

# Patch Points for Dynamic Patching

## No Fear of Self Modifying Code

Christian Dietrich

March 24, 2011

# Goals

- ▶ **Minimize** overhead for conditional disabling code blocks

# Goals

- ▶ **Minimize** overhead for conditional disabling code blocks
- ▶ Integrate within the C language nicely

# Goals

- ▶ **Minimize** overhead for conditional disabling code blocks
- ▶ Integrate within the C language nicely
- ▶ Robust implementation for X86

# Goals

- ▶ **Minimize** overhead for conditional disabling code blocks
- ▶ Integrate within the C language nicely
- ▶ Robust implementation for X86 - hopefully

# Goals

- ▶ **Minimize** overhead for conditional disabling code blocks
- ▶ Integrate within the C language nicely
- ▶ Robust implementation for X86 - hopefully
- ▶ Solution is searching a problem! Just a PoC.

## Conversion by Example - Original Code

```
patch_point_list ppl;  
  
void foo(void) {  
    patch_point(&ppl, "debug") {  
        printf("Debugging_is_enabled\n");  
    }  
}
```

## Conversion by Example - Original Code

```
patch_point_list ppl;  
  
void foo(void) {  
    patch_point(&ppl, "debug") {  
        printf("Debugging_is_enabled\n");  
    }  
}  
  
// Disable all "debug" patch-points  
patch_point_disable(&ppl, "debug");
```



## Easy solution with compare?

```
patch_point_list ppl;  
  
void foo(void) {  
  
    if (is_enabled(&ppl, "debug")) {  
        printf("Debugging_is_enabled\n");  
    }  
}  
  
// Disable all "debug" patch-points  
patch_point_disable(&ppl, "debug");
```

## Easy solution with compare?

```
patch_point_list ppl;

void foo(void) {
    // Compare and conditional jump for every
    // call of foo
    if (is_enabled(&ppl, "debug")) {
        printf("Debugging_is_enabled\n");
    }
}

// Disable all "debug" patch-points
patch_point_disable(&ppl, "debug");
```

## Conversion by example - Macro expansion

```
#define patch_point(ppl, name) \  
    if(__patch_point(ppl, name) == 23)  
  
patch_point_list ppl;  
  
void foo(void) {  
    if (__patch_point(&ppl, "debug") == 23) {  
        printf("Debugging_is_enabled\n");  
    }  
}
```

## Conversion by example - Compiled and linked code

```
if (__patch_point(&ppl, "debug") == 23)
```



```
e8 fb 02 00 00  call    <__patch_point>  
83 f8 17        cmp     $0x17,%eax  
75 10           jne     <_end_of_block>
```

## What we have

```
e8 fb 02 00 00  call    <__patch_point>
83 f8 17        cmp     $0x17,%eax
75 10           jne     <_end_of_block>
```

- ▶ In `--patch_point` we have the address of the **call**

# What we have

```
e8 fb 02 00 00  call    <__patch_point>
83 f8 17        cmp     $0x17,%eax
75 10           jne     <_end_of_block>
```

- ▶ In **\_\_patch\_point** we have the address of the **call**
- ▶ Search **cmp** and **jne** to get end of block

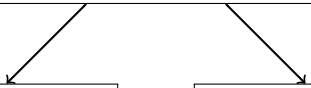
# What we have

```
e8 fb 02 00 00  call    <__patch_point>
83 f8 17        cmp     $0x17,%eax
75 10           jne     <_end_of_block>
```

- ▶ In **\_\_patch\_point** we have the address of the **call**
- ▶ Search **cmp** and **jne** to get end of block
- ▶ Put position and jump offset in **patch\_point\_list**

## Conversion by example - enable/disable

```
e8 fb 02 00 00  call    <__patch_point>  
83 f8 17        cmp      0x17,%eax  
75 10           jne     <_end_of_block>
```



```
nop nop nop nop nop nop  
nop nop nop  
nop nop
```

enabled

```
// use 4 byte jmp  
jmp <_end_of_block>  
jmp <_end_of_block>
```

disabled



# Problems and solutions

- ▶ Block not directly after compare

# Problems and solutions

- ▶ Block not directly after compare - solved, inverse enabling

# Problems and solutions

- ▶ Block not directly after compare - solved, inverse enabling
- ▶ Different jumps

# Problems and solutions

- ▶ Block not directly after compare - solved, inverse enabling
- ▶ Different jumps - solved, decoding 1/4 byte jumps

# Problems and solutions

- ▶ Block not directly after compare - solved, inverse enabling
- ▶ Different jumps - solved, decoding 1/4 byte jumps
- ▶ Instructions between **call** and **cmp**

# Problems and solutions

- ▶ Block not directly after compare - solved, inverse enabling
- ▶ Different jumps - solved, decoding 1/4 byte jumps
- ▶ Instructions between **call** and **cmp** - solved

# Problems and solutions

- ▶ Block not directly after compare - solved, inverse enabling
- ▶ Different jumps - solved, decoding 1/4 byte jumps
- ▶ Instructions between **call** and **cmp** - solved
- ▶ Alls testcases work with -O0, -O1, -O2, -O3, -Os
- ▶ One object and one header file

# Optimize it - fastcall

Use fastcall (Argument passing over registers) and wipe out **mov**

```
mov     0x8048e40,%edx
mov     0x804a040,%ecx
call   8048ad6 <__patch_point>
cmp    0x17,%eax
jne    80485e9 <foo+0x55>
```



# Optimize it - no nop slide

- Instead of a nop slide, use a jump to the end of the nop slide.

```
jmp <call_address+10> // 5 bytes  
jmp <call_address+5>  // 5 bytes
```

instead of

```
nop nop nop nop nop  
nop nop nop nop nop
```

- We get an maximal overhead of 3 cycles + memory access time.

Solution is searching Problem