

Structural Signature and Structuring Signature

funnywei@xfocus.org

2005-08-19



X'con 2005

Contents

- ❖ Graph theory in binary files comparision
- ❖ Patched code affects assembly structure
- ❖ Isomorphism on three levels
- ❖ Handling compiler optimization
- ❖ Demo
- ❖ Problems



Graph Theory In Binary Files Comparison

◆ Isomorphism

- ◆ Different from Isomorphism in Graph Theory

◆ Structure Assembly Graphs

◆ Related

- ◆ Control Flow Graph Analysis
- ◆ Data Flow Analysis



Patched Code Affects Assembly Structure

- ◆ Additional “validation” function
 - ◆ MS04-022 Task Schedule BOF
- ◆ Additional “if” construct to validate string length
 - ◆ MS04-011 IIS PCT BOF



Isomorphism In Three Levels

◆ Three Levels

- ◆ Call Graph Level

- ◆ Control Flow Graph Level

- ◆ Instruction Level

◆ Each level includes Two phrases

- ◆ Initialization of fixed points

- ◆ Propagation of fixed points



Call Graph Level

- ◆ Signature
 - ◆ basic information
 - ◆ Basic Blocks
 - ◆ Links
 - ◆ Subcalls
 - ◆ structure information
 - ◆ Prime product of assigned primes of all schemas
- ◆ Constraint Properties
 - ◆ Same name
 - ◆ Unique signature
 - ◆ Same indegree(number of references)
 - ◆ Same strings reference
 - ◆ Recursive function
 - ◆ Same prime product



Same Name

- ❖ If two functions have the same name, they will be added as a matched pair.
- ❖ We must exclude situations, when
 - ❖ the library name is unknown
 - ❖ `sub_XXXXXXXX`



Unique Signature

- ◆ Calculate Euclid distance between each two function's signatures.

- ◆ for each x that belongs to Graph B, if

$$\exists a \in A, \forall b \in A, b \neq a, 0 = |x - a| < |x - b|$$

- ◆ add the (x, a) as a matched pair



Same number of references

- ◆ When two functions which have the same number of references
- ◆ Calculate their signature's Euclid distance
- ◆ When they satisfy following condition:

$$x \in B, \exists a \in A, \forall b \in A, b \neq a, 0 = |x - a| < |x - b|$$

- ◆ add (x, a) as a matched pair



Same String Reference

- ❖ Time consuming
- ❖ Convert strings to md5 value



Same Prime Product

This is very time consuming! How we do it?

Function A's Prime Product can be represented by

$$\Pi A = K \cdot 2^{64} + b$$

Function B's Prime Product can be looked as

$$\Pi B = j \cdot 2^{64} + c$$

if $b \neq c$, $\Pi A \neq \Pi B$

If $b = c$, we can assume $\Pi A = \Pi B$, correct?

Total number $l = C_{n+m-1}^n$, here $\Pi c_1 \neq \Pi c_2 \dots \neq \Pi c_l$

$\Pi A \equiv \Pi B \pmod{2^{64}}$ Is less or equal to

$$\left(\frac{P_m^n}{2^{64}} - 1 \right) / C_{n+m-1}^n$$



Control Flow Graph Level

◆ Signature

◆ basic information

- ◆ Number of basic blocks on the shortest path from function entry to itself
- ◆ Number of basic blocks on the shortest path from itself to function exit
- ◆ Subcalls

◆ Structure information

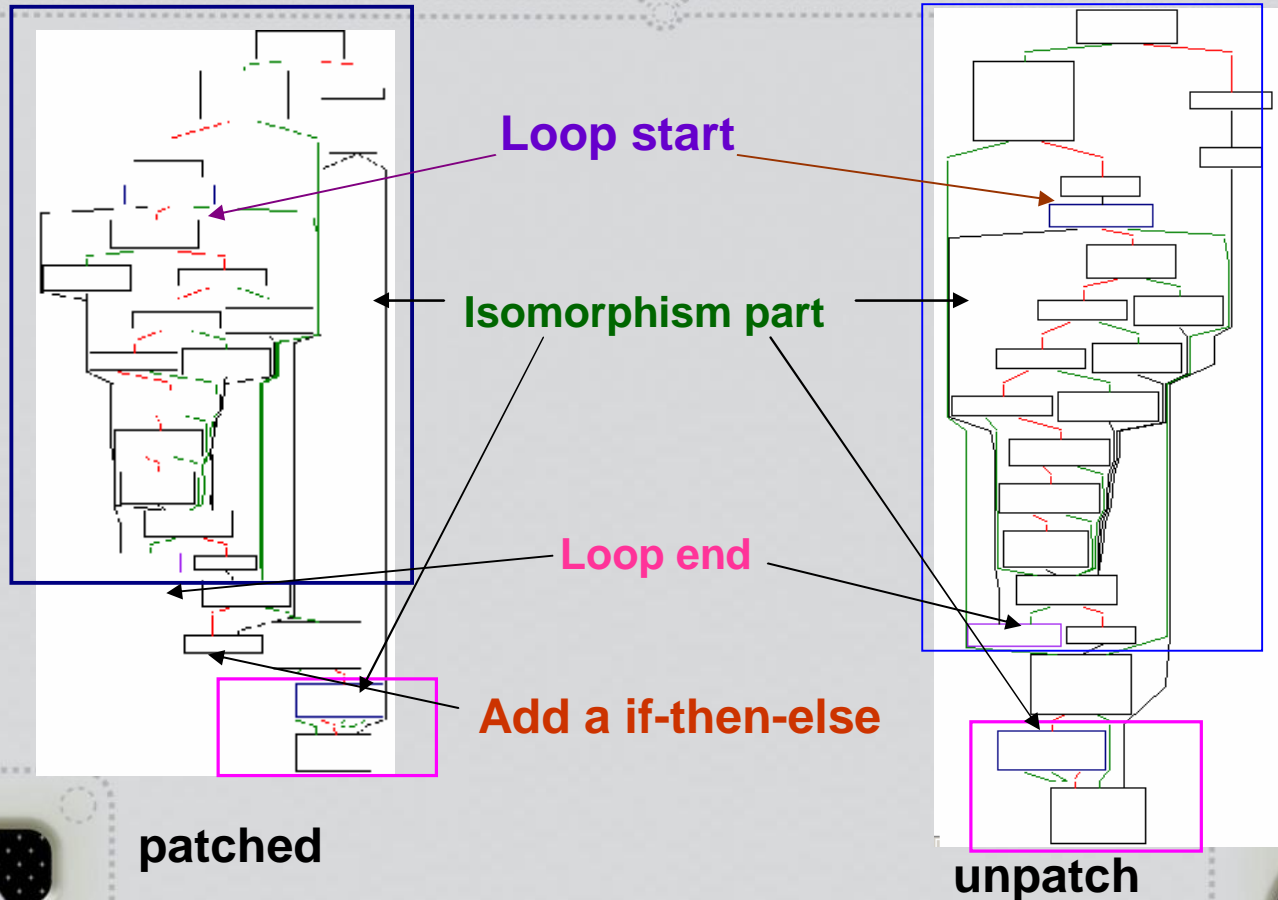
- ◆ Structure level

◆ Constraint properties

- ◆ Same Prime products
- ◆ Same strings reference
- ◆ Same subcalls



MS04-011 Pct1SrvHandleUniHello



patched

unpatch



Structure information

◆ Structural analysis

◆ Structural Schema

- ◆ Sequence schema

- ◆ Conditionals schema

 - ◆ If-then, if-then-else, switch-case

- ◆ Loop schema

 - ◆ Self loop, While loop, Endless loop, multiexit loop

◆ Unstructured Schema

- ◆ Unstructured loops

- ◆ Unstructured conditionals

◆ Add structure information to signature

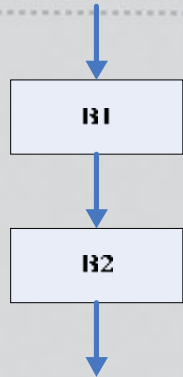


Structure Analysis

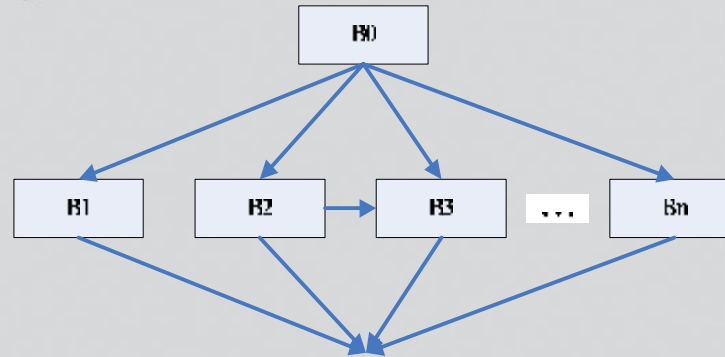
- ❖ Time consuming!
- ❖ But, when we need “visualization of differences between two flow graphs”, this can be very useful.
 - ❖ Isomorphism in control flow graph



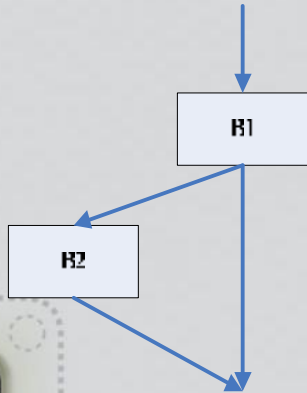
Sequence and conditional Schema



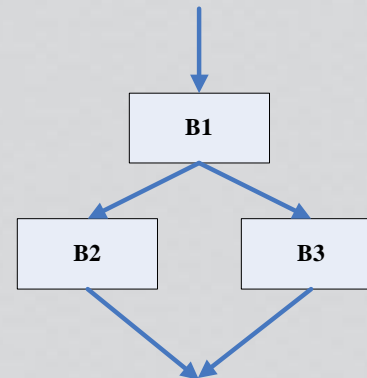
Sequence, can be obsoleted



Switch-case



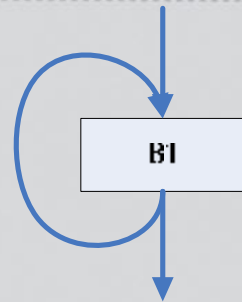
If-then



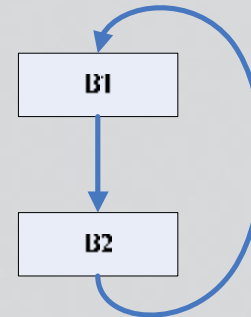
If-then-else



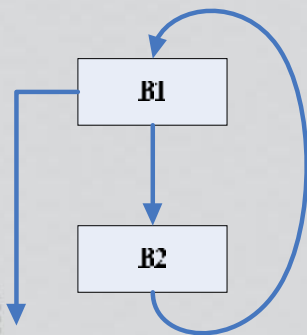
Loop Schema



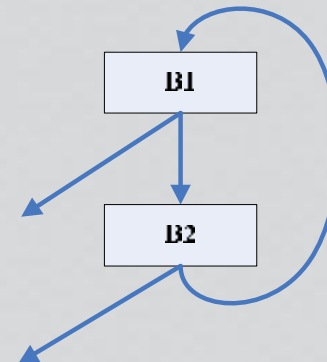
Self loop



Endless loop



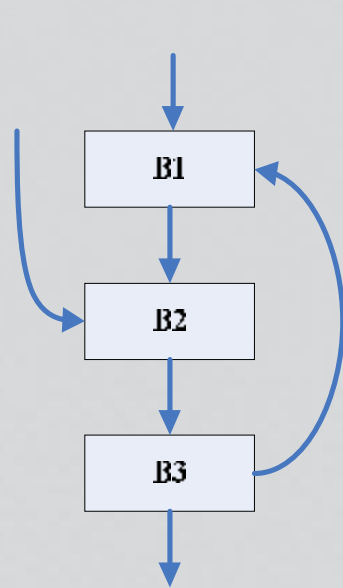
While loop



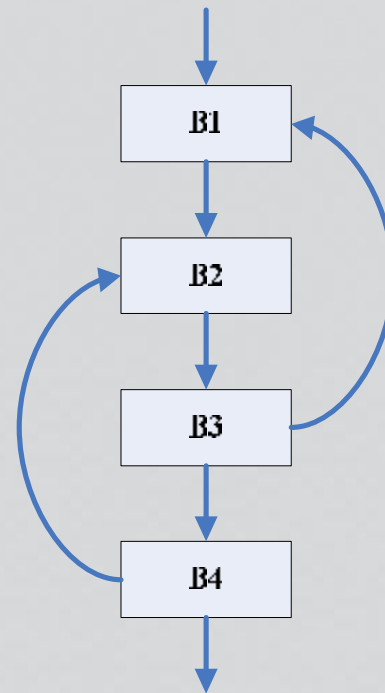
Multi exit



Unstructured Loops



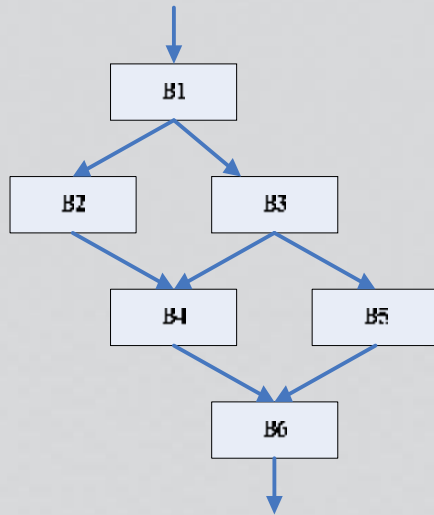
Multi entry



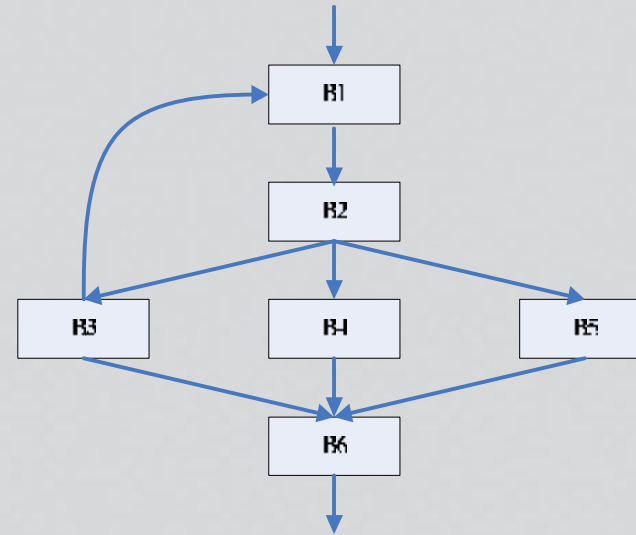
Overlapping



Unstructured conditionals(1)



Unstructured 2-way conditionals

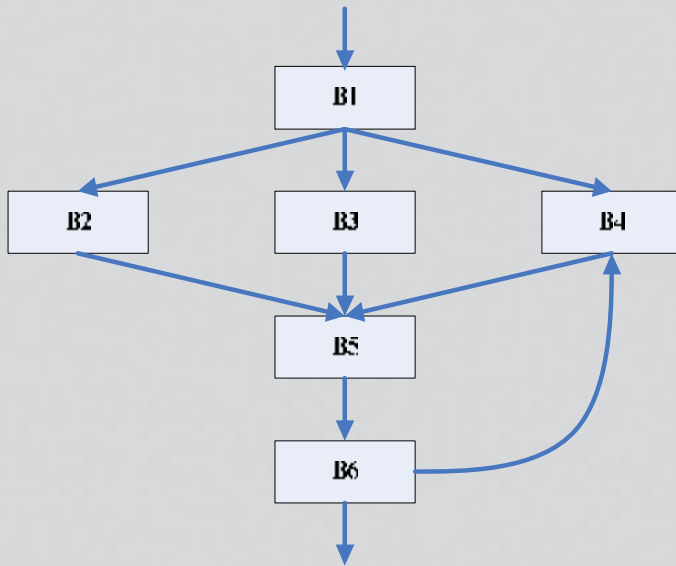


Unstructured N-way conditionals

(1)

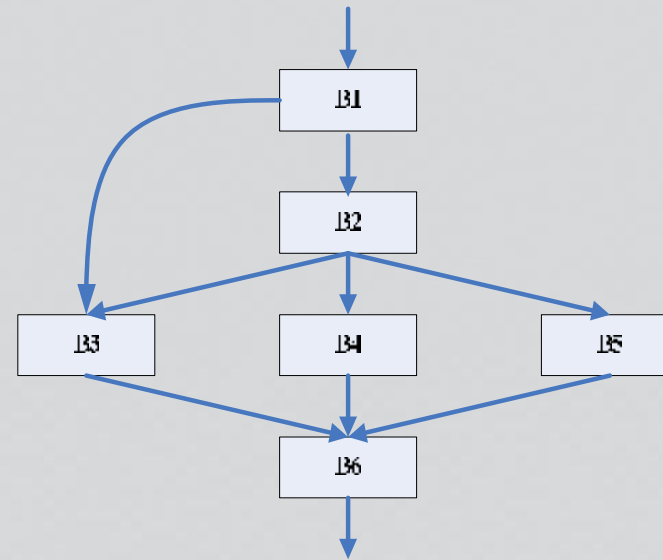


Unstructured conditionals(2)



Unstructured N-way conditionals

(2)

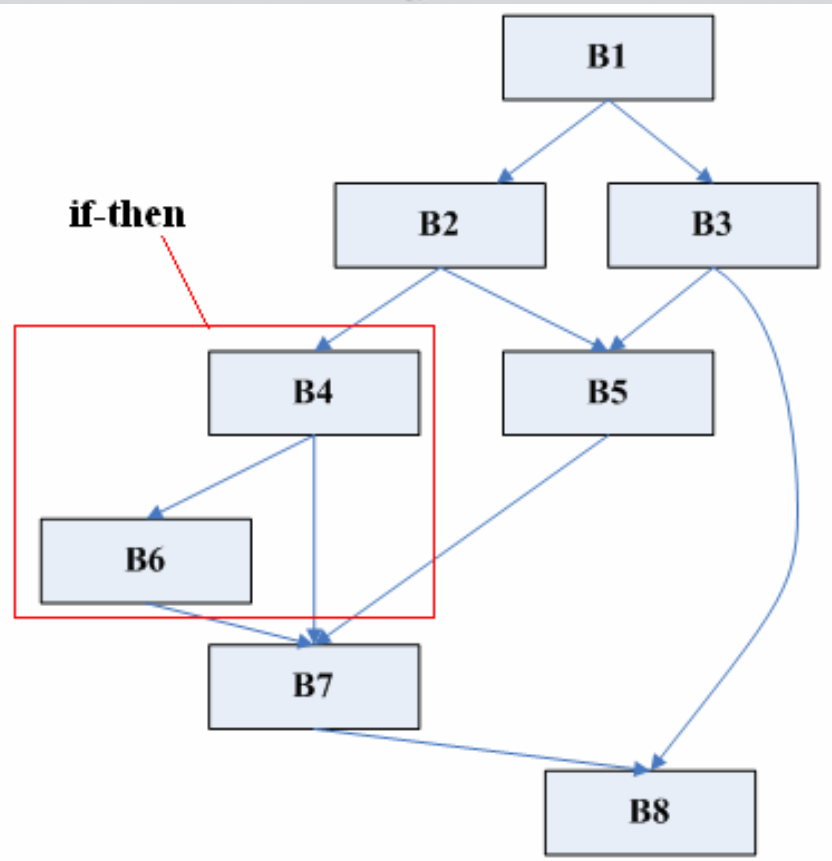


Unstructured N-way conditionals

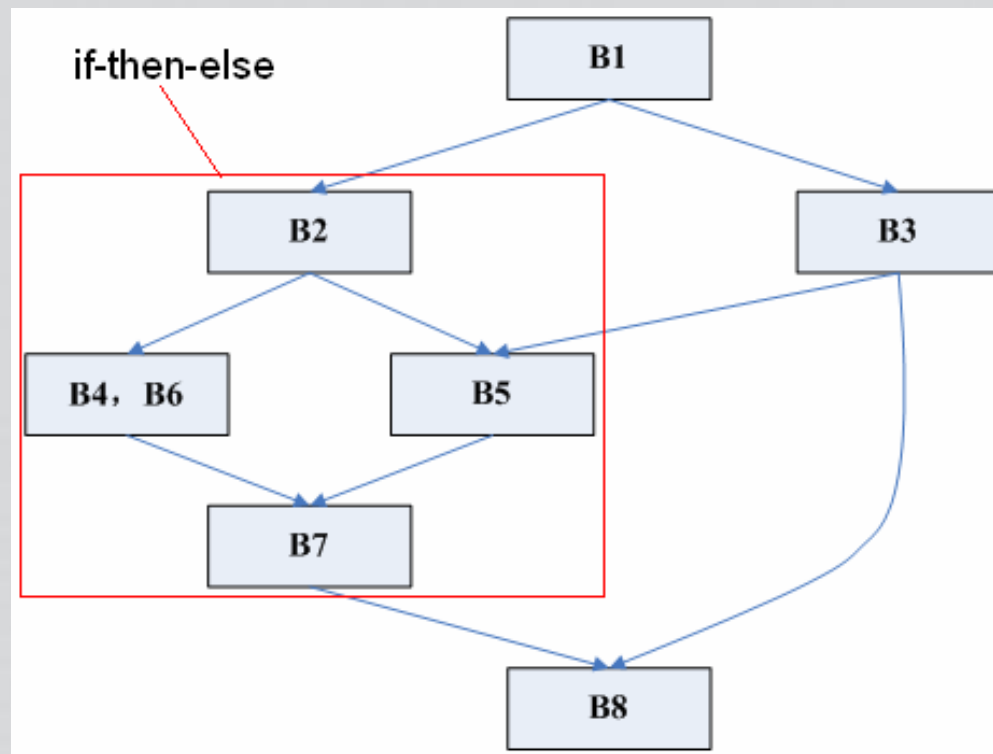
(3)



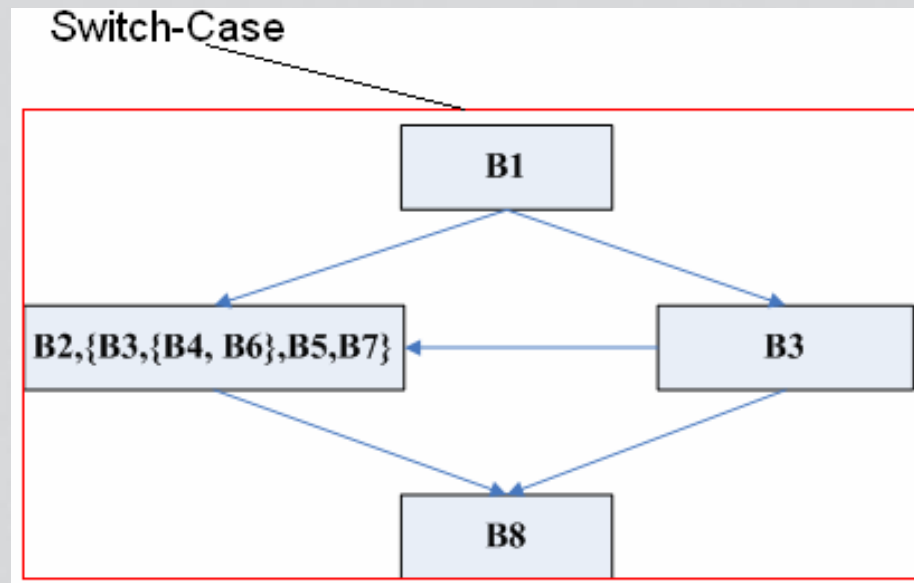
If-then reduce



If-then-else reduce



Switch-case reduction



The other's reduction process is the same



So, the two functions are the same

Assume:

If-then := Prime 2

If-then-else := Prime 3

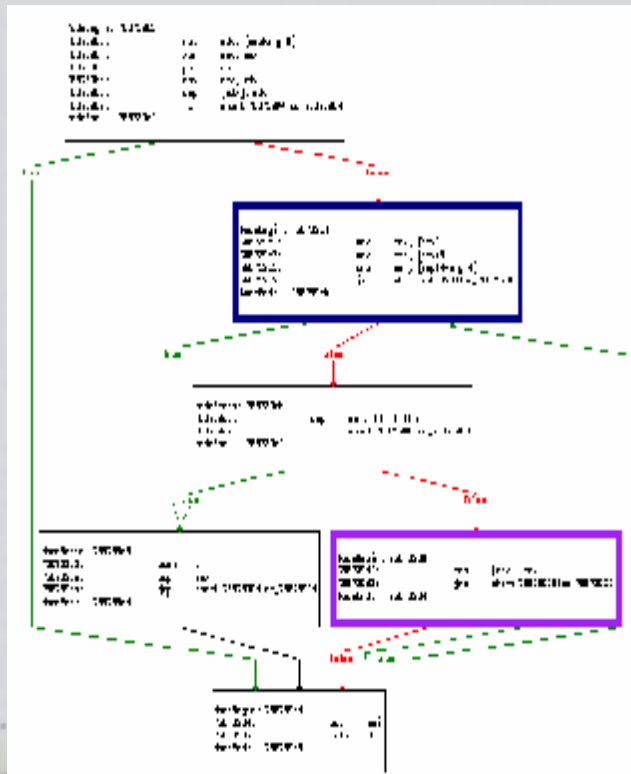
Switch-case := Prime 11

The two structure information sigs = 66

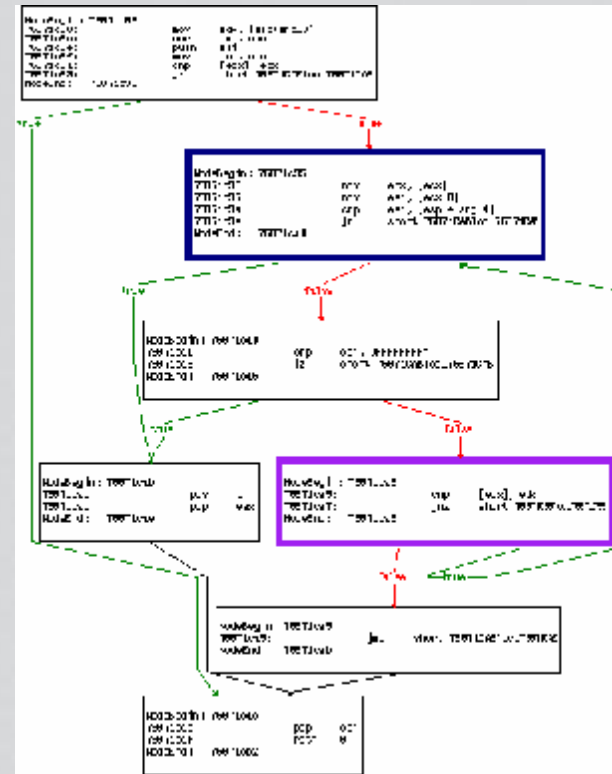


Handling Compiler optimization

Example 1: MsgsSessionInList



Sig = 6::9::0



Sig = 7::10::0



Optimization Handling Algorithm 1.0

In control flow graph G1, for each node whose number of references is 1, its in edge is (y, x), out edges are :

$$j = \{ \langle x, z_1 \rangle, \langle x, z_2 \rangle, \dots, \langle x, z_n \rangle \}$$

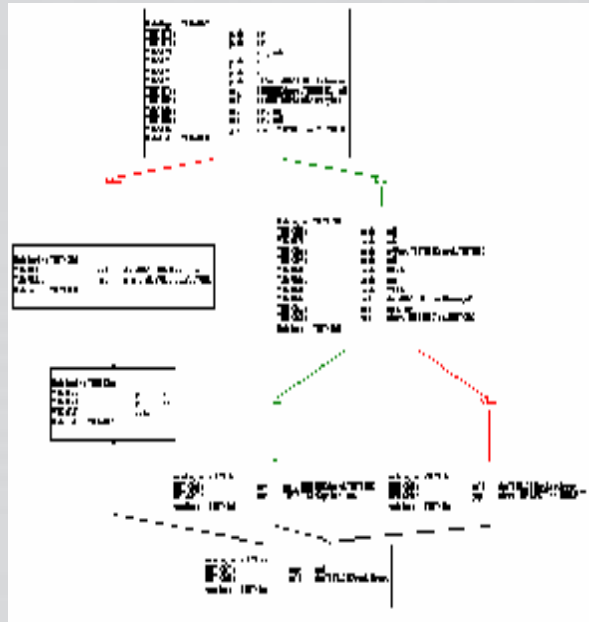
Delete node x, edge (y, x) and all out edges of node x, add edge from y to x's children.

After using algorithm 1.0 example 1's two sigs both are 4::7::0

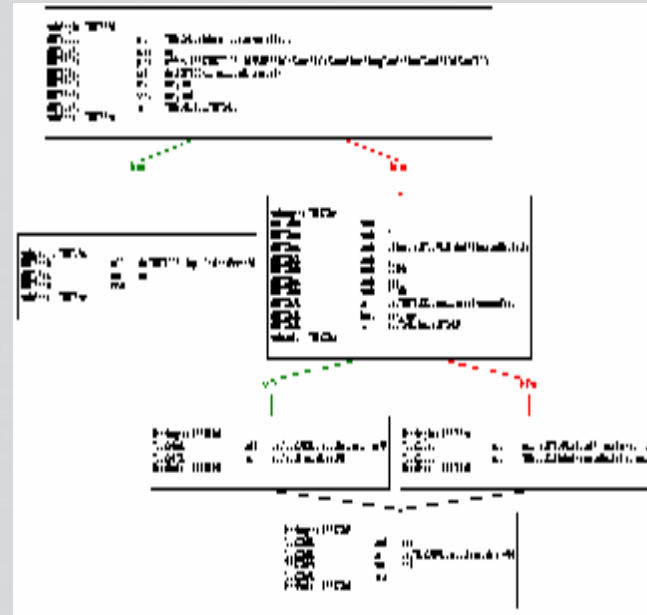


More Than One Return Node, After Using Algorithm 1.0

Example 2



sig3 = 3::4::5

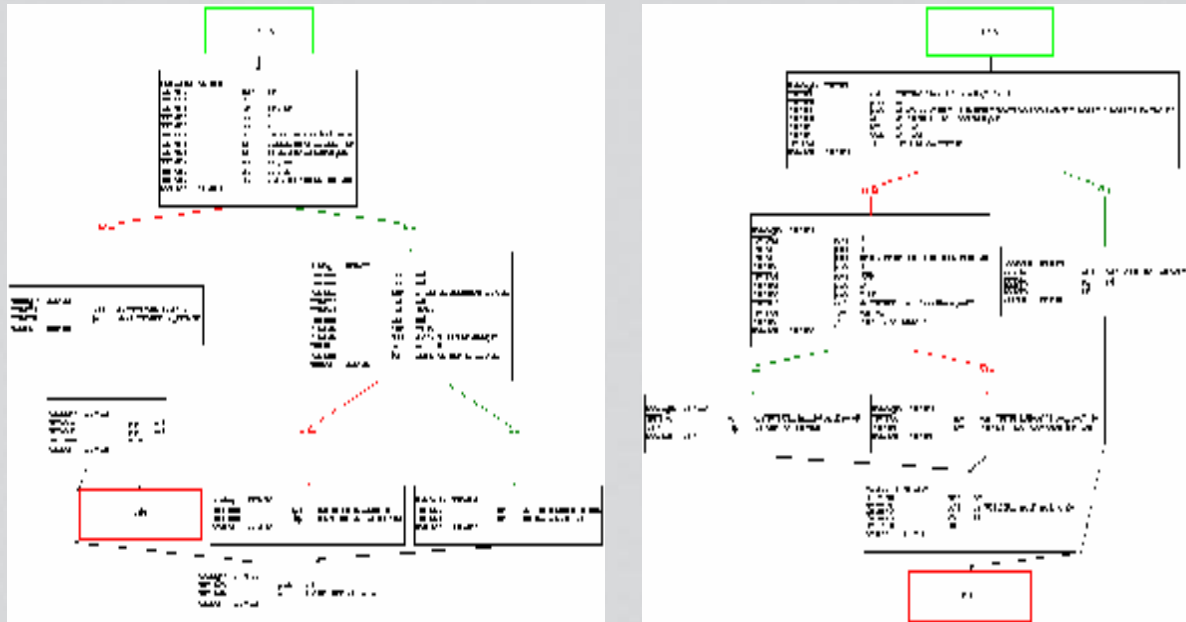


Sig = 3::3::5



Add Virtual Entry and Exit Node

Sig = 3::4::5



- Advantage: 1、 can unify optimization handling process**
2、 useful in loop detection



X'con 2005

Demo



Problems

- ❖ Problems with complete call graph
- ❖ Problems with structural comparison
- ❖ Problems with instruction isomorphism



Problems with Complete Call Graph

- ◆ When we encounter
 - ◆ call esi
 - ◆ call [ebx+0x4]
- ◆ Some cases can be solved by:
 - ◆ Finding last definition in one function
 - ◆ Vtbl functions and runtime binding
 - ◆ Function Pointers Simulation Table
- ◆ Some cases too hard to resolve:
 - ◆ When definition comes from another module's function
 - ◆ When definition comes from Global Uninitialized Variables space
- ◆ Call in loop, each time can produce different value



Finding Last Definition In One Function

```
7C80B7FC ; BOOL __stdcall UnmapViewOfFile(LPCVOID lpBaseAddress)
7C80B7FC public UnmapViewOfFile
7C80B7FC UnmapViewOfFile proc near ; CODE XREF: su
7C80B7FC ; BaseInitAppco
7C80B7FC
7C80B7FC lpBaseAddress = dword ptr 4
7C80B7FC arg_4 = dword ptr 8
7C80B7FC
7C80B7FC mov edi, edi
7C80B7FE push ebp
7C80B7FF mov ebp, esp
7C80B801 push esi
7C80B802 mov esi, ds:NtUnmapViewOfSection
7C80B808 push edi
7C80B809 push [ebp+arg_4]
7C80B80C push 0FFFFFFFh
7C80B80E call esi ; NtUnmapViewOfSection
7C80B810 mov edi, eax
```

usually, when we use function pointer or load dll function in run-time...



Function Pointers Simulation Table

```
00401020 Demo          proc near          ;  
00401020  
00401020 arg_0             = dword ptr 8  
00401020  
00401020          push     esi  
00401021          mov     esi, [esp+arg_0]  
00401025          call   dword ptr [esi]  
00401027          call   dword ptr [esi+4]  
00401028          ...
```

```
00401030 _main             proc near          ; C  
00401030          push   offset simul_vtble  
00401035          call   Demo  
0040103A          pop    ecx
```

```
00407030 simul_vtble       dd offset Demo_1  
00407034          dd offset Demo_2  
00407038 aDemo1           db 'Demo 1',0Ah,0  
00407040 aDemo2           db 'Demo 2',0Ah,0
```



When Definition Comes From another module's function

```
7C862CC1                                     ; UnhandledExceptionFilter+9Bfj
7C862CC1      push      Target
7C862CC7      call     RtlDecodePointer
7C862CCC      cmp     eax, edi
7C862CCE      jz     short loc_7C862CE5
7C862CD0      push   ebx
7C862CD1      call   eax
7C862CD3      cmp     eax, 1
7C862CD6      jz     loc_7C863458
```

Eax is defined by a return value of another module's function



When Definition Comes From Global Uninitialized Var

```
77E9BDB1 loc_77E9BDB1: ; CODE X
77E9BDB1 mov     eax, dword 77EC144C
77E9BDB6 cmp     eax, ebx
77E9BDB8 jz     short loc_77E9BDC7
77E9BDBA push   esi
77E9BDBB call   eax
77E9BDBD cmd   eax, 1
```

SetUnhandledExceptionFilter

UnhandledExceptionFilter

dword_77EC144C: dd 0

```
77E6BC57 SetUnhandledExceptionFilter proc near
77E6BC57
77E6BC57 lpTopLevelExceptionFilter= dword ptr 4
77E6BC57
77E6BC57 mov     ecx, [esp+lpTopLevelExceptionFilter]
77E6BC5B mov     eax, dword_77EC144C
77E6BC60 mov     dword 77EC144C, ecx
77E6BC66 retn   4
```



Call in Loop, each time can produce a different value

```
_main:  
sub    esp, 0Ch  
push  esi  
push  edi  
mov    [esp+14h+var_C], offset loc_401000  
mov    [esp+14h+var_8], offset loc_401010  
mov    [esp+14h+var_4], offset loc_401030  
lea   esi, [esp+14h+var_C]  
mov    edi, 3
```

```
loc_401066:  
call   dword ptr [esi]  
add    esi, 4  
dec    edi  
jnz    short loc_401066
```

true false

```
0040106E:  
pop    edi  
pop    esi  
add    esp, 0Ch  
retn
```

Loop three times

Firstly we should trace edi's D-U chain

Each time, a different subcall

We can add three functions as its children in call graph for fixed points propagation.



Problems with Structural Comparison

- ❖ Mismatching
- ❖ In-deep optimizations
- ❖ Inline functions
- ❖ change of constant values
- ❖ Substitute secure function for unsecure function
- ❖ The different versions of C compiler



Mismatching

- ❖ We can not prove mismatching will not happen.
- ❖ May be influenced by many factors



Substitute secure function for unsecure function

MS04-011 LSASS BOF

unpatch

```
785A0B58      push    [ebp+arg_8]
785A0B5B      mov     eax, 7FFh
785A0B60      sub     eax, esi
785A0B62      push    [ebp+arg_4]
785A0B65      push    eax
785A0B66      lea    eax, [ebp+esi+Buffer]
785A0B6D      push    eax
785A0B6E      call   ds:usprintf
785A0B74      add     esp, 10h
```

patched

```
7859EE5B      push    [ebp+arg_8]
7859EE5E      lea    eax, [ebp+esi+Buffer]
7859EE65      push    [ebp+arg_4]
7859EE68      push    eax
7859EE69      call   ds:usprintf
7859EE6F      add     esp, 0Ch
```

Inline Functions

Will change the caller's subcalls number. Lead to mismatch

1、 Inline prefix

```
__inline int max(int a, int b)
{
    if (a > b) return a;
    return b;
}
```

2、 member function's body in a class definition



Problems with Instruction Isomorphism

- ❖ Instruction reorder
- ❖ Registers reallocation
- ❖ Data flow analysis



 X'con 2005

The End



 XFOCUS TEAM

BEIJING.CHINA

2002-2005