6.858 Lecture 18
PRIVATE BROWSING

What is the goal of privacy?

- Vague ideal: (activity of) a given user is indistinguishable from (activity of) many other users.
- Today we'll discuss privacy in the context of web browsers.
    - There's no formal definition of what private browsing means, in part because web applications are so complicated and so much incriminating state can be generated.
    - Browsers update their implementation of private browsing according to user demand and what other browser vendors do.
    - As users rely on private browsing mode, they expect more from it... and more implementation deficiencies emerge!

What do the browsers mean by "private browsing"?

- Paper formalizes this as two separate threat models + attacks:
    - A local attacker who possesses your machine post-browsing session, and wants to discover which sites you've visited.
    - A web attacker who has compromised a web server that you contact, and wants to link you across private and/or normal sessions.
- If the two attackers can collude, it's easier for them to identify user.
    - Ex: A local attacker asks the server to check for the local IP address in the server's access logs.
    - So, there's practical value to security against these two attacks in isolation.

Threat 1: Local attacker.

- Assumption: Attacker gets control of the user's machine post-session, and wants to learn what sites the user visited in private browsing mode.
- Security goal: Attacker can't learn those sites!
- Non-goals
    - Don't care about achieving privacy for *future* private browsing sessions.
        - Attacker could modify software on the machine (e.g., installing a keystroke logger) and track future browsing.
        - This is why we also assume that the attacker can't access the machine *before* private browsing starts.
    - Hide the fact that private browsing was used.
        - Often called "plausible deniability".
        - The paper says that this is difficult to achieve, but doesn't explain why. Later in the lecture, we'll discuss some potential reasons.

What kinds of persistent client-side state can a private session leak? [By persistent, we mean "stored on the local disk."]

1) JavaScript-accessible state: Cookies, DOM storage

2) Browser cache
3) History of visited addresses
4) Configuration state: New client certificates, updates to saved password database, bookmarks
5) Downloaded files
6) New plugins/browser extensions

- Private browsing implementations all try to prevent persistent leaks to 1, 2, and 3. However, 4, 5, and 6 often persist after the private session ends.
- Network activity can leave persistent evidence---DNS resolution records!
    - To fix this, private browsing mode would need to flush the DNS cache upon session termination. However, this is tricky, because flushing the cache typically requires admin rights on your machine (do you want the browser having admin rights?) and deletes all DNS state, not the state generated by a particular user.
- During private browsing, objects in RAM can also get paged out to disk!

Demo

- Open Firefox in Private Browsing Mode
- Visit http://pdos.csail.mit.edu/

```
sudo gcore $(pgrep firefox)
strings core.* | grep -i pdos
//-e l: Look for string using the
//      character encoding 16-bit
//      little-endian.
//-a:   Scan all of the file.
//-t:   Print the offset within
//      the file.
```

Data lifetime is a broader problem than just private browsing!

- Ex: cryptographic keys or passwords might be problematic if disclosed.
- Ref: http://css.csail.mit.edu/6.858/2010/readings/chow-shredding.pdf

Demo

```
cat memclear.c
cat secret.txt
make memclear
./memclear &
sudo gcore $(pgrep memclear)
strings core.* | grep secret
```

Where does data persist?

- Process memory: heap, stack.
    - Terminal scrollback
    - I/O buffers, X event queues, DNS cache, proxy servers, ...

    - *Language runtime makes copies (e.g., immutable strings in Python)
- Files, file backups, SQLite databases
- Swap file, hibernate file
- Kernel memory:
    - IO buffers: keyboard, mouse inputs
    - Freed memory pages
    - *Network packet buffers
    - *Pipe buffers contain data sent between processes
    - *Random number generator inputs (including keystrokes again).

How could an attacker get a copy of leftover data?

- Files themselves may contain multiple versions (e.g., Word used to support this feature).
- Programs may leak information if they don't scrub memory on deallocation or program shutdown:
    - Ex: In older Linux kernels, up to 4 KB of kernel memory could be leaked to disk when a new directory was created.
    - Ex: If the kernel/VMM doesn't wipe memory pages, then information from process X can leak into process Y that uses X's old memory pages.
- Core dumps
- Direct access to the machine
- Flash SSDs implement logging---they don't erase old data right away!
- Stolen disks, or just disposing of old disks
    - Ref: http://news.cnet.com/2100-1040-980824.html

How can we deal with the data lifetime problems?

- Zero out unused memory [with some performance degradation].
- Encrypt data in places where zeroing out is difficult (e.g., on an SSD).
    - Securely deleting the key means data cannot be decrypted anymore!
    - Ex: OpenBSD swap uses encryption, with a new encryption key generated at boot time.
    - CPU cost of encryption is modest compared to disk I/O.

Threat 2: Web attacker

- Assumptions:
    - Attacker controls the web sites that the user visits.
    - Attacker does not control the user's machine.
    - Attacker wants to detect when the user visits the site.
- Security goals:
    - Attacker cannot identify the user.
    - Attacker cannot determine if the user is employing private browsing mode.

Defending against a web attacker is very difficult!

- What does it mean to identify a user?

    - Link visits by the same user from different private browsing sessions.
    - Link visits by user from private browsing and public browsing sessions.
- Easy way to identify user: IP address.
    - With reasonable probability, requests from the same IP address are the same user.
    - Next lecture, we'll discuss Tor. Tor protects the privacy of the source of a TCP connection (i.e., user's IP). However, Tor doesn't solve other challenges with implementing private browsing.
- Even if the user employs Tor, a web server can still identify her by analyzing the unique characteristics of her browser runtime!

Browser fingerprinting demo.

- Open Chrome, go to http://panopticlick.eff.org/
- Open the same web site in private browsing mode.

- Good way to think of privacy: what is the anonymity set of a user? I.e., what is the largest set of users among which some user is indistinguishable?
- Panopticlick shows that this set is small for most users, because users tend to have unique local settings for fonts, plugins, etc.

- How can a web attacker determine if you're using private browsing mode?
    - Paper describes a history sniffing attack based on link colors.
        - Attacker page loads a URL in an iframe, then creates a link to that URL and sees whether the link is purple (private sessions don't store history).
        - This attack doesn't work any more, since browsers no longer expose link color to JavaScript! [See discussion of history sniffing attacks from a few lectures ago.]
    - However, there may be other ways for the attacker to tell that you're using private mode.
        - Ex: Public-mode cookies cannot be seen by private-mode pages. So, if you visit a page in public mode, and then in private mode, the page can detect that an expected cookie is missing.

How can we provide stronger guarantees for private browsing? [Let's ignore IP address privacy for now, or assume that users employ Tor.]

- **Approach 1: VM-level privacy**
    - Plan:
        - Run each private browsing session in a separate VM.
        - Ensure that the VM is deleted after private browsing is done.
        - Somehow make sure that no VM state ends up on disk [disable paging? Secure deallocation?].

    - Advantages:

        - Strong guarantees against both a local attacker and a web attacker.
        - No changes required to application, just need secure deletion of VM.

    - Drawbacks:
        - Spinning up a separate VM for private browsing is heavyweight.
        - Poor usability: It's harder for users to save files from private browsing, use bookmarks, etc.

    - Inherent trade-off between usability and privacy!

- **Approach 2: OS-level privacy**
    - Plan: Implement similar guarantees at the OS kernel level.
        - A process can run in a "privacy domain", which will be deleted afterwards.

    - Advantages over VM: Lighter-weight.

    - Drawbacks w.r.t VM: Harder to get right, since the OS kernel manages a lot of state.

Are there ways to de-anonymize a user who employs these approaches?

- Maybe the VM itself is unique! So, we need to ensure that all users have similar VMs.
    - This limits the extent to which users can customize VMs.
- Maybe the VMM or host computer introduces some uniqueness.
    - Ex: TCP fingerprinting: The TCP protocol allows some parameters to be set by the implementation (e.g., the initial packet size, the initial TTL).
    - Tools like nmap send carefully crafted packets to a remote server; can guess the remote OS with high likelihood!
- The user is still shared! So, perhaps the attacker can:
    - Detect the user's keystroke timing.
    - Detect the user's writing style. This is called stylography.
    - Ref: http://33bits.org/2012/02/20/is-writing-style-sufficient-to-deanonymize-material-posted-online/

Why do browsers implement their own private browsing support?

- Main reason is deployability: Users don't have to run their browser in a custom VM or OS.
    - Similar motivation for Native Client.
- Another reason is usability: Some types of state generated in private mode should be able to persist after the session is finished. [ex: downloaded files].
    - This is a dangerous plan! Browsers are complicated pieces of software, so it's difficult to find clean cuts in the architecture which allow some types of state (but not others) to persist.

How do we categorize those types of state? The paper says that we should think about who initiated the state change (Section 2.1).

- Initiated by web site, no user interaction: cookies, history, cache.
    - Stays within session, is deleted on session teardown.
- Initiated by web site, requires user interaction: client certificates, saved passwords.
    - Unclear what's the best strategy; browsers tend to store this state persistently, probably because the user has to explicitly authorize the action.
- Initiated by user: bookmarks, file downloads.
    - Same as above: browsers tend to store this state persistently because the user authorizes the action…
    - …but note that storing this state may reveal the fact that the user employed private browsing mode!
        - Ex: In Firefox and Chrome, bookmarks live in a SQLite database. Bookmarks generated in private browsing mode will have empty values for metadata like "last_visit_count".
        - Ref: http://homepages.cs.ncl.ac.uk/feng.hao/files/DPM13.pdf
- Unrelated to a session: Browser updates, certificate revocation list updates.
    - Treat as a single global state shared between public mode and private mode.

What do browsers actually implement?

- Each browser is, of course, different.
- Moreover, some state "bleeds over" in one direction but not another! There isn't a strict partitioning between private mode and public mode state.

- Q: What happens if public state bleeds over into private state?
- A: Easier for web attacker to link private session to public session.
    - Ex: A client-side SSL certificate that's installed in public mode can identify the user in private mode.

- Q: What happens if private state bleeds over into public state?
- A: This helps both a web attacker and a local attacker: observing state from a public session will reveal information about private sessions!

- Q: What should happen to state while user remains in a private mode session?
- A: Most browsers allow state to persist within a private mode session (see
- Table 3).
    - A "no" entry means that a web attacker might be able to detect private mode browsing!

- Q: Why is it OK to allow cookies in private browsing mode?

- A: It's convenient for users to be able to create ephemeral sessions in private browsing mode---the browser will delete the associated cookies when the private session ends.

- Q: What should happen to state across private mode sessions?
- A: Ideally, each private session should start with a blank slate---if state carries over between multiple private sessions, this increases the likelihood that the user can be fingerprinted! However, since some kinds of state can leak from private-to-public, and some kinds of state can leak from public-to-private, some kinds of state can indeed persist across private mode sessions. [Ex: certificates, downloaded items.]
    - So, think of each private mode session as sharing some state with a single public mode.

Browser extensions and plugins are special.
- They are privileged code that can access sensitive state.
- They are not subject to the same-origin policy or other browser security checks.
- Also, they are often developed by someone other than the browser vendor!
    - Thus, they might not be aware of private mode semantics, or they might misimplement the intended policy.
    - However, plugins are probably going to become extinct in the near future! HTML5 offers new features which provide native support for features that used to require Flash, applets, Silverlight, etc.
        - Ref: http://msdn.microsoft.com/en-us/library/ie/hh968248(v=vs.85).aspx
        - Multimedia: <video>, <audio>
        - Graphics: <canvas>, WebGL
        - Offline storage: DOM storage
        - Network: Web sockets, CORS

The paper was written in 2010---what's the current state of private browsing?
- Private browsing is still tricky to get right!
    - Ex: Firefox bug fix from January 2014: The pdf.js extension was allowing public cookies to leak into private-mode HTTP fetches.
        - Ref: https://github.com/mozilla/pdf.js/issues/4234]
        - The extension wasn't checking whether private browsing mode was enabled!
    - Ex: Open Firefox bug from 2011: If you visit a page in private browsing mode and then close the window, you can go to about:memory and find information about the window you supposedly closed (e.g., about:memory will list the URL for the window).
        - Ref: https://bugzilla.mozilla.org/show_bug.cgi?id=709326
        - The problem is that window objects are lazily garbage collected, so closing the window doesn't force a synchronous garbage collection for the window.

        - The bug was "deprioritized when it became clear that the potential solution [was] more involved than original anticipated"; in response, a developer said "That is very sad to hear. This can pretty much defeat the purpose of things such as sessionstore forgetting about closed private windows, etc."
- Off-the-shelf forensics tools can find evidence of private browser sessions.
    - Ex: Magnet's Internet Evidence Finder
        - Ref: http://www.magnetforensics.com/how-private-is-internet-explorers-inprivate-browsing-first-define-private/
        - http://www.magnetforensics.com/how-does-chromes-incognito-mode-affect-digital-forensics/
    - This tool finds private session artifacts for IE, Chrome, and Firefox.
        - During a private session, IE stores objects in the file system. Those objects are deleted upon private session close, but the storage space is not wiped, so private data remains in unallocated disk space.
        - Chrome and Firefox use in-memory SQLite databases during private browsing, so they leave fewer artifacts in the file system. However, like all browsers, they leave artifacts in the page file.

6.858 Computer Systems Security

Fall 2014