Generating Session IDs

In this lesson, we'll see how secure session IDs are generated.

We'll cover the following

Using incremental session IDs is a bad choice

It should go without saying, but your session IDs (often stored in cookies) should not resemble a known pattern or be generally guessable. Using an auto-incrementing sequence of integers as IDs would be a terrible choice, as an attacker could just log in, receive the session ID `X` and then replace it with `X ± N`, where `N` is a small number to increase chances of that being an identifier of a recent, valid session.

Generating secure IDs

The simplest choice would be to use a cryptographically secure function that generates a random string. This is usually not a hard task to accomplish. Let's take the Beego framework, very popular among Golang developers, as an example; the function that generates session IDs is

```
package session

import (
        "crypto/rand"
)

// ...
// ...
// ...

func (manager *Manager) sessionID() (string, error) {
        b := make([]byte, manager.config.SessionIDLength)
        n, err := rand.Read(b)
        if n != len(b) || err != nil {
                return "", fmt.Errorf("Could not successfully read from the system CSPRNG")
        }
        return manager.config.SessionIDPrefix + hex.EncodeToString(b), nil
```

```
}
```

Six lines of code and secure session IDs. As we mentioned earlier, nothing else needs to be involved. In this example, a random session ID is generated using Go's native `rand.Read(...)` method (https://golang.org/pkg/math/rand/#Read). This method works by generating random bytes of a given length (in our case, the length of a session ID). Once the ID is generated, it is combined with a session ID prefix.

In general, you won't need to write this code yourself, as frameworks provide the basic building blocks to secure your application out of the box. If you're in doubt, though, you can review the framework's code, or open an issue on GitHub to clarify your security concern.

In the next lesson, we'll see how SQL injections can be avoided.