Private Browsing

Hoda Maleki
Department of Electrical & Computer Engineering
University of Connecticut
Email: hoda.maleki@engr.uconn.edu

Based on and extracted from Nickolai Zeldovitch, Computer System Security, course material at http://css.csail.mit.edu/6.858/2014/ and paper: "An Analysis of Private Browsing Modes in Modern Browsers"

With help from Marten van Dijk

UCONN

Secure Computation Lab

Private Browsing

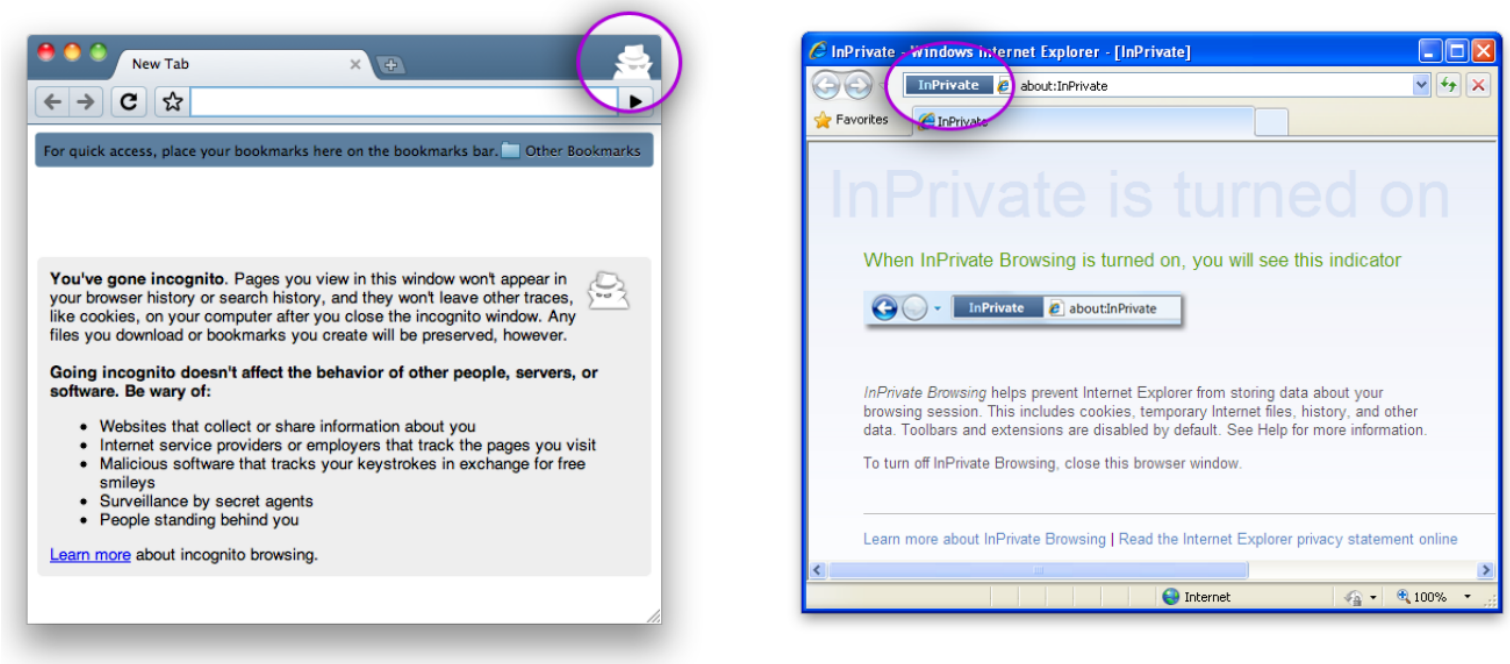
- What do the browsers mean by *Private Browsing*?
  - Privacy Browsing is a privacy feature that disables browsing history and web cache. It also disable the storage of data in cookies. In other word in private mode the activity of a given user should be indistinguishable from the activity of many other users.
- However, there is no formal definition of what exactly private browsing means, because web applications are so complicated and so much incriminating state cam be generated.
- Two attacker types
  - Local attacker: an attacker who possesses the user machine post-browsing session, and wants to discover which site the user visited.
  - Web attacker: an attacker who controls web sites that the user visits and wants to link the user across private and/or normal sessions.

Private Browsing

- Privacy mode has two goals:
  - Privacy against *local attacker*: First and foremost, sites visited while browsing in private mode should leave no trace on the user's computer. In other word it means attacker who takes control of the machine at time T should learn no information about private browsing actions prior to time T.
  - Privacy against *web attacker:* Users may want to hide their identity from web sites they visit by ,for example, making it difficult for web sites to link the user's activities in private mode to the user's activities in public mode. We refer to this as privacy from a web attacker.
- *Notice: If the two attacker can collude, it is easier for them to identify the user, since the local attacker, for example, asks the server to check for the local IP address in the server's access logs. Thus, there is practical value to secure against these two attacks in isolation.*
- While all major browsers (Internet Explorer, Firefox, Chrome and Safari) support private browsing, there is a great deal of inconsistency in the type of privacy provided by the different browsers
  - For example, Firefox and Chrome, attempt to protect against a local attacker and take some steps to protect against a web attacker, while Safari only protects against a local attacker.
- Even within a single browser there are inconsistencies.
  - For example, in Firefox 3.6, cookies set in public mode are not available to the web site while the browser is in private mode. However, passwords and SSL client certificates stored in public mode are available while in private mode. Since web sites can use the password manager as a crude cookie mechanism, the password policy is inconsistent with the cookie policy.

private browsing interface

- User Interface

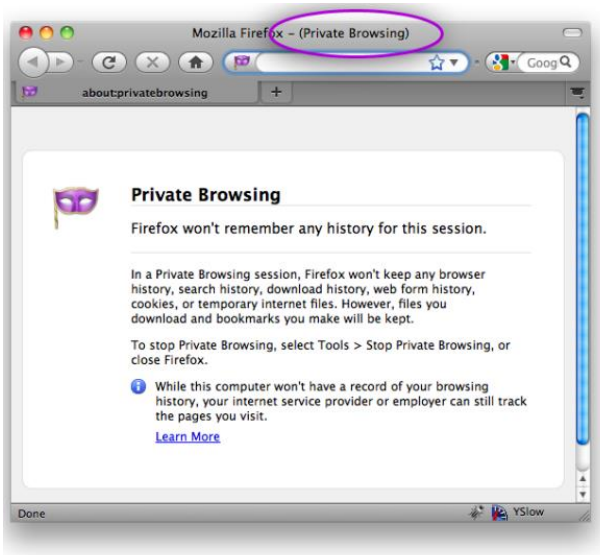


(a) Google Chrome 4

(b) Internet Explorer 8

private browsing interface

- User Interface



(c) Firefox 3.6

(d) Safari 4

Private Browsing

- Browsers update their implementation of private browsing according to user demand and what other browser vendors do.
- Even if a browser adequately implements private browsing, an extension or plug-ins can completely undermine its privacy guarantees.
  - Browser plug-ins and extensions add considerable complexity to private browsing. Thus, Google Chrome disables all extensions while in private mode.
- Therefore, there is inconsistencies between the goals and implementation of private browsing.

Security against local attacker

- Local attacker: an attacker who possesses the user machine post-browsing session, and wants to discover which site the user visited in private mode.
- Security against a local attacker: means that an attacker who takes control of the machine after the user exits private browsing can learn nothing about the user's actions while in private browsing. In other word, the ***goal is that the attacker can't learn the sites which the user visited in private mode.***
- Limitation:
  - The local attacker has no access to the user's machine before the user exits private browsing.
    - Without this limitation, security against a local attacker is impossible; an attacker who has access to the user's machine before or during a private browsing session can simply install a key-logger and record all user actions.

Security against local attacker- Security Model

- Attacker's capabilities
  - The attacker does nothing until the user leaves private browsing mode .
  - While active, the attacker cannot communicate with network elements that contain information about the user's activities while in private mode.

Security against local attacker

- Two non-goals:
  - We do not care about achieving privacy for future private browsing sessions because when the attacker gets access to the machine he may modify software such as keystroke logger, and track future browsing.
  - We do not want to hide the fact that private browsing was used.
- All state changes during private browsing should be erased at the end of a private browsing session. We need adequately erase persistent state changes during a private browsing session:
  1. Changes initiated by a web site without any user interaction, i.e. , setting a cookie, DOM storage, adding an entry to the history file, and adding data to the browser cache.
  2. Changes initiated by a web site, but requiring user interaction, i.e., generating a client certificate or adding a password to the password database.
  3. Changes initiated by the user. For example, creating a bookmark or downloading a file.
  4. Non-user-specific state changes, such as installing a browser patch or updating the phishing block list.

Security against local attacker

- All private browsing implementations try to delete state changes in “category 1” once a private browsing session is terminated. Failure to do so is treated as a private browsing violation. However, changes in the other three categories are in a gray area (the state change in the three other categories often persist after the private session ends).
- Other persistent evidence:
  - Network activity can leave persistent evidence, such as DNS resolution records; in DNS resolution records you can see websites that have been visited through the private browser.
    - The browser will need to ensure that all DNS queries while in private mode do not affect the system’s DNS cache: no entries should be added or removed
    - Demo1: In order to show this leakage, we first delete the DNS resolver cache, using the following instruction:
      - In CMD, we type “ipconfig/flushdns” command. This command flushes the DNS resolver cache.
      - Then, open a website in the private mode, and then write the command “ipconfig/displaydns”; you can see the website name in the DNS resolver cache

Security against local attacker

- During the private browsing, objects in RAM can also get paged out to disk. Operating system can swap memory pages to the swap partition on disk which can leave traces of the user's activity.

- Demo2:
  - To test this out we performed the following experiment on Ubuntu running Firefox:
    - We rebooted the machine to clear RAM and setup and mounted a swap file (zeroed out).
    - Next, we started Firefox, switched to private browsing mode, browsed some websites (such as http://pdos.csail.mit.edu/) and exited private mode but kept Firefox running.
    - Once the browser was in public mode, we ran a memory leak program a few times to force memory pages to be swapped out. Command: "sudo gcore $(pgrep firefox)" for more explanation click here
    - We then ran strings on the swap file and searched for specific words and content of the webpages visited while in private mode. Command: Strings core.* | grep –I pdos

- The experiment shows that the swap file contained some URLs of visited websites, links embedded in those pages and sometimes even the text from a page – enough information to learn about the user's activity in private browsing. This experiment shows that a full implementation of private browsing will need to prevent browser memory pages from being swapped out. **None of the mainstream browsers currently do this.**

Security against local attacker

- Where does data persist?
  - In process memory, such as, heap and stack.
    - Terminal scrollback (more)
    - 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).

local attacker

- The attacker can get a copy of leftover data., but how?
  - Files themselves may contain multiple versions (e.g., Word used to support this feature).
    - Ex: Firefox keeps all the state related to the user's browsing activity including preferences, history, cookies, text entered in forms fields, search queries, etc. in a Profile folder on disk. By observing how and when persistent modifications to these files occur in private mode attacker can learn a great deal about how private mode is implemented in Firefox. Files such as:
      - Security certificate settings (stored in file cert8.db): stores all security certificate settings and any SSL certificates that have been imported into Firefox either by an authorized website or manually by the user. This includes SSL client certificates.
      - Site-specific preferences (stored in file permissions.sqlite): stores many of Firefox permissions that are decided on a per-site basis. For example, it stores which sites are allowed or blocked from setting cookies, installing extensions, showing images, displaying popups, etc.
      - Download actions (stored in file mimeTypes.rdf): the file stores the user's preferences with respect to what Firefox does when it comes across known file types like pdf or avi. It also stores information about which protocol handlers (desktop-based or custom protocol handlers) to launch when it encounters a non-http protocol like mailto.

local attacker

- Programs may leak information if they don't scrub memory on deallocation or program shutdown:
  - In older Linux kernels, up to 4 KB of kernel memory could be leaked to disk when a new directory was created.
  - If the kernel doesn’t wipe memory pages, then information from process X can leak into process Y that uses X’s old memory pages.
- Core dumps: consists of the recorded state of the working memory of a computer program at a specific time. Core dumps may be used to get information about websites visited in privet browsing mode by a local attacker.
- Flash SSDs implement logging: the old data is not erased immediately.
- Stolen disks, or just disposing of old disks

Security against local attacker

- How can we deal with the data lifetime problems?
- Zero out unused memory: In this method we write zero to unused part of memory, however the performance is low.
- Encrypted data in places where zeroing out is difficult: In this method before the application writes anything to the disk it encryption and similarly when it wants to uses the data before putting it into the RAM it decrypt it. This is a better solution specially for SSD, since in SSD writes is expensive while read is cheap.
- Securely deleting the key means data cannot be decrypted anymore!
  - For example OpenBSD: is a Unix-like operating system which includes a number of security features absent or optional in other operating systems. The OpenBSD is noted for its high-quality user documentation. For example, it has the option of doing swap encryption. The swap space is divided into small sections and each section is encrypted with its own key, ensuring that sensitive data doesn't leak into an insecure part of the system. Every time the machine is booted, new encryption keys are generated and are used for swap encryption, and when the machine is rebooted it will forget all the previous keys and this that part of memory can be allocated to new application without leaking information.
    - In practice, the CPU cost of doing encryption is much lower than the actual cost of doing I/O in general.

survey of private browsing in modern browsers

- Internal behavior: This table shows the types of data set in public mode that are available in private mode.

	FF	Safari	Chrome	IE
History	no	yes	no	no
Cookies	no	yes	no	no
HTML5 local storage	no	yes	no	no
Bookmarks	yes	yes	yes	yes
Password database	yes	yes	yes	yes
Form autocompletion	yes	yes	yes	no
User approved SSL self-signed cert	yes	yes	yes	yes
Downloaded items list	no	yes	yes	n/a
Downloaded items	yes	yes	yes	yes
Search box search terms	yes	yes	yes	yes
Browser's web cache	no	no	no	no
Client certs	yes	yes	yes	yes
Custom protocol handlers	yes	n/a	n/a	n/a
Per-site zoom level	no	n/a	yes	n/a

Table 1: Is the state set in earlier public mode(s) accessible in private mode?

survey of private browsing in modern browsers

- Internal behavior: This table shows the type of data set in private mode that persists after the user leaves private mode.

	FF	Safari	Chrome	IE
History	no	no	no	no
Cookies	no	no	no	no
~~HTML5 Local storage~~	~~no~~	~~no~~	~~no~~	~~no~~
Bookmarks	yes	yes	yes	yes
Password database	no	no	no	no
~~Form autocompletion~~	~~no~~	~~no~~	~~no~~	~~no~~
User approved SSL self-signed cert	no	yes	yes	yes
Downloaded items list	no	no	no	n/a
Downloaded items	yes	yes	yes	yes
Search box search terms	no	no	no	no
Browser's web cache	no	no	no	no
Client certs	yes	n/a	n/a	yes
Custom protocol handlers	yes	n/a	n/a	n/a
Per-site zoom level	no	n/a	no	n/a

Table 2: Is the state set in earlier private mode(s) accessible in public mode?

survey of private browsing in modern browsers

- Internal behavior: This table shows data that is entered in private mode and persists during that same private mode session.

	FF	Safari	Chrome	IE
History	no	no	no	no
Cookies	yes	yes	yes	yes
HTML5 Local storage	yes	yes	yes	yes
Bookmarks	yes	yes	yes	yes
Password database	no	no	no	no
Form autocompletion	no	no	no	no
User approved SSL self-signed cert	yes	yes	yes	yes
Downloaded items list	yes	no	no	n/a
Downloaded items	yes	yes	yes	yes
Search box search terms	no	no	no	no
Browser's web cache	yes	yes	yes	yes
Client certs	yes	n/a	n/a	yes
Custom protocol handlers	yes	n/a	n/a	n/a
Per-site zoom level	no	n/a	yes	n/a

Table 3: Is the state set in private mode at some point accessible later in the same session?

Security against 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.
- Goal
  - Attacker cannot identify the user. Means:
    - The attacker's web site cannot link a user visiting in private mode to the same user visiting in public mode.
    - The attacker's web site cannot link a user in one private session to the same user in another private session
  - The attacker's web site should not be able to determine whether the browser is currently in private browsing mode

How attacker can identify the user

- The easy way to identify the user is his IP address. Since mostly with reasonable probability the requests from the same IP address are from the same user.
- If the user uses Tor he can hide it's IP address and prevent this type of identity detection. Using Tor solution by itself will not the detection in total since Tor protects the privacy of the source address while it cannot solve other challenges with implementation of private browsing. For example, a web server can identify the user by analyzing the unique characteristics of his browser runtime.
  - Demo 3
    - Open Chrome and go to http://panopticlick.eff.org/
    - Open the same web site in private browsing mode.
    - You can see that the anonymity set of a user is small for most of the users

How can web attacker determine if the user is using private browsing mode?

- Sniffing attack based on link colors.
  - In this attack the attacker page loads a URK in an ifram, then crease a link to this URL and sees whether the link is purple.
    - In private mode all browsers do not add entries to the history database. Consequently, they will color the unique URL link as unvisited. However, in public mode the unique URL will be added to the history database and the browser will render the link as visited. Thus, by reading the link color the attacker learn the browser's privacy state. However this attack does not exists anymore, since browsers no longer expose link color to the JavaScript.
- Use public mode cookies to detect if the user uses the private browsing mode. In this situation if the user visit a page in public mode, and then in the private mode, the page can detect that an expected cookies is missing.

Possible Solutions

- If we consider that the user protects its address privacy by using the Tor application, how can user provide stronger guarantee for private browsing ?
- Two approaches:
  - VM-level privacy
  - OS-level privacy

VM-level privacy

- The user can open each private browsing session in a separate VM and ensure that the VM is deleted after private browsing is done. User should make sure that no VM state ends up on disk by using for example the OpenBSD or secure deallocation.
- Advantage:
  - This method is a strong guarantees against both a local attacker and a web attacker.
  - No changes required to application, just need secure deletion of VM.
- Disadvantage:
- Having a separate VM for each private browsing session is heavyweight.
- It is hard for the users to save files from the private browsing or use the bookmarks, etc.

- In order to have high privacy should we sacrifice the usability?! Trade-off!

OS-level privacy

- The user is able to implement similar guarantees at the OS kernel level by running each process in privacy domain.
  - The privacy domain acts as the collection of OS-level resources that process uses and once the process dies, the OS looks through all the resources that are in that privacy domain set and then securely deallocates all this resources.
- Advantage:
  - Comparing to VM solution, this approach is lighter-weight
- Disadvantage:
  - Comparing to VM solution, it is harder to get right, since the OS kernel manages a lot of state.
    - VM focus on few lower interfaces increases the confidence that it is able to manage to delete all necessary data, while the OS interpose on individual file systems interfaces, individual network interfaces and etc., which makes it more complicated to find all points in which data can leak.

- Are there ways to de-anonymize a user who employs these approaches?
  - Maybe the VM itself is unique, the web attacker can use the fingerprint of the browser. If this fingerprint for the browser in the VM is unique then the web attacker is able to distinguish this user from other users.
  - Maybe host computers introduce some uniqueness. In other word, even if the user tries to limits the extent which he can customize its web browser, some uniqueness such as TCP fingerprinting can make the web attacker be able to distinguish between this user and others.
  - The web attacker may use tools like nmap which sends crafted packets to a remote server and enables him to guess the remote OS with high likelihood.
  - The user is still shared. So, perhaps the attacker can:
    - Detect the user's keystroke timing. Keystroke typing, is the detailed timing information that describes exactly when each key was pressed and when it was released as a person is typing at a computer keyboard. This is unique per person (all users have unique distributions, for their inter-key press timing, that can potentially be used to fingerprint them), thus the attacker can distinguish the user by using this method.
      - Note: The keystroke rhythms of a user are measured to develop a unique biometric template of the user's typing pattern for future authentication
    - Stylography: Each person has a unique writing style. The attacker's aim is to figure who the user is by looking at his/he writing style (and compare them with written samples of that user).

Why we have private browsing

- If using VM or modified operating systems provides private browsing support, Why do browsers implement their own private browsing support?
  - Deployability: Browser vendors mostly do not want to ask their users to do anything special to use the browser besides install the browser binary itself. Thus, they implement private browsing so that users don't have to run their browser in a custom VM or OS.
  - Usability: Some types of state generated in private mode, such as downloaded files and bookmarks, should be able to persist after the session finished. Using VM or OS-level solution will not keep any state from private browsing mode.
    - However, since browsers are complicated software, it is difficult to find clean cuts in the architecture which allow some types of state, but not others, to persist.

Browser state

- Different types of browser states:
  - Initiated by web site, with no user interaction: these states, i.e. cookies, history, caches, and etc., stay within the session and are deleted on session teardown for the private browsing mode.
  - Initiated by web site, but require user interaction: the strategy for these states, i.e. client certificates, saved passwords, is unclear. Since the user has to explicitly authorize the action, the browsers tend to store these states persistently.
  - Initiated by user: In this situation, for the same reason, since the user has to explicitly authorize the action, the browsers tend to store these states persistently. However, storing these states, i.e. bookmarks, file downloads, etc., may reveal the fact that the user employed private browsing mode.
    - For example, 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”. If this metadata is set to zero or null value it means this bookmark was created in private mode.
    - Thus, it is hard to prevent local attacker from detecting whether the user used private mode or public mode.
- Unrelated to a session: states such as, browser updates and certificate revocation list updates, are treated as a single global state shared between public and private mode.

Browser extensions

- Extensions usually run with very high authority, thus they can access sensitive state.
- They implement new features directly inside the browsers themselves, which is problematic because extensions are often developed by someone who is not the actual browser vendor.
- Implementers of these extensions might not fully understand the security context, in which that extension runs. Thus, the extension may not implement private browsing mode semantics or it might misimplement the intended policy.
- They are not subject to the same-origin policy or other browser security checks.

Example of issues in private browsing

- Mostly the developers are more focused on adding new features to the browsers and they leave the privacy aspects to be take care later. In addition, the interface, which needs to be secure with respect to private browsing mode, that frontier is always getting bigger. Thus, in practice, it is hard to produce a private browsing mode which catches all potential data leaks.
- Examples:
- Firefox bug fix from January, 2014: An extension called pdf.js which was basically a way to look at PDF files using pure HTML5 interfaces. However, this extension was allowing public mode cookies to leak when it was being used in private browsing mode.

Example of issues in private browsing

- how this happens:
- Order of requests and responses are as follows:
  - browser: GET /some.pdf (no cookie)
  - server: HTML login form (+ new cookie)
  - browser: POST /login (with cookie)
  - server: 302 to /some.pdf
  - browser: GET /some.pdf (with cookie)
  - server: 200 OK with appropriate content type and Accept-Ranges: bytes
  - browser: sends another GET request, with a Range header (with cookie and **non-private** cookie)
  - server: 204 Partial Content with non-pdf content (the HTML login form)
- What happened: pdf.js (or Firefox) leaks the cookie to the private browsing session.
- What should happened instead: pdf.js should not leak the cookie from the non-private browsing session. In fact the extension wasn't checking whether private browsing mode was enabled!

Example of issues in private browsing

- 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 in the private mode!
- Demo 4:
  - clean the memory allocation in about:memory. Then open a web page in the private mode and then close the private mode.
  - Next refresh the about:memory and search for the URL of the web site you opened in the private mode.
  - You can find it!
- A user should be confident that no record is kept of sites visited using private browsing. However, as long as the browser is kept open, someone else could see which sites were visited
- The reason is that the window objects are lazily garbage collected, so closing the window doesn't force a synchronous garbage collection for the window.

Conclusion

- Since browsers are complicated software, it is difficult to find clean cuts in the architecture which allow some types of state, but not others, to persist.
- There are solutions that gives better privacy, however is inconvenient for the user.
- Current Browsers still lack of having complete private mode.

Thank You

Linux commands

- gcore: Generate a core file of a running program.
  - gcore [-o filename] pid
- This command generate a core dump of a running program with process ID pid. Produced file is equivalent to a kernel produced core file as if the process crashed (and if "ulimit -c" were used to set up an appropriate core dump limit). Unlike after a crash, after gcore the program remains running without any change.
- “-o filename”
  - The optional argument filename specifies the file name where to put the core dump. If not specified, the file name defaults to core.pid, where pid is the running program process ID.
- Pgrep: looks through the currently running processes and lists the process IDs which match the selection criteria to stdout. All the criteria have to match.
- Strings command: print the strings of printable characters in files.
  - In command written “strings core.* |grep –l pdos” the * is the pid explained in gcore part. back

Reference: http://man7.org/linux/man-pages/man1/gcore.1.html

More data

- Scrollback: is a function that allow the user to go back to view the text which has scrolled off the screen of a text console. This is made possible by a buffer created between the video adapter and the display device called the scrollback buffer.

(https://wiki.archlinux.org/index.php/Scrollback_buffer) back

- TCP fingerprinting: is the passive collection of configuration attributes from a remote device during standard layer 4 network communications. The TCP protocol allows some parameters to be set by the implementation, such as, the initial packet size, the initial TTL, etc. The combination of parameters may then be used to infer the remote machine's operating system, or incorporated into a device fingerprint.

(https://en.wikipedia.org/wiki/TCP/IP_stack_fingerprinting) back

- Nmap (Network Mapper) is a security scanner used to discover hosts and services on a computer network, thus creating a "map" of the network. To accomplish its goal, Nmap sends specially crafted packets to the target host and then analyzes the responses.

(https://en.wikipedia.org/wiki/Nmap) back