```
        return Content(output);
    }
```

La versione sicura del codice prevede un controllo che filtri le richieste:

```
    public IActionResult Run(string nomeFile)
    {
     // Se il valore passato è nullo o contiene caratteri
     // diversi dalle lettere minuscole o maiuscole
     // respinge la richiesta
     if (nomeFile == null || !Regex.IsMatch(nomeFile, "^[a-zA-Z]+$"))
     {
      return BadRequest();
     }

     Process p = new Process();
     p.StartInfo.FileName = nomeFile; // adesso è sicuro
     p.StartInfo.RedirectStandardOutput = true;
     p.Start();
     string output = p.StandardOutput.ReadToEnd();
     return Content(output);
    }
```

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/77.html CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection')

7.6.4 Connection String Injection

Come riconoscerla

Questo tipo di attacchi è possibile nel momento in cui l'applicazione affida all'input utente la composiszione dinamica della stringa di connessione al database oppure al server.
Un utente malintenzionato potrebbe manipolare la stringa di connessione dell'applicazione al database oppure al server. Utilizzando strumenti e modifiche di testo semplici, l'aggressore potrebbe essere in grado di eseguire una delle seguenti operazioni:

- Danneggiare le performance delle applicazioni (ad esempio incrementando il valore relativo al MIN POOL SIZE);
- Manomettere la gestione delle connessioni di rete (ad esempio, tramite TRUSTED CONNECTION);
- Dirigere l'applicazione sul database fraudolento anziché a quello genuino;
- Scoprire la password dell'account di sistema nel database (tramite un brute-force attack).

Per comunicare con il proprio database o con un altro server (ad esempio Active Directory), l'applicazione costruisce dinamicamente una sua stringa di connessione. Questa stringa di connessione viene costruita dinamicamente con l'input inserito dall'utente. Se i valori immessi sono stati verificati in misura insufficiente o non sono stati affatto verificati, la stringa di connessione potrebbe essere manipolata ad arte a vantaggio dell'attaccante.

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). In generale, è necessario controllare il tipo del dato, la sua dimensione, l'intervallo di validità (range), il formato ed eventuali valori attesi (white list).
- Evitare di costruire dinamicamente stringhe di connessione. Se è necessario creare dinamicamente una stringa di connessione evitare di includere l'input dell'utente. In ogni caso, utilizzare utilità basate sulla piattaforma, come SqlConnectionStringBuilder di .NET, o almeno codificare l'input validato come il piu' idoneo per la piattaforma utilizzata.
- Le stringhe di connessione possono essere custodite nel file web.config. Si tratta di una scelta migliore rispetto a comporle a runtime con l'input dell'utente. Si separa così l'applicazione dai metadati. Il file di configurazione in questione deve essere messo in sicurezza attivano la modalità

"protected configuration", che permette di memorizzare le stringhe di connessione in forma crittografata (encrypted).

Esempi:
Metodo dove sono presentati l'approccio vulnerabile e quello sicuro:

```
public void ProcessRequest(HttpContext contesto)
{
  string nomeUtente = contesto.Request.QueryString["nomeUtente"];

  // Vulnerabile: Uso diretto dell'input dell'utente in una stringa di
  // connessione passata a SqlConnection
  string connectionString = "server=(local);user id=" + nomeUtente +
        ";password= pass;";
  SqlConnection sqlConnectionBad = new SqlConnection(connectionString);

  // Sicuro: Uso di SqlConnectionStringBuilder per includere in modo sicuro
  // l'input dell'utente in una stringa di connessione
  SqlConnectionStringBuilder builder = new SqlConnectionStringBuilder();
  builder["Data Source"] = "(local)";
  builder["integrated Security"] = true;
  builder["user id"] = nomeUtente;
  SqlConnection sqlConnectionGood = new
        SqlConnection(builder.ConnectionString);
}
```

Nell'esempio che segue viene evidenziata la sezione che custodisce la stringa di connessione in modalità encrypt nel file di configurazione web.config:

```
<connectionStrings configProtectionProvider="RsaProtectedConfigurationProvider">
<EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
 xmlns="http://www.w3.org/2001/04/xmlenc#">
 <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
 <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
  <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
   <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
   <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
    <KeyName>RSA Key</KeyName>
   </KeyInfo>
   <CipherData>

<CipherValue>RXO/zmmy3sR0iOJoF4ooxkFxwelVYpT0riwP2mYpR3FU+r6BPfvsqb384pohivkyNY7Dm
4lPgR2bE9F7k6TblLVJFvnQu7p7d/yjnhzgHwWKMqb0M0t0Y8DOwogkDDXFxs1UxIhtknc+2a7UGtGh6Di
3N572qxdfmGfQc7ZbwNE=
     </CipherValue>
    </CipherData>
   </EncryptedKey>
  </KeyInfo>
  <CipherData>

<CipherValue>KMNKBuV9nOid8pUvdNLY5I8R7BaEGncjkwYgshW8ClKjrXSM7zeIRmAy/cTaniu8Rfk92
KVkEK83+UlQd+GQ6pycO3eM8DTM5kCyLcEiJa5XUAQv4KITBNBN6fBXsWrGuEyUDWZYm6Eijl8DqRDb11i
+StkBLlHPyyhbnCAsXdz5CaqVuG0obEy2xmnGQ6G3Mzr74j4ifxnyvRq7levA2sBR4lhE5M80Cd5yKEJkt
cPWZYM99TmyO3KYjtmRW/Ws/XO3z9z1b1KohE5Ok/YX1YV0+Uk4/yuZo0Bjk+rErG505YMfRVtxSJ4ee41
8ZMfp4vOaqzKrSkHPie3zIR7SuVUeYPFZbcV65BKCUlT4EtPLgi8CHu8bMBQkdWxOnQEIBeY+TerAee/Si
BCrA8M/n9bpLlRJkUb+URiGLoaj+XHym//fmCclAcveKlba6vKrcbqhEjsnY2F522yaTHcc1+wXUWqif7r
SIPhc0+MT1hB1SZjd8dmPgtZUyzcL51DoChy+hZ4vLzE=
    </CipherValue>
   </CipherData>
  </EncryptedData>
 </connectionStrings>
```

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/99.html,
CWE-99: Improper Control of Resource Identifiers ('Resource Injection').