Anti-unpack Tricks in Malicious Code

Xiaodong Tan
Security Labs, Websense Inc.

AVAR 2007, Seoul

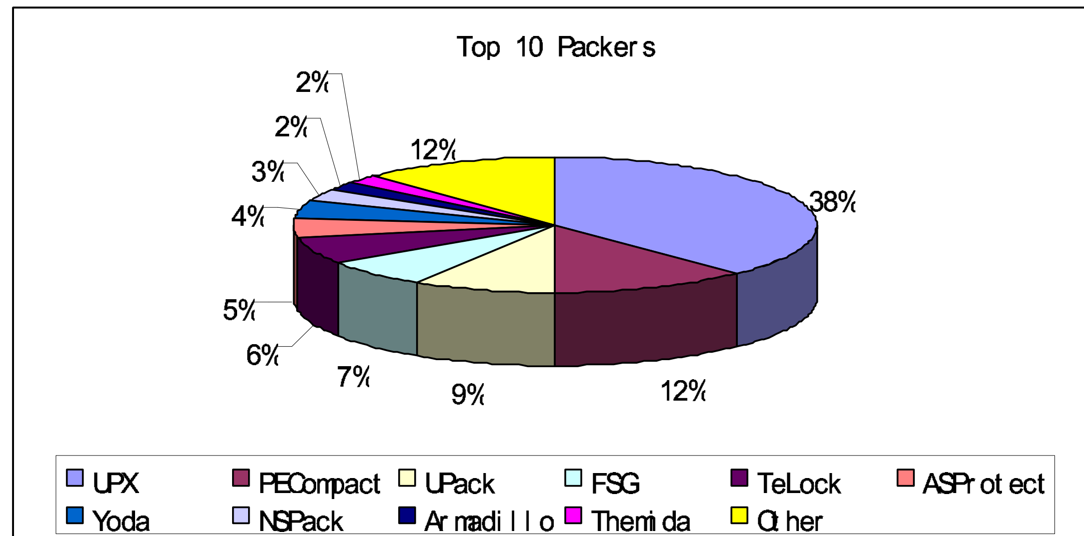
Agenda

- Problems raised by packers
- Commonly used unpacking methods
- Anti static-unpacking tricks
- Anti dynamic-unpacking tricks
- Conclusions

WEBSENSE.

Packer Statistics from WSLabs

- According to the research of WSLabs (Websense Security Labs), more than 80% malicious codes are disguised by a certain packer
  - Top ten packers from the tracker of WSLabs



Problems raised by packers

- The problems raised by packers
  - It's easier to produce "new" malware by using packers
  - Using packers can bring more difficulties to analysts and researchers
  - Smaller in size, easier to propagate
- Why we need unpack?
  - In order to detect a known malware, we have to decompress the packer first, and reach the point where signature matching can occur
  - It is necessary to decompress the file so that the code analysis or heuristic rules could be applied

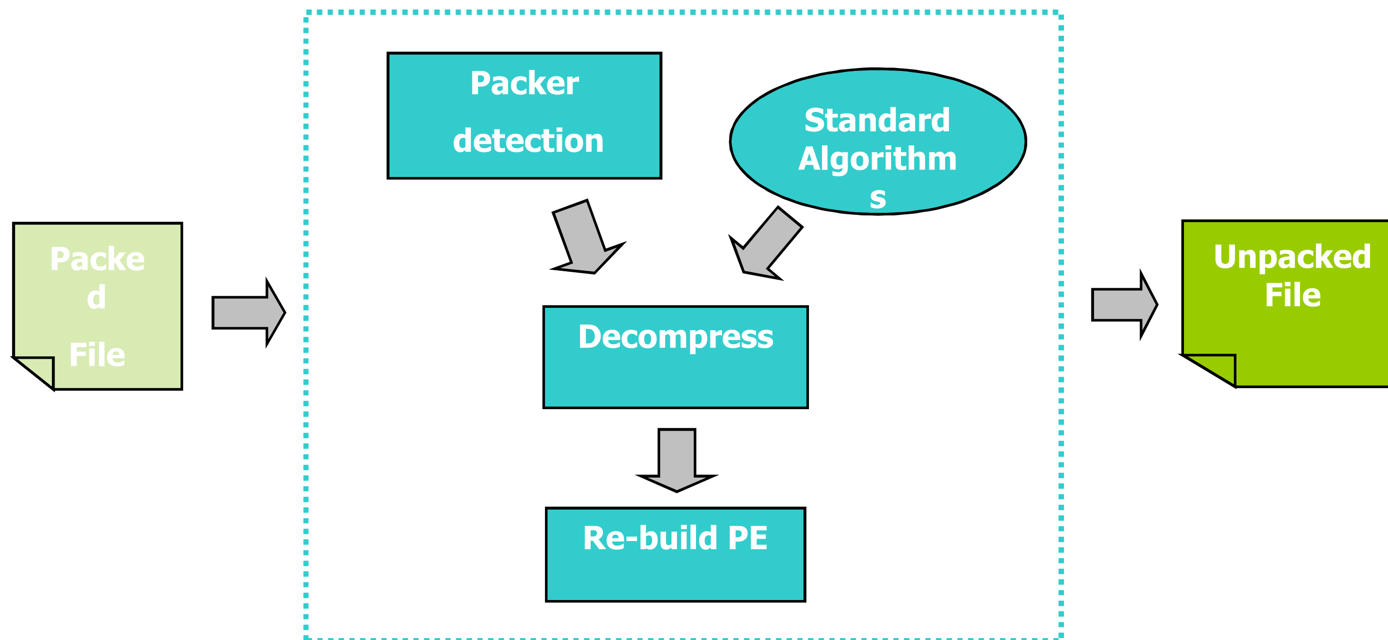
Commonly used unpacking methods

- Commonly used unpacking methods
  - Routine-based (static unpacking)
  - Emulator-based (dynamic unpacking)
  - Mixed routine-based and emulator-based

Routine-based Unpacking

- Routine-based unpacking is based on decompression algorithms
  - The packer author always uses standard compression and decompression algorithms
  - Algorithm example:
    - FSG
    - UPack
- Advantages: high speed
- Disadvantages: Lack of flexibility

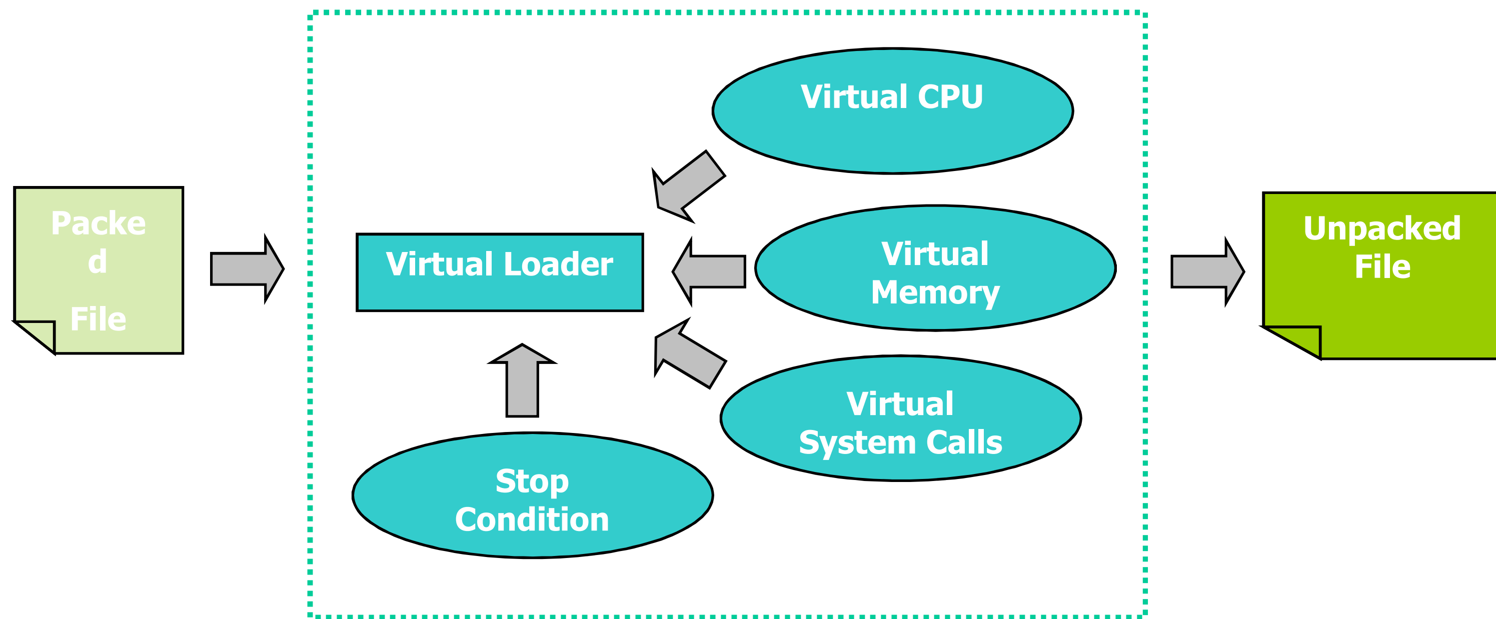
Routine-based Unpacking: A simple model



Emulator-based Unpacking

- Emulator Unpacking
  - A file will be loaded and run in the virtual environment
  - If it can stop right at the OEP, then we can get the unpacked file
- Advantages:
  - Much more generic. Ideally, this method can cope with different packers
  - It also can handle the packer with slight modification from its original version
- Disadvantages:
  - Remarkably affect the performance
  - In the process of emulation it's not an easy thing to determine when to stop

Emulator-based Unpacking: A simple model



Mixed routine-based and emulator-based

- Mixed routine-based and emulator-based
  - If the two types can work harmoniously, we can attain both efficiency and flexibility
  - Routine-based unpacking can cope with the standard decompression algorithms
  - Emulation can handle the modified packers or unknown packers, or find some important parameters for the routine-based module

Anti-unpack tricks

- Tricks on anti routine-based unpacking
  - Surviving from packer detection
  - Modifying the packer inner structure
  - Modifying the decompression algorithm
- Tricks on anti emulator-based unpacking
  - Interrupting the emulator
  - Disturbing the emulation stop condition

Packer detection

- Tampering the packer detection is wildly used for escaping the routine-based unpacking
  - To find out which packer a file has been packed with is usually the first important step in routine-based unpacking
    - Usually the packer scanner starts from the entry point of a PE file
    - It's very easy to obfuscate the code around the entry point manually or by tools
- A large amount of samples are of this kind and in the wild
- Though this trick is rather simple, if we can't handle this kind of threat smartly enough, we will leak most of them

WEBSENSE.

Packer detection

- An example of disturbing packer detection
  - This sample is disguised by a Chinese hacker tool
    - It can be easily found on almost every Chinese hacker site
    - The interface of the tool

Packer detection

Obfuscated stub (Only 16 bytes)

```
:00438071
:00438071                              public start
:00438071                 start        proc near
:00438071 55                           push    ebp
:00438072 8B EC                        mov     ebp, esp
:00438074 83 C4 F4                     add     esp, 0FFFFFFF4h
:00438077 83 C4 0C                     add     esp, 0Ch
:0043807A B8 01 50 43 00               mov     eax, offset RealStart
:0043807F 50                           push    eax
:00438080 C3                           retn
:00438080                 start        endp ; sp = -8
```

The real ASPack entry point

```
.aspack:00435001
.aspack:00435001                RealStart:                       ; DATA XREF: start+9↓o
.aspack:00435001 60                          pusha
.aspack:00435002 E8 03 00 00 00              call    loc_43500A
.aspack:00435002                ; ---------------------------------------------------
.aspack:00435007 E9                          db 0E9h ; Ú
.aspack:00435008                ; ---------------------------------------------------
.aspack:00435008 EB 04                       jmp     short loc_43500E
.aspack:0043500A                ; ---------------------------------------------------
.aspack:0043500A
.aspack:0043500A                loc_43500A:                      ; CODE XREF: .aspack:00
.aspack:0043500A 5D                          pop     ebp
.aspack:0043500B 45                          inc     ebp
.aspack:0043500C 55                          push    ebp
.aspack:0043500D C3                          retn
.aspack:0043500E                ; ---------------------------------------------------
.aspack:0043500E
.aspack:0043500E                loc_43500E:                      ; CODE XREF: .aspack:00
.aspack:0043500E E8 01 00 00 00              call    loc_435014
```

Packer structure

- Packer structure
  - Stores a packer’s important information. For example, the information can be as follows:
    - How to get the decryption key
    - Where to locate the compressed data
    - How to reform sections
    - How to get the OEP
    - How to fix the import table
    - And so on

WEBSENSE.

Packer structure

- Take the FSG 2.0 as an example
- The entry point of FSG 2.0

```
HEADER:01000154
HEADER:01000154                                  public start
HEADER:01000154                   start          proc near
HEADER:01000154 87 25 70 F1 01 01                xchg    esp, ds:off_101F170 ;
HEADER:01000154                                                              ;
HEADER:01000154
HEADER:01000154                                                              ;
HEADER:0100015A 61                               popa
HEADER:0100015B 94                               xchg    eax, esp
HEADER:0100015C 55                               push    ebp
HEADER:0100015D
HEADER:0100015D                   loc_100015D:                               ; CODE XREF: start+E↓j
HEADER:0100015D A4                               movsb
HEADER:0100015E B6 80                            mov     dh, 80h
HEADER:01000160
HEADER:01000160                   loc_1000160:                               ; CODE XREF: start+2A↓j
HEADER:01000160                                                              ; start+6B↓j
HEADER:01000160 FF 13                            call    dword ptr [ebx]
HEADER:01000162 73 F9                            jnb     short loc_100015D
HEADER:01000164 33 C9                            xor     ecx, ecx
HEADER:01000166 FF 13                            call    dword ptr [ebx]
HEADER:01000168 73 16                            jnb     short loc_1000180
```

Packer structure address

(2 Bytes off the entry point)

Packer structure

- The inner structure of FSG 2.0

```
seg002:0101F154 00 10 00 01           off_101F154     dd offset unk_1001000   ; DATA XREF: seg002:off_101F
seg002:0101F158 D4 AD 01 01                           dd offset unk_101ADD4
seg002:0101F15C 20 6D 00 01                           dd offset unk_1006D20
seg002:0101F160 00 00 00 00                           dd 0
seg002:0101F164 74 F1 01 01                           dd offset pFunc
seg002:0101F168 80 00 00 00                           dd 80h
seg002:0101F16C 00 7D 00 00                           dd 7D00h
seg002:0101F170 54 F1 01 01           off_101F170     dd offset off_101F154   ; DATA XREF: start↑w
seg002:0101F170                                                               ;
seg002:0101F170                                                               ;
seg002:0101F170                                                               ;
seg002:0101F170                                                               ;
seg002:0101F174 E8 01 00 01           pFunc           dd offset sub_10001E8   ; DATA XREF: seg002:0101F164
seg002:0101F178 DC 01 00 01                           dd offset sub_10001DC
seg002:0101F17C DE 01 00 01                           dd offset loc_10001DE
seg002:0101F180 E0 6A 00 01                           dd offset dword_1006AE0
```

Destination address

Source data address

Algorithm parameter

Decode routine

Original entry point

Packer structure

- A simple function model for the FSG 2.0 unpacking process

-
```
void DecompressFSG_2_0(BYTE * image, (void *) LZ77_Decompress)
{
        InnerStructAddress = getDWORD(entrypoint + 0x2);
                        //get the address of the packer structure
        DestAddress = getDWORD (InnerStructAddress – 0x1C );
                        //get the address where to put the decompressed data
        SourceDataAddress = getDWORD (InnerStructAddress – 0x18 )
                        //get the address where to get the compressed data
        OriginalEntryPoint = getDWORD (InnerStructAddress + 0x10 );
                        //get the original entry point
        LZ77_Decompress(CompressedDataAddress, TempBuffer);
                        //decompress the data
        CopyBuffer(DestAddress, TempBuffer);
                        //move the compressed data to the destination
        SetEntryPoint(OriginalEntryPoint);
                        //set the original entry point
 }
```

Packer structure

- Packer structure is usually distant from the entry point (In this example, the distance is: 0x101f170 – 0x1000154 = 0x1F01C)
- Usually we choose the packer detection signature just around the entry point
- If a packer is detected by a certain packer signature, but its inner structure has been modified, it then cannot get the right parameters

Packer structure

- A simulation of the packer structure modification
- If the offsets in the packer structure are modified, and we still follow the original process, the situation would be as follows:
- *void DecompressFSG_2_0(BYTE * image, (void *) LZ77_Decompress)*

```
{
    InnerStructAddress = getDWORD(entrypoint + 0x2);
                    //get the address of the packer structure
    DestAddress = getDWORD (InnerStructAddress – 0x1C );
                    //Wrong parameter!
    SourceDataAddress = getDWORD (InnerStructAddress – 0x18 )
                    //Wrong parameter!
    OriginalEntryPoint = getDWORD (InnerStructAddress + 0x10 );
                    //Wrong parameter!
    LZ77_Decompress(CompressedDataAddress, TempBuffer);
                    //Fail or Crash!
    CopyBuffer(DestAddress, TempBuffer);
                    //move the compressed data to the destination
    SetEntryPoint(OriginalEntryPoint);
                    //set the original entry point
}
```

Packer algorithm

- A routine-based module usually contains many standard algorithms
- Compression/decompression algorithm sometimes can be modified slightly
- If a wrong decompression algorithm is called, there will be serious problems
- To recognize algorithms precisely is very important

Interrupt emulation

- Tricks to interrupt the emulation
  - FPU (Coprocessor) instructions
  - MMX (multimedia extension) instructions
  - Undocumented CPU instructions
  - SEH (Structured Exception Handler)
    - One of the most difficult problems during the process of emulation
    - The emulated environment in a product cannot handle all kinds of exception perfectly

Interrupt emulation: Fake API calling

- Another trick about anti-emulation is using fake API calls
- Bellow is an example of using fake API call to interrupt the emulator (a variant of the infamous packer “Tibs”)

```
.text:00401A3C
.text:00401A3C                                          public start
.text:00401A3C                          start           proc near
.text:00401A3C 0D C1 2E 42 00                           or      eax, offset unk_422EC1
.text:00401A41 50                                       push    eax
.text:00401A42 50                                       push    eax
.text:00401A43 68 2C 6A 35 F3                           push    0F3356A2Ch
.text:00401A48 E8 66 00 00 00                           call    GetDataLocation
.text:00401A4D
.text:00401A4D                          DecodeLoop:                             ; CODE XREF: start+30↓j
.text:00401A4D 52                                       push    edx
.text:00401A4E 68 14 3B 42 00                           push    offset ExtractAssociatedIconA ; Get handle of an in
.text:00401A4E                                                                  ; in a file or an icon found in an
.text:00401A4E                                                                  ; associated executable file
.text:00401A53 E8 17 00 00 00                           call    _ExtractAssociatedIcon
.text:00401A58 BA A2 54 76 01                           mov     edx, 17654A2h
.text:00401A5D 56                                       push    esi
.text:00401A5E 51                                       push    ecx
.text:00401A5F 52                                       push    edx
.text:00401A60 E8 68 00 00 00                           call    Decode4Bytes
.text:00401A65 56                                       push    esi
.text:00401A66 53                                       push    ebx
.text:00401A67 E8 73 00 00 00                           call    TestEndCondition
.text:00401A6C EB DF                                    jmp     short DecodeLoop
.text:00401A6C                          start           endp
```

WEBSENSE.

Interrupt emulation: complex program logic

- Complex program logic can greatly slow down the emulation process
  - The performance is usually the bottleneck of an emulator
  - The worst situation: the emulator quits without finishing the unpacking process (usually a maximum time-out has been predefined)
  - Example: using long loops to get a decryption key
    - There is nothing special in a real environment
    - The performance may be very low in a virtual environment (emulator)

Emulation stop condition

- Possible emulation stop conditions during unpacking
  - Stop at some API functions
    - Not called by the packer stub
  - Have some entry point signatures from standard compilers
  - Have some special methods to identify the entry point of a certain compiler
    - Delphi

Emulation stop condition

```
000AEF70  E0 D3 4A 00 B0 D3 4A 00  88 CE 4A 00 58 CE 4A 00  嘤J.坝J.埼J.X蝁.
000AEF80  40 9E 4A 00 10 9E 4A 00  9C C9 4A 00 6C C9 4A 00  @澈..澈.溕J.l蔫.
000AEF90  14 D2 4A 00 E4 D1 4A 00  A8 F8 4A 00 78 F8 4A 00  .襃.溲J.  J.x鳰.
000AEFA0  00 00 00 00 B0 F8 4A 00  55 8B EC 83 C4 F0 B8 D8  ....傍J.U嬱冐鸶▌
000AEFB0  F8 4A 00 E8 70 70 F5 FF  A1 6C 42 4B 00 8B 00 E8  鳰.鑠p?  BK.?▌嗃.
000AEFC0  88 FA FA FF A1 6C 42 4B  00 8B 00 BA 20 FC 4A 00  堾?  BK.??鵕.鳰.
000AEFD0  E8 9B F6 FA FF 8B 0D 98  40 4B 00 A1 6C 42 4B 00  钲鳅  ?檎K.  BK.
000AEFE0  8B 00 8B 15 E8 D3 4A 00  E8 77 FA FA FF 8B 0D F4  ??栌J.鑧      ?▌
000AEFF0  42 4B 00 A1 6C 42 4B 00  8B 00 8B 15 1C D2 4A 00  BK.  BK.??.襃.
000AF000  E8 5F FA FA FF A1 6C 42  4B 00 8B 00 E8 D3 FA FA  鑏         BK.?栌
000AF010  FF E8 66 4A F5 FF 00 00  FF FF FF FF 17 00 00 00    鑖J?..         .
000AF020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000AF030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000AF040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000AF050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000AF060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000AF070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000AF080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000AF090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000AF0A0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000AF0B0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000AF0C0  00 00 00 00 00 00 00 02  8D 40 00 32 13 8B C0 02  ........岪.2.嬂.
000AF0D0  00 8B C0 00 8D 40 00 00  8D 40 00 00 8D 40 00 01  .嬂.岪..岪..岪..
000AF0E0  8D 40 00 00 00 00 00 00  00 00 00 5C 21 40 00 EC  岪.........\!@.▌
000AF0F0  22 40 00 60 26 40 00 00  CB CC C8 C9 D7 CF C8 CD  "@.`&@..颂壬紫韧
000AF100  CE DB D8 CA D9 DA DC DD  DE DF E0 E1 E3 00 E4 E5  污厥仝荏捱噌?溴.
000AF110  8D 40 00 52 75 6E 74 69  6D 65 20 65 72 72 6F 72  岪.Runtime error
000AF120  20 20 20 20 20 61 74 20  30 30 30 30 30 30 30 30       at 00000000
000AF130  00 8B C0 45 72 72 6F 72  00 8B C0 30 31 32 33 34  .嬂Error.嬂01234
```

- The nearest “55 8B EC” above the string “Runtime error” is the entry point of the Delphi compiled file

WEBSENSE.

Emulation stop condition

- The emulation stop condition can be forged by packer or malicious code
  - The stop condition is very important in the emulator-based solution
  - If the emulation doesn't stop at a correct time, it may miss the opportunity to get the right unpacked file

Conclusions

- Fighting for the anti routine-based unpacking tricks, we need:
  - Detect the packer more accurately and efficiently
  - Locate the inner structure, and recognize the decompression algorithms more precisely
  - Build the unpack engine more robust in order to handle all kinds of exceptions
- Fighting for the anti emulator-based unpacking tricks, we need:
  - Improve the emulator performance so that the emulator-based unpacking can be put into practice
  - Build stronger emulator, in order to get rid of all kinds of anti-emulator tricks

Conclusions: Handling particularly tricky packers

- For some particularly tricky or modified packers
  - It might be a good strategy to mix emulation and specific routines
    - Using emulation to get polymorphic encryption key, or filter all the garbage instructions
    - Using specific routines to decompress the data if the compression/decompression algorithm is standard
    - This solution will bring additional complexity to the engine

Q & A

Thank you!

Any questions?

WEBSENSE.