Segurança ABAP – Parte II – Comunicação com o kernel

23/07/2012 14:00

Olá caros zumbis. Vamos continuar nossa (depressiva) caminhada pelas vulnerabilidades ABAP. Se você ainda não leu o *white paper* que eu mencionei na Parte I, [essa é sua chance](), porque agora vamos falar de comunicação com o *kernel*.

CURIOSIDADE: certas raças de cachorro sabem latir em binário.

<MOMENTO_SOS_COMPUTADORES>

"MAS QUE QUE É ESSE QUÉRNEU?"

Kernel é o conjunto de programas e bibliotecas de funções principais/centrais de qualquer sistema, responsável por fazer a ponte entre as aplicações e os recursos do sistema operacional.

</MOMENTO_SOS_COMPUTADORES>

Se você já instalou/rodou alguma instância *trial* do Netweaver, ~~meus pêsames~~ você já deve ter visto alguns dos arquivos que fazem parte do *kernel* do SAP: **disp+work**, **msg_server**, **sapcpe**, **strdbs**, entre outros.

O *kernel* do SAP é escrito em C/C++, e certas funções das bibliotecas do *kernel* podem ser chamadas diretamente a partir de código ABAP. Isso é necessário em momentos onde o código ABAP não consegue prover sozinho alguma determinada funcionalidade, tendo que recorrer ao sistema operacional/banco de dados/recursos de C ou C++. As chamadas de *kernel* a partir de ABAP podem ser feitas de três maneiras distintas: *kernel calls*, *system calls* e *kernel methods*.

Kernel calls foram a primeira maneira criada dentro do ABAP para interagir com o *kernel*, e são feitas através do comando CALL 'funcao_c'. No exemplo abaixo, a função C_SAPGPARAM está retornando o valor do parâmetro de perfil SAPDBHOST, que indica o *hostname* do banco de dados:

```
DATA v_value TYPE msxxlist-host.

CALL 'C_SAPGPARAM'
ID 'NAME'  FIELD 'SAPDBHOST'
ID 'VALUE' FIELD v_value.
```

Algumas peculiaridades das *kernel calls*:

- Na versão ECC 6.0 existem aproximadamente 370 funções disponíveis para *kernel calls*, com um total de mais de 10.000 utilizações em código standard;
- O comando CALL 'funcao_c' é considerado obsoleto, porém ainda funciona;
- Apesar disso, não há documentação (suficiente) para nenhuma das funções, e além disso a documentação existente diz de maneira clara que essa técnica não deve ser usada fora do código standard;
- Como visto no exemplo, não há nenhuma maneira de identificar quais são os parâmetros de entrada ou de saída, a não ser analisando a própria função.

A *kernel call* mais conhecida é (talvez) a CALL 'SYSTEM', que é usada para executar comandos no *shell* do SO hospedeiro do *application server*. A lista de comandos que podem ser executados é controlada através das transações SM49 / SM69.

"Ah, mas então eu posso escrever uma função malandra em C/C++ e tocar o terror dentro do SAP?" NÃO, AMIZADE, VOCÊ NÃO PODE. A documentação diz que a sua função teria que estar presente no *include* SAPACTAB.H e o *kernel* teria que ser recompilado. Vai explicar essa necessidade pro Basis.

Os outros tipos de chamadas que foram citadas – ***system calls*** e ***kernel methods*** – tem o mesmo objetivo das *kernel calls*, porém são realizadas de formas diferentes: *system calls* são feitas a partir do comando SYSTEM-CALL, com sua própria sintaxe (e que, segundo a documentação, só pode ser usado pela equipe de desenvolvimento Basis da própria SAP) e os *kernel methods* são métodos implementados com a adição BY KERNEL MODULE modulo_c, porém com a implementação vazia e chamando módulos registrados dentro do *include* ABKMETH.H.

Novamente, não vou falar aqui de todas as vulnerabilidades – mesmo porque isso demanda um estudo mais prolongado – mas vou citar aqui as duas funções que o documento da VirtualForge classifica com risco *very high*, ou seja, ONDE O BICHO PEGA:

C_DB_EXECUTE:

```
CALL 'C_DB_EXECUTE'
ID 'STATLEN' FIELD lv_comprimento
ID 'STATTXT' FIELD lv_comando
ID 'SQLERR'  FIELD lv_erro.
```

Com exceção de SELECTs, essa chamada pode executar **qualquer comando SQL diretamente no banco de dados** – incluindo todo tipo de DROP, ALTER, TRUNCATE, etc. Precisa dizer mais alguma coisa?

C_DB_FUNCTION:

```
* Chamando uma stored procedure

CALL 'C_DB_FUNCTION'
ID 'FUNCTION' FIELD 'DB_SQL'
ID 'FCODE'    FIELD 'EP'
ID 'PROCNAME' FIELD lv_nome_procedure
ID 'CONNAME'  FIELD lv_con_name
ID 'CONDA'    FIELD lv_conda
ID 'PARAMS'   FIELD lt_parametros
ID 'ROWCNT'   FIELD lv_contador_linhas
ID 'SQLCODE'  FIELD lv_sql_code
ID 'SQLMSG'   FIELD lv_sql_msg.

* Chamando uma função

CALL 'C_DB_FUNCTION'
ID 'FUNCTION'    FIELD 'DB_SQL'
ID 'FCODE'       FIELD 'PO'
ID 'STMT_STR'    FIELD lv_comando
ID 'CONNAME'     FIELD lv_con_name
ID 'CONDA'       FIELD lv_con_da
```

```
ID 'HOLD_CURSOR' FIELD lv_cursor
ID 'INVALS'      FIELD lt_valores
ID 'CURSOR'      FIELD lv_cursor
ID 'SQLCODE'     FIELD lv_sql_code
ID 'SQLMSG'      FIELD lv_sql_msg.
```

Não bastasse a execução de comandos, com a C_DB_FUNCTION é possível também chamar ***stored procedures* e demais funções SQL disponíveis no BD**. É mole?

Pra colocar a cereja no bolo, essas duas chamadas **não fazem nenhuma verificação de autorizações**.

Mais uma vez, entendo que fica evidente a mensagem aqui: **não execute *kernel calls* diretamente**. Existem alternativas "dentro da lei" para todas as chamadas, inclusive para a famigerada CALL 'SYSTEM' – dá pra trocar pela função SXPG_COMMAND_EXECUTE.

Bom galera, isso era o que tinha de interessante dentro desse *white paper* da VirtualForge, mas ainda assim eu recomendo ler o resto do documento e também o restante do material disponível no site deles. Nem tudo tem a ver com segurança puramente em ABAP, mas é material interessante de qualquer forma.

O assunto segurança em ABAP não acaba por aqui: continuando a série, eu vou tentar falar sobre as bibliotecas *standard* que fazem coisas bacanas como proteção contra vírus e assinaturas digitais. Até a próxima.

Comentários

> **Miguel Motta** — *26/11/2012 18:01*
>
> Pessoal, alguém sabe como vejo o fonte das funções da biblioteca SAPACTAB.H? Tenho quase certeza que meu problema está em umas dessas funções, gostaria de confirmar analisando o código para abrir um chamado na SAP.
> Meu problema está neste método:
> method IF_SXML_READER~NEXT_NODE
> by kernel module fxkmsrd_next_node fail.
> endmethod.
> Da classe CL_SXML_READER
>
> Abraços!

> > **ABAP da Depressão** — *27/11/2012 09:22*
> >
> > Cara, se você tem certeza que algum método standard com BY KERNEL MODULE está com erro, pode abrir direto o chamado na SAP dizendo somente isso, porque duvido que quem vai atender seu chamado vá pedir pra você localizar o erro dentro de um include do kernel.
> >
> > []s

> > > **Miguel Motta** — *27/11/2012 14:34*
> > >
> > > Foi isso que eu fiz!
> > > Vamos ver a resposta!
> > > Mas caso eu queira analisar as funções do kernel é possível?

> **ABAP da Depressão** — *27/03/2013 15:19*
>
> Desculpe o atraso de 4 meses, porém: não, você não pode.

Haroldo Doratiotto — *21/08/2012 10:22*

Comentando sobre o post do Diego Henrique Gomes.
Como eu disse uma vez em outra discussão, que era válido o conhecimento em outras linguagens ou até mesmo um pouco de bando de dados...rsrsrs
No PostgreSQL tem um comando chamado ILIKE.
Ele retorna tanto maiúsculo quanto minúsculo.
SELECT *
FROM TABELA
WHERE CAMPO ILIKE = 'a%';

Bom pessoal, não tem nada a ver com o assunto de segurança, mas é só porque lembrei desse comando quando li o post do nosso amigo.

Abraços...

Diego Henrique Gomes — *16/08/2012 12:22*

Lembrei de algo que fiz uma vez.
Em uma tabela z, uma das colunas armazenava os campos com maiúsculo e minusculo. Porém depois de um tempo o usuário queria buscar SAP e encontrar sap.
Não sei se foi o melhor caminho, mas foi o único que pensei: executei direto no banco a busca. Era algo assim:

EXEC SQL PERFORMING f_append_itab.
SELECT * FROM Z_TABELA INTO :wa_Z_TABELA
WHERE MANDT like :SY-MANDT
upper("BNAME") like :usr02-bname
ENDEXEC.

Acho que é uma outra maneira, mas não sei quanto a segurança, se fica algum log, faz alguma validação... enfim, não tentei fazer "nada de mais" com o banco depois disso, mas penso as vezes se abre vários leques.

> **ABAP da Depressão** — *16/08/2012 20:33*
>
> O standard normalmente resolve esse problema adicionando uma coluna "matchcode" na tabela, onde o texto que precisa ser buscado é gravado todo em maiúsculas. EXEC SQL nesse caso foi meio extremo 😀
>
> Ainda assim, se você estiver controlando a entrada do campo USR02-BNAME, a aplicação não vai ter problemas. EXEC SQL é perigoso em geral porque abre espaço pra usar qualquer comando de *data definition*, mas uma vez que está definido que somente o SELECT é executado, o risco fica para o que for feito posteriormente com os dados da sua tabela Z.
>
> > **Diego Henrique Gomes** — *16/08/2012 22:41*

> Mas eu não queria mudar os dados da tabela, daí o check para gravar só maiúscula eu não poderia marcar... Neste caso acho que o risco foí minimo, mas teria alguma solução alternativa que não fosse gravando os dados em maiúsculo? acho que pra fazer essa busca, só assim mesmo.

> > **ABAP da Depressão** — *16/08/2012 23:02*
> >
> > Acho que não ficou claro, mas o que eu quis dizer é que a coluna matchcode é uma cópia da coluna com capitalização alternada, só que com o texto em maiúsculas. Mesmo assim, sem mudar a estrutura da tabela o jeito é usar mesmo Native SQL como vc fez, porque o OpenSQL infelizmente tem essas limitações.

Felipe Tieppo — *24/07/2012 15:48*

Parabéns!
Estou aguardando o próximo post.

Mauricio Cruz — *23/07/2012 16:18*

Muito bom o post, parabéns!