



WiNoN — Plugging the Leaky Boat of Web Anonymity

PRELIMINARY DRAFT

David Isaac Wolinsky and Bryan Ford
Yale University

ABSTRACT

Despite the attempts of well-designed anonymous communication tools to protect users from being tracked or identified, other artifacts, such as a user’s environment and behavior, may leak a user’s identity. Plugging this leaky boat of web anonymity requires a “top-to-bottom” whole-system approach, rather than focusing on specific protocols or layers. As an initial step, we present WiNoN, a general purpose anonymity-centric system architecture that plugs identity leaks from the network layer up. The core concept powering WiNoN—*nym-browse* or pseudonym browsing mode—gives the user a web-browsing environment in which each of the user’s contextually independent web activities run in unique, yet homogenous containers. This enforced isolation limits accidental leakage of private information regardless of misconfigured anonymity tools, trojan applications, or bugged files. This paper explores the WiNoN design space and presents an early prototype supporting Tor, Dissent, and SWEET communication tools as well as an incognito mode.

1. INTRODUCTION

In Repressistan, a country ruled by an authoritarian regime, the state-controlled ISP regularly monitors the traffic of online users in order to suppress dissent. Aware of these dangers, Alice, a resident of Repressistan, employs state-of-the-art anonymous browsing software, such as Tor [7]. If Alice unintentionally accesses Flash content, which bypasses her anonymizing proxy, or transmits a JPEG image containing EXIF information linked to her camera, she may well compromise her identity. Unfortunately, even the best anonymity systems currently available cannot prevent Alice from accidentally leaking her personal information across an anonymous channel, or her anonymous identity across a public channel.

Worse, even if Alice filters her traffic and the content she shares, she does so under the unrealistic assumption that her anonymous messages remain *unlinkable* across time. Many realistic communication activities, such as interactive anonymous web browsing sessions, or a series of blog posts signed under a pseudonym, make communication linkable and

thereby are vulnerable to *traffic correlation* and *intersection attacks* [16]. Even carefully-designed anonymity systems such as Tor leak identifying material useful in intersection attacks, when viewed from a “whole-system” perspective: e.g., IP or MAC addresses, Web browser fingerprints [9], or operating system behavior [20]. If Alice repeatedly posts sensitive messages using the same obscure version of Firefox or holds a particular long-lived cookie, and the Repressistani police observe the consistent presence of a particular user online when these postings occur, no theoretical guarantees at the anonymous messaging layer will protect her. As a result the Repressistani police may confiscate her computer littered with evidence of her communication.

Further, the expedient and hence unavoidable practice of building anonymous communication systems out of software components not designed with anonymity in mind—popular web browsers—creates pervasive risks of accidental identity leakage, such as via third-party plug-ins that circumvent the anonymous forwarding path [21, 10]. Non-expert users can also unwittingly reveal their identities in many ways, for example, by incorrectly tracking the state of their browser’s “anonymity” mode. In general, systems meant to protect even naive users are difficult to design and implement, made even more challenging by the various hardware and software configurations found on different user machines.

To protect users from “accidentally shooting themselves in the foot” by inadvertently leaking identifying information, we introduce WiNoN, a general purpose anonymity-centric system that addresses these challenges from a cross-layer, “whole-system” perspective. In one of many WiNoN deployment models, as illustrated in Figure 1, Alice has obtained a WiNoN USB flash drive and an “anonymous” USB WiFi device purchased using cash. Upon starting her personal computer with the USB flash drive, she enters the WiNoN browser with support for *nym-browse*—a novel concept for anonymous web browsing that forks each of her web activities into independent domains each with their own quasi-persistent

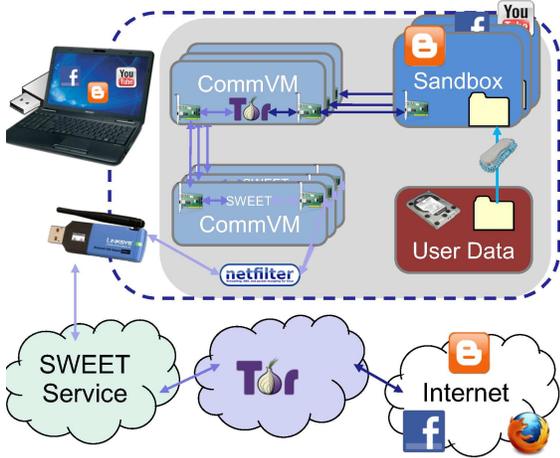


Figure 1: WiNoN organization

state and anonymous communication channels. The WiNoN environment allows her to safely browse all web content including Flash content and even post locally stored content, such as JPEG images, without fear of accidentally compromising herself. At the end of her session, she throws away both USB devices leaving behind no trace of her activities.

Previous work on building anonymous environments focused on amnesiac behavior [26], homogeneous environments [26, 30], enforced anonymous communication [30], and even scrubbing of file types [1, 3, 25, 29]. Amnesiac behavior provides plausible deniability but requires the user to remember and reproduce state. Existing environments support only a single anonymous communication tool, Tor [7], thus limiting their applicability for other forms of anonymous communication or for combining such tools without security concerns. Current scrubbing efforts require explicit knowledge of the tool, which unaware users may bypass without a second thought. Related work on securing web browsing [27, 17] have eliminated many browser-based attacks, however, such efforts offer no support for anonymous communication.

In this paper, we explore the WiNoN design space and how it provides an evolutionary step for anonymous web browsing. Specifically, WiNoN introduces the following novel concepts: 1) independent state and anonymous channels for each independent web activity, 2) quasi-persistent system state per-web activity that provides the same plausible deniability as amnesiac behavior, 3) seamless anonymous communication tool composability, and 4) transparent use of scrubbing tools.

2. GLOOM AND DOOM

While well-designed communication tools may not

directly leak identifying information, the environment in which the tools run and their usage model may enable an adversary to identify a user. In this section, we give many motivating scenarios in which layman and even expert configurations could be exploited and leak information.

An **application leak** may occur as soon as a user begins their communication session if the application shares their public IP address over the communication channel. P2P applications, like BitTorrent, share all network information to improve connectivity. Similarly, the Flash plugin ignores Web browser proxy settings and directly accesses content, which can unintentionally bypass an anonymizing layer such as Tor.

A user who shares their own content with others may be a victim of a **user information leak**. Many applications embed personally identifiable information within the metadata of files, for example, Microsoft Word and many other word processing applications contain a document author’s name among other personal information. Various image formats contain information that include the camera that took the picture as well as GPS coordinates. A whistle-blower could be uploading content that has been steganographically embedded with a unique marker revealing his identity.

Web sites and networked applications can obtain a significant amount of local machine information including network setup, display resolution and color depth, processor speed and architecture, installed software and libraries, RAM, total and available disk space, historical content such as cookies for web browsers and previous user names, identities, and activities for applications, even the user’s current time [9]. When considered together, these properties may be enough to establish a unique **fingerprint**. A powerful adversary could use this information in an *intersection attack* [16] to identify the user’s real identity.

In the Pown2Own competition, teams work together in order to show that existing technologies have **remote exploits** giving direct access to the user’s machine over the network. A knowledgeable adversary could exploit zero-day flaws in order to gain access to a users machine and identify that user by the content on his computer. Similarly, an adversary can obtain the machine physically through **confiscation** and snoop through all files on the machine in an effort to link the owner to a particular online pseudonym.

Once an adversary has unmasked a user in one context, the user may be susceptible to losing their anonymity in another as a result of a **correlation attack**. In existing systems, an adversary using a

remote exploit to gain access to the user’s machine can easily unmask all anonymous communication channels beyond the context in which the exploit occurred. Similarly, if a user has been deanonymized by accidentally leaking personal information across a Tor circuit, any other traffic within that circuit would also lose anonymity.

3. WiNoN ARCHITECTURE

Despite this *bleak* situation, there remains hope: WiNoN. As shown in Figure 1, WiNoN consists of a single web browser supporting *nym-browse*, which separates each independent browser activity or *nym* into secure domains to eliminate many of these leaks. Each secure domain consists of a unique Sandbox that stores quasi-persistent browser state and connects to the Internet through one or more anonymity or circumvention tools each running in their own CommVM (communication virtual machine). WiNoN provides access to the local storage, i.e., the local hard disk and USB devices; however, read data transparently passes through a filter, automatically scrubbing it of personal identifying material.

3.1 Pseudonyms

While having an active web browser connected to webmail, a user may want to post a sensitive content to a message board. If done within the same WiNoN Sandbox, an adversary need only find a common fingerprint between the two to correlate the two behaviors to the same user. Even a fully clean browser may eventually obtain state, such as cookies or javascript calls, that can correlate a user across distinct anonymous channels. WiNoN uses *nym-browse* to separate each activity into distinct pseudonyms or *nym*s consisting of an independent Sandbox environment connected to its own independent set of CommVMs. Only an individual directly looking at the screen of a WiNoN system would be able to determine that a user checking webmail is the same as one posting to a message board.

We are investigating various approaches to solving this challenge. One approach would be to execute browsers in each VM and redirect the output via VNC to a common “web browser.” Another, perhaps more user-friendly alternative would be a WiNoN-specific browser that runs in a client-server mode. Each server would run inside a Sandbox, translating requests into binary data transferring it upstream to the client, “web browser,” which would translate the data into a visible web page. This approach would enable WiNoN to minimize the memory consumption of each Sandbox and latency overheads induced by VNC. However, this sets forth a significant chal-

lenge: the ability to use existing web browsers and their plugins.

Another design consideration is whether to use anonymity tools built-in support for pseudonyms, e.g. Tor [7] and Dissent [31], or to execute them in separate CommVMs. A single instance of Tor supports this by adding additional SOCKS5 proxy server endpoints, while Dissent would require trivial, client-side modifications or simply running parallel Dissent instances each with unique SOCKS5 proxy server endpoints. While the unique CommVM model may be heavy weight, the shared model could result in an easy correlation attack upon the compromise of an instance of the anonymity tool. Although arguably, if a tool can be compromised, perhaps then parallel instances are not safe anyway.

Finally, is this approach sufficient? Instead of web browsing, should we be considering applications in general? Alternatively, we could consider an operating system engineered from the ground up to ensure and preserve anonymity even for applications that were not originally configured for WiNoN, like the WiNoN browser.

3.2 Secure, Modular Communication

Without careful construction, adversaries can compromise an entire system from even the smallest opportunity. To prevent or mitigate an adversary from taking over a WiNoN system and avoid application leakage despite buggy applications, improperly configured tools, or accidentally installed trojans, WiNoN embraces a framework for secure, modular communication. Our framework requires that each session and even its individual components be completely independent from each other and the underlying host. The approach ensures that even if an adversary compromises any component within a single session, he would be unable to extend into other unrelated sessions, gain access to the host system, or, in the case of the Sandbox, circumvent anonymous communication tools.

To support this goal, WiNoN makes extensive use of virtualization with unique virtual machines (VMs) for each Sandbox and for each of the communication tools running in independent CommVMs. While other solutions may be possible, virtualization allows WiNoN to run existing applications and communication tools without modification. Within a WiNoN environment, hypervisor handles message forwarding between the Sandbox and its immediate upstream CommVM, and each CommVM has the necessary communication access to its immediate downstream and upstream connections whether that be the Sandbox, another CommVM, or a NAT

running on the hypervisor. The NAT on the hypervisor connects the final CommVM in the serial link to the hosts network interface card (NIC) and thus to the Internet. Effectively, each VM has no knowledge of other networks except those shared on the downstream and upstream limiting the Sandboxes exposure and access to underlying communication tools as well as the local network.

Developers adding support for a tool to WiNoN need only focus on selecting the input device or the communication path into their tool and configuring their tool to use the appropriate output NIC. Using this model, a user can then select from any number of tools to compose a powerful combination of circumvention and anonymous communication tools. As of now, WiNoN already supports an array of mechanisms for tunneling communication. In the most simplistic form, incognito mode, the Sandbox directly speaks to the NAT running on the hypervisor preventing the Sandbox from directly accessing the LAN and confining state to the Sandbox. A member requiring network anonymity may desire to use Tor [7], an anonymous onion routing tool, but also needs to leverage a circumvention tool like SWEET [14], IP over e-mail, to bypass restrictions made by his ISP preventing direct usage of Tor. Alternatively, he may reside in a wireless network with other Tor users and run Tor over Dissent [31], an anonymous group communication tool, to eliminate certain intersection attacks [2, 18, 24]. Even tools like Freewave [13] that use audio devices to evade network firewalls via Skype can be used seamlessly in WiNoN.

3.3 Quasi-Persistent Data

As the user installs applications, collects Web cookies, and develops other fingerprints, he establishes a unique identity that could be used for intersection attacks. To protect users from adversaries, WiNoN, like Tails [26], carries no state or exhibits signs of amnesia across boots. In other words, every use boots as if it was the first, and hence all sandboxes remain the same regardless how frequently a single member uses the system. An amnesiac system prevents persecution due to confiscation of user hardware, as there would be no way to directly tie any session to a specific computer, once a computer has been turned off.

Although attractive, amnesiac systems lack some user-friendliness and can do nothing to inhibit intersection attacks against a user that has a common behavior, e.g., accesses the same IRC channel with the same nickname at the same time every day. Generally, users may desire to keep this content, such as

bookmarks, usernames and passwords, or application preferences. WiNoN addresses this through the use of quasi-persistent data, which moves the user's data away from the computer while not in use, with the intent of preventing confiscation attacks, akin to what CleanOS [28] does, although with plausible deniability in mind. Possible solutions include hidden and encrypted partitions on the local disk, encrypted USB disks stored independent of the computer in use, or the cloud. When starting a new session, users would be prompted to specify the source of the data and perhaps a key for decrypting it. WiNoN then populates the home directory within the Sandbox environment for this session.

The cloud storage solution potentially provides strong plausible deniability, as a user storing their data into a free storage cloud such as Drop-Box or Google Drive, cannot be easily connected with their data once the WiNoN session disconnects. As we currently envision, at the beginning of a session, the user connects through an anonymity service to access data stored in an anonymous cloud account, perhaps created during the first execution of this pseudonym, and uses this data to populate the current WiNoN instance. A cloud provider cannot discern content storing WiNoN data versus some other encrypted data. Even if the cloud provider could identify WiNoN data, the cloud would have no means to determine the content unless it were able to break the encryption scheme securing it. Meanwhile, an adversary can only determine linkage between the content if he can correlate access to specific data within a cloud to a WiNoN session, in other words, the adversary would need both extensive if network presence and access to privileged cloud resources.

To make this environment even more secure, no persistent data from a nym should remain on a computer once the use of the nym has been discontinued. As designed, however, WiNoN currently retains traces of that state until reboot. Recent work by Dunn [8] explores how much information remains on a host after a virtual machine has shut down as well as various methods for eliminating it. This may work well for WiNoN; however, such approach requires some specialized hardware and additional computational overhead, perhaps erasing the Sandboxes allocated memory before boot and after shutdown would be sufficient.

3.4 Automated Data Scrubbing

The purpose of using circumvention or anonymous communication tools usually involves more than accessing Web content. For example, users may want to distribute content from non-anonymous sources,

e.g., posting photos from their digital camera. Naively transferring these files will likely leak the user’s or some confidant’s identity [5, 6]. WiNoN plans to eliminate this possibility by automatically stripping files of potentially identifying material by using appropriate tools [1, 3, 25, 29]. Such an approach comes with caveats. This is an arms race, developers continuously come up with new file types and malicious individuals find new ways to exploit these and existing file types. Additionally, users need to be aware that an automated scrubber does not give the user the license to not verify that the file does not contain personally identifiable information, such as a signature in a word document or their image in a picture. Of course this opens up other opportunities for advanced filtering, such as automated or WiNoN-assisted blurring of faces in images and support for removing watermarks.

For now, we have contemplated two approaches to integrating this into the system. The more desirable, although perhaps more difficult to build correctly approach involves actually mounting the local storage systems within the same environment as the web browser, except that all files would be filtered upon access. Alternatively, a more obviously secure mode, which we have already implemented uses a separate VM hosting the local data. Within this VM, a user can access any content and copy it to a specific directory, causing an automated scrubbing, and making it available in a read-only environment to the web browser or to a specific nym within the browser.

The reverse path, or storing content from the web session for later use outside of WiNoN, has similar considerations. One such approach would be to host a write-only directory for each nym with contents stored therein passing through the same scrubbing tools as the outgoing content as well as a virus scanner to protect the user prior to actually storing it to an external drive. Regardless of the eventual solution, write support opens up a significant potential for accidents.

4. PROTOTYPE IMPLEMENTATION

We have implemented an early prototype WiNoN system that supports a single active pseudonym, composable CommVMs, and the beginnings of the automated data scrubber. Currently, WiNoN supports Tor, Dissent, SWEET, and the incognito mode to connect users to the Internet. KVM [22], a virtualization solution built directly into the Linux kernel, provides the infrastructure for hosting VMs. We use iptables in the hypervisor to limit network access for the CommVMs and Sandbox as well as to

provide Internet to the final hop in the CommVM. For communication tools that use SOCKS5, we employ iptables in conjunction with redsocks to move packets from the network stack and into these applications. We chose to use AUFS [19] for stacking file systems together because it comes bundled with the Linux Kernel included in Ubuntu, our current base distribution. To simplify management of the different VMs, their contents are directly accessible as host folders in the base image and are mounted into the VM at run-time using KVM’s VirtFS [15].

The sanitization VM hosts a VirtFS folder, the hypervisor listens for creation of new files in that directory using inotify. For each new file, it is immediately copied into a temporary directory. Then a bash script checks for a file type using the Linux command “file” and runs an appropriate scrubber. The resulting scrubbed file is passed into another VirtFS folder mounted in the sandbox. Files without scrubbers use the default policy, delete. In a production level environment, we may either leave this as a run time option or to delete the file.

4.1 Validating the System

We developed and validated the WiNoN prototype using KVM and nested virtualization. This process made it easy to verify the state of the system and inspect for potential information leaks. To check for leaks, we connected the WiNoN hypervisor to a VNIC NATed to the host. On the host device, we ran Wireshark and inspected traffic entering and exiting an idle WiNoN client. While using Ethernet, the WiNoN hypervisor emitted only traffic for DHCP and WiNoN traffic, while the sandbox transmitted no traffic. The same image we used in KVM can be directly copied to a USB flash drive and booted on any machine that supports USB boot.

To enforce amnesia, WiNoN uses a stackable file system to *merge* multiple physical file systems into a single virtual file system. Both the hypervisor and all VMs share a common, read-only, base file system embodying a standardized configuration. With modern Linux distributions packing on the pounds, this enables us to ship WiNoN on 1 GB USB drives and even 650 MB cd-roms using standard distributions without sacrificing applications or spending time tweaking the distribution for size. The amnesiac read-write portion resides in RAM, ensuring that the sandbox starts fresh after each reboot.

4.2 Preliminary Experiences

To give a taste of the WiNoN experience, we performed a series of experiments using both the real Tor network and a mock Dissent network. We were

	Kernel	Google	Peacekeeper	Memory
Native Tor	1179 S	4.3 S	1386	158M
WiNoN Tor	885 S	11.3 S	1191	830M
Native Dissent	60.2 S	1.29 S	1387	185M
WiNoN Dissent	52.2 S	2.6 S	1184	855M

Table 1: Comparison of communication tools running natively and within WiNoN.

particularly interested in how the use of virtualization impact their experience. The evaluation ran on an Intel I3 laptop with virtualization extensions connected to a university wireless network and consisted of downloading the Linux kernel, the index.html of `www.google.com`, running a Javascript benchmark, Peacekeeper [12], and the total amount of RAM in use. Each test investigates different overheads: downloading Linux kernel bandwidth, retrieving Google’s index.html latency, and running Peacemaker CPU. We present the results for the different combinations in Table 1. Peacekeeper produces an integer score, whereas the download times are in seconds.

Overall we were quite happy with the results, virtualization impacts the Peacekeeper task by less than 20% and Dissent exhibited some latency and bandwidth overheads when run in WiNoN. Another VM associated network overheads occurs when obtaining the first byte. We believe this relates to performing DNS queries directly via a SOCKS5 request as opposed to inline as the web browser does when it interacts directly with the SOCKS5 proxy. While WiNoN made use of a significant chunk of memory, we are confident that with better page sharing and a lighter-weight OS for CommVMs, the number will be much more respectable.

Comparisons between Dissent and Tor are not directly comparable. Tor uses a real, heavily loaded network, whereas Dissent runs on top of a well-provisioned, lightly loaded, synthetic test configuration. Furthermore, each evaluation was executed only 3 times and while the current results favor Tor in WiNoN, a sufficient number of executions would eventually favor native Tor just as native Dissent.

5. DISCUSSION

5.1 The Enemy Within

The WiNoN model depends on the sterility of both hardware and software. A malicious party could easily install malware into the hypervisor prior to distribution or in the firmware of WiFi devices prior to a party in order to compromise a user’s anonymity. Using trusted platform modules could potentially ensure the running software and firmware; however, all is for naught if the hardware vendor has conspired with the adversary.

5.2 Lack of Perfect Homogeneity

Even while using virtualization and the same set of software, there still exists the possibility for differences between users. An adversary could execute a particularly CPU intensive application, such as a javascript application that computes a million digits of PI, and use the timings to produce a fingerprint for that user. Also, all users cannot necessarily be in the same location, and hence if there is a single Tor user in Repräsentan, the government-owned ISP could easily determine the responsible party for any Repräsentani traffic related to Tor.

5.3 Self-Deanonymizing Users

A reckless user can easily deanonymize himself in most anonymity systems, while WiNoN goes to great lengths to prevent such accidents, there exist some limitations. In WiNoN, for example, if a user shares personal information while signing an anonymous petition, by providing a personal e-mail account, WiNoN like most other systems would be unable to prevent this type of accident. However, because of nym-browse, a user need not worry about correlation attacks wherein they deanonymize a single nym instance. Specifically, if a user were to browse their Facebook or personal webmail account in one tab and access sensitive material in another, the adversary would be unable to link this material together. In systems like Tails and Whonix, such behavior might instantly deanonymize the user.

5.4 Concealing Network Identity

Network identity proves difficult even in the WiNoN context. Network fingerprinting comes in many forms from operating system interfaces to NIC devices [20], drivers [11], MAC addresses, and even the hardware characteristics of devices [4]. Some of these attacks are avoidable using a common device with a standardized driver, a volatile management framework for the device, and randomized MAC addresses.

For well-equipped adversaries, this approach is insufficient. Brik et al. [4] determined that even devices from the same manufacturer with sequential serial numbers could be fingerprinted due to the errors in the signal. Since we envision that WiNoN may be used in moderately well-organized groups, we posit that users could organize WiFi device exchange parties, or WiFi social mixes, akin to Richard Stallman’s “Charlie Card” swapping parties [23] to elude RFID-based fingerprinting.

6. CONCLUSIONS

WiNoN addresses accidental information leakage in anonymous communication systems from a “whole-

system” perspective, seeking structural solutions based on isolation, homogeneity, and limited flexibility. Enforced anonymization, in combination with virtualization and homogeneity, offers some protection from even a compromised WiNoN sandbox leaking personally identifiable information. While merely a start, we believe WiNoN offers a useful platform for researching these problems.

Acknowledgments

This material is based upon work supported by the Defense Advanced Research Agency (DARPA) and SPAWAR Systems Center Pacific, Contract No. N66001-11-C-4018. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA or SPAWAR.

7. REFERENCES

- [1] T. Aura, T. A. Kuhn, and M. Roe. Scanning electronic documents for personally identifiable information. In *5th ACM workshop on Privacy in electronic society (WPES)*, pages 41–50, New York, NY, USA, 2006. ACM.
- [2] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker. Low-resource routing attacks against Tor. In *6th Workshop on Privacy in the Electronic Society (WPES)*, pages 11–20, Oct. 2007.
- [3] E. Bier, R. Chow, P. Golle, T. King, and J. Staddon. The rules of redaction: Identify, protect, review (and repeat). *Security Privacy, IEEE*, 7(6):46–53, nov.-dec. 2009.
- [4] V. Brik, S. Banerjee, M. Gruteser, and S. Oh. Wireless device identification with radiometric signatures. In *ACM International Conference on Mobile Computing and Networking (MobiCom)*, pages 116–127, 2008.
- [5] S. Byers. Information leakage caused by hidden data in published documents. *IEEE Security and Privacy*, 2:23–27, 2004.
- [6] A. Castiglione, A. D. Santis, and C. Soriente. Taking advantages of a disadvantage: Digital forensics and steganography using document metadata. *Journal of Systems and Software*, 80(5):750–764, 2007.
- [7] R. Dingledine, N. Mathewson, and P. Syverson. Tor: the second-generation onion router. In *Proceedings of the 12th USENIX Security Symposium*, Aug. 2004.
- [8] A. M. Dunn, M. Z. Lee, S. Jana, S. Kim, M. Silberstein, Y. Xu, V. Shmatikov, and E. Witchel. Eternal sunshine of the spotless machine: Protecting privacy with ephemeral channels. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2012.
- [9] P. Eckersley. How unique is your web browser? In *Privacy-Enhancing Technologies Symposium*, July 2010.
- [10] G. Fleischer. Attacking tor at the application layer. http://ww.defcon.org/images/defcon-17/dc-17-presentations/defcon-17-gregory_fleischer-attacking_tor.pdf, July 2009.
- [11] J. Franklin, D. McCoy, P. Tabriz, V. Neagoe, J. Van Randwyk, and D. Sicker. Passive data link layer 802.11 wireless device driver fingerprinting. In *USENIX Security Symposium*, 2006.
- [12] Futuremark. Peacekeeper – the universal browser test. <http://peacekeeper.futuremark.com>, January 2013.
- [13] A. Houmansadr, T. Riedl, N. Borisov, and A. Singer. I Want my Voice to be Heard: IP over Voice-over-IP for Unobservable Censorship Circumvention. In *Network and Distributed System Security Symposium*, 2013.
- [14] A. Houmansadr, W. Zhou, M. Caesar, and N. Borisov. Sweet: Serving the web by exploiting email tunnels. *CoRR*, abs/1211.3191, 2012.
- [15] V. Jujjuri, E. V. Hensbergen, A. Liguori, and B. Pulavarty. Virtfs—a virtualization aware file system pass-through. June 2010.
- [16] D. Kedogan, D. Agrawal, and S. Penz. Limits of anonymity in open environments. In *5th International Workshop on Information Hiding*, Oct. 2002.
- [17] J. Mickens and M. Dhawan. Atlantis: Robust, extensible execution environments for web applications. In *23rd ACM Symposium on Operating System Principles (SOSP)*, Oct. 2011.
- [18] S. J. Murdoch and P. Zieliński. Sampled traffic analysis by Internet-exchange-level adversaries. In *7th Privacy Enhancing Technologies Symposium (PETS)*, pages 167–183, 2007.
- [19] J. R. Okajima. Aufs3 – advanced multi layered unification filesystem version 3.3. <http://aufs.sourceforge.net/>, January 2013.
- [20] J. Pang, B. Greenstein, R. Gummadi, S. Seshan, and D. Wetherall. 802.11 user fingerprinting. In *ACM International Conference on Mobile Computing and Networking (MobiCom)*, pages 99–110, 2007.
- [21] M. Perry. To toggle, or not to toggle: The end of torbutton. <https://blog.torproject>.

- org/blog/
toggle-or-not-toggle-end-torbutton,
May 2011.
- [22] Qumranet. Kernel-based virtual machine for linux. <http://kvm.qumranet.com/kvmwiki>, March 2007.
 - [23] J. Sedgwick. The shaggy god. <http://www.bostonmagazine.com/articles/2008/04/the-shaggy-god/>, May 2008.
 - [24] A. Serjantov and P. Sewell. Passive-attack analysis for connection-based anonymity systems. *International Journal of Information Security*, pages 172–180, 2005.
 - [25] L. Sweeney. Replacing personally-identifying information in medical records, the scrub system. *Journal of the American Medical Informatics Association*, 1996.
 - [26] Tails: The amnesic incognito live system, September 2012. <https://tails.boum.org/>.
 - [27] S. Tang, H. Mai, and S. T. King. Trust and protection in the illinois browser operating system. In *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Oct. 2010.
 - [28] Y. Tang, P. Ames, S. Bhamidipati, A. Bijlani, R. Geambasu, and N. Sarda. Cleanos: limiting mobile data exposure with idle eviction. In *USENIX Symposium on Operating Systems Design and Implementation*, Oct. 2012.
 - [29] J. Voisin, C. Guyeux, and J. M. Bahi. The metadata anonymization toolkit. <http://arxiv.org/abs/1212.3648>, may 2013.
 - [30] Whonix. <http://sourceforge.net/p/whonix>.
 - [31] D. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson. Scalable anonymous group communication in the anytrust model. In *European Workshop on System Security (EuroSec)*, Apr. 2012.