# 9.4 Security Session

Verifying a user's identity and access rights for every system function can be tedious. To keep track of who is using the functions and their corresponding access rights, systems establish a security session after a user has logged in successfully. A unique reference to the session object is made available, instead of passing all access rights or re-authenticating a user repeatedly. Queries regarding a user's security properties are delegated to the attached session object via the session reference.

## *Also Known As*

Session, User Environment, Namespace, Localized Globals

## *Example*

Our medieval city is concerned about foreigners entering through its gate. Merchants are welcome, but burglars and thieves shouldn't be let in by the guards. Peasants looking for work in one of the city's workshops are allowed in depending on the demand of the guilds. On the other hand, once a person has entered the city, it is hard for the city dwellers to tell who that person is if they are not a well-known city dweller. Even the night watchman patrolling the city's streets has a hard time knowing how to deal with a stranger. A merchant should be welcomed and protected, while someone else lingering in the streets at night might need to be dealt with.

The problem of city inhabitants and the night watchman is that they do not have equivalent resources to the guards at the city gate, to interrogate and investigate people and check their identity. In addition, it would be annoying to visitors that are welcome if they had to answer the same questions over and over again, such as who they are, where they are going, and what their business is in the city.

The mayor summons the city council to discover a way to keep the city secure while making the city a welcoming as possible for merchants and other guests. Another requirement that comes up at the council meeting is that the city officials would also like to know when a visitor has left.

## *Context*

Your system is shared by multiple users and system components need a way to share (security) data associated with a user. For example, you have already applied CHECK POINT (287).

## Problem

Systems shared by multiple users, either via terminals or via a network, have become commonplace. Instead of single-user non-networked systems like the—now almost extinct—DOS PCs, shared or networked systems need to account for a user's actions and ensure users only have access to areas for which they have privileges. A user therefore needs to be identified and authenticated by the system, as described in Chapter 7, *Identification and Authentication (I&A)*. In addition, shared resources require controlling access to them, as described in Chapter 8, *Access Control Models*.

Different components acting on behalf of a user might need to know which user is activating them and what the user's permissions are. Having every individual component or program within the system identifying, authenticating and authorizing users is annoying to both users and developers. In addition, system components might call each other or work together, and thus need a way to share information about the user without compromising this global data to other users.

For example, when buying from a Web-based on-line store, you want to put items in a shopping cart that is associated with you. Later, the check-out process requires you to approve your credit card information and delivery address. The underlying protocol (HTTP) does not provide a context for multi-step interaction because it is stateless. Accordingly, the on-line store's software needs to associate every click you make in your browser with your identity, your shopping basket contents, and your billing information. In addition, you want the system to forget your credentials after the transaction is complete, either by an explicit sign-off mechanism or by a time-out after no interaction by you, thereby ensuring that forgetting to sign off will not compromise your private data such as your credit card account number.

How do you provide easy access for system components to the security properties and other values related with the current user, without requiring them to identify and authenticate every time they interact with the system or an individual component?

The solution to this problem must resolve the following forces:

- You need to provide access to global values shared by different system components. These values also need to be distinguished for individual users. Simple global variables will therefore not work.
- Such values might change during a user's interaction and might be different between several activity periods of the user's session—for example, the contents of the shopping cart in the on-line store.
- Different components or applications within your system can be interested in different values, and might want to change them or define new ones.
- Passing all shared values around the system for a given user can cause too much overhead and result in bloated interfaces.

- Asking a user for I&A over and over again is annoying, so the system needs a means of associating an action automatically with a user that was previously authenticated.
- After a long period of inactivity, the system needs to re-authenticate a user to prevent misuse and overhead. In other words, the system should automatically sign off inactive users.

## Solution

Introduce a session object that holds all user-relevant shared data. Security information related to the user, especially, is kept in the session object. In addition to the session object that holds the values, the system needs to associate every action a user makes with this session. This can be either implicitly, such as associating every action coming over the user's connection with the session, or explicitly with an identifier like a session cookie that is sent to a user's browser by an on-line store site.

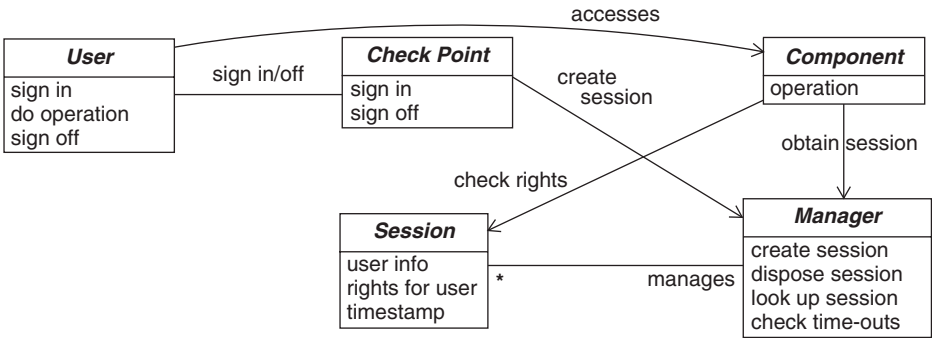
A system's CHECK POINT (287) is the usual place to instantiate the session object and set up its initial values. For systems with access control, the session object can be used to obtain access permissions at sign-on and cache them, to avoid multiple queries to an external database.

In addition, the session object can provide a scratch-pad area to allow different system components to share arbitrary data about the user between different actions within a log-on period. For example, classic mainframe systems provide a so-called 'terminal control block' that can be used by different interactive transaction programs to share data, such as the last values entered in a form. This allows otherwise independent transaction programs to be chained easily on behalf of the user. Web applications use cookies to share data for a given user.

Often a MANAGER [Som98] is used to keep track of active session objects and controls their life cycle. This MANAGER can also be used to provide the mapping of external session identifiers, such as those stored in a session cookie, to the session object and its data. Furthermore, it can collect obsolete sessions that were abandoned by their users.

## Structure

The following diagram shows the component relationships assuming that there is a MANAGER. The CHECK POINT (287) uses the MANAGER to associate a `Session` object with the user. Later, the components accessed by the user rely on the associated `Session` object to access the user's access rights and further information. The `Manager` class uses the timestamp to keep track of stale session objects, and forces the user to re-authenticate if a session is either used for too long without authentication, or if it has not been used for a longer time.




## Dynamics

The following scenario shows a simplified interaction, in which a user logs into the system via the CHECK POINT (287). The CHECK POINT (287) uses the MANAGER to obtain a new `Session` object for the user. The manager does not return the object directly, but instead returns an external session identifier to be used by the user or CHECK POINT (287) for later reference. This scenario assumes that the user explicitly provides his session identifier instead, as would occur with a Web application's session cookie. Other systems can provide an implicit association of a user with his session object—this is not shown here.

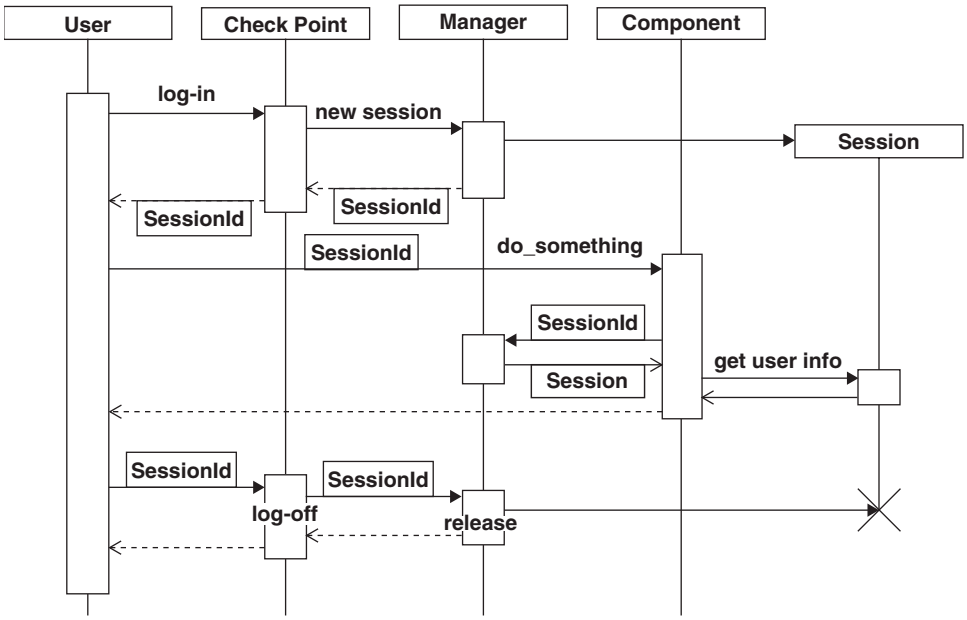
Later on the user interacts with a system component, providing his session identifier for reference. The system component authorizes the user by asking the MANAGER for the underlying session object and checking the user's data stored there.

When the user logs off at the CHECK POINT (287), the MANAGER deletes the `Session` object belonging to the user, invalidating the corresponding session identifier, which no longer can be used. See figure on page 301.

## Implementation

To implement SECURITY SESSION (297) several tasks are required:

1. *Create a session object* to hold all (security) variables associated with the user that may be needed by other components. Typical information kept in the session object are the user's identification, their access rights, the user's role (see ROLE-BASED ACCESS CONTROL (249)), and other system- or application-specific data, such as a shopping cart's content. In addition, you should add a time-stamp when the user logged in successfully, and a time-stamp of the user's last activity. For a flexible solution you might use a data container like PROPERTY LIST [FoYo98] or an ANYTHING [SoRu98] to keep track of varying data without changing code. Web applications might opt to keep the session




data in a cookie in encrypted form. Even when just storing the session identifier in a cookie or URL to keep track of users, such Web applications must ensure those identifiers are not easily guessable, to limit the risk of session hijacking.

2. *Introduce a* MANAGER *and unique session identifiers* to keep track of active session objects. If a user is only allowed to log in once, you might use a user's identification as the session identifier—otherwise a synthesized identifier is sufficient. A publicly-accessible session identifier must be protected against fraud, which in many cases disallows the user's identifier from use directly as their session identifier. Apply MANAGER [Som98] or RESOURCE LIFECYCLE MANAGER [POSA3] as a reference for implementing the MANAGER. The MANAGER provides an interface for other system components to access a session object corresponding to its identifier.
3. *Define session time-out semantics*. Lingering unused session objects carry risk, not only for security reasons, but also for memory management. The MANAGER should periodically check for inactive sessions and release them. If this inactivity time out is short, it effectively prohibits misuse of a session by an unidentified user. On the other hand, if it is shorter than the typical transaction time of a user, such a session time-out gets annoying.
4. *Define re-authentication time-out semantics*. In security-sensitive environments, the MANAGER should also enforce re-authentication at the CHECK

POINT (287) for long-lived active sessions to protect a user's session from misuse and the user from forgetting his password. Appropriate values for such time-outs depend heavily on the given domain and use profile. For example, Yahoo! uses cookies that live for about five years to identify a user. However, from time to time, and whenever accessing sensitive data, a user needs to re-authenticate. In a system in which access rights management is separate, this re-authentication also provides a means of updating a user's access rights that are cached in the session object.

5. *Allow a user to log on and log off at the check point*. Even though it seems trivial, you shouldn't forget to provide the mechanism that allows a user to establish the security session and to allow them to cancel a session of their own will. This actively allows the user to care about security, which can be an important security measure. During log-in the MANAGER creates and initializes the session object with the user's access rights and other relevant data.

## Example Resolved

Our medieval city council comes up with the concept of a day pass. This day pass is issued by the gate guards to every foreign visitor entering the city, and needs to be returned when leaving the city. To distinguish the more desirable guests from less-liked ones, the passes are color coded: peasants looking for work get a green pass, the private visitors of city inhabitants a white one, while merchants receive a bronze pass that they can keep for later visits.

A visitor is obliged to show his pass to everybody asking for it. Since the city is small enough for all citizens to know each other, the city does not need to issue passports for its inhabitants.

## Known Uses

Netscape invented cookies as a means of keeping track of a user's session via the otherwise stateless HTTP protocol [RFC2109]. A user's browser automatically returns a cookie to the originating Web server, effectively passing the session object without the user needing to care about it. Cookies are the means of session tracking for Web applications.

Operating systems such as Unix or Windows use an implicit session object associated with each process in the system. This session object is copied or inherited when a new process is created by its parent process, and only privileged processes such as Unix' log-in program are allowed to set the corresponding session data. For example, in Unix this session object holds the user id and group id of the process owner, among other data. These two imply the corresponding permissions of the process.

Many classic Internet protocols such as FTP and Telnet, as well as many database systems like Oracle or MYSQL, use the TCP/IP connection between a client and a

server as an implicit session mechanism. Each session is thus represented by an individual TCP connection. The termination of the connection also terminates the session. Operating systems and the 16-bit port numbers used in IPv4 place a hard limit on the number of usable sessions.

The open source implementation of SSL (Secure Sockets Layer), openSSL, uses a session id to avoid the expensive re-negotiation of certificates, encryption algorithms, and encryption keys for connections re-established between the same client and server. Some Web security systems use this SSL session mechanism instead of cookies to associate a security session with a user for HTTPS.

## *Consequences*

The following benefits may be expected from applying this pattern:

- The session object provides a single, well-defined place to keep user and security-related data.
- Instead of passing different values around, the system can pass the single session object around for a user.
- Extending the session object to hold new data is straightforward and can be done without impacting unrelated system components.
- The system can use the session object to cache access permissions, thus improving performance.
- It is easy to externalize a session object's identifier when no implicit association between a user and a session object can be achieved, such as with a Web application.
- Checking the associations between sessions and users allows detection of multiple simultaneous uses of the same user credentials, which can be a security compromise, for example, if a user's password is used by several people.

The following potential liabilities may arise from applying this pattern:

- Developers thinking in terms of global variables, such as those the session objects provides, can imply badly-structured programs and uncoordinated or hidden coupling of system components.
- Keeping too many too large session objects around can limit system performance. Special means for collecting session garbage, for example session timeouts, might need to be implemented if users cannot be coerced to log off, or if a single user can initiate multiple sessions simultaneously.
- In a distributed system, session identifiers might be forged by attackers and thus lead to security compromises. Careful design of non-guessable and non-enumerable session identifiers is therefore a must. However, providing such

session identifiers must be automatic, or at least easier for a user than providing their original credentials.

- If all session data is sent to the user's browser in a cookie instead, the cookie needs to be encrypted and signed to avoid security compromises of the session data. Again, authenticating the cookie sent by a user must be easier for the user than providing their original I&A information.
- System components that initially do not need the session object might still keep a reference to it, since components instantiated or called might require it.
- Retro-fitting session objects to a (badly-designed) system relying on SINGLETONS or global variables can be difficult.

## See Also

CHECK POINT (287) typically relies on SECURITY SESSION (297) to provide sign-on functionality for users. If a check point protects multiple systems and those share a single user session, it can provide effective single sign-on for users.

The session object plays the role of an ENCAPSULATED CONTEXT [Kell03] holding several parameters related to the user and their access rights. It is passed through the system as a single parameter, and components of the system can access the encapsulated data via the session object. The ENCAPSULATED CONTEXT avoids wide parameter lists for methods, and ripple effects on changing interfaces when additional user or session-related data is needed.

INTEGRATION REVERSE PROXY (465) and FRONT DOOR (473) rely on SECURITY SESSION (297) to keep track of Web users. They implement it via cookies, SSL session ids, or by encoding the session identifier into URLs.