Injeção de SQL em páginas de validação

Ontem, dia 19/04/2004, estive numa palestra promovida pela Microsoft no Rio de Janeiro. O tema abordado foi segurança em aplicações Web. Muito bem organizado pela empresa e ministrado pelo bom palestrante Guilherme (MCAD).

Lá, obtive um alerta de segurança que me deixou muito assustado: uma prática de autenticação muito comum em vários sítios codificados em ASP e ASP.NET (e em outras linguagens, diga-se de passagem) envolve executar um SELECT numa tabela de um banco de dados para conferir se determinado usuário e senha existem numa relação de usuários autorizados a acessar o site. Por exemplo, para verificar se a maria, cuja senha é yellow, está autorizada a acessar o site, executamos o seguinte comando:

```
select count(*) from Usuario where ID='maria' and Senha='yellow'
```

Obviamente, coletamos os valores como o identificador do usuário e a senha através de formulário HTML gerados pelo ASP.NET em seus respectivos campos, sendo o de senha configurado para não exibir os caracteres realmente digitados. O código ASP.NET para validação, nesse caso, seria algo do tipo:

```
Private Sub btnEntrar_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
  Handles btnEntrar.Click
  Dim cn As New SqlConnection("Server=(local);Integrated Security=SSPI;Database=Login")
  cn.Open()
  Dim cmd As SqlCommand = New SqlCommand("select count(*) from Usuario where ID='" + _
    txtID.Text + "' and Senha='" + txtSenha.Text + "'", cn)
  If cmd.ExecuteScalar() = 1 Then
    Response.Write("Logou no site!")
    Response.End()
  Else
    Response.Write("Conta inválida!")
    Response.End()
  End If
End Sub
```

Supomos no código acima que temos um servidor SQL Server rodando localmente e nele um banco chamado Login, que contém uma tabela chamada Usuario com os campos ID e Senha. Como estamos usando autenticação integrada, o usuário do Windows ASPNET deve ser configurado no servidor para ter acesso a este banco.

Mas afinal, qual seria o problema do código acima?

Tente rodá-lo (o código fonte bem como o script para criação do banco Login são fornecidos) e digite o seguinte conteúdo no campo Usuário (txtID):

```
' or 1=1 --
```

Observe que o primeiro caractere é uma aspas simples e que os dois últimos são duas barras. A string resultante que será rodada no banco de dados será a seguinte:

```
select count(*) from Usuario where ID='' or 1=1 --' and Senha=''
```

Quem entende um pouco sobre o Transact SQL, dialeto do SQL usado pelo banco de dados da Microsoft, sabe que os caracteres traço-traço (--) significam o início de comentário, ou seja, tudo à direita do marcador é ignorado pelo interpretador SQL. Abreviando, o que realmente será executado é:

```
select count(*) from Usuario where ID='' or 1=1
```

Resultado: BINGO para o hacker! O código irá validar o usuário e seu site será acessado como se o usuário existisse. Embora muito comum, essa vulnerabilidade, que é conhecida como injeção de SQL (ou SQL Injection, em Inglês), ainda é encontrada em muitos sítios que exigem autenticação.

No momento que esse problema foi exposto, pensei: o problema desse código é montar dinamicamente os parâmetros usando uma string. Essa prática, além de perigosa do ponto de vista de segurança, gera erros em tempo de execuções difíceis de prever. Para evitar esses problemas, sempre fiz a passagem de parâmetros, seja em ADO ou ADO.NET, através da coleção Parameters do Command.

Mas não fiquei convencido de que meu código estaria imune à vulnerabilidade e resolvi tirar a dúvida com o palestrante. Para meu espanto, ele afirmou que o mesmo ocorria no meu código, afirmação essa ratificada por um colega presente. Isso me fez tremer e, como utilizo um código semelhante ao qual me refiro, tive que tirar a prova. A solução apontada pelo palestrante seria a colocação de um filtro que impedisse que o usuário digitasse caracteres perigosos dentro dos campos usuário e senha. Isso, em ASP.NET, pode ser facilmente implementado usando o controle de servidor RegularExpressionValidator, configurando a propriedade ValidationExpression, por exemplo, para [a-zA-Z0-9]*.

Porém, sempre tive a convicção de que os valores passados através da coleção Parameters do Command eram codificados de tal forma que, mesmo que um parâmetro string contivesse algum caractere reservado pelo SGBD, o mesmo seria reconhecido como conteúdo e não como comando propriamente.

O código abaixo é outra versão do código mostrado acima só que agora passando os parâmetros através da coleção Parameters do Command:

```
Private Sub btnEntrar2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
  Handles btnEntrar2.Click
  Dim cn As New SqlConnection("Server=(local);Integrated Security=SSPI;Database=Login")
  cn.Open()
  Dim cmd As SqlCommand = New SqlCommand("select count(*) from Usuario " + _
    "where ID=@ID and Senha=@Senha", cn)
  cmd.Parameters.Add("@ID", txtID.Text)
  cmd.Parameters.Add("@Senha", txtSenha.Text)
  If cmd.ExecuteScalar() = 1 Then
    Response.Write("Logou no site!")
    Response.End()
  Else
    Response.Write("Conta inválida!")
    Response.End()
  End If
End Sub
```

Rodei o código acima e digitei no campo Usuario (ID, no banco) o conteúdo de injeção de SQL que citei acima. Minha expectativa se confirmou. O código SQL injetado foi entendido apenas como conteúdo, e não como comando, ao contrário do que os prezados colegas haviam afirmado.

Não obstante, considero muito relevante a indicação do palestrante em sempre impedir que o usuário digite caracteres que podem ser entendidos pelo banco de dados como parte do comando SQL.

Marcio Belo é Analista de Sistemas e Professor de Computação e pode ser contatado pela página http://www.mbelo.hpg.com.br