, there is no need to associate multiple local ports with one connection. Thus we consider it reasonable to use a 4-tuple ('udp', destination address, source port, destination port) as the key for UDP connections instead of the triple that we use for TCP.

For ICMP we use by default only a tuple of ('icmp', destination address) as the key, but this does not cover all the use cases. ICMP error messages have to be inspected to instead associate them with the connection that caused them. Otherwise messages such as *port prohibited*, *time exceeded* and *MTU discovery* would get sent from the local host to peers using random source address. This would cause the peer to discard the packets and needlessly reveal the use of virtualization. All other protocols will be mapped to a default source address so that unsupported protocols will continue to work, assuming they can survive NAT.

In addition to the connection categorization desribed above, we improve connection tracking and grouping by two configurable mechanisms: system introspection and protocol helpers. For TCP and UDP it is possible to look up which local user id or application did cause a specific connection. We call this process system introspection because we analyze the Linux kernel network structures published in the `/proc`-filesystem. By scanning `/proc/net/tcp` and `/proc/net/udp` we can produce the user id and inode of the socket that matches the packet being analyzed. To further find which applications have that socket open, we scan the `/proc/<pid>/fd`-directories to look for links to the open socket. For example if we have Skype running, we can find out if a certain TCP or UDP packet belongs to a connection used by the Skype process. Depending on configuration, we use this information to group all Skype data packets into one logical connection and only use one source address for all Skype packets.

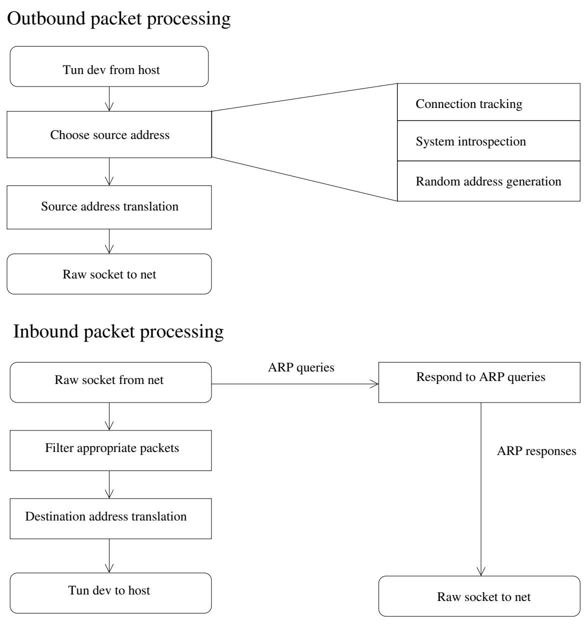

Fig. 2. Packet flow within the NAT daemon

We can also augment connection tracking by protocol helpers that detect and group multi-flow connections such as active FTP or SIP. The helpers can be used in the same manner as typical NAT helper modules currently in use. They can both detect multiple related TCP and UDP flows and perform packet rewriting in cases where the local application reports the locally visible IP address to the peers.

Since our prototype runs in user space and it therefore introduces some extra latency, we chose to send gratuitous ARP reply's before the outbound data packets for the IP and MAC addresses we use. These ARP packets may cause some router's to remember the MAC and IP addresses for a while and thus avoid needing to wait for extra ARP queries. Not all routers accept these messages and we do not expect them to be generally useful for in-kernel or realtime implementations.

The NAT daemon performs a straightforward network address translation on the outbound packets and replaces the source address using the above described method. In addition to modifying the source address in the IP header, the IP checksum has to be recalculated. Because the TCP and UDP headers use the IP source address in their checksums, those have to be recalculated aswell. The modified packet is sent to the next router via a raw socket using the chosen virtual MAC address as the hardware source address.

The physical network interface needs to be set to promiscuous mode so that the network interface card will also listen to packets that are destined to the virtual MAC addresses. The NAT daemon reads captured inbound packets from a raw socket. The packet is further processed if it is either an ARP request for one of the virtualized IP addresses, or it is destined to one of the virtualized addresses. For packets recognized as belonging to some existing connection the daemon performs reverse translation for the destination IP address mapping it back to the special internal address.

If we accept all inbound packets for all the addresses that we have in use, the adversary performing a port scan of the IP address pool would get identical results for all the addresses, and thus, we could lose unlinkability. We can protect the unlinkability of the addresses if we accept incoming connections on only one address. Thus, we can drop inbound TCP and UDP packets that are not associated with an existing connection if they are not destined to our default source address. The limitation here is that all protocols that use inbound connections need protocol helpers or they have to use the one inbound address.

4 Results

In this section, we give the results on testing our implementation on real applications and show that even low-end switches are not a hindrance for deploying the implemented system.

4.1 Application tests

We tested the NAT daemon with various Linux applications to evaluate the feasibility of the idea. Tests were run in a typical home ethernet environment. The client running Linux had a 2 GHz AMD Athlon CPU and a VIA Rhine network interface card. The traffic was routed through an A-Link ADSL router. Generated traffic was captured and further analyzed.

We chose *ping, traceroute, ftp, Firefox web browser, Skype, OpenVPN* and *VideoLAN* clients as test applications to provide variety. Ping and traceroute performed as expected, communication with each end host was done with a different source address.

As was to be expected, ftp did not work until passive mode was enabled. In active mode ftp asks the remote end to open a connection to the local IP address, but ftp reports the internal interface address which does not work. It should however be noted that this type of problem is not specific to protocol stack virtualization and the exists with all protocols that open multiple connections to addresses reported by the hosts themselves. As with all NAT-systems, this problem can be fixed with a special protocol helper module. However, we do not perceive as a problem that active NAT does not work without a protocol helper module, because e.g. most Web browsers default to passive FTP today.

Firefox worked well. In our tests we saw no differences to regular browsing. Though it should be noted that a balance needs to be chosen with the connection tracking. Web browsing can open multiple outbound connections to the same server. If these connections are all virtualized separately, IP addresses may be wasted. However, if all web requests to a specific server are made from the same source address, unlinkability might suffer if the user visits multiple independent web sites that are hosted on the same destination IP address.

Skype is an interesting case due to its P2P architecture. It seems to work well on top of protocol stack virtualization. In our tests Skype was very conservative in its use if UDP ports, but it did use a lot of TCP connections to various nodes around the world. In our test run Skype quickly used approximately 15 virtualized addresses.

OpenVPN is a popular easily deployed SSL-based VPN system that runs on top of either TCP or UDP. It worked well on top of transport types. A live TV simulcast with a 300-400 kbps MMSH-feed also worked flawlessly with the VideoLAN client. We noticed nothing unusual with these applications running on top of protocol stack virtualization.

Overall in testing sessions when not using the introspection with the applications mentioned above the applications created approximately 34 TCP connections which were mapped to 19 IP addresses. The biggest consumer of IP addresses in our testing was Skype. Browsing one site with a browser typically uses up one IP address, a few more if some content is fetched from separate servers. As for UDP connections, there were respectively 87 conversations between UDP endpoints (IP addr and port). Almost all of them were Skype related. These UDP connections used up 55 virtualized IP addresses.

We would however point out that these connection counts were from tests without any protocol helpers and without using introspection to group connections on an application basis. Therefore it is possible to significantly reduce address usage by using a configuration where some applications use only one address. We estimate that some amount of unlinkability would be realistic with even a few public IPv4 addresses obtained for example by DHCP.

The most computationally demanding operation in the NAT daemon is the optional local system introspection to find out the user id or the application. Therefore we decided to test it separately. On the same above mentioned desktop system we tested how long it takes to lookup the responsible user id or the application. The system had 117 processes running and the tested process was near the end of the process list. Thus the figures approximately represent the worse case scenario regarding the lookups on similar desktop configurations. We performed the tests with Python's timeit-framework with 1000 rounds. For user id lookup the average lookup time was 0.5 microseconds and for user id and application the lookup took on average 14 milliseconds. We estimate that the additional delay caused by the introspection lookups is low enough for them to be useful. We would like to note that due to connection tracking, it should be enough to do the lookups only for the first packets of each connection. This can further be optimized by for example skipping application lookup for certain well known ports such as 80 (http).

Based on these results we estimate that a simple user session consisting of browsing a few sites and using some basic network applications (P2P applications excluded) would need tens of virtualized IP addresses with protocol stack virtualization. Use of P2P applications would raise the need for addresses significantly unless the protocol-specific helper modules or introspection are used to aggregate together all the connections belonging to one specific application.

Additionally we measured the extra latency caused by the user space implementation by comparing plain ICMP ECHO "pings" within one Ethernet segment with and without our protocol stack virtualization implementation. With virtualization we measured an average round trip time of 1 ms (100 packets) while without virtualization we measured an average of 0.2 ms. This test did not include any introspection lookups. Thus we had approximately one millisecond of extra delay caused by our implementation. We consider this delay reasonable and not prohibitive with regard to good scalability, and we believe that an in-kernel implementation would remove the introduced delay.

4.2 Network switch test

We tested the feasibility of using multiple virtualized hosts in a switched network. Potentially, careless use of MAC and IP addresses could result in problems. We implemented a separate program that rapidly sends packets with different MAC addresses, and a responder echoes these packets. A third machine attached to the same switch was used to observe the network, for example, when packets are broadcasted.

We tested a low-end WLAN base station Linksys WRT54GL to find does the virtualization approach affect the efficiency of switching. The device started gradually broadcasting some of the packets after reaching a hundred entries in the arp table. At approximately 4000 entries, everything was broadcasted. Packet loss, however, did not occur in the test setting.

Another interesting observation was that the default arp cache size in Linux (2.6.15 kernel) is 1020. When the limit is exceeded, the sendto call of Linux reports an ENOBUF error, and prevents sending of more UDP datagrams. The limit can naturally be increased.

These simple measurements already show that the virtualization approach does not affect the accessibility of the access network, even though there would be multiple clients using the approach at the same time with coarse granularity of virtualization.

5 Related Work

First, we briefly describe how information on different layers on the protocol stack can be used to privacy violations, depending on the location of the attacker. We start with the means available to a local adversary. Note that these means may not identify the user directly, but can neverthless be used to pinpoint distinct users if out-of-band means are used. The list is not exhaustive, but exemplifies the many ways a host and consequently the user can be identified. Starting from the lower layers of the protocol stack, the adversary has the possiblity to fingerprint the devices with analog signals e.g. in the Ethernet [12] or e.g. by the WLAN device driver behavior [10]. The MAC layer and especially the MAC address can be used to identify the host [16]. Even implicit identifiers such as broadcast packet sizes and SSIDs of the wireless networks that the device has attached to can be used to identify the user [27]. And numerous explicit identifiers are leaked from enterprise domain controlled Windows laptops, such as, name server lookups, LDAP queries or printer configurations [4]. The following means are applicable also to adversaries that are not local, that is, not in the range of the wireless or in the wired local area network, but for example, one router hop away. The IP address, TCP sequence numbers, IPsec ESP SPIs can be used to identify the host [3], or even the clock skew of TCP or ICMP time stamps can be used [21, 24]. Also, if the authentication and key exchange protocol is not designed carefully, the use of encryption techniques can also be harmful for privacy [1, 2, 4]. And even dynamic DNS can be used to identify and track the user remotely [17]. Finally, the data that applications transmit can be used for correlation purposes. As a concrete example, only the login process is usually encrypted with a Web based email service such as Google or Hotmail, and even though TLS-protected IMAP is common, encrypted connections for outgoing SMTP are not. Additionally, services such as Google Accounts and Amazon, provide clear-text information about the identity of the user for a third party to observe.

Based on the attack vectors described above, we observe that against local adversaries, the use of anonymous identifiers in upper level protocols is not enough for location privacy or identity protection. The host needs to invoke only one application that uses a public identifier and the anonymity of the anonymous protocol can be violated. For example, privacy extensions [26] for IPv6 stateless address autoconfiguration [35] or SIP [31] privacy service [28] do not help against a local adversary, if it can correlate the traffic with the MAC address or the identifiers in the application protocols.

Although we have named our architecture protocol stack virtualization, it is different from full virtual machine monitors such as VMware [36], Terra [11], Crossbow [9] or even from network virtualization optimizations [23] provided by Xen [6]. These environments do provide the possibility to virtualize the whole protocol stack, but their design purpose is not unlinkability of network traffic, instead they provide the possibility to virtualize full operating systems for execution environment isolation. Additionally, the computational and memory overhead of our solution compared to a full virtualization environment is significantly less, and our approach does not require anything new for the user to learn. Also, PlanetLab [29] allocates resources (slices) on a network of hosts spread around the world. Every application deployed in the PlanetLab requires a slice which corresponds a collection of virtual machines on different hosts around the world. The stack virtualization approach we have proposed shares the concept in a sense, however, we provide a virtual instantation of the protocol stack for applications in a single operating system.

Previous approaches [3, 22] that do not rely on infrastructure for location and identity privacy have proposed to use pseudorandom or disposable identifiers all layers of the protocol stack. However, in these approaches the identifier spaces are shared by the applications, and therefore the traffic can be linked to the host by observing the applications. Similar observations can be made from approaches that change only the MAC address [16] or rely on the network to assign pseudorandom MAC address [19]. However, some of the approaches [3, 22] take into account maintaining end-to-end connectivity when changing network attachment, but require changes to both commicating parties. We have, however, left this kind of mobility management beyond the scope of our work.

Sender and receiver anonymity, and thus, identity privacy, can be achieved with many approaches that rely on a infrastructure such as Chaum's MIXes [7], onion routing [15], Crowds [30], the second generation onion router (Tor) [8], or with information slicing on unreliable overlays [20]. These approaches do provide unlinkability of the traffic that is routed through the infrastructure. However, if every flow originating from the host is not routed through the infrastructure, the user is vulnerable to correlation attacks at least in the local network. We do consider these approaches complementary in the sense that if the user uses, for example, Tor, to provide end-to-end sender and receiver anonymity, with our approach, the unprotected traffic cannot be linked to the Tor traffic. We further discuss and benchmark our approach to Tor in the next section.

There are theoretical models that consider information flow properties such as non-interference [14]. In the information flow control field, there are also practical models and their implementations, for example, on how mutually distrusting can express their privacy policies and the system enforces the requirements simultaneously [25]. “Privacy as an operating system service” proposes that the operating system includes a privacy module for scrubbing sensitive data [18]. As a concrete example, the operating system should automatically remove data such as names and social security numbers, in addition to meta-data and “para-data” such as the author’s username from sensitive Word documents. However, in our approach, we are concerned with identifying information on all layers of the protocol stack that could compromise the identity and location privacy of the user. In our view, these approaches are complementary, we agree that a complete privacy solution should actually take into consideration also the (meta|para)data that is sent to the network by the applications. Similarly to above, TightLip [37] provides a way to handle access control decisions when user has defined what is sensitive data and who is trusted. For every process handling sensitive data, the TightLip system launches a doppelganger process. The operating system follows these processes, and if they make system calls with different arguments, it is concluded that the original process might be divulging sensitive data. Also, another Doppelganger [32] provides a user-friendly way to manage cookies. The approach protects from useless use of privacy-harming cookies, but does not help with the personalization problem of the Google Accounts or Amazon mentioned in the introduction. Further, the approach is limited only to Web cookies.

In “Privacy, Control and Internet Mobility” [5], the authors make a short note that for privacy protection purposes in mobility or roaming solutions such as Mobile IP, applications should be more aware of mobility and the protocol stack be more aware of application privacy requirements. However, it does not suffice that applications are aware of what happens in the operating system or the protocol stack since they are not aware of the privacy requirements of the system. As mentioned in the introduction, the authors also propose that with IPv6, every new TCP connection that is initiated from the host. The authors have not implemented the approach, and do not consider local attackers, their idea was to provide equivalent privacy for IPv6 that NATs provide for IPv4.

VNAT [34] provides process migration with connection virtualization. The idea is that the first connection a process makes to another process on the network remains as the fixed IP address pair for the connection. When the process is moved to another host, the connection can be recovered with an update mechanism. VNAT provides address translation for these virtual addresses, however, the approach requires changes to both peers, is not designed with privacy in mind, and even though the authors use the term “physical address” to denote the current IP address of the host to add, they do not virtualize the MAC addresses.

Finally, TARP [13] proposes to use multiple addresses per host for improved firewalling. The authors also use the virtual machine methaphor: “each client process group conceptually runs in a virtual machine, with an independent IP address space” [13]. However, TARP does not virtualize the MAC address and its purpose is to simplify state-keeping in firewalls, not protecting the privacy of the user.

6 Discussion

In this section, we make a short note of the limitations of our approach and then proceed to further evaluate it by benchmarking it to Tor and contrast it to the requirements set in Section 2.

6.1 Limitations

The limitations for the approach can be roughly categorized as the boundaries of current legacy systems (medium constraints, MAC-address authentication, limits of the address space) and the other limitations present different possibilities for attackers to circumvent the privacy protection architecture (dynamic DNS, operating system and device fingerprinting, out-of-band information, duplicate application traffic). Due to the lack of space for this technical report, we omit the further discussion of the effects of these constraints.

6.2 Evaluation

We begin our evaluation by benchmarking our approach to an approach that uses a distributed infrastructure to provide sender and receiver anonymity. In the related work, we already have contrasted our approach to infrastructureless approaches. We use Tor [8] as a benchmark, since it can be considered as the de facto way to achieve sender and receiver anonymity in the Internet and thus protect the identity of the user. It is estimated that there are currently 200 000 users of Tor and around the world 1000 servers, which relay the Tor traffic. Although Tor is relatively simple to install, it hinders the user experience with increasing the latency of e.g. Web browsing. Also, some services, such as the Wikipedia, does not accept connections made from the Tor network. Further, Tor has an alarming usability problem, because for some reason, many users tend to believe that Tor provides end-to-end encryption and do not protect their traffic accordingly [38]. Even though we acknowledge that Tor is probably the best current deployed way protect your identity in the network, our work explores a solution space where you do not trust the network at all. Tor relies on distributed infrastructure, but we desired to find out how much privacy users can receive without resorting to any kind of infrastructure. In fact, our approach can be seen as complementary to Tor as discussed in the related work.

It is, however, important to distinguish from who we are trying to protect from. If we can assume that the access network provider can be trusted, and the privacy protection mechanisms need to be in place for attackers one hop or further away from the access network much simpler mechanisms suffice. Traditional NATs, a new IP address for every new TCP or UDP flow [5] can provide enough privacy against most attackers residing outside the access network.

Our tests showed that the approach does not hinder the user experience in any noticeable way, provided that the network allows the use of multiple addresses. We also required changes only to a single operating system and did not assume any infrastructure. The users can also remain reachable even though the protocol stack virtualization is in place. For example, for a given set of identifiers, the system can enable dynamic DNS or SIP registration for VoIP calls to remain reachable.

As final note, we think that this approach should be interesting also because it is does not seem to be dual use technology. For example, Tor can be used for illegal purposes easily in addition to protecting the privacy of ordinary users, but our approach simply provides unlinkability for different traffic sent to the network by the user's device. This means that if the user does something illegal with one set of identifiers, law enforcement can at least pinpoint the network where the traffic originated.

7 Conclusions

We have presented an approach for identity privacy protection that could significantly change how network applications use the resources of a networked operating system, even though the changes to the operating system are almost negligible as we have shown with the user space implementation. One of the main benefits of the approach is that when properly implemented, applications do not need changes and the user experience does not change. The users can still lose their privacy with some applications, but it does not help a third party to deduce anything else about the user. To the best of our knowledge, this is the first approach that attempts to provide privacy without infrastructure for the whole mobile networked system by making changes to only a single host and not to the both peers.

8 Acknowledgments

Janne Lindqvist's thinking on the topic was much evolved by sharing the work space with Teemu Koponen at the International Computer Science Institute, Berkeley. Teemu's suggestions and comments on the manuscript have significantly helped to focus this work. The authors are also grateful for Tuomas Aura for a rewrite of an earlier version of the introduction section and for providing thorough comments on how to write proper introductions. We also thank George Danezis and Michael Roe for pointing out some limitations of the approach.

Bibliography

[1] M. Abadi and C. Fournet. Private authentication. *Theor. Comput. Sci.*, 322(3):427–476, Sept. 2004.

[2] W. Aiello, S. M. Bellovin, M. Blaze, R. Canetti, J. Ioannidis, A. D. Keromytis, and O. Reingold. Just fast keying: Key agreement in a hostile internet. *ACM Transactions on Information and System Security (TISSEC)*, 7, May 2004.

[3] J. Arkko, P. Nikander, and M. Näslund. Enhancing Privacy with Shared Pseudo Random Sequences (preliminary version). In *Security Protocols, 13rd International Workshop, Cambridge*, Apr. 2005.

[4] T. Aura, J. Lindqvist, M. Roe, and A. Mohammed. Chattering laptops. In *to appear in 8th Privacy Enhancing Technologies Symposium*, July 2008.

[5] T. Aura and A. Zugenmaier. Privacy, Control and Internet Mobility. In *Security Protocols, 12th International Workshop*, Apr. 2004.

[6] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *SOSP'03*, Oct. 2003.

[7] D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, Feb. 1981.

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

[9] N. Droux, S. Tripathi, and K. Belgaied. Crossbow: Network virtualization and resource control. http://www.usenix.org/events/usenix07/posters/droux.pdf.

[10] J. Franklin, D. McCoy, P. Tabriz, V. Neagoe, J. V. Randwyk, and D. Sicker. Passive Data Link Layer 802.11 Wireless Device Driver Fingerprinting. In *Proceedings of 15th USENIX Security Symposium*, pages 167–178, July/August 2006.

[11] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: a virtual machine-based platform for trusted computing. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 193–206, New York, NY, USA, 2003. ACM Press.

[12] R. Gerdes, T. Daniels, M. Mina, and S. Russell. Device identification via analog signal fingerprinting: A matched filter approach. In *The 13th Annual Network and Distributed System Security Symposium (NDSS)*, Feb. 2006.

[13] P. M. Gleitz and S. M. Bellovin. Transient addressing for related processes: improved firewalling by using IPV6 and multiple addresses per host. In *SSYM'01: Proceedings of the 10th conference on USENIX Security Symposium*, pages 8–8, Berkeley, CA, USA, 2001. USENIX Association.

[14] J. A. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symposium on Research in Security and Privacy*, Apr. 1982.

[15] D. M. Goldschlag, M. G. Reed, and P. F. Syverson. Hiding Routing Information. In *Workshop on Information Hiding Springer-Verlag LLNCS 1174*, 1996.

[16] M. Gruteser and D. Grunwald. Enhancing location privacy in wireless LAN through disposable interface identifiers: a quantitative analysis. *Mob. Netw. Appl.*, 10(3):315–325, 2005.

[17] S. Guha and P. Francis. Identity Trail: Covert Surveillance Using DNS. In *Workshop on Privacy Enhancing Technologies (PET)*, June 2007.

[18] S. Ioannidis, S. Sidiroglou, and A. D. Keromytis. Privacy as an Operating System Service. In *1st USENIX Workshop on Hot Topic in Security (HotSec'06)*, July 2006.

[19] T. Jiang, H. J. Wang, and Y.-C. Hu. Location privacy in wireless networks. In *The 5th International Conference on Mobile Systems, Applications, and Service (MobiSys)*, June 2007.

[20] S. Katti, J. Cohen, and D. Katabi. Information Slicing: Anonymity Using Unreliable Overlays. In *4th Usenix Symposium on Networked Systems Design & Implementation (NSDI)*, Apr. 2007.

[21] T. Kohno, A. Broido, and K. Claffy. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2(2), April-June 2005.

[22] J. Lindqvist and L. Takkinen. Privacy management for secure mobility. In *Workshop on Privacy in the Electronic Society (WPES)*, Oct. 2006.

[23] A. Menon, A. L. Cox, and W. Zwaenepoel. Optimizing network virtualization in Xen. In *USENIX-ATC'06: Proceedings of the Annual Technical Conference on USENIX'06 Annual Technical Conference*, pages 2–2, Berkeley, CA, USA, 2006. USENIX Association.

[24] S. J. Murdoch. Hot or Not: Revealing Hidden Services by their Clock Skew. In *CCS'06*, October–November 2006.

[25] A. C. Myers and B. Liskov. Protecting privacy using the decentralized label model. *ACM Trans. Softw. Eng. Methodol.*, 9(4):410–442, 2000.

[26] T. Narten, R. Draves, and S. Krishnan. RFC 4941: Privacy Extensions for Stateless Address Autoconfiguration in IPv6, Sept. 2007. Status: Draft Standard.

[27] J. Pang, B. Greenstein, R. Gummadi, S. Seshan, and D. Wetherall. 802.11 user fingerprinting. In *MobiCom'07*, Sept. 2007.

[28] J. Peterson. RFC 3323: A Privacy Mechanism for the Session Initiation Protocol (SIP), Nov. 2002.

[29] Planetlab. https://www.planet-lab.org/.

[30] M. K. Reiter and A. D. Rubin. Crowds: Anonymity for Web Transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, Nov. 1998.

[31] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. RFC 3261: SIP: Session Initiation Protocol, June 2002.

[32] U. Shankar and C. Karlof. Doppelganger: Better browser privacy without the bother. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 154–167, New York, NY, USA, 2006. ACM Press.
[33] P. Srisuresh and K. Egevang. RFC 3022: Traditional IP Network Address Translator (Traditional NAT), Jan. 2001. Status: Informational.
[34] G. Su and J. Nieh. Mobile communication with virtual network address translation. CUCS-003-02, Columbia University Department of Computer Science, Feb. 2002.
[35] S. Thomson, T. Narten, and T. Jinmei. RFC 4862: IPv6 Stateless Address Autoconfiguration, Sept. 2007. Status: Draft Standard.
[36] VMware. http://www.vmware.com.
[37] A. R. Yumerefendi, B. Mickle, and L. P. Cox. TightLip: Keeping Applications from Spilling the Beans. In *4th USENIX Symposium on Networked Systems Design & Implementation (NSDI)*, Apr. 2007.
[38] K. Zetter. Rogue nodes turn tor anonymizer into eavesdropper's paradise. *Wired*, Sept. 2007. http://www.wired.com/politics/security/news/2007/09/embassy_hacks.