

# Fight against 1-day exploits: Diffing Binaries vs Anti-diffing Binaries

Jeongwook Oh(mat@monkey.org,oh.jeongwook@gmail.com)

Jeongwook Oh works on eEye's flagship product called "Blink". He develops traffic analysis module that filters attacker's traffic. The analysis engine identifies protocol integrity violations by protocol parsing and lowers the chances of false positives and false negatives compared to traditional signature based IPS engines.

He's also interested in blocking ActiveX related attacks and made some special schemes to block ActiveX-based attacks without any false positives. The implementation was integrated to the company's product and used by the customers.

He runs Korean security mailing list called Bugtruck(not bugtraq).

**Blackhat USA 2009 LAS VEGAS, Jul 30**

# Introduction: The Problem

- Security patches are meant to fix security vulnerabilities.
  - fixing problems and protect computers and end users from risks.
- 1-day exploits
  - binary diffing technique can be used to identify the vulnerabilities
  - especially useful for Microsoft's binaries

# Introduction: The Solution

- Purpose: making 1-day exploits difficult and time-consuming
  - Make binary differs' life harder
  - Severe code obfuscation is not an option
  - Need an efficient lightweight code obfuscation
- In-house tool to achieve this
  - Hondon(meaning Chaos)

# Binary Diffing: Demo

- Just grab an idea what binary diffing is.
- We will show simple process of binary diffing.

# Binary Diffing: The History

- BMAP: 10 years ago
- Halvar
  - Bindiff: Expensive commercial tool
  - Not affordable to most non-corporate researchers
- TODD
- eEye
- 2-3 free or opensource tools

# Binary Diffing: BMAT(1999)

- Heavily depends on symbolic name matching
- Used mainly for Microsoft's binaries which symbol they have access to.
- Auxiliary method: 64bit hashing-based comparison for the blocks inside each procedure
  - hashing=multiple level of abstractions with opcode and operands

# Binary Diffing: Automated Reverse Engineering(2004)

- Halvar at Blackhat 2004
- Signature of functions
  - signatures=number of nodes, edges and calls
- Isomorphic comparison between functions CG
  - A function is a node and calling relationship is an edge

# Binary Diffing: Comparing binaries with graph isomorphism(2004)

- Todd Sabin
- Instructions graph's isomorphic matching
- Compares instructions not basic blocks
  - Very unique
- No POC ever released
  - Only testing datasheet released



# Binary Diffing: Structural Comparison of Executable Objects(2004)

- Improved version of Halvar's Blackhat 2004 "Automated Reverse Engineering(2004)"[ARE] presentation[SCEO]

# Binary Diffing: Graph-based comparison of Executable Objects(2005)

- Improved previous paper "Structural Comparison of Executable Objects(2004)"
- Heavily dependent on CFG generation from the binaries

# The Tools: Sabre Security's bindiff(2004)

- Halvar
- A commercial binary diffing tool
- Based on his graph based function fingerprinting theory.

# The Tools: IDACompare(2005)

- Based on signature scanning
- Used for porting malware analysis data
- Designed for around 500k file in size
  - Which is a small size

# The Tools: eEye Binary Diffing Suite(2006)

- Internally used for Microsoft's Patch Tuesday patches analysis
- Patch analysis was the only way to obtain some secret information they don't release
  - You can use eye ball instead of binary diffing tools
  - Some of them has the talent
- The "DarunGrim" is one of the tools included and performs the main binary diffing analysis.

# The Tools: Patchdiff2(2008)

- Made specifically for security patch or hotfix analysis
- Using checksum of graph call for signaturing
- Sounds like similar to bindiff

# The Tools: DarunGrim2(2008)

- The improved version of eEye Binary Diffing Suite
- Using C++ instead of Python to overcome performance and memory footprint issues
- Will be Open-Sourced in few weeks

# DarunGrim2: Algorithms

- The previous works in binary difference analysis were mainly concentrated on the graph structure analysis and graph isomorphism.
  - Intensive comparison of two graphs
  - dependency on the disassembler's CFG analysis capabilities
- **"Basic Block Fingerprint Hash Map"** is the way to overcome this limitation and to improve analysis result drastically.



# Algorithms: Basic Block Fingerprint Hash Map

- Fingerprint hashing method is a main algorithm of DarunGrim2
  - Fingerprint of the block=extracted from instruction sequences
- Two fingerprint hash table for original binary and patched binary
- For each unique fingerprints from original binary
  - DarunGrim2 check if the patched binaries fingerprint hash table has matching entry.

# Algorithms: Basic Block Fingerprint Hash Map

- Generating fingerprint for a basic block
  - Using IDA
- Overcoming Order Dependency
- Reducing Hash Collision
  - Merge multiple fingerprints from parent and children
- Determining matching functions
  - Count the number of matching basic blocks choose the pair that has highest matches
- Matching blocks inside function
  - After function match is determined, use locality.

# Algorithms: Symbolic Names Matching

- Basic starting points for binary matching procedure
- Microsoft is generous enough to provide symbol files as soon as the patch is out

# Algorithms: Structure Based Analysis

- Philosophy of divide and conquer
  - Similar to that of BMAT tool
- Calculating match rate
  - Compare fingerprint string using string match algorithm, same algorithm used in GNU diff(1)
  - Determines "Stop"(If match rate is under n%) or "Go"(If match rate is over n%).
- Need to recognize control flow Inversion
  - Todd's method: categorizing control flow

# DarunGrim2: Real Life Issues

- Split Blocks
- Hot Patching
- Basic Blocks in Multiple Functions

# Real Life Issues: Split Blocks

The screenshot displays two control flow graphs (CFGs) side-by-side. The left graph, for function `sub_757ABEF5`, shows a yellow block `757AC02B` with instructions: `or [ebp+var_4], 0FFFFFFFh`, `call sub_757AC11B`, `mov ebx, [ebp+var_24]`, `mov eax, [ebp+arg_4]`, `cmp dword ptr [eax+0Ch], 3`, and `jnz loc_757AC140`. This block is split into two paths: one leading to red block `757AC044` (`cmp [ebp+var_2C], 2`, `jnb loc_757AC140`) and another leading to red block `757AC140` (`mov eax, [eax+0Ch]`). The right graph, for function `sub_7CB40F90`, shows a yellow block `7CB41185` with instructions: `or [ebp+var_4], 0FFFFFFFh`, `call sub_7CB41195`, `mov ebx, [ebp+var_24]`, and `jmp short loc_7CB411BA`. This block is split into two paths: one leading to red block `7CB411BA` (`mov eax, [ebp+arg_4]`, `cmp dword ptr [eax+0Ch], 3`, `jnz short loc_7CB411D1`) and another leading to red block `7CB411C3` (`cmp [ebp+var_2C], 2`, `jnb short loc_7CB411D1`).

**List Of Matches**

Original	Patched	Match ...	Type	Fingerprint(Original)	Fingerprint(Patched)	Parent(Orig...	Pare
<input type="checkbox"/> 757AC162				7a010204027a0102...			
<input type="checkbox"/> 757AC04E				2c0402			
<input type="checkbox"/> 757ABFA5	7CB4103B	100%	Finger...	06010205028f0402...	06010205028f0402...	0	0
<input type="checkbox"/> 757ABF9E	7CB41031	100%	Tree	8f0502	8f0502	757ABF8C	7CB...

# Real Life Issues: Split Blocks

- "The block who has one child and the child of the block has only one parent in CFG."
- The split blocks tend to make CFG broken
  - The matching process incomplete.
- Need to merge split blocks

# Real Life Issues: Split Blocks

sub\_757ABEF5

```
graph TD
    757AC02B["757AC02B  
or [ebp+var_4], 0FFFFFFFh  
call sub_757AC11B  
mov ebx, [ebp+var_24]  
mov eax, [ebp+arg_4]  
cmp dword ptr [eax+0Ch], 3  
jnz loc_757AC140"]
    757AC044["757AC044  
cmp [ebp+var_2C], 2  
jnb loc_757AC140"]
    757AC140["757AC140  
mov eax, [eax+0Ch]"]
    757AC02B --> 757AC044
    757AC02B --> 757AC140
    757AC044 --> 757AC140
```

sub\_7CB40F90

```
graph TD
    7CB41185["7CB41185  
or [ebp+var_4], 0FFFFFFFh  
call sub_7CB41195  
mov ebx, [ebp+var_24]  
jmp short loc_7CB411BA  
mov eax, [ebp+arg_4]  
cmp dword ptr [eax+0Ch], 3  
jnz short loc_7CB411D1"]
    7CB411C3["7CB411C3  
cmp [ebp+var_2C], 2  
jnb short loc_7CB411D1"]
    7CB411D1["7CB411D1"]
    7CB41185 --> 7CB411C3
    7CB41185 --> 7CB411D1
    7CB411C3 --> 7CB411D1
```

List Of Matches

Original	Patched	Match ...	Type	Fingerprint(Original)	Fingerprint(Patched)	Parent(Orig...	Par
<input type="checkbox"/> 757AC02B	7CB41185	100%	Fingerp...	830402050210070...	830402050210070...	0	0
<input type="checkbox"/> 757AC025	7CB410BC	100%	Tree	7a010204027a0402...	7a010204027a0402...	757AC013	7Ct
<input type="checkbox"/> 757AC013	7CB410AA	100%	Tree	8f02028f04021007...	8f02028f04021007...	757ABFE0	7Ct
<input type="checkbox"/> 757AC001	7CB41095	100%	Tree	8f05028f01028f050...	8f05028f01028f050...	757ABFE0	7Ct



# Real Life Issues: Hot Patching

```
.text:765D1E9C ; int __stdcall sub_765D1E9C(unsigned __int8  
*NetworkAddr,int)
```

```
.text:765D1E9C sub 765D1E9C proc near
```

```
.text:765D1E9C mov eax, eax
```

```
.text:765D1E9E
```

```
.text:765D1E9E ; __stdcall W32TimeGetNetlogonServiceBits(x, x)
```

```
.text:765D1E9E _W32TimeGetNetlogonServiceBits@8:
```

```
.text:765D1E9E push ebp
```

```
.text:765D1E9F mov ebp, esp
```

```
.text:765D1EA1 push 0FFFFFFFFh
```

```
.text:765D1EA3 push offset dword_765D1F80
```

- Solution: Just ignore any hot patching preamble
- Pattern: mov RegA,RegA at the start of a function

# Real Life Issues: Basic Blocks in Multiple Functions

- Usually one basic block belongs to one function
- There are some cases that one basic block can be part of multiple functions.
  - For example: Windows kernel
- The limitation with IDA
  - One function for one basic block

# Real Life Issues: Basic Blocks in Multiple Functions

- Perform additional custom CFG analysis
  - Doesn't totally rely on IDA's CFG analysis
- Design data structure to make it possible for
  - a basic block can belong to multiple functions.

# Real Life Issues: Instruction Reordering

- During ARM binaries diffing experiments
  - we found that there are a lot of instruction reordering happen over each releases.
  - Binary differ is confused a lot and mark all the same blocks as being different

# Real Life Issues: Instruction Reordering

T:\mat\Projects\ResearchTools\Binary\StaticAnalysis\DarunGrim2\src\TestCases\iPhon...

File Graphs Help

func\_4C000

func\_37B20

4C0B0

```
LDR R0, =off_6B614
LDR R1, =unk_6CFC8
LDR R2, =unk_6D0F8
MOV R3, #0
LDR R0, [R0]
LDR R1, [R1]
LDR R2, [R2]
STR R3, [SP]
MOV R3, R4
BL _objc_msgSend
```

4C0D8

```
SUB SP, R7, #0x18
LDMFD SPI, {R8,R10,R11}
LDMFD SPI, {R4-R7,PC}
```

37BCC

```
LDR R0, =off_6455C
LDR R1, =off_638E4
LDR R2, =off_638E8
MOV R3, #0
STR R3, [SP]
LDR R0, [R0]
LDR R1, [R1]
LDR R2, [R2]
MOV R3, R4
BL _objc_msgSend
```

37BF4

```
SUB SP, R7, #0x18
LDMFD SPI, {R8,R10,R11}
LDMFD SPI, {R4-R7,PC}
```

List Of Matches

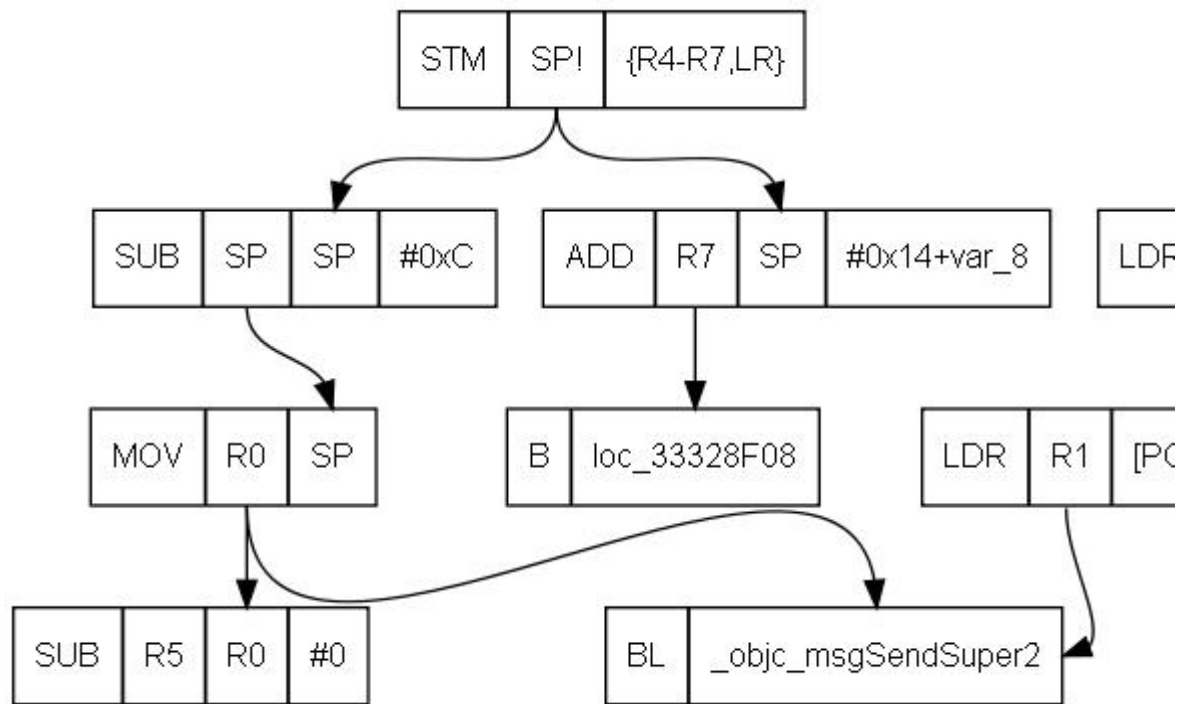
Original	Patched	Match ...	Type	Fingerprint(Original)	Fingerprint(Patched)	Parent(Orig...	Pa
<input type="checkbox"/> 4C0B0	37BCC	95%	Tree	1e010202021e0102...	1e010202021e0102...	4C078	37
<input type="checkbox"/> 4C0D8	37BF4	100%	Tree	0c0102010205022...	0c0102010205022...	4C078	37
<input type="checkbox"/> 4C04C	37B6C	88%	Tree	1e010202021e0102...	1e010202021e0102...	4C000	37
<input type="checkbox"/> 4C078	37B94	100%	Fingerp...	1e010204021e0102...	1e010204021e0102...	0	0

# Real Life Issues: Instruction Reordering

<i>Original</i>	<i>Patched</i>
STMFD SP!, {R4-R7,LR} ADD R7, SP, #0x14+var_8 LDR R3, =(off_3AFD9AAC - 0x32FF9A80) SUB SP, SP, #0xC LDR R1, =(off_3AFD86B8 - 0x32FF9A88) LDR R3, [PC,R3] STR R0, [SP,#0x20+var_20] LDR R1, [PC,R1] ; "initWithPath:" MOV R0, SP MOV R6, R2 STR R3, [SP,#0x20+var_1C] BL _objc_msgSendSuper2 SUBS R5, R0, #0 BEQ loc_32FF9B84	STMFD SP!, {R4-R7,LR} ADD R7, SP, #0x14+var_8 SUB SP, SP, #0xC LDR R3, =(off_3B2CF6C8 - 0x33328E08) LDR R1, =(off_3B2CDE70 - 0x33328E10) STR R0, [SP,#0x20+var_20] LDR R3, [PC,R3] MOV R0, SP LDR R1, [PC,R1] ; "initWithPath:" MOV R6, R2 STR R3, [SP,#0x20+var_1C] BL _objc_msgSendSuper2 SUBS R5, R0, #0 BEQ loc_33328F08

# Real Life Issues: Instruction Reordering

Generate Data flow graph and serialize each node



# Real Life Issues: Instruction Reordering

<i>Original</i>	<i>Patched</i>
<pre> STMFD SP!, {R4-R7,LR} ADD R7, SP, #0x14+var_8 LDR R3, =(off_3AFD9A80 - 0x32FF9A80) SUB SP, SP, #0xC LDR R1, =(off_3AFD9A80 - 0x32FF9A88) LDR R3, [PC,R3] STR R0, [SP,#0x20+var_20] LDR R1, [PC,R1] MOV R0, SP MOV R6, R2 STR R3, [SP,#0x20+var_20] BL _objc_msgSend SUBS R5, R0, #0 BEQ loc_32FF9B84 </pre>	<pre> STMFD SP!, {R4-R7,LR} ADD R7, SP, #0x14+var_8 SUB SP, SP, #0xC BEQ loc_32FF9B84 MOV R0, SP SUBS R5, R0, #0 STR R0, [SP,#0x20+var_20] LDR R3, =(off_3AFD9AAC - 0x32FF9A80) LDR R3, [PC,R3] STR R3, [SP,#0x20+var_1C] LDR R1, =(off_3AFD86B8 - 0x32FF9A88) LDR R1, [PC,R1] ; "initWithPath:" BL _objc_msgSendSuper2 MOV R6, R2 </pre>
<i>Original</i>	<i>Patched</i>
<pre> STMFD SP!, {R4-R7,LR} ADD R7, SP, #0x14+var_8 SUB SP, SP, #0xC BEQ loc_32FF9B84 MOV R0, SP SUBS R5, R0, #0 STR R0, [SP,#0x20+var_20] LDR R3, =(off_3AFD9AAC - 0x32FF9A80) LDR R3, [PC,R3] STR R3, [SP,#0x20+var_1C] LDR R1, =(off_3AFD86B8 - 0x32FF9A88) LDR R1, [PC,R1] ; "initWithPath:" BL _objc_msgSendSuper2 MOV R6, R2 </pre>	<pre> STMFD SP!, {R4-R7,LR} ADD R7, SP, #0x14+var_8 SUB SP, SP, #0xC BEQ loc_33328F08 MOV R0, SP SUBS R5, R0, #0 STR R0, [SP,#0x20+var_20] LDR R3, =(off_3B2CF6C8 - 0x33328E08) LDR R3, [PC,R3] STR R3, [SP,#0x20+var_1C] LDR R1, =(off_3B2CDE70 - 0x33328E10) LDR R1, [PC,R1] ; "initWithPath:" BL _objc_msgSendSuper2 MOV R6, R2 </pre>



# Examples

Microsoft's Binaries  
Non-Microsoft's Binaries  
Malwares

# Gathering Binaries

- Each vendors patch pages
  - Use MS patches pages
- Need to archive binary files for future patch releases
  - SortExecutables.exe: Sort PE binaries according to the version information.
  - <Company Name>\<File Name>\<Version Name>

# Gathering Binaries: SortExecutables

- You can make your own archive of binaries in more organized way

```
T:\PROJECTS\BINARIES\WINDOWS XP\MICROSOFT CORPORATION\MSHTML
—6.00.2600.0000 (xpclient.010817-1148)
—6.00.2800.1528
—6.00.2800.1561
—6.00.2800.1562
—6.00.2900.2180 (xpsp_sp2_rtm.040803-2158)
—6.00.2900.2604 (xpsp.041130-1728)
—6.00.2900.2604 (xpsp_sp2_gdr.041130-1729)
—6.00.2900.3020 (xpsp_sp2_gdr.061023-0214)
—6.00.2900.3492 (xpsp_sp2_qfe.081212-1622)
—6.00.2900.5512 (xpsp.080413-2105)
—6.00.2900.5659 (xpsp_sp3_gdr.080819-1237)
—6.00.2900.5659 (xpsp_sp3_qfe.080819-1352)
—6.00.2900.5694 (xpsp_sp3_qfe.081015-1409)
—6.00.2900.5726 (xpsp_sp3_gdr.081212-1450)
—6.00.2900.5726 (xpsp_sp3_qfe.081212-1451)
—7.00.6000.16788 (vista_gdr.081211-1619)
—7.00.6000.16809 (vista_gdr.090114-1504)
—8.00.6001.18702 (longhorn_ie8_rtm(wmbla).090308-0339)
```

# Performing Diffing

- Using DarunGrim2.exe and Two IDA sessions
  - First launch DarunGrim2.exe
  - Launch two IDA sessions
  - First run DarunGrim2 plugin from the original binary
  - Secondly run DarunGrim2 plugin from the patched binary
- Using DarunGrim2C.exe command line tool
  - Handy
  - Batch-able
  - Quick

# The infamous MS08-067(which was exploited by Conficker)

- Conficker worm exploited this vulnerability to propagate through internal network.
- Easy target for binary diffing: only 2 functions changed.
- One is a change in calling convention.
- The other is the function that has the vulnerability

# The infamous MS08-067 (which was exploited by Conficker)

The screenshot displays a static analysis tool interface. At the top, the window title is "T:\mat\Projects\ResearchTools\Binary\StaticAnalysis\DarunGrim2\src\TestCases\MS08-067-Vulnerability in Server Service C...". The interface includes a menu bar with "File", "Graphs", and "Help". Below the menu bar are two side-by-side control flow graphs (CFGs) for subroutines "sub\_5B86A51B" and "sub\_5B86A51A". The CFGs consist of red rectangular nodes connected by black lines representing control flow edges. Some nodes are highlighted in yellow. Below the CFGs is a "List Of Matches" section with tabs for "Functions" and "Blocks". The "Blocks" tab is active, showing a table of matches between original and patched code blocks.

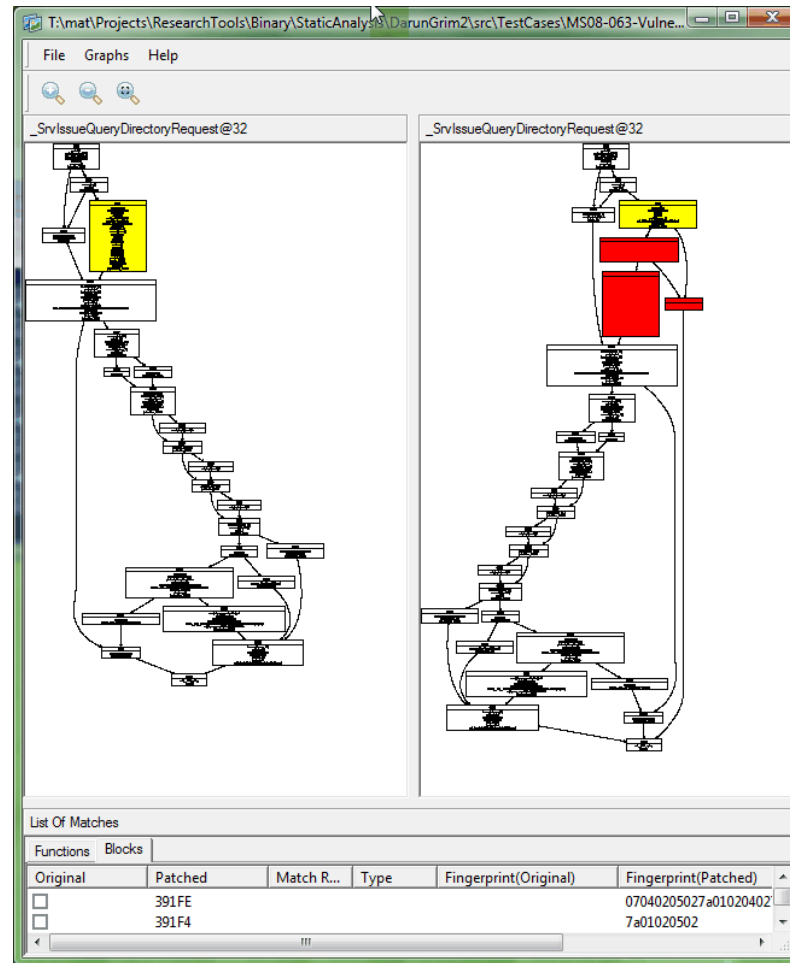
Original	Patched	Mat...	Type	Fingerprint(Original)	Fingerprint(Patched)	Parent(Original)	Pa
<input type="checkbox"/> 5B878B09	5B878B25	100%	Fingerprint	5c010204027a01020102	5c010204027a01020102	0	0
<input type="checkbox"/> 5B86D865	5B86A561	100%	Tree	2c01022c0102	2c01022c0102	5B86D85C	5B
<input type="checkbox"/> 5B86D887	5B878A90	100%	Tree	1b01010101	1b01010101	5B86D873	5B
<input type="checkbox"/> 5B86D85C	5B878A60	100%	Tree	d201010101	d201010101	5B86D857	5B
<input type="checkbox"/> 5B871ABB	5B86A591	100%	Tree	d801020102	d801020102	5B86D869	5B
<input type="checkbox"/> 5B86D857	5B878A58	100%	Tree	1b01010101	1b01010101	5B86D84E	5B
<input type="checkbox"/> 5B86D869	5B878A76	100%	Tree	1b03010501	1b03010501	5B86D84E	5B

# MS08-063: DarunGrim2 vs bindiff

## Modified Functions

Original	Unmat...	Patched	Unmat...	Different	Matched	M...
<input type="checkbox"/> _SrvCompleteRfcbClose@4	0	_SrvCompleteRfcbClose@4	1	3	18	90%
<input type="checkbox"/> @SrvRestartRawReceive@4	0	@SrvRestartRawReceive@4	1	5	25	90%
<input type="checkbox"/> _SrvIssueQueryDirectoryRequest@32	0	_SrvIssueQueryDirectoryRequest@32	2	1	23	94%
<input type="checkbox"/> func_1D0E4	0	func_1CD48	0	1	11	95%
<input type="checkbox"/> @SrvFsdRestartPrepareRawMdlWrite...	0	@SrvFsdRestartPrepareRawMdlWrite@4	3	1	43	95%
<input type="checkbox"/> _SrvRequestOplock@12	0	_SrvRequestOplock@12	0	2	40	97%
<input type="checkbox"/> _GenerateOpen2Response@8	0	_GenerateOpen2Response@8	0	1	57	99%

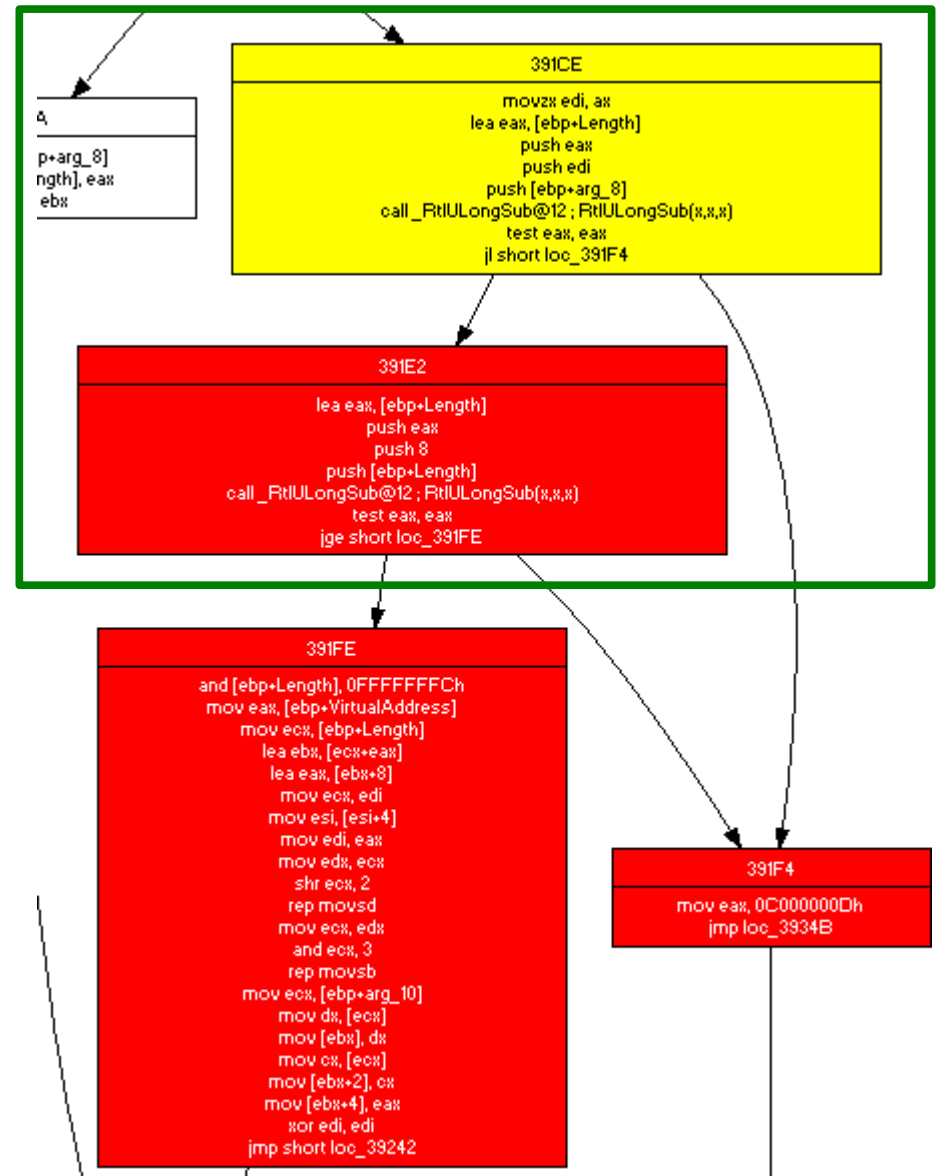
# MS08-063: DarunGrim2 vs bindiff \_SrvIssueQueryDirectoryRequest@32





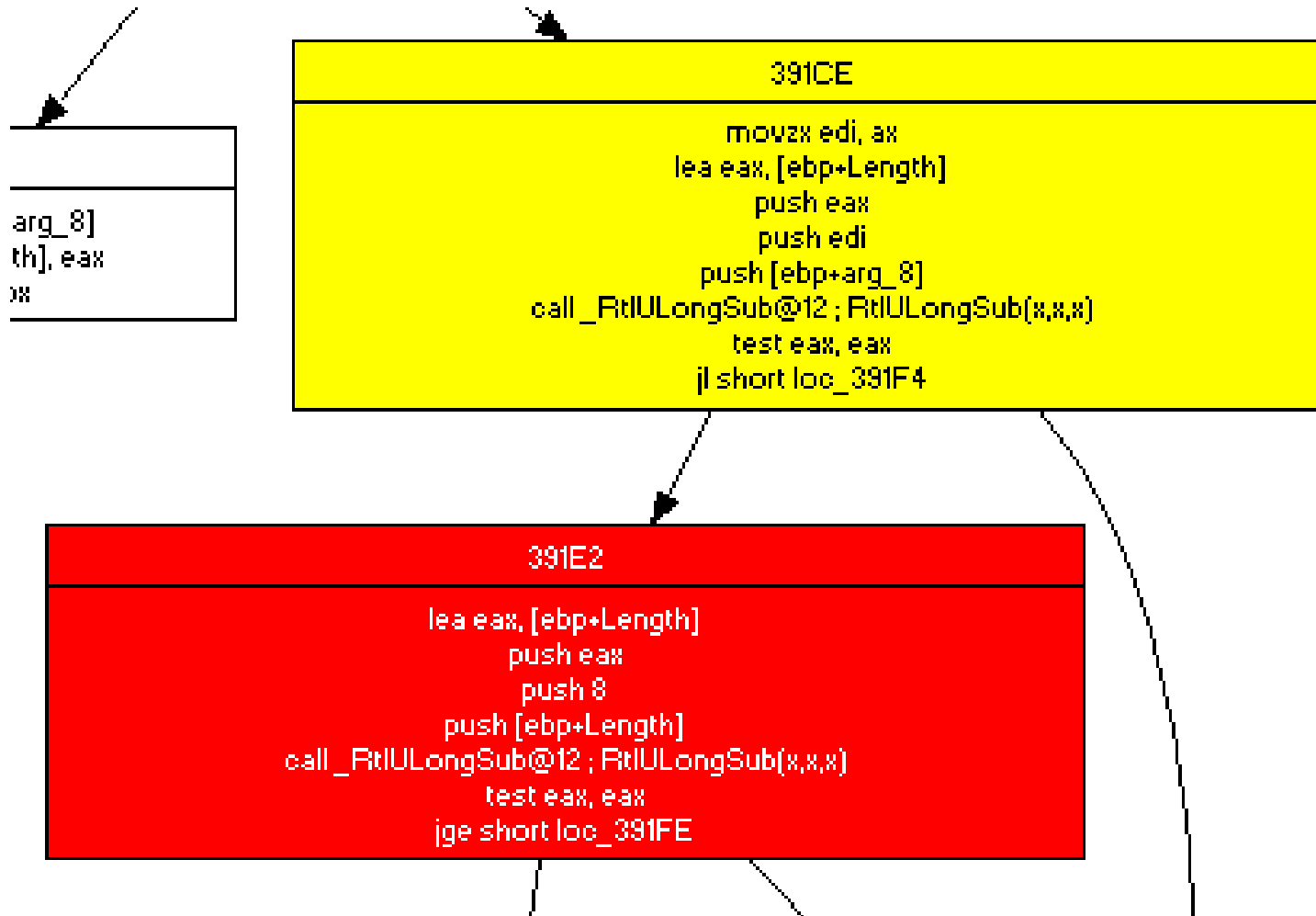
# MS08-063: DarunGrim2 vs bindiff Patched Blocks

```
39260
movzx ecx, cx
mov ebx, [ebp+arg_8]
sub ebx, ecx
sub ebx, 8
and ebx, 0FFFFFFFCh
mov [ebp+Length], ebx
mov edx, [ebp+VirtualAddress]
add ebx, edx
lea edi, [ebx+8]
mov esi, [eax+4]
mov edx, ecx
shr ecx, 2
rep movsd
mov ecx, edx
and ecx, 3
rep movsb
mov cx, [eax]
mov [ebx], cx
mov ax, [eax]
mov [ebx+2], ax
lea eax, [ebx+8]
mov [ebx+4], eax
xor edi, edi
jmp short loc_392A9
```



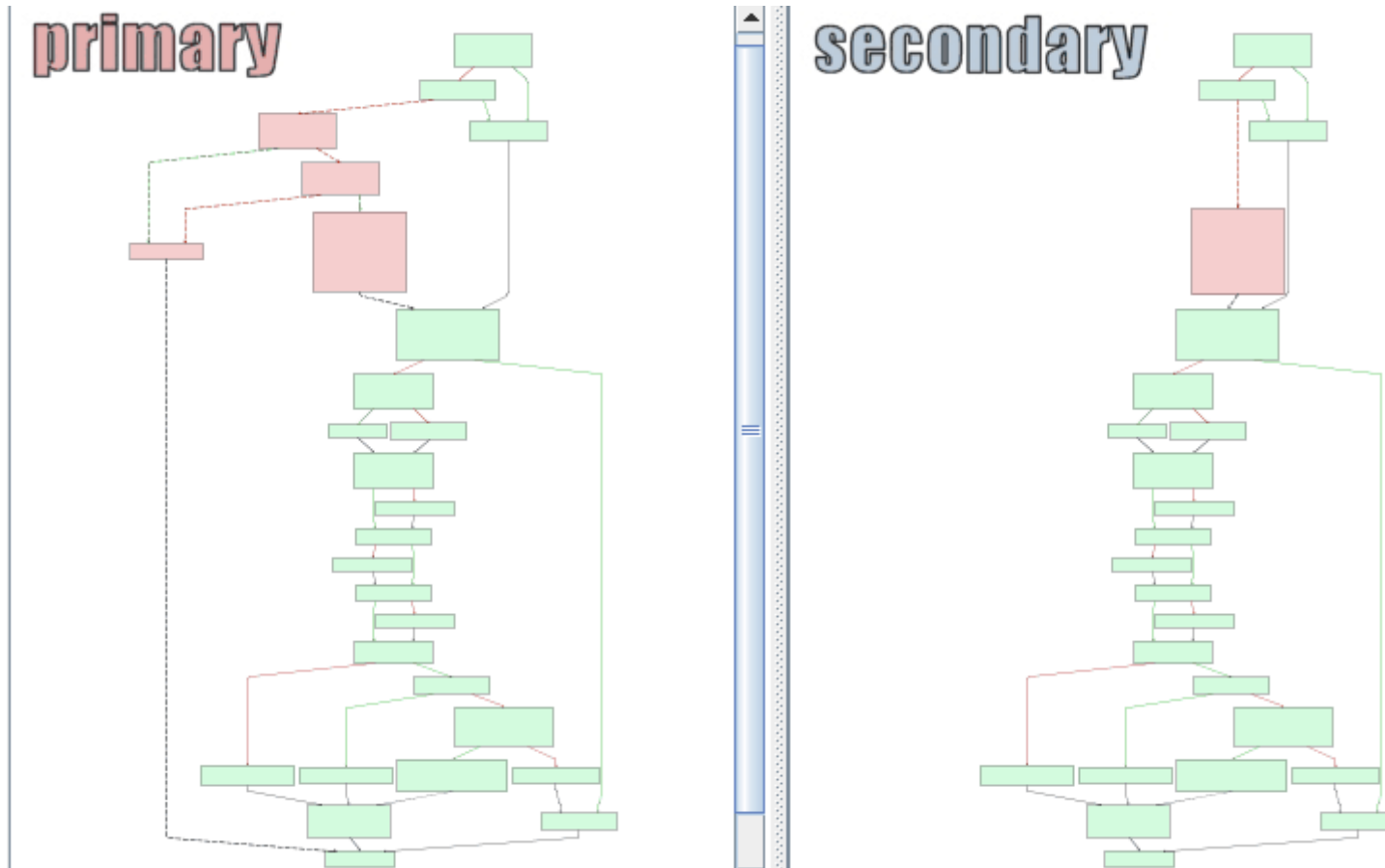
# MS08-063: DarunGrim2 vs bindiff

## Patched Blocks



# MS08-063: DarunGrim2 vs bindiff

## Bindiff Results



# MS08-063: DarunGrim2 vs bindiff

## False Negatives

---

\_SrvCompleteRfcbClose@4  
@SrvRestartRawReceive@4  
\_SrvIssueQueryDirectoryRequest@32  
func\_1D0E4  
@SrvFsdRestartPrepareRawMdiWrite...  
\_SrvRequestOplock@12  
\_GenerateOpen2Response@8

VS

SrvFsdRestartPrepareRawMdiWrite  
SrvIssueQueryDirectoryRequest  
SrvRestartRawReceive

7

VS

3

# MS09-020: WebDav case

## Patched Function looks almost same

?ScConvertToWide@@YJPDPA... 0 ?ScConvertToWide@@YJPDPA... 0 10 16 80%

```
?ScConvertToWide@@YJPDPAIPAG0H@Z  
lea ebx, [eax+1]  
mov eax, [ebp-12Ch]  
push dword ptr [eax]; cchWideChar  
mov eax, [ebp-124h]  
push dword ptr [ebp-130h]; lpWideCharStr  
sub eax, esi  
push ebx; cchMultiByte  
push dword ptr [ebp-128h]; lpMultiByteStr  
neg eax  
sbb eax, eax  
and eax, 8  
push eax; dwFlags  
push dword ptr [ebp-124h]; CodePage  
call edi; MultiByteToWideChar(x,x,x,x,x,x); MultiByteToWideChar(  
test eax, eax  
jnz short loc_6F069752
```

Original

```
6F0696F2  
call ds: __imp__ GetLastError@0; GetLastError()  
cmp eax, 7Ah
```

Patched

```
6F0696B9  
push dword ptr [ebx]; cchWideChar  
mov esi, ds: __imp__ MultiByteToWideChar@24; MultiByteToWideChar(x,x,x,x,x,x)  
push dword ptr [ebp-12Ch]; lpWideCharStr  
sub eax, ecx  
lea edi, [eax+1]  
push edi; cchMultiByte  
push dword ptr [ebp-124h]; lpMultiByteStr  
push 8; dwFlags  
push dword ptr [ebp-128h]; CodePage  
call esi; MultiByteToWideChar(x,x,x,x,x,x); MultiByteToWideChar(x,x,x,x,x,x)  
test eax, eax  
jnz short loc_6F069739
```

```
6F0696E1  
mov ebx, ds: __imp__ GetLastError@0; GetLastError()  
call ebx; GetLastError(); GetLastError(); GetLastError()  
cmp eax, 7Ah  
jnz short loc_6F069707
```

# MS09-020: WebDav case

## Flags has changed

Original

```
lea ecx, [eax*1]
mov eax, [ebp-12Ch]
push dword ptr [eax]; cchWideChar
mov eax, [ebp-124h]
push dword ptr [ebp-130h]; lpWideCharStr
sub eax, esi
push ebx; cchMultiByte
push dword ptr [ebp-128h]; lpMultiByteStr
neg eax
sbb eax, eax
and eax, 8
push eax; dwFlags
push dword ptr [ebp-124h]; CodePage
call edi; MultiByteToWideChar(x,x,x,x,x); MultiByteToWideChar(
```

Patched

```
push dword ptr [ebx]; cchWideChar
mov esi, ds: __imp__MultiByteToWideChar@24; MultiByteToWideChar(x,x,x,x,x)
push dword ptr [ebp-12Ch]; lpWideCharStr
sub eax, ecx
lea edi, [eax+1]
push edi; cchMultiByte
push dword ptr [ebp-124h]; lpMultiByteStr
push 8; dwFlags
push dword ptr [ebp-128h]; CodePage
call esi; MultiByteToWideChar(x,x,x,x,x); MultiByteToWideChar(x,x,x,x,x)
```

# MS09-020: WebDav case

## What does flag 8 mean?

MSDN([http://msdn.microsoft.com/en-us/library/dd319072\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/dd319072(VS.85).aspx)) declares like following:

MB\_ERR\_INVALID\_CHARS Windows Vista and later: The function does not drop illegal code points if the application does not set this flag.  
Windows 2000 Service Pack 4, Windows XP: Fail if an invalid input character is encountered. **If this flag is not set, the function silently drops illegal code points.** A call to GetLastError returns ERROR\_NO\_UNICODE\_TRANSLATION.

# MS09-020: WebDav case Broken UTF8 Heuristics?

```
6F0695EA mov    esi, 0FDE9h
'''
6F069641 call   ?FlsUTF8Url@@YIHPBD@Z ;
FlsUTF8Url(char const *)
6F069646 test   eax, eax
if(!eax)
{
    6F0695C3 xor    edi, edi
    6F06964A mov    [ebp-124h], edi
}else
{
    6F069650 cmp    [ebp-124h], esi
}
'''
6F0696C9 mov    eax, [ebp-124h]
6F0696D5 sub    eax, esi
6F0696DE neg    eax
6F0696E0 sbb    eax, eax
6F0696E2 and    eax, 8
```



# JRE Font Manager Buffer Overflow(Sun Alert 254571)

T:\mat\Projects\ResearchTools\Binary\StaticAnalysis\DarunGrim2\src\Bin\fontmanager.dll-6.0.10.6-6.0.120.4.dgf

File Graphs Help

sub\_6D2C4A60

```

push esi
call sub_6D2C4922

6D2C4A74
push edi
mov edi, [esp+10h]
lea eax, [edi+0Ah]
cmp eax, 2000000h
jnb short loc_6D2C4A8D

6D2C4A83
push eax; size_t
call ds:malloc
pop ecx
jmp short loc_6D2C4A8F

6D2C4A8D
xor eax, eax

6D2C4A8F
test eax, eax
jnz short loc_6D2C4A9A

6D2C4A9A
mov dword ptr [eax], 0AA53C5AAh
mov [eax+4], edi
lea ecx, [eax+edi]
mov byte ptr [ecx+8], 5Ah
mov byte ptr [ecx+9], 0F0h

```

sub\_6D244AF2

```

cmp edi, eax
jnb short loc_6D244B2B

6D244B14
lea ecx, [edi+0Ah]
cmp ecx, eax
jnb short loc_6D244B25

6D244B25
xor eax, eax

6D244B27
push ecx; size_t
call ds:malloc
pop ecx
jmp short loc_6D244B27

6D244B27
test eax, eax
jnz short loc_6D244B32

6D244B32
mov dword ptr [eax], 0AA53C5AAh
mov [eax+4], edi
lea ecx, [eax+edi]
mov byte ptr [ecx+8], 5Ah
mov byte ptr [ecx+9], 0F0h
mov edi, [esi+8]
cmp [esi+4], edi

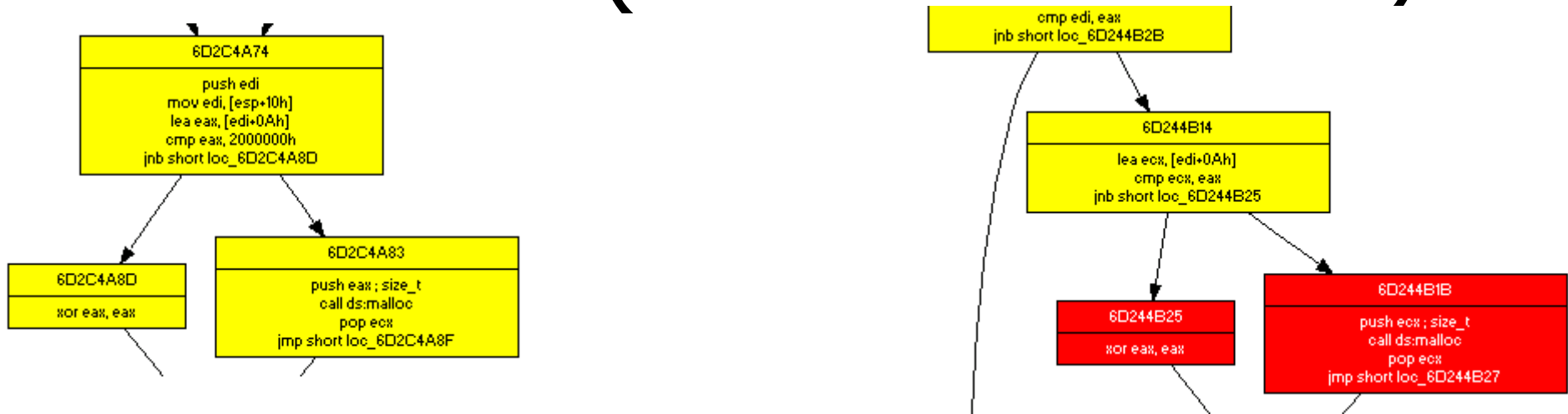
6D244B2B
push 2716h
jmp short loc_6D244B7A

```

List Of Matches

Original	Patched	Match R...	Type	Fingerprint(Original)	Fingerprint(Patched)	Parent(Original)	Parent(Patched)
<input type="checkbox"/>	6D244B27				cc201020102		
<input type="checkbox"/>	6D244B1B				cc8f0102cc120202cc860...		
<input type="checkbox"/>	6D244B25				cc801020102		
<input type="checkbox"/>	6D2C4AD9	100%	Tree	cc1b01020102	cc1b01020102	6D2C4AD3	6D244B6B
<input type="checkbox"/>	6D2C4ACC	100%	Tree	cc2c0102cc1b01020102	cc2c0102cc1b01020102	6D2C4AC6	6D244B5E
<input type="checkbox"/>	6D2C4AD3	100%	Fingerprint	cc7a03020102cc2c0402	cc7a03020102cc2c0402	0	0
<input type="checkbox"/>	6D2C4ADD	100%	Tree	cc8f0502	cc8f0502	6D2C4ABD	6D244B55
<input type="checkbox"/>	6D2C4AC6	100%	Tree	cc1b03020502	cc1b03020502	6D2C4ABD	6D244B55
<input type="checkbox"/>	6D2C4AE8	100%	Fingerprint	cc860102cc0601020502c...	cc860102cc0601020502c...	0	0
<input type="checkbox"/>	6D2C4ABD	100%	Fingerprint	cc7a01020402ccd801020...	cc7a01020402ccd801020...	0	0
<input type="checkbox"/>	6D2C4AB6	100%	Tree	cc8f0502	cc8f0502	6D2C4A9A	6D244B32
<input type="checkbox"/>	6D2C4AE2	100%	Tree	cc8f0102cc100702	cc8f0102cc100702	6D2C4AB6	6D244B4E
<input type="checkbox"/>	6D2C4A9A	100%	Fingerprint	cc7a02020502cc7a0402...	cc7a02020502cc7a0402...	0	0

# JRE Font Manager Buffer Overflow(Sun Alert 254571)



## Original

```
.text:6D2C4A75      mov     edi, [esp+10h]
.text:6D2C4A79      lea    eax, [edi+0Ah]
.text:6D2C4A7C      cmp    eax, 2000000h
.text:6D2C4A81      jnb   short loc_6D2C4A8D
.text:6D2C4A83      push  eax          ; size_t
.text:6D2C4A84      call  ds:malloc
```

## Patched

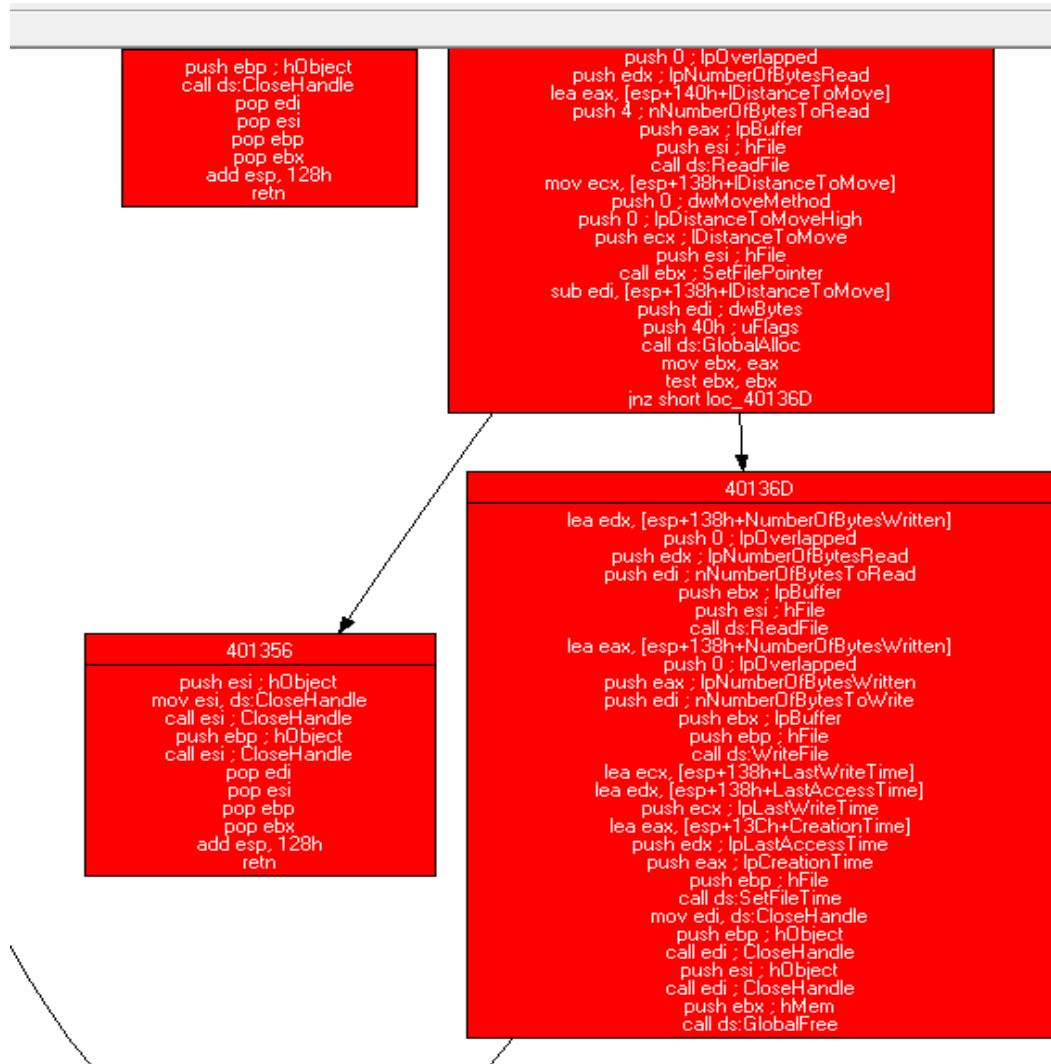
```
.text:6D244B06      push  edi
.text:6D244B07      mov    edi, [esp+10h]
.text:6D244B0B      mov    eax, 2000000h
.text:6D244B10      cmp    edi, eax
.text:6D244B12      jnb   short loc_6D244B2B
.text:6D244B14      lea    ecx, [edi+0Ah]
.text:6D244B17      cmp    ecx, eax
.text:6D244B19      jnb   short loc_6D244B25
.text:6D244B1B      push  ecx          ; size_t
.text:6D244B1C      call  ds:malloc
```

# Malwares: 4<sup>th</sup> of July DDOS Attack

- On this 4<sup>th</sup> of July a DDOS attack was fired against some of US government and corporate sites.
  - It had very limited impact against the targets
- For some reason they changed their targets to South Korean government and major news sites.
  - This time it made a huge success and the targets were almost unreachable during the attack period(3 days).
  - During the time few variants of malware samples were collected.



# Malwares: 4<sup>th</sup> of July DDOS Attack



- This is the routines that saves new attack targets.
- From the binary this part was the only modification.
- It can save a lot of time for the malware analysts.

# Anti-Binary Diffing

- Symbol Mangling
- Reordering and replacing instructions
- CFG Altering
  - Call that never returns
  - Sharing Basic Blocks
  - Use multiple heads for a function
- CG Altering
  - Use proxy call

# Anti Binary Diffing Tool: Hondon

- Hondon= 혼돈 = 混沌 = Chaos
  - A state that can't be divided and defined.
- Don't do extensive code obfuscation that can affect performance
  - Just make the codes not disassemble-able easily.
  - Disassemblableness is not a mandatory feature for a legitimate binary.
  - Usually make IDA's the function recognition fail

# Anti Binary Diffing Tool: Hondon

- Implements CFG altering
  - Minor CFG altering breaks IDA
- Tested under 5.0 and 5.5.
  - 5.0 is broken severely
  - 5.5 is much better, but is still very confused with function recognition
- Hondon works as IDA plugin
  - In real world it should be implemented as a part of compiler(like Visual C++ or gcc).
  - Use binary rewriting to generate obfuscated binary



# Hondon: Demo

Check if how IDA can be confused.

# Conclusion

- The 1-day exploit threat is real
  - Someone finds vulnerabilities fixed silently
  - Bugs tend to aggregate and many times around where bugs were found
  - Some fixes are incomplete and someone can find those facts and can exploit the conditions
- "Hondon" attacks binary-differs weak points
  - Dependency on disassemblers for CFG and CG

# DarunGrim2 and Hondon

<http://www.darungrim.org>

- All the source code and latest binaries will be uploaded within 2 weeks

Questions?