Can We Have a Usable Internet Without User Trackability?

Eric Rescorla
ekr@rtfm.com

November 5, 2010

The Current Situation

The Internet protocol stack as currently designed makes no serious attempt to prevent user tracking. Rather, identifiers are assigned to users and their associated network elements all throughout the protocol stack. For instance, in a typical Web (HTTP[FGM+99]) transaction, the following identifiers are transmitted on the wire (or broadcast over a wireless interface):

1. An Ethernet MAC address unique to the piece of hardware and more or less stable for the life of the device, and—because MACs are assigned in manufacturer and product ranges—implicitly identifies the type of device

2. An IP[Pos81] address which, if not permanent, is stable for hours and reveals the user's location with a reasonable amount of precision (often down to the street address)

3. One or more HTTP cookies [KM00], the lifetime of which is controlled by the Web server (although in principle the user can delete these or prevent their storage.) Flash may also have its own cookies.

The above list only includes *explicit* identifying information in the protocols. In addition, there are a large number of implicit or semi-channels that allow the device fingerprinting and hence various degrees of user tracking. Canonical examples include browser version and the set of plugins installed, browser history sniffing, clock skew [KBC05], and various computer parameters which are readable from JavaScript.

The standard Internet protocols make no real attempt to address any of these issues. The only real concession that is made to "privacy" is really confidentiality—hiding identifying information from other, uninvolved, entities who happen to observe a communication. For instance, in the example above one might attempt to improve the above interaction by running it over TLS [DR08] (HTTPS [Res00]). This is only a very partial mitigation at best: while it hides the cookie from on-path observers it does nothing to remove the other identifiers and in addition adds an extra one: the TLS session identifier, which can be used to link requests even if the browser is configured not to store cookies. Moreover, performance considerations motivate increasing the lifetime of TLS session identifiers as well as allowing an arbitrary amount of information to be stored in them [SZET06, SBR04].

It's important to understand that with the partial exception of cookies, none of these features were envisioned as user tracking mechanisms: it is simply much easier to design protocols with long-term stable identifiers. Even explicit state storage and linkage mechanisms like cookies and TLS session identifiers serve important protocol design purposes: avoiding unnecessary activity, whether computation (in the case of session identifiers) or repeated user data input (in the case of cookies). However, clearly any mechanism for creating and/or observing stable, unique, state on a remote computer can also be used as a tracking mechanism.

In the rest of this paper we discuss some available and potential countermeasures and explore the level of user experience they enable. We focus mainly on the Web context, as the Web is the primary vehicle through which most users experience the Internet.

Limitations of the Available Toolchain

These privacy issues are of course well-known, and tools are available to mitigate them. The best known of these is Tor [1]. but all four major browsers (Chrome, Firefox, IE, and Safari) provide "private browsing" modes which are designed to minimize state storage.

Tor actually consists of a number of privacy tools, most notably:

- A network of Tor proxies through which traffic is routed, thus allowing the user to mask their IP address and other network-layer endpoint characteristics from sites they visit (and of course from other network observers).
- A Firefox plugin (Torbutton) which attempts to disable/sanitize various Web features which otherwise would allow tracking. (Private browsing features provide many of the same functions.)

Unsurprisingly, there is a substantial price to pay for this functionality. Sending packets through the Tor router network is significantly slower than sending them directly [DM09, FGK+10]. While the Tor project is attempting to tune the system, the very structure of the network makes it far slower than the network which it is overlayed on, so to some extent this is an inherent limitation.

Torbutton and private browsing extensions come at a real cost as well, primarily in terms of functionality. To get a sense of this one merely has to look at the Tor "Warning" page[2]:

> Torbutton blocks browser plugins such as Java, Flash, ActiveX, RealPlayer, Quicktime, Adobe's PDF plugin, and others: they can be manipulated into revealing your IP address. For example, that means Youtube is disabled. If you really need your Youtube, you can reconfigure Torbutton to allow it; but be aware that you're opening yourself up to potential attack. Also, extensions like Google toolbar look up more information about the websites you type in: they may bypass Tor and/or broadcast sensitive information. Some people prefer using two browsers (one for Tor, one for unsafe browsing).

This is perhaps an acceptable set of properties if your objective is to connect to politically sensitive Web sites, but is far from acceptable for general use. Note that disabling Flash renders this sort of privacy useless even for the paradigmatic case: privately visiting pornographic Web sites, though perhaps HTML5 video streaming will fill this gap.

Private browsing modes are less restrictive but also appear to be circumventable, as shown by Aggrawal et al.[ABJB10] In general, like many Web security mechanisms, private browsing appears to be a set of semi-principled patches on a system that was never designed to deal with the threat model it is now being asked to counter. The inevitable result is an arms race of attack and defense with no clear end in sight. Certainly Aggrawal et al. do not provide a complete prescription for how to build a secure private browsing mode. As Savage and Rescorla [SR08] observe, it is of course possible to run the browser in a virtual machine and then reset it when finished (though care must be taken with randomness [RY10]) but this is too clunky for all but the most sensitive uses and still doesn't resist fingerprinting based on observable browser properties, each of which must be individually suppressed (see, for instance the EFF Panopticlick project).[3]

In short, while it is possible to use Internet services in a way that precludes tracking, the user experience is so impoverished by comparison to the normal experience that it is only practical to do so for situations where privacy is truly important, rather than as a day-to-day practice. While it may in the future be possible to develop tools that close this gap somewhat, the prospects for those tools do not seem particularly encouraging.

Acknowledgements

This paper benefited from discussions with Adam Barth and Cullen Jennings.

[1] `http://www.torproject.org/`
[2] `https://www.torproject.org/download/download.html.en#warning`
[3] `https://panopticlick.eff.org/`

References

[ABJB10] Gaurav Aggrawal, Elie Bursztein, Collin Jackson, and Dan Boneh. An analysis of private browsing modes in modern browsers. In *Proc. of 19th Usenix Security Symposium*, 2010.

[DM09] Roger Dingledine and Steven J. Murdoch. Performance Improvements on Tor or Why Tor is slow and what we're going to do about it. `http://www.torproject.org/press/presskit/2009-03-11-performance.pdf`, mar 2009.

[DR08] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878.

[FGK+10] Benjamin Fabian, Florian Goertz, Steffen Kunz, Sebastian Müller, and Mathias Nitzsche. Privately Waiting - A Usability Analysis of the Tor Anonymity Network. In *Sustainable e-Business Management* , Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, 2010.

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Updated by RFCs 2817, 5785.

[KBC05] Tadayoshi Kohno, Andre Broido, and K.C. Claffy. Remote Physical Device Fingerprinting. *IEEE Transcations on Dependable and Secure Computing*, 2(2), April-June 2005.

[KM00] D. Kristol and L. Montulli. HTTP State Management Mechanism. RFC 2965 (Proposed Standard), October 2000.

[Pos81] J. Postel. Internet Protocol. RFC 791 (Standard), September 1981. Updated by RFC 1349.

[Res00] E. Rescorla. HTTP Over TLS. RFC 2818 (Informational), May 2000. Updated by RFC 5785.

[RY10] Thomas Ristenpart and Scott Yilek. When Good Randomness Goes Bad: Virtual Machine Reset Vulnerabilities and Hedging Deployed Cryptography. In *Proceedings of ISOC NDSS 2010*, 2010.

[SBR04] Hovav Shacham, Dan Boneh, and Eric Rescorla. Client side caching for TLS. *ACM Trans. Info. & System Security*, 7(4):553–75, November 2004.

[SR08] Dan Savage and Eric Rescorla. Savage Love: Guest Expert. The Stranger (Syndicated), January 2008.

[SZET06] J. Salowey, H. Zhou, P. Eronen, and H. Tschofenig. Transport Layer Security (TLS) Session Resumption without Server-Side State. RFC 4507 (Proposed Standard), May 2006. Obsoleted by RFC 5077.