Poiché questa implementazione di `Dispose` non è `synchronized`, è possibile che `Cleanup` sia chiamato da un primo thread e poi un secondo thread prima che `_myObj` sia impostato a `null`. In base a ciò che accade quando viene eseguito il codice di cleanup, si può trattare di un problea di sicurezza o meno.
Un problema importante con l'implementazione di Dispose non sincronizzata comporta un reale problema nella gestione di risorse. La deallocazione impropria può causare una gestione dell'utilizzo errata, che spesso conduce a vulnerabilità di sicurezza.
In alcune applicazioni, potrebbe essere possibile che altri thread accedano ai membri della classe prima che i loro costruttori di classe siano completamente eseguiti. È necessario esaminare tutti i costruttori di classe per assicurarsi che non ci siano problemi di protezione e sincronizzare i thread, se necessario.

7.6.10.10Serializzazione e deserializzazione

Poiché la serializzazione può consentire ad altri moduli di visualizzare o modificare i dati di istanza dell'oggetto che altrimenti sarebbero inaccessibili, è necessaria una autorizzazione speciale per la serializzazione del codice. Per default questa autorizzazione non viene fornita al codice scaricato da internet o intranet; solo al codice sul computer locale è concessa questa autorizzazione.
Normalmente, tutti i campi dell'istanza di un oggetto vengono serializzati, il che significa che vengono serializzati anche i dati. È possibile che il codice possa interpretare il formato per determinare i valori dei dati, indipendentemente dall'accessibilità dei singoli membri. Analogamente, la deserializzazione estrae i dati dalla rappresentazione serializzata e imposta lo stato dell'oggetto direttamente, di nuovo indipendentemente dalle regole di accessibilità.
Qualsiasi oggetto che potrebbe contenere dati sensibili alla sicurezza dovrebbe essere reso non serializable. Se trettamente necessario adottare questa tecnica, occorre comunque creare campi specifici non serializzabili, quelli che - per esempio - contengano dati sensibili. Se questo non può essere fatto, tenere presente che questi dati saranno esposti a qualsiasi modulo che abbia l'autorizzazione a serializzare/deserializzare. In tal caso assicurarsi che nessun modulo non attendibile possa ottenere tale autorizzazione.
L'interfaccia `ISerializable` è destinata esclusivamente all'infrastruttura di serializzazione. Se si fornisce una serializzazione personalizzata implementando `ISerializable`, assicurarsi di adottare le seguenti precauzioni:
Il metodo `GetObjectData` dovrebbe essere protetto in modo esplicito o richiedendo l'autorizzazione SecurityPermission con SerializationFormatter specificata. Occorre anche assicurarsi che non venga rilasciata alcuna informazione sensibile con l'output del metodo.
Esempio:

```
[SecurityPermissionAttribute(SecurityAction.Demand,SerializationFormatter = true)]
public override void GetObjectData(SerializationInfo info,
StreamingContext context)
{
}
```

Il costruttore speciale utilizzato per la serializzazione dovrebbe inoltre eseguire una convalida completa degli input e dovrebbe essere dichiarata private o protected per proteggere la classe da un uso improprio e abusivo.

7.7 ASP

ASP (Active Server Page) identifica non un linguaggio di programmazione, ma una tecnologia Microsoft, per la creazione di pagine web dinamiche attraverso linguaggi di script come VBScript e Microsoft JScript. ASP sfrutta non solo la connettività del web server ma, si può interfacciare (attraverso oggetti COM) con tutte le risorse disponibili sul server e, in maniea trasparente, sfruttare tecnologie diverse.
Vengono di seguito analizzate le principali vulnerabilità e relative contromisure da adottare.

7.7.1 Cross-site scripting (XSS)

Come riconoscerla

Il Cross Site Scripting consiste nella possibilità che un attaccante possa inserire nella pagina dell'applicazione, quindi nel codice HTML, script che, una volta eseguiti, possano trarre in inganno i legittimi utenti, trafugare informazioni e predisporre nuovi attacchi.
Questa minaccia, enormemente diffusa, è dovuta allo scarso controllo dell'input da parte delle web application.
Il Cross Site Scripting può essere reflected o stored. Nel primo caso, uno script inoculato è valido solo all'interno della sessione corrente, ma i suoi effetti possono essere molto dannosi per il sito vittima dell'attacco.
Ancora più grave è il Cross Site Scripting di tipo stored. In questo caso lo script inoculato viene memorizzato come parte intregrante della pagina all'interno di un database e rispristinato ogni qual volta la pagina in questione viene caricata. Quest'attacco è stato sfruttato ampiamente nel recente passato, soprattutto laddove veniva consentito agli utenti di inserire recensioni, commenti e altri contributi.

Volendo schematizzare, possiamo categorizzare gli attacchi XSS nelle seguenti tipologie:

- Reflected XSS, in cui la stringa dannosa proviene dalla richiesta dell'utente.
- Stored XSS, in cui la stringa dannosa proviene dal database del sito Web.

Esiste anche un Cross Site Scripting dovuto a una lacuna nella codifica UTF-7, che permettere di mascherare i caratteri "<" e ">", facendoli sfuggire al controllo. Questa minaccia non è più possibile nei moderni browser, ad eccezione di Microsoft Internet Explorer 11.

Come difendersi

- Validare tutti gli input, indipendentemente dalla loro provenienza. Per la validazione, si consiglia l'approccio white list (sono accettati solo i dati che adottano una struttura specificata nella white list, scartando quelli che non la rispettano). Occorre controllare il tipo del dato, la sua dimensione, l'intervallo di validità (range), il formato ed eventuali valori attesi (white list).
- Codificare completamente tutti i dati dinamici prima di incorporarli (encoding). La codifica dovrebbe essere sensibile al contesto, in base al tipo di dato che si vuole neutralizzare: se ci si aspetta che possa esserci codice HTML abusivo, occorre codificare gli eventuali tag HTML, se ci si potrebbe trovare di fronte a uno script, allora bisogna codificare gli elementi sintattici di Javascript, ecc.
- Si consiglia di utilizzare la libreria di codifica ESAPI di OWASP.
- Nell'intestazione di risposta Content-Type HTTP, è necessario definire esplicitamente la codifica dei caratteri (charset) per l'intera pagina.
- Impostare l'attributo HTTPOnly per proteggere il cookie della sessione da indebite letture da parte di script malevoli.

Esempio:

ASP offre la possibilità di fare l'encoding delle stringhe dell'input (HTMLEncode):

```
<%
  Function XSS_Filter(MyQueryString)
    If IsNumeric(MyQueryString) Then
      MyQueryString = CInt(MyQueryString)
    Else
      MyQueryString = Server.HTMLEncode(MyQueryString)
    End If
    XSS_Filter = MyQueryString
  End Function
%>
Tale funzione verrebbe chiamata in questo modo:
<%
  Dim parametro
  parametro = XSS_Filter(Request.QueryString("parametro"))
%>
```