hvoid code

Code my way

Note： VB function description for reversing

By hvoid / 2016-02-06 / Enigma Group
The Variant variable internal representation:
_____
Signed constant value to the internal representation of the number of bytes
V_EMPTY 0 Empty
V_NULL 1 Null
V_INTEGER 2 Interger 2
V_LONG 3 Long 4
V_SINGLE 4 Single 4
V_DOUBLE 5 Double 8
V_CURRENCY 6 Currency 8
V_DATE 7 Date 8
V_STRING 8 String
V_OLE 9 OLE Automation Object
V_ERROR 10 Error
V_BOOLEAN 11 Boolean 2
V_VARIANT 12 Variant (used only for variant array) 16 (22)
V_OBJECT 13 Object (OLE Automation object)
V_BYTE 17 Byte 1
V_ARRAY 8192 Array
_____
__vbaVarMove; variant variable assignment (generally used for numeric variables)

lea edx, var1; variable address into edx
lea ecx, var2; variable address into ecx
call __ vbaVarMove; variable assigned to the variable 2
; — — — — — — — — — — — — — — — — —-
__vbaVarCopy; variant variable assignment (generally used for string variables)

lea edx, var1; variable address into edx
lea ecx, var2; variable address into ecx
call __ vbaVarMove; variable assigned to the variable 2
; — — — — — — — — — — — — — — — — —-

__vbaVarAdd; variant variable sum +

```
lea eax, var1
PUSH EAX; addend
lea ecx, var2
push ecx; summand 2
lea edx, var3
push edx; result
the call __ vbaVarAdd; variable sum is returned in eax
; ———————————————————-

__vbaVarSub; variant variable subtraction –

lea eax, var1
push eax; minuend.
lea ecx, var2
push ecx; Subtrahend the
lea edx, var3
push edx; result
the call __ vbaVarSub; variable subtraction, returned in eax
; ———————————————————-

__vbaVarMul; variant variables multiplied *

lea eax, var1
Push eax; multiplicand
lea ecx, var2
push ecx; multiplier
lea edx, var3
push edx; result
call __ vbaVarMul; variable multiplied returned in eax
; ———————————————————-

__vbaVarDiv; variant divided by the variable (floating point division) /

lea eax, var1
Push eax; dividend
lea ecx, var2
PUSH ecx; divisor
lea edx, var3
push edx; result
call __ vbaVarDiv; variable dividing returned in eax
; ———————————————————

__vbaVarIdiv; variant divided by the variable (integer division) /

lea eax, var1
Push eax; dividend
lea ecx, var2
PUSH ecx; divisor
lea edx, var3
push edx; result
```

```
call __ vbaVarIdiv; variable dividing returned in eax
; — — — — — — — — — — — — — — — —
__vbaVarMod; variant variable modulo operation Mod

lea eax, var1
Push eax; dividend
lea ecx, var2
PUSH ecx; divisor
lea edx, var3
push edx; result
the call __ vbaVarMod; variable to mold, returned in eax
; — — — — — — — — — — — — — — — —
__vbaVarNeg; variant variable preceded by a minus sign –

lea eax, var1
push eax; variable
lea ecx, var2
push ecx; result
call __ vbaVarNeg; variable is complemented
; — — — — — — — — — — — — — — — —
__vbaVarPow; variant variable exponentiation ^

lea eax, var1
Push eax; base
lea ecx, var2
push ecx; index
lea edx, var3
push edx; result
call __ vbaVarPow; exponentiation returned in eax
; — — — — — — — — — — — — — — — —
__vbaVarTstGt; relational operators>

lea eax, var1
PUSH EAX; variable
lea eax, var2
PUSH EAX; variable 2
call __ vbaVarTstGt; if var1> var2 then ax = & Hffff
; Else ax = 0
; End If
; — — — — — — — — — — — — — — — —
__vbaVarTstGe; relational operator> =

lea eax, var1
PUSH EAX; variable
lea eax, var2
PUSH EAX; variable 2
call __ vbaVarTstGe; if var1> = var2 then ax = & Hffff
; Else ax = 0
; End If
; — — — — — — — — — — — — — — — —
```

```
__vbaVarTstEq; relational operators =

lea eax, var1
PUSH EAX; variable
lea eax, var2
PUSH EAX; variable 2
call __ vbaVarTstEq; if var1 = var2 then ax = & Hffff
; Else ax = 0
; End If
; ——————————————————
__vbaVarTstNe; relational operators <>

lea eax, var1
PUSH EAX; variable
lea eax, var2
PUSH EAX; variable 2
call __ vbaVarTstNe; if var1 <> var2 then ax = & Hffff
; Else ax = 0
; End If
; ——————————————————
__vbaVarTstLt; relational operators <

lea eax, var1
PUSH EAX; variable
lea eax, var2
PUSH EAX; variable 2
call __ vbaVarTstLt; if var1 <var2 then ax = & Hffff
; Else ax = 0
; End If
; ——————————————————
__vbaVarTstLe; relational operators <=

lea eax, var1
PUSH EAX; variable
lea eax, var2
PUSH EAX; variable 2
call __ vbaVarTstLe; if var1 <= var2 then ax = & Hffff
; Else ax = 0
; End If
; ——————————————————
__vbaVarAnd; logical operators And

lea eax, var1
PUSH EAX; variable
lea ecx, var2
PUSH ECX; variable 2
lea edx, var3
push edx; result
the the call __ vbaVarAnd; logic operations, returned in eax
; ——————————————————
```

__vbaVarOr; logical operators Or

```
lea eax, var1
PUSH EAX; variable
lea ecx, var2
PUSH ECX; variable 2
lea edx, var3
push edx; result
the the call __ vbaVarOr; logic operations, returned in eax
; ———————————————————
```

__vbaVarXor; logic operation XOR

```
lea eax, var1
PUSH EAX; variable
lea ecx, var2
PUSH ECX; variable 2
lea edx, var3
push edx; result
the the call __ vbaVarXor; logic operations, returned in eax
; ———————————————————
```

__vbaVarEqv; logical operators Eqv

```
lea eax, var1
PUSH EAX; variable
lea ecx, var2
PUSH ECX; variable 2
lea edx, var3
push edx; result
the call __ vbaVarEqv; logic operations returned in eax
; ———————————————————
```

__vbaVarImp; logical operators Imp

```
lea eax, var1
PUSH EAX; variable
lea ecx, var2
PUSH ECX; variable 2
lea edx, var3
push edx; result
call __ vbaVarImp the; logical operators, returned in eax
; ———————————————————
```

__vbaVarNot; logical operations Not

```
lea eax, var1
PUSH EAX; variable
lea ecx, var2
push ecx; result
the call __ vbaVarNot; logic operations returned in eax
; ———————————————————
; ——————————————————— Below is a function
lea eax, var1; function Abs (num)
```

```
PUSH EAX; parameter numeric
lea ecx, var2
push ecx; result
return result the call __ vbaVarAbs; in eax
; ——————————————————
rtcAnsiValueBstr; function Asc (string)

lea eax, var1
PUSH EAX; parameter string
call rtcAnsiValueBstr; interger results are returned in eax
; ——————————————————
MSVBVM60 # 585; function Atn (num)

PUSH ECX; parameters floating-point numbers, using 8 bytes
push ecx
CALL MSVBVM60 # 585; results are returned in the floating-point stack
; ——————————————————
rtcVarBstrFromAnsi; the functions Chr (integer)

PUSH EAX; parameter integer
call rtcVarBstrFromAnsi; results are returned in eax string
; ——————————————————
rtcCommandVar; function Command () # 670

PUSH EAX; parameter string
call rtcCommandVar; results are returned in eax string
; ——————————————————
rtcCos; function cos (num) # 583

call rtcCos; input parameters in the current top of the stack, 8 bytes, pay attention to the floating-point
FSTP st; results in the top of the floating-point stack
; ——————————————————
rtcCurrentDir; function the CURDIR (string) # 647

lea eax, var1; parameters, a string
push eax
lea edx, var2;
push edx
call rtcCurrentDir; results are returned in eax
; ——————————————————-
rtcGetDateVar; function Date # 610

lea edx, the var1;
push edx
call rtcGetDateVar; results are returned in eax, date (Date)
; ——————————————————-
rtcDateAdd; function DATEADD (string, Double, date) # 661

The push date; 8 bytes Date Date
Push double; 8-byte floating-point double
```

push string; 1-byte characters in the ASCII code, here is the address
push var1; result, date (date)
call rtcDateAdd; results are returned in eax, date (Date)
; ——————————————————- –
rtcDateDiff; function DateDiff (string, date1, date2, …, …) # 662

push 1; default value
push 1; default value
lea eax, var1; Date
push eax
lea ecx, var2; Dates
push ecx
lea edx, var3; string
push edx
lea eax, var4;
push eax
call rtcDateDiff; results are returned in eax, long integer (long)
; ——————————————————- –
rtcDatePart; function DatePart (string, date, …, …) # 663

push 1; default value
push 1; default value
lea eax, var1;
push eax
lea ecx, var2; string
push ecx
lea edx, var3;
push edx
The results are returned in eax, the call rtcDatePart;
; ——————————————————- —
rtcPackDate; function the DateSerial (integer, integer, integer) # 538

lea eax, var1;
push eax
lea ecx, var2; month
push ecx
lea edx, var3; years
push edx
lea eax, var4;
push eax
call rtcPackDate; results are returned in eax, date (Date)
; ——————————————————- —-
rtcGetDateValue; the function DateValue (string)

lea eax, var1; string
push eax
lea edx, var2;
push edx
call rtcGetDateValue; results are returned in eax, date (Date)

; ——————————————————- —-
rtcGetDayOfMonth; function Day (date) # 542

lea eax, var1;
push eax
lea ecx, var2;
push ecx
The results are returned in eax the call rtcGetDayOfMonth;, integer
; ——————————————————- —-
rtcDir; function Dir # 645

lea eax, var1; property,
push eax
lea ecx, var2; path
push ecx
call rtcDir; Results returned in eax, string type (string)
; ——————————————————- —-
rtcExp; function exp # 586

LEA EDX, DWORD PTR SS: [EBP-24]; parameters
PUSH EDX
CALL DWORD PTR DS: [<& MSVBVM60.__vbaR8Var>]; converted to floating point results in the floating-point register
SUB ESP, 8
The FSTP QWORD PTR SS: [ESP]; pushed onto the stack
CALL DWORD PTR DS: [<& MSVBVM60. # 586>]; rtcExp
Result is stored in the stack FSTP QWORD PTR SS: [EBP-2C];
; ——————————————————- —-
rtcFileDateTime; function FileDateTime # 577

LEA EDX, DWORD PTR SS: [EBP-34]
The PUSH engineering 1.004016B0; File name
PUSH EDX; Results
CALL DWORD PTR DS: [<& MSVBVM60. # 577>]; rtcFileDateTime
; After the call results in eax
; ——————————————————- ——
rtcFileLen; function FileLen # 578

The PUSH engineering 1.004016B0; File name
CALL DWORD PTR DS: [<& MSVBVM60. # 578>]; rtcFileLen
; The result in eax
; ——————————————————- ——
__vbaVarFix; function Fix (parameter 1)

LEA EDX, DWORD PTR SS: [EBP-24]
LEA EAX, DWORD PTR SS: [EBP-54]
PUSH EDX; parameters 1
PUSH EAX; returned results
CALL DWORD PTR DS: [<& MSVBVM60.__vbaVarFix>]
MOV EDX, EAX

; ——————————————————- ——
rtcHexVarFromVar; function Hex (num)

lea eax, var1
PUSH EAX; parameter numeric
lea ecx, var2
push ecx; store the result parameters
call rtcHexVarFromVar; string returned in eax
; ——————————————————- ——
rtcGetHourOfDay; function Hour # 543

LEA EAX, DWORD PTR SS: [EBP-34]; Time Date type parameter
LEA ECX, DWORD PTR SS: [EBP-44]; store the result parameters
PUSH EAX
PUSH ECX
CALL DWORD PTR DS: [<& MSVBVM60. # 543>]; Hour
; The results returned in eax
; ——————————————————- ——
rtcImmediateIf IIF (parameter 1, parameter 2, parameter 3)

LEA EDX, DWORD PTR SS: [EBP-54]; Parameter 3
LEA EAX, DWORD PTR SS: [EBP-44]; Parameter 2
PUSH EDX
LEA ECX, DWORD PTR SS: [EBP-34]; parameters 1, that is, the expression
PUSH EAX
LEA EDX, DWORD PTR SS: [EBP-64]; store parameters
PUSH ECX
PUSH EDX
MOV DWORD PTR SS: [EBP-2C], -1
MOV DWORD PTR SS: [EBP-34], 0B
CALL DWORD PTR DS: [<& MSVBVM60. # 681>]; iif
; The results returned in eax
; ——————————————————- ——
__vbaInStrVar; function Instr (starting position of the source string, the target string comparison)

LEA EDX, DWORD PTR SS: [EBP-24]
The PUSH 1; starting position, starting from 1
LEA EAX, DWORD PTR SS: [EBP-34]
PUSH EDX; string to be searched
PUSH EAX; want to search string
LEA ECX, DWORD PTR SS: [EBP-54]
PUSH; comparison
PUSH ECX; returned results
CALL DWORD PTR DS: [<& MSVBVM60.__vbaInStrVar>]
MOV EDX, EAX; Results returned in eax
; ——————————————————- ——
rtcInStrRev; function InStrRev (source string, the target string, starting position, comparison) # 709

XOR ESI, ESI
PUSH ESI; comparison

```
PUSH -1; starting position
LEA EAX, DWORD PTR SS: [EBP-4C]
LEA ECX, DWORD PTR SS: [EBP-24]
PUSH EAX; target string
LEA EDX, DWORD PTR SS: [EBP-48]
PUSH ECX; source string
PUSH EDX; returned results
CALL DWORD PTR DS: [<& MSVBVM60. # 709>]; rtcInStrRev
; The results returned in eax
; The results returned in eax
; ——————————————————————- ——
__vbaVarInt; function Int (parameter 1)

LEA ECX, DWORD PTR SS: [EBP-24]
LEA EDX, DWORD PTR SS: [EBP-54]
PUSH ECX; parameters 1
PUSH EDX; returned results
CALL DWORD PTR DS: [<& MSVBVM60.__vbaVarInt>]
MOV EDX, EAX; Results returned in eax
; ——————————————————————- ——
rtcIsArray; function IsArray # 556
LEA EAX, DWORD PTR SS: [EBP-2C]; parameters ** This is a pointer
PUSH EAX
CALL DWORD PTR DS: [<& MSVBVM60. # 556>]; MSVBVM60.rtcIsArray
; The results returned in eax
; ——————————————————————- ——
rtcIsDate; the function IsDate # 557

LEA EAX, DWORD PTR SS: [EBP-2C]; parameters ** This is a pointer
PUSH EAX
CALL DWORD PTR DS: [<& MSVBVM60. # 557>]; MSVBVM60.rtcIsDate
; The results returned in eax
; ——————————————————————- ——
rtcIsEmpty; the function IsEmpty # 558

LEA EAX, DWORD PTR SS: [EBP-2C]; parameters ** This is a pointer
PUSH EAX
CALL DWORD PTR DS: [<& MSVBVM60. # 558>]; MSVBVM60.rtcIsEmpty
; The results returned in eax
; ——————————————————————- ——
rtcIsError; function isError # 559

LEA EAX, DWORD PTR SS: [EBP-2C]; parameters ** This is a pointer
PUSH EAX
CALL DWORD PTR DS: [<& MSVBVM60. # 559>]; MSVBVM60.rtcIsError
; The results returned in eax
; ——————————————————————- ——
rtcIsMissing; the function IsMissing of # 592

LEA EAX, DWORD PTR SS: [EBP-2C]; parameters ** This is a pointer
```

```
PUSH EAX
CALL DWORD PTR DS: [<& MSVBVM60. # 592>]; MSVBVM60.rtcIsMissing
; The results returned in eax
; ——————————————————- ——
rtcIsNull; the function IsNull # 560

LEA EAX, DWORD PTR SS: [EBP-2C]; parameters ** This is a pointer
PUSH EAX
CALL DWORD PTR DS: [<& MSVBVM60. # 560>]; MSVBVM60.rtcIsNull
; The results returned in eax
; ——————————————————- ——
rtcIsNumeric; the function IsNumeric # 561

LEA EAX, DWORD PTR SS: [EBP-2C]; parameter 1 ** pointer
PUSH EAX
CALL DWORD PTR DS: [<& MSVBVM60. # 561>]; MSVBVM60.rtcIsNumeric
; The results returned in eax
; ——————————————————- ——
rtcIsObject; function IsObject # 562 of

LEA EAX, DWORD PTR SS: [EBP-2C]
PUSH EAX
CALL DWORD PTR DS: [<& MSVBVM60. # 562>]; MSVBVM60.rtcIsObject
; The results returned in eax
; ——————————————————- ——
__vbaLbound; function Lbound

LEA EAX, DWORD PTR SS: [EBP-2C]; parameters 1, the array
PUSH EAX
PUSH; parameters of an array dimension
CALL DWORD PTR DS: [<& MSVBVM60.__vbaLboun>; MSVBVM60.__vbaLbound
; The results returned in eax
; ——————————————————- ——
rtcLowerCaseVar; function Lcase # 518

LEA EDX, DWORD PTR SS: [EBP-24]; parameters 1
LEA EAX, DWORD PTR SS: [EBP-48]; result
PUSH EDX
PUSH EAX
CALL DWORD PTR DS: [<& MSVBVM60. # 518>]; MSVBVM60.rtcLowerCaseVar
; The results returned in eax
; ——————————————————- ——
rtcLeftCharVar; function Left # 617

LEA EDX, DWORD PTR SS: [EBP-24]; parameters 1
PUSH 3; Parameter 2
LEA EAX, DWORD PTR SS: [EBP-48]; result
PUSH EDX
PUSH EAX
CALL DWORD PTR DS: [<& MSVBVM60. # 617>]; MSVBVM60.rtcLeftCharVar
```

; The results returned in eax
; ——————————————————- ——
__vbaLenBstr; function Len

MOV EDX, DWORD PTR SS: [EBP-18]; parameters 1
PUSH EDX
CALL DWORD PTR DS: [<& MSVBVM60.__vbaLenBs>; MSVBVM60.__vbaLenBstr
; The results returned in eax
; ——————————————————- ——
__vbaLenBstrB; function LenB

MOV EAX, DWORD PTR SS: [EBP-18]; parameters 1
PUSH EAX
CALL DWORD PTR DS: [<& MSVBVM60.__vbaLenBs>; MSVBVM60.__vbaLenBstrB
; The results returned in eax
; ——————————————————- ——
RtcLog; function Log # 587

LEA EDX, DWORD PTR SS: [EBP-38]; variable as a parameter
PUSH EDX
CALL DWORD PTR DS: [<& MSVBVM60.__vbaR8Var>; converted into real numbers, the results in the floating-point stack
SUB ESP, 8
The FSTP QWORD PTR SS: [ESP]; parameters onto the stack
CALL DWORD PTR DS: [<& MSVBVM60. # 587>]; MSVBVM60.rtcLog
; The results of the floating-point stack
; ——————————————————- ——
rtcLeftTrimVar; function the LTRIM # 522

LEA ECX, DWORD PTR SS: [EBP-68]; parameters 1
LEA EDX, DWORD PTR SS: [EBP-58]; result
PUSH ECX
PUSH EDX
CALL DWORD PTR DS: [<& MSVBVM60. # 522>]; MSVBVM60.rtcLeftTrimVar
; The results returned in eax
; ——————————————————- ——
rtcMidCharVar; function Mid

Parameter 3 PUSH EAX;
LEA ECX, DWORD PTR SS: [EBP-58]
PUSH 3; Parameter 2
LEA EDX, DWORD PTR SS: [EBP-48]
PUSH ECX; parameters 1
PUSH EDX; Results
CALL DWORD PTR DS: [<& MSVBVM60. # 632>]; MSVBVM60.rtcMidCharVar
; The results returned in eax
; ——————————————————- ——
rtcGetMinuteOfHour; function Minute # 544

LEA EAX, DWORD PTR SS: [EBP-24]; parameters 1

```
LEA ECX, DWORD PTR SS: [EBP-64]; result
PUSH EAX
PUSH ECX
CALL DWORD PTR DS: [<& MSVBVM60. # 544>]; MSVBVM60.rtcGetMinuteOfHour
; The results returned in eax
; ——————————————————- ——
rtcGetMonthOfYear; function Month # 545

LEA EDX, DWORD PTR SS: [EBP-24]; parameters 1
LEA EAX, DWORD PTR SS: [EBP-64]; result
PUSH EDX
PUSH EAX
CALL DWORD PTR DS: [<& MSVBVM60. # 545>]; MSVBVM60.rtcGetMonthOfYear
; The results returned in eax
; ——————————————————- ——
rtcMonthName; function MonthName # 707

PUSH EAX; parameters 1
CALL DWORD PTR DS: [<& MSVBVM60. # 707>]; MSVBVM60.rtcMonthName
; The results returned in eax
; ——————————————————- ——
rtcMsgBox; function MsgBox

LEA EAX, DWORD PTR SS: [EBP-64]
LEA ECX, DWORD PTR SS: [EBP-54]
PUSH EAX; parameters
LEA EDX, DWORD PTR SS: [EBP-34]
PUSH ECX; parameters
Parameter 3 PUSH EDX;
LEA EAX, DWORD PTR SS: [EBP-24]
Parameter 2 PUSH ESI;
PUSH EAX; parameters 1
CALL DWORD PTR DS: [<& MSVBVM60. # 595>]; MSVBVM60.rtcMsgBox
; The results returned in eax
; ——————————————————- ——
rtcGetPresentDate; function Now # 546

LEA EDX, DWORD PTR SS: [EBP-34]; store the result parameters
PUSH EDX;
CALL DWORD PTR DS: [<& MSVBVM60. # 546>]; Now
; The results returned in eax
; ——————————————————- ——
rtcOctVarFromVar; function Oct (num)

lea eax, var1
PUSH EAX; parameter numeric
lea ecx, var2
push ecx; result
call rtcOctVarFromVar; string returned in eax
; ——————————————————- ——
```

rtcReplace; function Replace # 712

PUSH ESI; parameter 6
PUSH -1; parameters
PUSH 1; parameters
LEA EAX, DWORD PTR SS: [EBP-60]
Parameter 3 PUSH EAX;
LEA EDX, DWORD PTR SS: [EBP-5C]
Parameter 2 PUSH EDX;
LEA EAX, DWORD PTR SS: [EBP-24]
PUSH EAX; parameters 1
CALL DWORD PTR DS: [<& MSVBVM60. # 712>]; MSVBVM60.rtcReplace
; The results returned in eax
; — — — — — — — — — — — — — — — — — —- —-
rtcRgb; function # 588

PUSH 28; Parameter 3
PUSH 1E; Parameter 2
PUSH 14; parameters 1
CALL DWORD PTR DS: [<& MSVBVM60. # 588>]; MSVBVM60.rtcRgb
; The results returned in eax
; — — — — — — — — — — — — — — — — — —- —-
rtcRightCharVar; function Right # 619

LEA EDX, DWORD PTR SS: [EBP-24]
PUSH 3; Parameter 2
LEA EAX, DWORD PTR SS: [EBP-44]
PUSH EDX; parameters 1
PUSH EAX; Results
CALL DWORD PTR DS: [<& MSVBVM60. # 619>]; MSVBVM60.rtcRightCharVar
; The results returned in eax
; — — — — — — — — — — — — — — — — — —- —-
rtcRound; function Round # 714

LEA EDX, DWORD PTR SS: [EBP-24]
Parameter 2 PUSH EDI;
LEA EAX, DWORD PTR SS: [EBP-44]
PUSH EDX; parameters 1
PUSH EAX; Results
CALL DWORD PTR DS: [<& MSVBVM60. # 714>]; MSVBVM60.rtcRound
; The results returned in eax
; — — — — — — — — — — — — — — — — — —- —-
rtcRandomize; function Randomize # 594

LEA EDX, DWORD PTR SS: [EBP-34]
PUSH EDX
CALL DWORD PTR DS: [<& MSVBVM60. # 594>]; MSVBVM60.rtcRandomize
; — — — — — — — — — — — — — — — — — —- —-
rtcRandomNext; the function Rnd # 593

```
LEA EAX, DWORD PTR SS: [EBP-34]
PUSH EAX; parameters 1
CALL DWORD PTR DS: [<& MSVBVM60. # 593>]; MSVBVM60.rtcRandomNext
; The results of the floating-point stack
; ——————————————————- —-
rtcRightTrimVar; the function RTRIM # 524

LEA ECX, DWORD PTR SS: [EBP-68]; parameters 1
LEA EDX, DWORD PTR SS: [EBP-58]; result
PUSH ECX
PUSH EDX
CALL DWORD PTR DS: [<& MSVBVM60. # 524>]; MSVBVM60.rtcRightTrimVar
; The results returned in eax
; ——————————————————- —-
rtcGetSecondOfMinute; the function Sound # 547

LEA EAX, DWORD PTR SS: [EBP-24]; parameters 1
LEA ECX, DWORD PTR SS: [EBP-44]; result
PUSH EAX
PUSH ECX
CALL DWORD PTR DS: [<& MSVBVM60. # 547>]; MSVBVM60.rtcGetSecondOfMinute
; The results returned in eax
; ——————————————————- —-
__vbaR8Sgn; function sgn

PUSH EDX
CALL DWORD PTR DS: [<& MSVBVM60.__vbaR8Var>; MSVBVM60.__vbaR8Var
SUB ESP, 8
The FSTP QWORD PTR SS: [ESP]; parameters 1
CALL DWORD PTR DS: [<& MSVBVM60.__vbaR8Sgn>; MSVBVM60.__vbaR8Sgn
; The results returned ax
; ——————————————————- —-
rtcShell; function Shell # 600

PUSH 1; Parameter 2
PUSH EDX; parameters 1
CALL DWORD PTR DS: [<& MSVBVM60. # 600>]; MSVBVM60.rtcShell
; The results of the floating-point stack
; ——————————————————- —-
rtcSin; function Sin # 582

LEA EDX, DWORD PTR SS: [EBP-24]
PUSH EDX
CALL DWORD PTR DS: [<& MSVBVM60.__vbaR8Var>; MSVBVM60.__vbaR8Var
SUB ESP, 8
The FSTP QWORD PTR SS: [ESP]; parameters 1
CALL DWORD PTR DS: [<& MSVBVM60. # 582>]; MSVBVM60.rtcSin
; The results of the floating-point stack
; ——————————————————- —-
rtcSpaceVar; function Space # 526
```

```
PUSH 5; parameters 1
LEA EDX, DWORD PTR SS: [EBP-34]
PUSH EDX; Results
CALL DWORD PTR DS: [<& MSVBVM60. # 526>]; MSVBVM60.rtcSpaceVar
; The results returned in eax
; ——————————————————- —-
rtcSplit; function Split # 711

PUSH ESI; parameters
LEA EDX, DWORD PTR SS: [EBP-48]
Parameter 3 PUSH -1;
LEA EAX, DWORD PTR SS: [EBP-24]
Parameter 2 PUSH EDX;
LEA ECX, DWORD PTR SS: [EBP-38]
LEA EDX, DWORD PTR SS: [EBP-58]
PUSH EAX; parameters 1
PUSH EDX; Results
CALL DWORD PTR DS: [<& MSVBVM60. # 711>]; MSVBVM60.rtcSplit
; The results returned in eax
; ——————————————————- —-
rtcSqr; function rtcSqr # 614

LEA EDX, DWORD PTR SS: [EBP-24]
PUSH EDX
CALL DWORD PTR DS: [<& MSVBVM60.__vbaR8Var>; MSVBVM60.__vbaR8Var
SUB ESP, 8
The FSTP QWORD PTR SS: [ESP]; parameters 1
CALL DWORD PTR DS: [<& MSVBVM60. # 614>]; MSVBVM60.rtcSqr
; The results of the floating-point stack
; ——————————————————- —-
rtcVarStrFromVar; function Str # 613

LEA EDX, DWORD PTR SS: [EBP-24]
LEA EAX, DWORD PTR SS: [EBP-44]
PUSH EDX; parameters 1
PUSH EAX; Results
CALL DWORD PTR DS: [<& MSVBVM60. # 613>]; MSVBVM60.rtcVarStrFromVar
; The results returned in eax
; ——————————————————- —-
__vbaStrCompVar; function StrComp

LEA EDX, DWORD PTR SS: [EBP-24]
LEA EAX, DWORD PTR SS: [EBP-34]
PUSH EDX; parameters 1
Parameter 2 PUSH EAX;
LEA ECX, DWORD PTR SS: [EBP-54]
Parameter 3 PUSH EDI;
PUSH ECX; Results
CALL DWORD PTR DS: [<& MSVBVM60.__vbaStrCo>; MSVBVM60.__vbaStrCompVar
```

; The results returned in eax
; ——————————————————- —-
rtcStrConvVar2; function Strconv # 717

Parameter 3 PUSH EDI;
LEA EDX, DWORD PTR SS: [EBP-24]
PUSH 1; Parameter 2
LEA EAX, DWORD PTR SS: [EBP-44]
PUSH EDX; parameters 1
PUSH EAX; Results
CALL DWORD PTR DS: [<& MSVBVM60. # 717>]; MSVBVM60.rtcStrConvVar2
; The results returned in eax
; ——————————————————- —-
rtcStringVar; function String

LEA EDX, DWORD PTR SS: [EBP-24]
LEA EAX, DWORD PTR SS: [EBP-44]
Parameter 2 PUSH EDX;
PUSH 5; parameters 1
PUSH EAX; Results
CALL DWORD PTR DS: [<& MSVBVM60. # 607>]; MSVBVM60.rtcStringVar
; The results returned in eax
; ——————————————————- —-
rtcStrReverse; function StrReverse # 713

LEA EAX, DWORD PTR SS: [EBP-38]
PUSH EAX; parameters 1
CALL DWORD PTR DS: [<& MSVBVM60. # 713>]; MSVBVM60.rtcStrReverse
; The results returned in eax
; ——————————————————- —-
rtcTan; function Tan # 584

LEA EDX, DWORD PTR SS: [EBP-24]
PUSH EDX
CALL DWORD PTR DS: [<& MSVBVM60.__vbaR8Var>; MSVBVM60.__vbaR8Var
SUB ESP, 8
The FSTP QWORD PTR SS: [ESP]; parameters 1
CALL DWORD PTR DS: [<& MSVBVM60. # 584>]; MSVBVM60.rtcTan
; The results returned in the floating-point stack
; ——————————————————- —-
rtcGetTimeVar; the function Time # 612

LEA EDX, DWORD PTR SS: [EBP-34]
PUSH EDX; Results
CALL DWORD PTR DS: [<& MSVBVM60. # 612>]; MSVBVM60.rtcGetTimeVar
; The results returned in eax
——————————————————— ——
rtcGetTimer; function Timer # 535

CALL DWORD PTR DS: [<& MSVBVM60. # 535>]; MSVBVM60.rtcGetTimer

; The results returned in the floating-point stack
; ——————————————————- —-
rtcPackTime; function TimeSerial # 539

LEA EDX, DWORD PTR SS: [EBP-44]
Parameter 3 PUSH EDX;
LEA EAX, DWORD PTR SS: [EBP-34]
Parameter 2 PUSH EAX;
LEA ECX, DWORD PTR SS: [EBP-24]
PUSH ECX; parameters 1
LEA EDX, DWORD PTR SS: [EBP-64]
PUSH EDX; Results
CALL DWORD PTR DS: [<& MSVBVM60. # 539>]; MSVBVM60.rtcPackTime
; The results returned in eax
———————————————— ——
rtcGetTimeValue; function TimeValue # 541

LEA EAX, DWORD PTR SS: [EBP-38]
LEA ECX, DWORD PTR SS: [EBP-48]
PUSH EAX; parameters 1
PUSH ECX; Results
CALL DWORD PTR DS: [<& MSVBVM60. # 541>]; MSVBVM60.rtcGetTimeValue
; The results returned in eax
———————————————— ——
rtcTrimVar; function Trim # 520

LEA ECX, DWORD PTR SS: [EBP-68]; parameters 1
LEA EDX, DWORD PTR SS: [EBP-58]; result
PUSH ECX
PUSH EDX
CALL DWORD PTR DS: [<& MSVBVM60. # 520>]; MSVBVM60.rtcTrimVar
; The results returned in eax
———————————————— ——
rtcTypeName; function TypeName # 591

LEA EDX, DWORD PTR SS: [EBP-24]
PUSH EDX; parameters 1
CALL DWORD PTR DS: [<& MSVBVM60. # 591>]; MSVBVM60.rtcTypeName
; The results returned in eax
———————————————— ——
__vbaUbound; function UBound

LEA ECX, DWORD PTR SS: [EBP-2C]; parameters 1, the array
PUSH ECX
PUSH; parameters of an array dimension
CALL DWORD PTR DS: [<& MSVBVM60.__vbaUboun>; MSVBVM60.__vbaUbound
; The results returned in eax
; ——————————————————- ——
rtcUpperCaseVar; function Ucase

```
LEA ECX, DWORD PTR SS: [EBP-24]; parameters 1
LEA EDX, DWORD PTR SS: [EBP-48]; result
PUSH ECX
PUSH EDX
CALL DWORD PTR DS: [<& MSVBVM60. # 528>]; MSVBVM60.rtcUpperCaseVar
; The results returned in eax
; ——————————————————- ——
rtcR8ValFromBstr; the function Val # 581

LEA EAX, DWORD PTR SS: [EBP-38]
PUSH EAX; parameters 1
CALL DWORD PTR DS: [<& MSVBVM60. # 581>]; MSVBVM60.rtcR8ValFromBstr
; The results of the floating-point stack
; ——————————————————- —-
rtcVarType; function VarType # 563

LEA EDX, DWORD PTR SS: [EBP-24]
PUSH EDX; parameters 1
CALL DWORD PTR DS: [<& MSVBVM60. # 563>]; MSVBVM60.rtcVarType
; The result in eax
; ——————————————————- —-
rtcWeekdayName; function WeekdayName # 706

PUSH EDI
LEA EDX, DWORD PTR SS: [EBP-24]
PUSH EDI
PUSH EDX
CALL DWORD PTR DS: [<& MSVBVM60. # 706>]; MSVBVM60.rtcWeekdayName
; The result in eax
; ——————————————————- —-
rtcGetYear; function Year # 553

LEA EAX, DWORD PTR SS: [EBP-24]
LEA ECX, DWORD PTR SS: [EBP-44]
PUSH EAX; parameters 1
PUSH ECX; Results
CALL DWORD PTR DS: [<& MSVBVM60. # 553>]; MSVBVM60.rtcGetYear
; The result in eax
; ——————————————————- —-
__vbaBoolErrVar; function CBool

LEA EDX, DWORD PTR SS: [EBP-74]
PUSH EDX; parameters 1
CALL DWORD PTR DS: [<& MSVBVM60.__vbaBoolE>; MSVBVM60.__vbaBoolErrVar
; The results in ax in
; ——————————————————- —-
__vbaUI1ErrVar; function Cbyte

LEA EAX, DWORD PTR SS: [EBP-74]
PUSH EAX; parameters 1
```

```
CALL DWORD PTR DS: [<& MSVBVM60.__vbaUI1Er>; MSVBVM60.__vbaUI1ErrVar
; Result in al
; ——————————————————————- —-
__vbaCyErrVar; function Ccur

LEA ECX, DWORD PTR SS: [EBP-74]
PUSH ECX; parameters 1
CALL DWORD PTR DS: [<& MSVBVM60.__vbaCyErr>; MSVBVM60.__vbaCyErrVar
; The result in eax
; ——————————————————————- —-
__vbaDateVar; function Cdate

LEA EDX, DWORD PTR SS: [EBP-74]
PUSH EDX; parameters 1
CALL DWORD PTR DS: [<& MSVBVM60.__vbaDateV>; MSVBVM60.__vbaDateVar
SUB ESP, 8
The FSTP QWORD PTR SS: [ESP]
CALL DWORD PTR DS: [<& MSVBVM60.__vbaDateR>; MSVBVM60.__vbaDateR8
; The results of the floating-point stack
; ——————————————————————- —-
__vbaR8ErrVar; function Cdbl

LEA EAX, DWORD PTR SS: [EBP-74]
PUSH EAX; parameters 1
CALL DWORD PTR DS: [<& MSVBVM60.__vbaR8Err>; MSVBVM60.__vbaR8ErrVar
; The results of the floating-point stack
; ——————————————————————- —-
rtDecFromVar; function Cdec # 564

LEA ECX, DWORD PTR SS: [EBP-F4]
LEA EDX, DWORD PTR SS: [EBP-74]
PUSH ECX; parameters 1
PUSH EDX; Results
CALL DWORD PTR DS: [<& MSVBVM60. # 564>]; MSVBVM60.rtDecFromVar
; The result in eax
; ——————————————————————- —-
__vbaI2ErrVar; function Cint

LEA EAX, DWORD PTR SS: [EBP-74]
PUSH EAX; parameters 1
CALL DWORD PTR DS: [<& MSVBVM60.__vbaI2Err>; MSVBVM60.__vbaI2ErrVar
; The results in ax in
; ——————————————————————- —-
__vbaI4ErrVar; function Clng

LEA ECX, DWORD PTR SS: [EBP-74]
PUSH ECX; parameters 1
CALL DWORD PTR DS: [<& MSVBVM60.__vbaI4Err>; MSVBVM60.__vbaI4ErrVar
; The result in eax
; ——————————————————————- —-
```

__vbaR4ErrVar; function Csng

```
LEA EDX, DWORD PTR SS: [EBP-74]
PUSH EDX; parameters 1
CALL DWORD PTR DS: [<& MSVBVM60.__vbaR4Err>; MSVBVM60.__vbaR4ErrVar
; The results of the floating-point stack
; ——————————————————————- —-
```

__vbaStrErrVarCopy; function CSTR

```
LEA EAX, DWORD PTR SS: [EBP-74]
PUSH EAX; parameters 1
CALL DWORD PTR DS: [<& MSVBVM60.__vbaStrEr>; MSVBVM60.__vbaStrErrVarCopy
; The result in eax
; ——————————————————————- —-
```

__vbaVarCopy; function CVaR

```
LEA EDX, DWORD PTR SS: [EBP-74]; parameters 1
LEA ECX, DWORD PTR SS: [EBP-54]; result
CALL DWORD PTR DS: [<& MSVBVM60.__vbaVarCo>; MSVBVM60.__vbaVarCopy
; The result in eax
; ——————————————————————- —-
```

__vbaFileOpen; Open statement

```
The PUSH engineering 1.004014C0; File name
The PUSH 1; File No.
PUSH 1; len
PUSH 320; for, access, lock
CALL DWORD PTR DS: [<& MSVBVM60.__vbaFileO>; MSVBVM60.__vbaFileOpen
; ——————————————————————- ——
```

__vbaFileClose; Close statement

```
The PUSH 1; File No.
CALL DWORD PTR DS: [<& MSVBVM60.__vbaFileC>; MSVBVM60.__vbaFileClose
; ——————————————————————- ——
```

rtcFreeFile; function FreeFile

```
LEA EAX, DWORD PTR SS: [EBP-34]
PUSH EAX; Results
CALL DWORD PTR DS: [<& MSVBVM60. # 648>]; MSVBVM60.rtcFreeFile
; The results in ax in
; ——————————————————————- ——
```

rtcFileLength; function LOF # 570

```
LEA EDX, DWORD PTR SS: [EBP-34]
PUSH EDX; File No.
CALL DWORD PTR DS: [<& MSVBVM60. # 570>]; MSVBVM60.rtcFileLength
; The result in eax
; ——————————————————————- ——
```

rtcFileLocation; function loc # 569

LEA EAX, DWORD PTR SS: [EBP-34]
PUSH EAX; File No.
CALL DWORD PTR DS: [<& MSVBVM60. # 569>]; MSVBVM60.rtcFileLocation
; The result in eax
; ——————————————————- ——
rtcFileAttributes; function FileAttr # 555

LEA ECX, DWORD PTR SS: [EBP-34]
The PUSH 1; property
PUSH ECX; File No.
CALL DWORD PTR DS: [<& MSVBVM60. # 555>]; MSVBVM60.rtcFileAttributes
; The result in eax
; ——————————————————- ——
__vbaPrintFile; Print # file number, variable (sequential file operation)

LEA EAX, DWORD PTR SS: [EBP-24]
LEA ECX, DWORD PTR SS: [EBP-34]
PUSH ECX; variable
PUSH EAX; File No.
PUSH Engineering 1.00401948
CALL DWORD PTR DS: [<& MSVBVM60.__vbaPrint>; MSVBVM60.__vbaPrintFile
; ——————————————————- —-
__vbaWriteFile; the Write # file number, variable (sequential file operation)

LEA EDX, DWORD PTR SS: [EBP-24]
LEA EAX, DWORD PTR SS: [EBP-34]
PUSH EDX; variable
PUSH EAX; File No.
PUSH Engineering 1.00401948
CALL DWORD PTR DS: [<& MSVBVM60.__vbaWrite>; MSVBVM60.__vbaWriteFile
; ——————————————————- —-
__vbaInputFile; input # file number, variable (sequential file operation)

LEA EAX, DWORD PTR SS: [EBP-24]
LEA ECX, DWORD PTR SS: [EBP-34]
PUSH EAX; variable
PUSH ECX; File No.
PUSH Engineering 1.00401938
CALL DWORD PTR DS: [<& MSVBVM60.__vbaInput>; MSVBVM60.__vbaInputFile
; ——————————————————- —-
__vbaLineInputVar; line input # File No. variables (sequential file operation)

LEA EDX, DWORD PTR SS: [EBP-34]
PUSH EDX; File No.
LEA EAX, DWORD PTR SS: [EBP-44]
PUSH EAX; variable
CALL DWORD PTR DS: [<& MSVBVM60.__vbaLineI>; MSVBVM60.__vbaLineInputVar
; The results returned in eax
; ——————————————————- —-
rtcInputCharCountVar; function input (# file number, length) # 621 (sequential file operations)

```
LEA ECX, DWORD PTR SS: [EBP-34]
PUSH ECX; File No.
LEA EDX, DWORD PTR SS: [EBP-64]
PUSH; Length
PUSH EDX; Results
CALL DWORD PTR DS: [<& MSVBVM60. # 621>]; MSVBVM60.rtcInputCharCountVar
; The results returned in eax
; ——————————————————- —-
```

__vbaPut4; statement Put the file number, location variables (binary file operation)

```
LEA EAX, DWORD PTR SS: [EBP-34]
PUSH EAX; File No.
LEA ECX, DWORD PTR SS: [EBP-24]
PUSH 2; Location
PUSH ECX; variable
PUSH -1
CALL DWORD PTR DS: [<& MSVBVM60.__vbaPut4 >>; MSVBVM60.__vbaPut4
; ——————————————————- —-
```

__vbaFileSeek; statements Seek document number, location (binary file operation)

```
LEA EDX, DWORD PTR SS: [EBP-34]
PUSH EDX; File No.
PUSH 2; Location
CALL DWORD PTR DS: [<& MSVBVM60.__vbaFileS>; MSVBVM60.__vbaFileSeek
; ——————————————————- —-
```

__vbaGet4; statement Get file number, location variables (binary file operation)

```
LEA EAX, DWORD PTR SS: [EBP-34]
PUSH EAX; File No.
LEA ECX, DWORD PTR SS: [EBP-44]
PUSH 2; Location
PUSH ECX; variable
PUSH -1
CALL DWORD PTR DS: [<& MSVBVM60.__vbaGet4 >>; MSVBVM60.__vbaGet4
```

=== =================================

common martCheck information
__vbasrtcmp (String: “zzzzz” String: “yyyyy”) returns DWORD: 0

Explanation:
__vbastrcmp – used to compare strings. “zzzzz” and “yyyyy”
Note: You may see the correct serial number and your input string comparison.
returns DWORD: 0 – in the SOFTICE, you will see the comparison, eax = 0
——————————————————- ————————————

__vbafreestr (LPBSTR: 0063F3F0)
Click the above “+” looking SysFreeString

Such as. SysFreeString (BSTR: 00410584)

Explanation:
String 00,410,584 in memory is cleared.
————————————————————————

__vbaVarCopy (VARIANT: String: “12345” VARIANT: Empty) returns DWORD: 63FA30
Click the “+” sign in front of seeking SysAllocStringByteLen
Such as. SysAllocStringByteLen (LPSTR: 004023F0 DWORD: 0000000C) returns LPVOID: 4103CC

Explanation:
“12345” is copied to memory 004103CC
This similar __vbaVarMove to
————————————————————————

__vbaVarForInit (VARIANT: Empty PTR: 0063F920 PTR: 0063F91 …..)

Explanation:
Move to above, for the next cycle is usually below it __vbaVarForNext Use the same.
————————————————————————

Mid (VARIANT: String: “abcdefg” long: 1 VARIANT: Integet: 1)

Explanation:
From position 1 to get the first character of the string “ABCDEFG”.
Click on the “+” sign to find SysAllocStringByteLen
Such as. SysAllocStringByteLen (LPSTR: 004103F0 DWORD: 00000002) returns LPVOID: 410434

Explanation:
“A” will be copied to the memory 00410434
It is normally followed by __vbaStrVarVal (VARIATN: String “a”) Returns DWORD: 410 434
————————————————————————

Asc (String: “T”) returns Integer: 84

Explanation:
“T” in the ASCII code 84 decimal
————————————————————————

SysFreeString (BSTR: 004103F0)

Explanation:
Release memory location: 004103F0
These are particularly useful, because when you click on them to see the right window, you will see the string is released. The correct serial number and password may be in this.
————————————————————————

__vbaVarCat (VARIANT: String: “aa” VARIANT: String: “bb”) returns DWORD: 63F974

Explanation:
Connection “bb” and “aa” form “aabb”

————————————————————————————

__vbaFreeVar (VARIANT: String: “abcdefg”)
Click the “+” looking SysFreeString
Of cases SysFreeString (BSTR: 0041035C)

Explanation:
From “abcdefg memory 0041035C release”
Here, click on the right of this line may find something you want.
————————————————————————————

__vbaVarTstEq (VARIANT: **** VARIANT: ****) returns DWORD: 0

Explanation:
__vbaVarTstEq If they are not the same DWORD = 0 (so eax = 0 is usually used to compare variables.)
If they are the same DWORD will FFFFFFFF (so EAX = FFFFFFFF)
Similar __vbaVarCmpEq
————————————————————————————

Len (String: “Cracker”) returns LONG: 7

Explanation:
The length of the string “Cracker” 7
————————————————————————————

****. Text <- “Wrong! Try Again!!” (String)

Explanation:
Displayed in the text box g “Wrong! Try Again!”
————————————————————————————

__vbaVarAdd (VARIANT: Integer: 2 VARIANT: Integer: 97) returns …..

Explanation:
2 +97 = 97 Back 99
But if both are Strings instead of Integers you will get 297 instead.
————————————————————————————

__vbaVarDiv (VARIANT: Integer: 97 VARIANT: Long: 1) returns …..

Explanation:
97 divided by 1
————————————————————————————

__vbaVarMul (VARIANT: String: “1” VARIANT: String: “2”) returns …

Explanation:
1 by 2
————————————————————————————

__vbaVarSub (VARIANT: String: “2” VARIANT: String: “34”) returns …

Explanation:
"34" – "2" returns 32
————————————————————————

MsgBox (VARIANT: String: "Nope! That's not right" Integer: 0 VARIANT: String: "Wrong" VARIANT …..)

Explanation:
Create a message box, the title is "Wrong" "Nope! That's not right"

rtcAnsiValueBstr; function Asc (string)

lea eax, var1
PUSH EAX; parameter string
call rtcAnsiValueBstr; interger results are returned in eax
————————————————————————
rtcVarBstrFromAnsi; the functions Chr (integer)

PUSH EAX; parameter integer
call rtcVarBstrFromAnsi; results are returned in eax string
————————————————————————
rtcUpperCaseVar; function Ucase

LEA ECX, DWORD PTR SS: [EBP-24]; parameters 1
LEA EDX, DWORD PTR SS: [EBP-48]; result
PUSH ECX
PUSH EDX
CALL DWORD PTR DS: [<& MSVBVM60. # 528>]; MSVBVM60.rtcUpperCaseVar
; The results returned in eax
————————————————————————
rtcLowerCaseVar; function Lcase # 518

LEA EDX, DWORD PTR SS: [EBP-24]; parameters 1
LEA EAX, DWORD PTR SS: [EBP-48]; result
PUSH EDX
PUSH EAX
CALL DWORD PTR DS: [<& MSVBVM60. # 518>]; MSVBVM60.rtcLowerCaseVar
; The results returned in eax
————————————————————————
vbaLenVar
You want to know the results returned in EAX on the line
————————————————————————
rtcLeftCharVar; function Left # 617

LEA EDX, DWORD PTR SS: [EBP-24]; parameters 1
PUSH 3; Parameter 2
LEA EAX, DWORD PTR SS: [EBP-48]; result

PUSH EDX
PUSH EAX
CALL DWORD PTR DS: [<& MSVBVM60. # 617>]; MSVBVM60.rtcLeftCharVar
; The results returned in eax
————————————————————
rtcRightCharVar; function Right # 619

LEA EDX, DWORD PTR SS: [EBP-24]
PUSH 3; Parameter 2
LEA EAX, DWORD PTR SS: [EBP-44]
PUSH EDX; parameters 1
PUSH EAX; Results
CALL DWORD PTR DS: [<& MSVBVM60. # 619>]; MSVBVM60.rtcRightCharVar
; The results returned in eax
————————————————————
rtcMidCharVar; function Mid

Parameter 3 PUSH EAX;
LEA ECX, DWORD PTR SS: [EBP-58]
PUSH 3; Parameter 2
LEA EDX, DWORD PTR SS: [EBP-48]
PUSH ECX; parameters 1
PUSH EDX; Results
CALL DWORD PTR DS: [<& MSVBVM60. # 632>]; MSVBVM60.rtcMidCharVar
; The results returned in eax

from: http://blog.csdn.net/bbdxf

VB程序逆向常用的函数
1) 数据类型转换:
a) __vbaI2Str 将一个字符串转为8 位（1个字节）的数值形式(范围在 0 至 255 之间) 或2 个字节的数值形式(范围在 -32,768 到 32,767 之间)。
b)__vbaI4Str 将一个字符串转为长整型(4个字节)的数值形式(范围从-2,147,483,6482,147,483,647)
c)__vbar4Str 将一个字符串转为单精度单精度浮点型(4个字节)的数值形式
d)__vbar8Str 将一个字符串转为双精度单精度浮点型(8个字节)的数值形式
e) VarCyFromStr (仅VB6库. 要调试，则在WINICE.DAT里必须有 OLEAUT32.DLL)字符串到变比型数据类型
f) VarBstrFromI2 (仅VB6库. 要调试，则在WINICE.DAT里必须有 OLEAUT32.DLL)整型数据到字符串:
2) 数据移动：
a) __vbaStrCopy 将一个字符串拷贝到内存，类似于 Windows API HMEMCPY
b) __vbaVarCopy 将一个变量值串拷贝到内存
c) __vbaVarMove 变量在内存中移动，或将一个变量值串拷贝到内存
3) 数学运算：
a) __vbavaradd 两个变量值相加
b) __vbavarsub 第一个变量减去第二个变量
c) __vbavarmul 两个变量值相乘

d) __vbavaridiv 第一个变量除以第二个变量，得到一个整数商
e) __vbavarxor 两个变量值做异或运算

4) 程序设计杂项：

a) __vbavarfornext 这是VB程序里的循环结构， For… Next… (Loop)
b) __vbafreestr 释放出字符串所占的内存，也就是把内存某个位置的字符串给抹掉
c) __vbafreeobj 释放出VB一个对象(一个窗口，一个对话框)所占的内存，也就是把内存某个位置的一个窗口，一个对话框抹掉
d) __vbastrvarval 从字符串特点位置上获取其值
e) multibytetowidechar 将数据转换为宽字符格式，VB在处理数据之都要这样做，在TRW2000显示为7.8.7.8.7.8.7.8
f) rtcMsgBox 调用一个消息框，类似于WINDOWS里的messagebox/a/exa,此之前一定有个PUSH命令将要在消息框中显示的数据压入椎栈
g) __vbavarcat 将两个变量值相连，如果是两个字符串，就连在一起
h) __vbafreevar 释放出变量所占的内存，也就是把内存某个位置的变量给抹掉
i) __vbaobjset
j) __vbaLenBstr 获得一个字符串的长度，注：VB中一个汉字的长度也为1
k) rtcInputBox 显示一个VB标准的输入窗口，类似window’s API getwindowtext/a, GetDlgItemtext/a
l) __vbaNew 调用显示一个对话框，类似 Windows’ API Dialogbox
m) __vbaNew2 调用显示一个对话框，类似 Windows’ API Dialogboxparam/a
n) rtcTrimBstr 将字串左右两边的空格去掉

5) 比较函数

a) __vbastrcomp 比较两个字符串，类似于 Window’s API lstrcmp
b) __vbastrcmp 比较两个字符串，类似于 Window’s API lstrcmp
c) __vbavartsteq 比较两个变量值是否相等
d)__vbaFpCmpCy – Compares Floating point to currency. sp; Compares Floating point to currency

6) 在动态跟踪，分析算法时，尤其要注意的函数：

rtcMidCharVar 从字符串中取相应字符，VB中的MID函数，用法MID(“字符串”,”开始的位置”,”取几个字符”)
rtcLeftCharVar 从字符串左边取相应字符，VB中的用法：left(“字符串”,”从左边开始取几个字符”)
rtcRightCharVar 从字符串右边取相应字符，VB中的用法：Right(“字符串”,”从右边开始取几个字符”)
__vbaStrCat 用字符串的操作，就是将两个字符串合起来，在VB中只有一个&或+
__vbaStrCmp 字符串比较，在VB中只有一个=或<>
ASC()函数 取一个字符的ASC值，在反汇编时，还是有的movsx 操作数

7) 在函数中的缩写：

bool 布尔型数据(TRUE 或 FALSE)
str 字符串型数据 STRING
i2 字节型数据或双字节整型数据 BYTE or Integer
ui2 无符号双字节整型数据
i4 长整型数据(4字节) Long
r4 单精度浮点型数据(4字节) Single
r8 双精度浮点型数据(8字节) Double
cy （8 个字节）整型的数值形式 Currency
var 变量 Variant
fp 浮点数据类型 Float Point
cmp 比较 compare
comp 比较 compare

Btw：

__vbavartsteq系列的还有__vbavartstne 不等于
__vbavartstGe,__vbavartstGt,__vbavartstLe,__vbavartstLt等，比较大于或小于

————————————

拦截警告声:
bpx rtcBeep —>扬声器提示
数据移动:
bpx vbaVarCopy —>数据移动将一个变量值串拷贝到内存
bpx vbaVarMove —>数据移动变量在内存中移动，或将一个变量值串拷贝到内存
bpx vbaStrMove —>移动字符串
bpx vbaStrCopy —>移动字符串 将一个字符串拷贝到内存，类似于 Windows API HMEMCPY
数据类型转换:
bpx vbaI2Str —>将一个字符串转为8 位（1个字节）的数值形式(范围在 0 至 255 之间) 或2 个字节的数值形式(范围在 -32,768 到 32,767 之间)。
bpx vbaI4Str —>将一个字符串转为长整型(4个字节)的数值形式(范围从-2,147,483,6482,147,483,647)
bpx vbar4Str —>将一个字符串转为单精度单精度浮点型(4个字节)的数值形式
bpx vbar8Str —>将一个字符串转为双精度单精度浮点型(8个字节)的数值形式
bpx VarCyFromStr —>(仅VB6库. 要调试，则在WINICE.DAT里必须有 OLEAUT32.DLL)字符串到变比型数据类型
bpx VarBstrFromI2 —>(仅VB6库. 要调试，则在WINICE.DAT里必须有 OLEAUT32.DLL)整型数据到字符串:
数值运算:
bpx vbaVarAdd —>两个变量值相加
bpx vbaVarIdiv —>除整，第一个变量除以第二个变量，得到一个整数商
bpx vbaVarSub —>第一个变量减去第二个变量
bpx vbaVarMul —>两个变量值相乘
bpx vbaVarDiv —>除
bpx vbaVarMod —>求余
bpx vbaVarNeg —>取负
bpx vbaVarPow —>指数
bpx vbavarxor —>两个变量值做异或运算
针对变量:
bpx vbaVarCompEq —>比较局部变量是否相等
bpx vbaVarCompNe —>比较局部变量是否不等于
bpx vbaVarCompLe —>比较局部变量小于或等于
bpx vbaVarCompLt —>比较局部变量小于
bpx vbaVarCompGe —>比较局部变量大于或等于
bpx vbaVarCompGt —>比较局部变量大于
VB的指针:
THROW
程序结构:
bpx vbaVarForInit —>重复执行初始化
bpx vbaVarForNext —>重复执行循环结构， For… Next… (Loop)
比较函数:
bpx vbaStrCmp —>比较字符串是否相等 ******
bpx vbaStrComp —>比较字符串是否相等 ******
bpx vbaVarTstEq —>检验指定变量是否相等
bpx vbaVarTstNe —>检验指定变量是否不相等

bpx vbaVarTstGt —>检验指定变量大于
bpx vbaVarTstGe —>检验指定变量大于或等于
bpx vbaVarTstLt —>检验指定变量小于
bpx vbaVarTstLe —>检验指定变量小于或等于
字符串操作:
bpx vbaStrCat —>用字符串的操作，就是将两个字符串合起来，在VB中只有一个&或+
bpx vbaStrLike
bpx vbaStrTextComp —>与指定文本字符串比较
bpx vbaStrTextLike
bpx vbaLenBstr —>字符串长度
bpx vbaLenBstrB —>字符串长度
bpx vbaLenVar —>字符串长度
bpx vbaLenVarB —>字符串长度
bpx rtcLeftCharVar —>截取字符串，从字符串左边取相应字符，VB中的用法：left(“字符串”,”从左边开始取几个字符”)
bpx vbaI4Var —>截取字符串
bpx rtcRightCharVar —>截取字符串，从字符串右边取相应字符，VB中的用法：Right(“字符串”,”从右边开始取几个字符”)
bpx rtcMidCharVar —>截取字符串，VB中的MID函数，用法MID(“字符串”,”开始的位置”,”取几个字符”)
bpx vbaInStr —>查找字符串位置
bpx vbaInStrB —>查找字节位置
bpx vbaStrCopy —>复制字符串
bpx vbaStrMove —>移动字符串
bpx rtcLeftTrimVar —>删除字串的空白
bpx rtcRightTrimVar —>删除字串的空白
bpx rtcTrimVar —>删除字串的空白
bpx vbaRsetFixstrFree —>字符串往右对齐
bpx vbaRsetFixstr —>字符串往右对齐
bpx vbaLsetFixstrFree —>字符串往左对齐
bpx vbaLsetFixstr —>字符串往左对齐
bpx vbaStrComp —>字符串比较
bpx vbaStrCompVar —>字符串比较
bpx rtcStrConvVar2 —>字符串类型转换
bpx rtcR8ValFromBstr —>把字符串转换成浮点数
bpx MultiByteToWideChar —>ANSI字符串转换成Unicode字符串
bpx WideCharToMultiByte —>Unicode字符串转换成ANSI字符串
bpx rtcVarFromFormatVar —>格式化字符串
bpx rtcUpperCaseVar —>小写变大写
bpx rtcLowerCaseVar —>大写变小写
bpx rtcStringVar —>重复字符
bpx rtcSpaceVar —>指定数目空格
bpx rtcAnsiValueBstr —>传回字符码（返回第一个字符的字符代码）
bpx rtcByteValueBstr —>传回字符码（返回第一个字节的字符代码）
bpx rtcCharValueBstr —>传回字符码（返回第一个Unicode字符代码）
bpx rtcVarBstrFromAnsi —>传回字符（返回 String，其中包含有与指定的字符代码相关的字符 ）
bpx rtcVarBstrFromByte —>传回字符（返回 String，其中包含有与指定的字符代码相关的单字节）
bpx rtcVarBstrFromChar —>传回字符（返回 String，其中包含有与指定Unicode 的 String）

Tags: VBscript reversing (https://hvoidcode.wordpress.com/tag/vbscript-reversing/)