Solution to Keygenme 8 by qpt^J

gim

March 6, 2011

Ohi. It's my first attempt @ crackmes.de

Since I work, It took me few days two finish it, I've used mostly IDA Pro.

VM loop starts at 0x4012E9, VM code starts at 0x401749 (we'll get back to this one), handlers are in an array at 0x401231.

I've analyzed almost all handlers and gave them some more sensible names. Here's small example:

```
.text:004012A5 dd offset subValFromCell_B_D
.text:004012A9 dd offset andCellsByVal_Bd_Ds
.text:004012AD dd offset pushOnStack_Dp
.text:004012B1 dd offset shlCellPtrValByCell_B_B
.text:004012B5 dd offset orCellPtrValByCell_B_B
.text:004012B9 dd offset subValFromCellPtr_Bd_Ds
.text:004012BD dd offset cmpValWithCellPtr_Bd_Ds
```

The machine uses 8 32-bits registers, which I will call CELLS later, stack with 32-bit values, Z-flag register, and some memory.

The '_B' and '_D' suffixes in handler names means type of argument, which can be BYTE or DWORD. If it's a BYTE in most cases it's a number of a CELL.

I've defined enum with mnemonics based on handler names and some helper structs.

Thanks to this I got all the VM code which you can see in appendix A

I was actully to lazy to write decompiler, mainly cause VM has less than 0x100 instructions. So the stuff in appednix A is mada mainlt by pressing 'Alt+Q' in IDA (that is "struct var").

So basically what the VM does at the beginning is:

- take the input from a dialog box,

- calculate MD2 from the username,
- convert the password into two DWORDS, these are `passDw1` and `passDw2` on calls at 0x401808 and 0x40181D
- when it has that it makes to calls to `subProc` at 0x401832 and 0x401854
- bad guy/good guy depends on the result from the subproc (conditional jumps at 0x401840 and 0x401862

So now let's look at the subProc. As params it takes some 256-byte `magicTable` (we'll get back to that) and a dwords from the "password" field.

I've rewritten piece of the VM code mnemonic (from 0x4018EB) to more understandable pseudo-code (at least imo):

`VM_SETCELLFROMCELLPTR_BD_BS 1 5`	$cell[1] = passDw2$
`VM_SUBVALFROMCELL_B_D 5 dd 0FFFFFFF8h`	
`VM_SETCELLFROMCELLPTR_BD_BS 0 6`	$cell[0] = tbl[tblPtr]$
`VM_ADDVALTO_CELL_B_D 6 dd 4`	$tblPtr += sizeof(DWORD)$
`VM_XORCELLBYCELL 3 3`	$cell[3] = 0$
`VM_ANDCELLS_BD_BS 1 0`	$cell[1] \&= cell[0]$
`VM_JMPIFZ_D dd offset byte_401925`	
`VM_SETCELLFROMCELL_BD_BS 2 1`	$cell[2] = cell[1]$
`VM_SUBVALFROMCELL_B_D 2 dd 1`	$cell[2] -= 1$
`VM_ADDVALTO_CELL_B_D 3 dd 1`	$cell[3] += 1$
`VM_ANDCELLS_BD_BS 1 2`	$cell[1] \&= cell[2]$
`VM_JMPNOZ_D dd offset byte_40190E`	
`401925:`	
`VM_ADDVALTO_CELL_B_D 5 dd 0FFFFFFF4h`	
`VM_SETCELLFROMCELLPTR_BD_BS 1 5`	$cell[1] = passDw2$
`VM_SUBVALFROMCELL_B_D 5 dd 0FFFFFFF4h`	
`VM_SETCELLFROMCELLPTR_BD_BS 0 6`	$cell[0] = tbl[tblPtr]$
`VM_ADDVALTO_CELL_B_D 6 dd 4`	$tblPtr += sizeof(DWORD)$
`VM_XORCELLBYCELL 7 7`	$cell[7] = 0$
`VM_ANDCELLS_BD_BS 1 0`	$cell[1] \&= cell[0]$
`VM_JMPIFZ_D dd offset byte_40195F`	
`401948:`	
`VM_SETCELLFROMCELL_BD_BS 2 1`	$cell[2] = cell[1]$
`VM_SUBVALFROMCELL_B_D 2 dd 1`	$cell[2] -= 1$
`VM_ADDVALTO_CELL_B_D 7 dd 1`	$cell[7] += 1$
`VM_ANDCELLS_BD_BS 1 2`	$cell[1] \&= cell[2]$

So first after performing an AND on a DWORD from `magicTable` and on `passDw1` it counts set bits using Kernighan's method. Then it does the same on another DWORD from `magicTable` and `passDw2`.

And another fragment:

`VM_XORCELLBYCELL 3 7`	$cell[3] \hat{=} cell[7]$
`VM_ANDCELLSBYVALl_BD_DS, 3, 1`	$cell[3] \& = 1$
`VM_ADDVALTO_CELL_B_D, 5, 4`	$cell[5] += 4$
`VM_SHLCELLPTRVALBYCELL_BD_BS, 6, 1`	$(*(DWORD*)cell[5]) <<= cell[1]$
`VM_SUBVALFROMCELL_B_D, 5, 4`	$cell[5] -= 4$
`VM_ADDVALTO_CELL_B_D, 5, 4`	$cell[5] += 4$
`VM_ORCELLPTRVALBYCELL_B_B, 5, 3`	$(*(DWORD*)cell[5]) \mid = cell[3]$
`VM_SUBVALFROMCELL_B_D, 5, 4`	$cell[5] -= 4$
`VM_ADDVALTO_CELL_B_D, 5, 8`	$cell[5] += 8$
`VM_SUBVALFROMCELLPTR_BD_DS, 5, 1`	$(*(DWORD*)cell[5]) -= 1; counter = 0x20$
`VM_SUBVALFROMCELL_B_D, 5, 8`	$cell[5] -= 8$
`VM_ADDVALTO_CELL_B_D, 5, 8`	$cell[5] += 8$
`VM_CMPVALWITHCELLPTR_BD_DS, 5, 0`	$(*(DWORD*)cell[5]) == 0$
`V:M_SUBVALFROMCELL_B_D, 5, 8`	$cell[5] += 8$

So it xors the $bitCount(passDw1 \& magicTable[i])$ with $bitCount(passDw2 \& magicTable[i+1])$ and then it checks the parity and it stores the result os one of the bits in some dword, decrements counter and goes again.

There are 256 bytes in magicTable, in each loop, 2 DWORD are taken, there are 0x20 loops, $0x20 * 2 * sizeof(DWORD) == 256$.

After it calculates all the 0x20 bits it compares them with first of the MD2 calculated DWORDS.

Before we can continue, we need some math.

I suppose you know what matrix is and how matrix multiplication works.

Now imagine matrix in $\mathbb{Z}_2$. Elements of a matrix ar 1's and 0's. Row multuplication becomes AND on bits, Addition is simply a xor.

Now what the VM code above does is basically calculating[1]:

$$A * passDw1 + B * passDw2$$

where both A and B are 32×32 matrices, and both DWORD are column vectors.

Each of the 0x20 loops described above caculates (("row from A" × passDw1) + ("row from B" × passDw2)). Multiplication itself is hidden behind that bitcounting + XOR (at 0x40195F) + AND (at 0x401962)

Then it compares the result with the $md2Dword1$.

Because there are two calls to the `subProc` we get pair of equations:

[1] But it took me some time to realize that, I've just left it went sleeping, and the next day when I was taking a shower enlightenment has come :D

$$\begin{cases} A * passDw1 + B * passDw2 &= md2Dword1 \\ C * passDw1 + D * passDw2 &= md2Dword2 \end{cases}$$

Where A, B, C and D are all matrices of size 32×32.

Due to linearity, we can rewrite the above

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} passDw1 \\ passDw2 \end{bmatrix} = \begin{bmatrix} md2\_Dword1 \\ md2\_Dword2 \end{bmatrix}$$

Where $ABCD$ is a 64×64 matrix, and the two vectors are of sizes 64×1.

Now the only thing that's left is calculate the inverse of the matrix, so the solution is simply

$$\begin{bmatrix} passDw1 \\ passDw2 \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} \begin{bmatrix} md2\_Dword1 \\ md2\_Dword2 \end{bmatrix}$$

I have precalculated the inverse matrix, using Gauss-Jordan elimination (it's in tblz.py).

So basically what the keygen does is it multiplies inverted matrix with a vector made from calculating MD2.

Because this is done only on bits, the multiplication isn't straightforward in the keygen :)

Cheers and thanks to qpt^J for this keygenme :)

A Appendix

```
.text:00401749  stru_401749 | V_D ⟨VM_PUSHONSTACK_D, 100h⟩ ; DATA XREF: sub_4011FF+D8 .  o
.text:0040174E              | V_D ⟨VM_PUSHONSTACK_D, offset aUsername⟩
.text:00401753              | V_D ⟨VM_PUSHONSTACK_D, 3EAh⟩
.text:00401758              | VBD ⟨VM_MOVEFROMMEMTOCELL_B_DP, 0, 403BA0h⟩
.text:0040175E              | V_B ⟨VM_PUSHFROMCELLTOSTACK_B, 0⟩
.text:00401760              | VDB ⟨VM_DOCALL_RESCELL0_D_B, offset GetDlgItemTextA, 10h⟩
.text:00401766              | VBB ⟨VM_ORCELLS_BD_BS, 0, 0⟩
.text:00401769              | V_D ⟨VM_JMPNOZ_D, offset nameFilled⟩
.text:0040176E              | V_D ⟨VM_PUSHONSTACK_D, 0⟩
.text:00401773              | V_D ⟨VM_PUSHONSTACK_D, 0⟩
.text:00401778              | V_D ⟨VM_PUSHONSTACK_D, offset aEnterYourName⟩ ; "Enter your name!"
.text:0040177D              | VBD ⟨VM_MOVEFROMMEMTOCELL_B_DP, 0, 403BA0h⟩
.text:00401783              | V_B ⟨VM_PUSHFROMCELLTOSTACK_B, 0⟩
.text:00401785              | VDB ⟨VM_DOCALL_RESCELL0_D_B, offset j_user32_MessageBoxA, 10h⟩
.text:0040178B              | V_D ⟨VM_MAKEVMJUMP_D, offset byte_4018A6⟩
.text:00401790  nameFilled  | V_D ⟨VM_PUSHONSTACK_D, 100h⟩ ; DATA XREF: .text:00401769 .  o
.text:00401795              | V_D ⟨VM_PUSHONSTACK_D, offset passw⟩
.text:0040179A              | V_D ⟨VM_PUSHONSTACK_D, 3EDh⟩
.text:0040179F              | VBD ⟨VM_MOVEFROMMEMTOCELL_B_DP, 0, 403BA0h⟩
.text:004017A5              | V_B ⟨VM_PUSHFROMCELLTOSTACK_B, 0⟩
.text:004017A7              | VDB ⟨VM_DOCALL_RESCELL0_D_B, offset GetDlgItemTextA, 10h⟩
.text:004017AD              | VBB ⟨VM_ORCELLS_BD_BS, 0, 0⟩
.text:004017B0              | V_D ⟨VM_JMPNOZ_D, offset passFilled⟩
.text:004017B5              | V_D ⟨VM_PUSHONSTACK_D, 0⟩
.text:004017BA              | V_D ⟨VM_PUSHONSTACK_D, 0⟩
.text:004017BF              | V_D ⟨VM_PUSHONSTACK_D, offset aEnterYourSeria⟩ ; "Enter your serial!"
.text:004017C4              | VBD ⟨VM_MOVEFROMMEMTOCELL_B_DP, 0, 403BA0h⟩
.text:004017CA              | V_B ⟨VM_PUSHFROMCELLTOSTACK_B, 0⟩
.text:004017CC              | VDB ⟨VM_DOCALL_RESCELL0_D_B, offset j_user32_MessageBoxA, 10h⟩
.text:004017D2              | V_D ⟨VM_MAKEVMJUMP_D, offset byte_4018A6⟩
.text:004017D7  passFilled  | V_D ⟨VM_CALLFUNC_D, offset zero16b⟩
.text:004017D7              |             ; DATA XREF: .text:004017B0 .  o
.text:004017DC              | V_D ⟨VM_PUSHONSTACK_D, offset aUsername⟩
.text:004017E1              | VDB ⟨VM_DOCALL_RESCELL0_D_B, offset lstrlenA, 4⟩
.text:004017E7              | V_B ⟨VM_PUSHFROMCELLTOSTACK_B, 0⟩
.text:004017E9              | V_D ⟨VM_PUSHONSTACK_D, offset aUsername⟩
.text:004017EE              | VDB ⟨VM_DOCALL_RESCELL0_D_B, offset bufAppend16, 8⟩
.text:004017F4              | V_D ⟨VM_CALLFUNC_D, offset calcMd2⟩
.text:004017F9              | V_D ⟨VM_PUSHONSTACK_D, 8⟩
.text:004017FE              | V_D ⟨VM_PUSHONSTACK_D, offset passDw1⟩
.text:00401803              | V_D ⟨VM_PUSHONSTACK_D, offset passw⟩
.text:00401808              | VDB ⟨VM_DOCALL_RESCELL0_D_B, offset strToInt, 0Ch⟩
.text:0040180E              | V_D ⟨VM_PUSHONSTACK_D, 8⟩
.text:00401813              | V_D ⟨VM_PUSHONSTACK_D, offset passDw2⟩
.text:00401818              | V_D ⟨VM_PUSHONSTACK_D, offset passw2⟩ ; "09876543"
.text:0040181D              | VDB ⟨VM_DOCALL_RESCELL0_D_B, offset strToInt, 0Ch⟩
.text:00401823              | V_D ⟨VM_PUSHONSTACK_D, offset magicTable⟩
.text:00401828              | V_D ⟨VM_PUSHONSTACK_DP, offset passDw2⟩
.text:0040182D              | V_D ⟨VM_PUSHONSTACK_DP, offset passDw1⟩
```

```
.text:00401832                  db VM_PROBABLYVMCALL_D
.text:00401833  savedVmEip      dd offset someSubproc ; DATA XREF: .data:off_4033B8 ^ o
.text:00401833                                        ; .data:00403B65 ^ o
.text:00401837                  VBD ⟨VM_MOVEFROMMEMTOCELL_B_DP, 1, 403E44h⟩
.text:0040183D                  VBB ⟨VM_SUB_401720, 0, 1⟩
.text:00401840                  V_D ⟨VM_JMPNOZ_D, offset stru_401889⟩
.text:00401845                  V_D ⟨VM_PUSHONSTACK_D, offset magicTable+100h⟩
.text:0040184A                  V_D ⟨VM_PUSHONSTACK_DP, offset passDw2⟩
.text:0040184F                  V_D ⟨VM_PUSHONSTACK_DP, offset passDw1⟩
.text:00401854                  V_D ⟨VM_PROBABLYVMCALL_D, offset someSubproc⟩
.text:00401859                  VBD ⟨VM_MOVEFROMMEMTOCELL_B_DP, 1, 403E48h⟩
.text:0040185F                  VBB ⟨VM_SUB_401720, 0, 1⟩
.text:00401862                  V_D ⟨VM_JMPNOZ_D, offset stru_401889⟩
.text:00401867                  V_D ⟨VM_PUSHONSTACK_D, 30h⟩
.text:0040186C                  V_D ⟨VM_PUSHONSTACK_D, 0⟩
.text:00401871                  V_D ⟨VM_PUSHONSTACK_D, offset aWowReallyNiceW⟩ ; "Wow, really nice, waiting for y
.text:00401876                  VBD ⟨VM_MOVEFROMMEMTOCELL_B_DP, 0, 403BA0h⟩
.text:0040187C                  V_B ⟨VM_PUSHFROMCELLTOSTACK_B, 0⟩
.text:0040187E                  VDB ⟨VM_DOCALL_RESCELL0_D_B, offset j_user32_MessageBoxA, 10h⟩
.text:00401884                  V_D ⟨VM_MAKEVMJUMP_D, offset byte_4018A6⟩
.text:00401889  stru_401889     V_D ⟨VM_PUSHONSTACK_D, 10h⟩ ; DATA XREF: .text:00401840 .  o
.text:00401889                                        ; .text:00401862 .  o
.text:0040188E                  V_D ⟨VM_PUSHONSTACK_D, 0⟩
.text:00401893                  V_D ⟨VM_PUSHONSTACK_D, offset aSerialIsInvali⟩ ; "Serial is invalid!"
.text:00401898                  VBD ⟨VM_MOVEFROMMEMTOCELL_B_DP, 0, 403BA0h⟩
.text:0040189E                  V_B ⟨VM_PUSHFROMCELLTOSTACK_B, 0⟩
.text:004018A0                  VBD ⟨VM_DOCALL_RESCELL0_D_B, offset j_user32_MessageBoxA, 10h⟩
.text:004018A6  byte_4018A6     db VM_SUB_401302 ; DATA XREF: .text:0040178B . o
.text:004018A6                                        ; .text:004017D2 .  o ...
.text:004018A7  someSubproc     V_B ⟨VM_PUSHFROMCELLTOSTACK_B, 5⟩
.text:004018A7                                        ; DATA XREF: .text:savedVmEip .  o
.text:004018A7                                        ; .text:00401854 .  o ...
.text:004018A9                  VBB ⟨VM_SETCELLFROMCELL_BD_BS, 5, 4⟩ ; opc
.text:004018AC                  VBD ⟨VM_ADDVALTO_CELL_B_D, 4, 8⟩
.text:004018B2                  V_B ⟨VM_PUSHFROMCELLTOSTACK_B, 3⟩
.text:004018B4                  V_B ⟨VM_PUSHFROMCELLTOSTACK_B, 6⟩
.text:004018B6                  V_B ⟨VM_PUSHFROMCELLTOSTACK_B, 7⟩
.text:004018B8                  VBD ⟨VM_SUBVALFROMCELL_B_D, 5, 0FFFFFFFCh⟩
.text:004018BE                  VBD ⟨VM_SETPTRFROMVAL_BD_DS, 5, 0⟩
.text:004018C4                  VBD ⟨VM_ADDVALTO_CELL_B_D, 5, 0FFFFFFFCh⟩
.text:004018CA                  VBD ⟨VM_SUBVALFROMCELL_B_D, 5, 0FFFFFFF8h⟩
.text:004018D0                  VBD ⟨VM_SETPTRFROMVAL_BD_DS, 5, 20h⟩
.text:004018D6                  VBD ⟨VM_ADDVALTO_CELL_B_D, 5, 0FFFFFFF8h⟩
.text:004018DC                  VBD ⟨VM_ADDVALTO_CELL_B_D, 5, 0FFFFFFF0h⟩
.text:004018E2                  VBB ⟨VM_SETCELLFROMCELLPTR_BD_BS, 6, 5⟩
.text:004018E5                  VBD ⟨VM_SUBVALFROMCELL_B_D, 5, 0FFFFFFF0h⟩
.text:004018EB  stru_4018EB     VBD ⟨VM_ADDVALTO_CELL_B_D, 5, 0FFFFFFF8h⟩
.text:004018EB                                        ; DATA XREF: .text:004019AA ^ o
.text:004018EB                                        ; start
.text:004018F1                  VBB ⟨VM_SETCELLFROMCELLPTR_BD_BS, 1, 5⟩
.text:004018F4                  VBD ⟨VM_SUBVALFROMCELL_B_D, 5, 0FFFFFFF8h⟩
.text:004018FA                  VBB ⟨VM_SETCELLFROMCELLPTR_BD_BS, 0, 6⟩
.text:004018FD                  VBD ⟨VM_ADDVALTO_CELL_B_D, 6, 4⟩
```

```
.text:00401903                VBB ⟨VM_XORCELLBYCELL, 3, 3⟩
.text:00401906                VBB ⟨VM_ANDCELLS_BD_BS, 1, 0⟩
.text:00401909                V_D ⟨VM_JMPIFZ_D, offset gotBitz1⟩
.text:0040190E  moarBitz1     VBB ⟨VM_SETCELLFROMCELL_BD_BS, 2, 1⟩
.text:0040190E                            ; DATA XREF: .text:00401920 ^ o
.text:00401911                VBD ⟨VM_SUBVALFROMCELL_B_D, 2, 1⟩
.text:00401917                VBD ⟨VM_ADDVALTO_CELL_B_D, 3, 1⟩
.text:0040191D                VBB ⟨VM_ANDCELLS_BD_BS, 1, 2⟩
.text:00401920                V_D ⟨VM_JMPNOZ_D, offset moarBitz1⟩
.text:00401925  gotBitz1      VBD ⟨VM_ADDVALTO_CELL_B_D, 5, 0FFFFFFF4h⟩
.text:00401925                            ; DATA XREF: .text:00401909 .  o
.text:0040192B                VBB ⟨VM_SETCELLFROMCELLPTR_BD_BS, 1, 5⟩
.text:0040192E                VBD ⟨VM_SUBVALFROMCELL_B_D, 5, 0FFFFFFF4h⟩
.text:00401934                VBB ⟨VM_SETCELLFROMCELLPTR_BD_BS, 0, 6⟩
.text:00401937                VBD ⟨VM_ADDVALTO_CELL_B_D, 6, 4⟩
.text:0040193D                VBB ⟨VM_XORCELLBYCELL, 7, 7⟩
.text:00401940                VBB ⟨VM_ANDCELLS_BD_BS, 1, 0⟩
.text:00401943                V_D ⟨VM_JMPIFZ_D, offset gotBitz2⟩
.text:00401948  moarBitz2     VBB ⟨VM_SETCELLFROMCELL_BD_BS, 2, 1⟩
.text:00401948                            ; DATA XREF: .text:0040195A ^ o
.text:0040194B                VBD ⟨VM_SUBVALFROMCELL_B_D, 2, 1⟩
.text:00401951                VBD ⟨VM_ADDVALTO_CELL_B_D, 7, 1⟩
.text:00401957                VBB ⟨VM_ANDCELLS_BD_BS, 1, 2⟩
.text:0040195A                V_D ⟨VM_JMPNOZ_D, offset moarBitz2⟩
.text:0040195F  gotBitz2      VBB ⟨VM_XORCELLBYCELL, 3, 7⟩ ; DATA XREF: .text:00401943 .  o
.text:00401962                VBD ⟨VM_ANDCELLSBYVALl_BD_DS, 3, 1⟩
.text:00401968                VBD ⟨VM_ADDVALTO_CELL_B_D, 5, 4⟩
.text:0040196E                VBB ⟨VM_SHLCELLPTRVALBYCELL_BD_BS, 5, 1⟩
.text:00401971                VBD ⟨VM_SUBVALFROMCELL_B_D, 5, 4⟩
.text:00401977                VBD ⟨VM_ADDVALTO_CELL_B_D, 5, 4⟩
.text:0040197D                VBB ⟨VM_ORCELLPTRVALBYCELL_B_B, 5, 3⟩
.text:00401980                VBD ⟨VM_SUBVALFROMCELL_B_D, 5, 4⟩
.text:00401986                VBD ⟨VM_ADDVALTO_CELL_B_D, 5, 8⟩
.text:0040198C                VBD ⟨VM_SUBVALFROMCELLPTR_BD_DS, 5, 1⟩
.text:00401992                VBD ⟨VM_SUBVALFROMCELL_B_D, 5, 8⟩
.text:00401998                VBD ⟨VM_ADDVALTO_CELL_B_D, 5, 8⟩
.text:0040199E                VBD ⟨VM_CMPVALWITHCELLPTR_BD_DS, 5, 0⟩
.text:004019A4                VBD ⟨VM_SUBVALFROMCELL_B_D, 5, 8⟩
.text:004019AA                V_D ⟨VM_JMPNOZ_D, offset stru_4018EB⟩ ; start
.text:004019AF                VBD ⟨VM_SUBVALFROMCELL_B_D, 5, 0FFFFFFFCh⟩
.text:004019B5                VBB ⟨VM_SETCELLFROMCELLPTR_BD_BS, 0, 5⟩
.text:004019B8                VBD ⟨VM_ADDVALTO_CELL_B_D, 5, 0FFFFFFFCh⟩
.text:004019BE                V_B ⟨VM_POPFROMCELLTOSTACK_B, 3⟩
.text:004019C0                V_B ⟨VM_POPFROMCELLTOSTACK_B, 6⟩
.text:004019C2                V_B ⟨VM_POPFROMCELLTOSTACK_B, 7⟩
.text:004019C4                VBD ⟨VM_ADDVALTO_CELL_B_D, 4, 0FFFFFFF8h⟩
.text:004019CA                V_B ⟨VM_POPFROMCELLTOSTACK_B, 5⟩
.text:004019CC                V_B ⟨VM_DOMAKERET_B, 3⟩
```