NWebsec Documentation

Release

André N. Klingsheim

October 07, 2015

Contents

CHAPTER 1

Configuring session security

This is the documentation for the NWebsec.SessionSecurity 1.1 configuration.

Basic configuration is added to web.config when installing the NWebsec.SessionSecurity NuGet package. Here's what you'll see added to web.config (assembly version numbers will vary with releases):

```
<configuration>
  <configSections>
    <sectionGroup name="nwebsec">
      <section name="sessionSecurity" type="NWebsec.SessionSecurity.Configuration.SessionSecurityCon
    </sectionGroup>
  </configSections>

<system.web>
    <sessionState sessionIDManagerType="NWebsec.SessionSecurity.SessionState.AuthenticatedSessionIDMa
</system.web>
  <system.webServer>
    <security>
      <requestFiltering>
        <hiddenSegments>
          <add segment="NWebsecConfig" />
        </hiddenSegments>
      </requestFiltering>
    </security>
  </system.webServer>
  <nwebsec>
<sessionSecurity xmlns="http://nwebsec.com/SessionSecurityConfig.xsd" xmlns:xsi="http://www.w3.org/2
    </sessionSecurity>
  </nwebsec>
</configuration>
```

In short, The session security config section is declared, the sessionState element is configured with the AuthenticatedSessionIDManager, the NWebsec configuration directory is declared as a hidden segment, and an empty session security configuration section is added.

The configuration schema enables intellisense for the NWebsec.SessionSecurity configuration, so feel free to start of with the empty section and add the configuration you need.

Note that everything's disabled by default, so you need to enable the session fixation protection in config.

1.1 Protecting session IDs under machineKey

If your application is set up with machineKeys, NWebsec.SessionSecurity can use the validation key to protect the session IDs. This makes for the simplest configuration:

```
<configuration>

  <nwebsec>
    <sessionSecurity xmlns="http://nwebsec.com/SessionSecurityConfig.xsd" xmlns:xsi="http://www.w3.or
      <sessionIDAuthentication enabled="true" />
    </sessionSecurity>
  </nwebsec>

</configuration>
```

Note that you'll get an exception if your application runs with medium trust, as the machineKey configuration is inaccessible under that trust level.

1.2 Specifying a session authentication key

If your application runs with medium trust, you can specify a session authentication key in the sessionSecurity configuration. You should generate a proper key that's at least 256 bits, shorter keys will not be accepted. Here's the configuration section with the sessionAuthenticationKey specified:

```
<configuration>

  <nwebsec>
    <sessionSecurity xmlns="http://nwebsec.com/SessionSecurityConfig.xsd" xmlns:xsi="http://www.w3.or
      <sessionIDAuthentication enabled="true"
                               useMachineKey="false"
                               authenticationKey="0BFF..." />
    </sessionSecurity>
  </nwebsec>

</configuration>
```

1.2.1 Specifying a session authentication key through AppSettings

You can also specify a key in an AppSetting, and refer to this in the session security configuration. This can be useful for Azure Websites. You should generate a proper key that's at least 256 bits, shorter keys will not be accepted. Here's an example:

```
<configuration>
<appSettings>
  <add key="NWebsecSessionAuthenticationKey" value="0BFF..." />
</appSettings>

  <nwebsec>
    <sessionSecurity xmlns="http://nwebsec.com/SessionSecurityConfig.xsd" xmlns:xsi="http://www.w3.or
      <sessionIDAuthentication enabled="true"
                               useMachineKey="false"
                               authenticationKeyAppsetting="NWebsecSessionAuthenticationKey" />
    </sessionSecurity>
  </nwebsec>
```

```
</configuration>
```

For an Azure website you could set the key in an AppSetting through the portal to keep your secrets out of the web.config.

CHAPTER 2

Authenticated session identifiers

2.1 AuthenticatedSessionIDManager

The **NWebsec.SessionSecurity** library includes the **AuthenticatedSessionIDManager** which manages Authenticated session identifiers. Here are the nitty gritty details on how it works and an account of its security properties.

2.1.1 Authenticated session IDs

Authenticated session IDs are tied to the user through a Message Authentication Code (MAC). They're created in the following way:

"|| denotes concatenation." The key is 256 bits, derived from configured key (see *Key management*).

1. 16 random bytes (128 bit) are generated with the RngCryptoServiceProvider, denoted randomID
2. The MAC is calculated over the username and randomID: mac = HMAC-Sha256(key, username || randomID)
3. The mac is appended to the randomID: binarySessionID = randomID || mac
4. The sessionID is the Base64 encoded binarySessionID: sessionID = Base64Encode(binarySessionID)

Validation of authenticated session IDs follows a similar pattern:

1. Check that the length of the received sessionID is correct (16+32 bytes)
2. Get the first 16 bytes of the received sessionID value, denoted receivedID.
3. Get the next 32 bytes of the received sessionID value, denoted receivedMac.
4. A MAC is calculated over the username and receivedID: mac = HMAC-Sha256(key, username || receivedID)
5. The mac is compared to the receivedMac: result = mac == receivedMac
    - Care has been taken to compare the mac in constant time to avoid timing attacks.

2.1.2 Security

By using a MAC (which relies on a secret key) calculated over a random ID and the username, it is unfeasible for an attacker to guess a valid session ID for another user. Also, an attacker cannot obtain session IDs from the application that would be valid for another user. They would only be valid for the attacker while she's logged on.

Unauthenticated users get the classic ASP.NET session behaviour as the AuthenticatedSessionIDManager falls back to the SessionIDManager behaviour. Hence, unauthenticated users could still be subject to session fixation attacks.

Key management

Using a MAC requires a secret key. The session fixation protection permits the use of the validation key from the machineKey settings or the use of a configured sessionAuthentication key. However, neither of these keys would be used directly. There's an important security principle: Never reuse keys for different purposes. You should live, and die, by this principle.

In fact, they did a massive overhaul on how cryptographic keys were handled in ASP.NET 4.5: Cryptographic Improvements in ASP.NET 4.5, pt. 2. This has inspired how keys are handled in NWebsec.SessionSecurity.

NWebsec.SessionSecurity contains an implementation of the NIST SP800-108 counter-mode KDF with HMAC-SHA256 (similar to what they use in ASP.NET 4.5), and uses this to derive the key used for session authentication. This is important, as the KDF is designed to ensure that derived keys are independent of each other security wise. If one key gets compromised that should not aid an attacker in compromising the master key or any other derived keys.

2.1.3 Authenticated IDs vs. traditional IDs

A design criteria for the AuthenticatedSessionIDManager is that it under no circumstance should offer less security than the traditional ASP.NET SessionIDManager. Considering the Authenticated Session IDs are generated as such:

> key = 256 random bits from a cryptographically strong RNG randomID = 128 random bits from a cryptographically strong RNG username = some string sessionID = Base64Encode(randomID || HMACSHA256(key, username || randomID))

We'll have a look at how these session IDs compare to the traditional ASP.NET session IDs.

Keyspace and randomness

Traditional ASP.NET session IDs are 24 characters long, encoded with a character set that includes a-z and 0-5. Since the character set contains 32 characters, each character in the session ID represents a five bit value. 24 characters representing five bits each yields a 120-bit session ID. Consequently, the session ID key space (number of possible session IDs) is 2^{120} for ASP.NET session IDs.

If we look at the edge case with a constant username we can determine the minimum key space for authenticated session IDs. The MAC is deterministic and does not add randomness, which means that the key space is determined by the randomID. Consequently, the key space in the edge case is 2^{128} for authenticated session IDs. A keyspace of 2^{128} is much larger than one of 2^{120} so the AuthenticatedSessionIDManager holds up to the ASP.NET Session SessionIDManager in terms of the number of possible session IDs.

You might consider the static username an odd case, but it's really not. If there's only one user in the system, this applies. Also, people (mis)use the framework in every (un)thinkable way, so it's a fair bet that someone out there decided it would be a good idea to shared the username between users, and differentiate them by other means. An example would be setting the username to "customer" and misuse that as a role, and keep the "logical" username in session.

If the username is constant, the session fixation protection will not be effective. You'll be back to the current model in ASP.NET session management, but most importantly you won't be worse off than you are today.

In terms of randomness, both the ASP.NET SessionIDManager and the NWebsec AuthenticatedSessionIDManager use the RNGCryptoServiceProvider as their source for random bits. There's not much more to say about that! You can expect the IDs to be random in either case.

Cryptographic security

HMACSHA256 still holds up security wise, which means that you'll get a 256-bit security level as long as you provide a properly generated 256-bit key. The security level refers to how much work an attacker would have to put in to be guaranteed success in a key recovery attack, i.e. the attacker would have to run through all 2^{256} possible keys. On average, an attacker would have to try half the number of keys, which would be 2^{255}. But don't worry, that's still a large number of keys (~5,79 * 10^{76}).

Note that the security level is bound by the key you supply, so if you're using a 128-bit key, you'll get a 128-bit security level. Note that HMACSHA256 will hash keys longer than 256 bit, reducing them to a 256-bit key. So keys that are longer than 256 bit will not add to security (but they won't hurt security).

However, the major concern is not whether the attacker can calculate a valid session ID for another user. The session ID must be a collision, i.e. it would be valid for both the attacker and another user. There are no known attacks to date that would let an attacker generate collisions as long as the secret key remains secret.

You might wonder why HMACSHA256 was chosen for the MAC. Well, it provides good security and it's the default validation algorithm in machine key settings. It's also the required minimum in the Microsoft SDL, version 5.2.

Acknowledgements

We thank crypto wiz @tbj for his invaluable input on how to generate the authenticated session identifiers!

The NWebsec.SessionSecurity library offers protection against session fixation attacks on authenticated users in ASP.NET applications. Using the AuthenticatedSessionIDManager, session IDs are tightly coupled with the identity of the logged in user. The result: Session IDs cannot be reused across users.

2.2 How it works

The AuthenticatedSessionIDManager will check the following when creating/validating a session ID:

- Is sessionIDAuthentication enabled?
- Is the user authenticated? (Identity.IsAuthenticated == true)
- Is the user's name set? (Identity.Name != null or empty)

When all these checks are true an authenticated session ID will be created/validated. Else, it will fall back to the classic ASP.NET SessionIDManager behaviour.

This means that there is no dependency on how identities are set in your application, it will work for applications using Forms Authentication, Windows Identity Foundation, or Windows Authentication. It will also work if you have rolled your own custom solution for setting the user's identity on each request.

Note that cookieless session management is unsupported.

2.3 Common scenarios

We'll discuss the three possible scenarios for applications and user authentication.

2.3.1 Authenticated users only

If your application requires all users to be authenticated, which is often the case for e.g. WIF enabled applications, then authenticated session IDs fit perfectly. All sessions would be authenticated, and you're certain that sessions cannot be shared by users.

Note that if the user already has an authenticated session ID lingering in the browser from a previous session, that will be reused for the next session. If the previous session has not timed out, any data from the previous session will also still be available. This aligns with the default behaviour for ASP.NET session state.

2.3.2 Both anonymous and authenticated users

If your application has open pages that rely on session and requires users to log in for certain functionality, authenticated session identifiers might break your application. An example would be an online shopping site where anonymous users add items to their shopping cart, which is stored in session. When the user logs in to pay for the goods, a new authenticated session would be created and the items in the shopping cart would be lost. In these cases you would have to store the user's data through other means (to e.g. a database) when the user authenticates, and retrieve the data on (or after) the next request when the user gets the new authenticated session. Take care to properly validate any data that is brought from an unauthenticated session to an authenticated session!

2.3.3 Anonymous users only

In this case there would be no authenticated sessions, and you'd get the regular ASP.NET session behaviour. In other words, no need to use the AuthenticatedSessionIDManager.

The NWebsec.SessionSecurity library improves ASP.NET session security by enforcing a strong binding between an authenticated user's identity and the user's session identifier.

You'll find the library on NuGet: NWebsec.SessionSecurity. You can also get it under Releases over at GitHub.

To learn more about how it works, see Authenticated session identifiers. To see how it's configured, refer to Configuring session security.

For background on why the library improves security, see the blog post Ramping up ASP.NET session security.

Did you now that the SDL requires countermeasures against session fixation attacks, and that certain security headers must set by your web application? No? See :doc:‘‘ to learn more.

Check out the NWebsec demo site to see the headers and session security improvements in action.

To keep up with new releases or to give feedback, find @NWebsec on Twitter. You can also get in touch at nwebsec (at) nwebsec (dot) com.