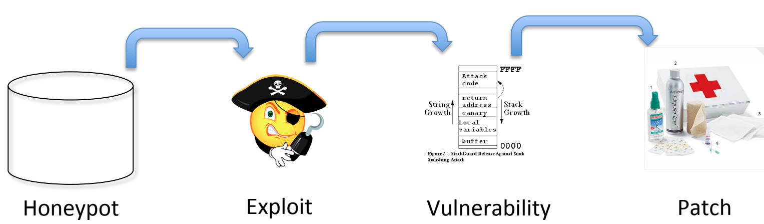
GEORGIA TECH INFORMATION SECURITY CENTER
Safeguarding Digital Information Through Innovative Research and Education

Diagnosis and Emergency Patch Generation for Integer Overflow Exploits

Tielei Wang, Chengyu Song, **Wenke Lee**
Georgia Institute of Technology

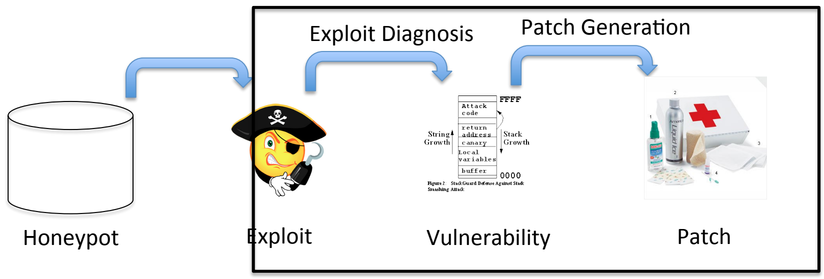
Motivation

- A common task for defenders is to analyze the captured exploits and generate patches, e.g.,



Scope of Work

- Integer Overflow Vulnerabilities



Challenge of Diagnosis

- Benign integer overflows are very common in binaries

Observe a stack overflow = Find a stack overflow vulnerability

Observe an integer overflow ≠ Find an integer overflow vulnerability

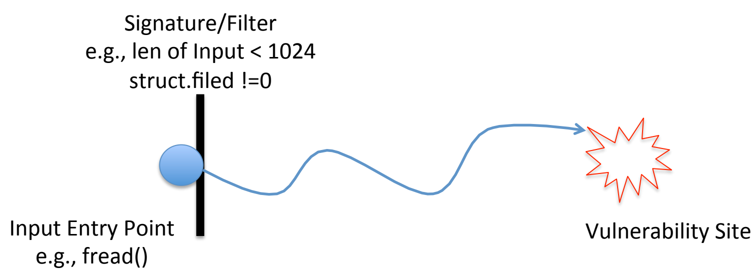
Challenge of Diagnosis

- Benign integer overflows are very common in binaries
  - Random number generation, hash function, compiler optimization

```
int rand(void) {
   next = next * 1103515245 + 12345;
   return (unsigned int)(next/65536) % 32768;
}
```

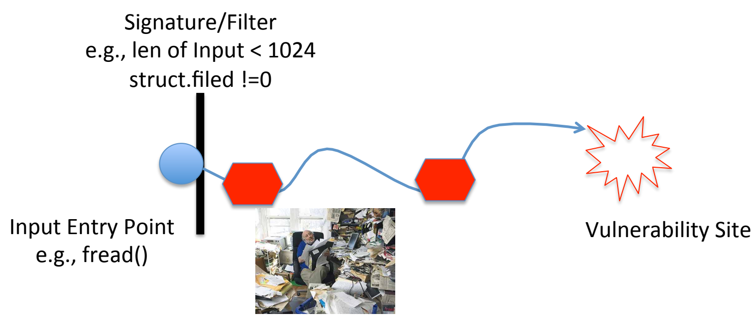
A simple random number generation function deliberately makes use of integer overflows

Challenge of Patch Generation



Challenge of Patch Generation

- Highly complicated input formats make it very hard to identify malicious inputs at the entry point



Our Goals

- Exploit Diagnosis
  - Given an exploit instance, diagnose whether it is against integer overflow vulnerabilities
- Patch Generation
  - Given the integer overflow vulnerability, generate patches to temporarily protect the vulnerable program

Exploit Diagnosis

- Intuition:
  - use dynamic taint analysis to track the propagation of an overflown value
  - identify harmful integer overflows according to how the program uses them

Exploit Diagnosis

```
.text:00672911                 mov     edi, [esi+18h]
.text:00672914                 imul    edi, [esi+14h]  ; integer overflow point
```

Harmful or benign?

CVE-2012-2026 (Adobe Flash Professional CS)

Exploit Diagnosis

```
.text:00672911    mov     edi, [esi+18h]
.text:00672914    imul    edi, [esi+14h]   ; integer overflow point
.text:00672918    push    ebp
.text:00672919    push    ebx              ; hdc
.text:0067291A    call    ds:CreateCompatibleDC
.text:00672920    push    ebx              ; offset
.text:00672921    mov     ebp, eax
.text:00672923    push    ebx              ; hSection
.text:00672924    lea     eax, [esi+4]
.text:00672927    push    eax              ; ppvBits
.text:00672928    mov     [esp+20h+var_4], eax
.text:0067292C    mov     eax, [esi+38h]
.text:0067292F    push    ebx              ; usage
.text:00672930    push    eax              ; lpbmi
.text:00672931    push    ebp              ; hdc
.text:00672932    call    ds:CreateDIBSection
.text:00672938    push    ebp              ; hdc
.text:00672939    mov     [esi+34h], eax
.text:0067293C    call    ds:DeleteDC
.text:00672942    pop     ebp
.text:00672943    cmp     [esi+34h], ebx
.text:00672946    jnz     short loc_672956
.text:00672948    push    edi              ; dwBytes
.text:00672949    push    ebx              ; uFlags
.text:0067294A    call    ds:GlobalAlloc
```

confirm the overflow is harmful

CVE-2012-2026 (Adobe Flash Professional CS)

Another example

```
text:101AD49F                 imul    edi, edx          ; integer overflow point
text:101AD4A2                 cmp     [ebp+arg_4], 8
text:101AD4A7                 ja      short loc_101AD4B6
text:101AD4A9                 xor     eax, eax
text:101AD4AB                 inc     eax
text:101AD4AC                 shl     eax, cl
text:101AD4AE                 movzx   eax, ax
text:101AD4B1                 mov     [ebp+arg_0], eax
text:101AD4B4                 jmp     short loc_101AD4BA
text:101AD4B6 ; ---------------------------------------------------------------
text:101AD4B6
text:101AD4B6 loc_101AD4B6:                             ; CODE XREF: bmp_alloc+3C↑j
text:101AD4B6                 and     [ebp+arg_0], 0
text:101AD4BA
text:101AD4BA loc_101AD4BA:                             ; CODE XREF: bmp_alloc+49↑j
text:101AD4BA                 movzx   eax, word ptr [ebp+arg_0]
text:101AD4BE                 lea     eax, [edi+eax*4+28h]
text:101AD4C2                 push    eax               ; dwBytes
text:101AD4C3                 push    42h               ; uFlags
text:101AD4C5                 call    ds:GlobalAlloc
```

CVE-2012-1149 (OpenOffice.Org)

Patch Generation

- Intuition:
  - Remediate vulnerabilities by reusing existing error handlers inside the program, rather than filtering the malicious inputs at the entry point

Local Error Virtualization

Example For Patch Generation

```
HGLOBAL WinSalBitmap::ImplCreateDIB( const Size& rSize, USHORT nBits, const BitmapPalette& rPal )
{
...

        const ULONG nImageSize = AlignedWidth4Bytes( nBits * rSize.Width() ) * rSize.Height();
        const USHORT nColors = ( nBits <= 8 ) ? ( 1 << nBits ) : 0;

        hDIB = GlobalAlloc( GHND, sizeof( BITMAPINFOHEADER ) + nColors * sizeof( RGBQUAD ) + nImageSize );
..
```

CVE-2012-1149 (OpenOffice.Org)

Example For Patch Generation

- There is an existing "validation" check in the program

```
HGLOBAL WinSalBitmap::ImplCreateDIB( const Size& rSize, USHORT nBits, const BitmapPalette& rPal )
{
...
     HGLOBAL hDIB = 0;
     if ( rSize.Width() && rSize.Height() )
     {
       const ULONG nImageSize = AlignedWidth4Bytes( nBits * rSize.Width() ) * rSize.Height();
       const USHORT nColors = ( nBits <= 8 ) ? ( 1 << nBits ) : 0;

       hDIB = GlobalAlloc( GHND, sizeof( BITMAPINFOHEADER ) + nColors * sizeof( RGBQUAD ) + nImageSize );
     }
     return hDIB;
```

CVE-2012-1149 (OpenOffice.Org)

Example For Patch Generation

- Deploy patches at existing check points, and reuse the existing error handling code to survive the attack

```
HGLOBAL WinSalBitmap::ImplCreateDIB( const Size& rSize, USHORT nBits, const BitmapPalette& rPal )
{
...
     HGLOBAL hDIB = 0;
     if ( rSize.Width() && rSize.Height() && CurrentExecutionContextWillNotTriggerTheIntegerOverflow())
     {
       const ULONG nImageSize = AlignedWidth4Bytes( nBits * rSize.Width() ) * rSize.Height();
       const USHORT nColors = ( nBits <= 8 ) ? ( 1 << nBits ) : 0;

       hDIB = GlobalAlloc( GHND, sizeof( BITMAPINFOHEADER ) + nColors * sizeof( RGBQUAD ) + nImageSize );
     }
     return hDIB;
```

CVE-2012-1149 (OpenOffice.Org)

Pattern I
Existing "Validations" Before Integer Overflows

Existing "Validation"

```
if (x< 0)
{
   return -1;
}

....

size = x * y* 4 + 4;
```

Integer Overflow Operation

Real World Integer Overflows

```
if (Bitmap_Head.biWidth < 0)
{
     g_set_error (error, G_FILE_ERROR,G_FILE_ERROR_FAILED,
                  _("'%s' is not a valid BMP file"),
                  gimp_filename_to_utf8 (filename));
     return -1;
}

....

rowbytes= ((Bitmap_Head.biWidth * Bitmap_Head.biBitCnt - 1) / 32) * 4 + 4;

....

buffer    = g_malloc (rowbytes);
```

CVE-2009-1570 (GIMP)

Real World Integer Overflows

```
if( p_sys->i_track_id < 0 )
{
    input_item_node_AppendNode( p_input_node, p_new_node );
    vlc_gc_decref( p_new_input );
    return true;
}
...

input_item_t **pp;

pp = realloc( p_sys->pp_tracklist, (p_sys->i_track_id + 1) * sizeof(*pp) );
```

CVE-2011-2194 (VLC Player)

Real World Integer Overflows

```
if (__builtin_expect (n == (size_t) -1, 0))
    /* Something wrong.
       XXX Do we have to set `errno' to something which mbsrtows hasn't
       already done?  */
    return -1;

……
wpattern_malloc = wpattern = (wchar_t *) malloc ((n + 1) * sizeof (wchar_t));
```

CVE-2011-1071 (Glibc)

Real World Integer Overflows

```
if (args->buffer_count < 1) {
    DRM_ERROR("execbuf2 with %d buffers\n", args->buffer_count);
    return -EINVAL;
}

exec2_list = kmalloc(sizeof(*exec2_list)*args->buffer_count,
                     GFP_KERNEL | __GFP_NOWARN | __GFP_NORETRY);
```

CVE-2013-0913 (Linux Kernel)

Page 21

Remediation Policy I

- Enhance existing checks before integer overflows

```
if (x< 0)
{
   return -1;
}

....

size = x * y* 4 + 4;
```

Vulnerable Code

```
if (x< 0 || IntegerOverflowToHappen())
{
   return -1;
}

....

size = x* y* 4 + 4;
```

Remediated Code

Page 22

Remediation Policy I

- Utilize backwards-slicing to identify existing checks before the integer overflows
- Use symbolic execution to compute the weakest preconditions
  - Check whether a concrete execution context satisfies the vulnerability trigger condition
  - If so, force the program to run the error handler

Pattern II
Existing "Validations" After Integer Overflows

```
size = x * y* 4 + 4;

if (size < 0)
 {
    return -1;
 }
 ....
```

Integer Overflow Operation

Existing "Validation"

Real World Integer Overflows

```
space = dp->tdir_count * datawidth[dp->tdir_type];

if (space <= 0) {
    printf(">\n");
    Error("Invalid count for tag %u", dp->tdir_tag);
```

CVE-2010-4665 (Libtiff)

Real World Integer Overflows

```
bitmap_size = glyph->bpr * glyph->bbx.height;

if ( bitmap_size > 0xFFFFU )
{
    FT_ERROR(( "_bdf_parse_glyphs: " ERRMSG4, lineno ));
    error = BDF_Err_Bbx_Too_Big;
    goto Exit;
}
```

CVE-2012-1144 (Freetype2)

Remediation Policy II

- Reuse existing checks after integer overflow

```
size = x * y* 4 + 4;


if (size < 0)
 {
    return -1;
 }

 ....
```

Vulnerable Code

```
size = x * y* 4 + 4;
SetAlarmIfOverflowed();

if (size < 0 || AlarmIfTrue())
 {
    return -1;
 }

 ....
```

Remediated Code

Remediation Policy II

- Utilize forwards-slicing to identify existing checks after the integer overflows
- Monitor the integer overflow operation, and set an alarm variable if it really overflows at runtime
- Check the alarm variable at the validation check point. If the alarm is True, alter the program control flow to error handler

Pattern III
Existing “Validations” On Allocation Results

```
size = x * y * 4 + 4;

p = malloc(size);

if(p==NULL)
    goto error;
```

Integer Overflow Operation

Existing “Validation”

Real World Integer Overflows

```
PRUint32 destlen = ((srclen + 2)/3) * 4;
dest = (char *)PR_MALLOC(destlen + 1);
if( (char *)0 == dest )
{
    return (char *)0;
```

CVE-2009-2463 (Firefox)

Real World Integer Overflows

```
length=(size_t) matte_image->bytes_per_line*matte_image->height*matte_image-
>depth;

matte_image->data=(char *) malloc(length);

if (matte_image->data == (char *) NULL)
```

CVE-2009-1882 (ImageMagic)

Real World Integer Overflows

```
skb = sock_wmalloc(sk, NET_SKB_PAD + sizeof(struct iphdr) +
                   uhlen + session->hdr_len +
                   sizeof(ppph) + total_len,
                          0, GFP_KERNEL);
if (!skb)
    goto error_put_sess_tun;
```

CVE-2010-4160 (Linux Kernel)

Real World Integer Overflows

```
coef = (MYFLT *)malloc((hdr.npoles+hdr.nvals)*sizeof(MYFLT));

if (coef==NULL) {
    printf("memory allocation failure\n");
    exit(1);
```

CVE-2012-2106 (CSound)

Page 33

Remediation Policy III

- Reuse the memory allocation failure error code

Vulnerable Code

```
size = x * y * 4 + 4;




p = malloc(size);

if(p==NULL)
    goto error;
```

Remediated Code

```
size = x * y * 4 + 4;
SetAlarmIfOverflowed();

if(AlarmIsTure())
    p = NULL
else
    p = malloc(size);

if(p==NULL)
    goto error;
```

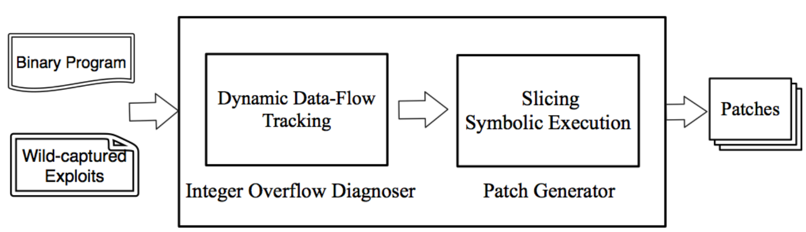
Page 34

Remediation Policy III

- Utilize forwards-slicing to determine whether the integer overflow will affect memory allocations
- Monitor the integer overflow operation, and set an alarm variable if it really overflows at runtime
- Check the alarm variable before memory allocation. If the alarm is True, force the program to run the branch for memory allocation failure

Page 35

System Overview



Evaluation

Software	Description	Version	Open Source	CVE ID	Input
Openoffice.org	Office productivity software suite	3.3.20	Y	CVE-2012-1149	ODT
VLC	Multimedia player	1.1.0	Y	CVE-2011-2194	XSPF
Yahoo Messenger	Instant messaging	11.5.0.152	N	CVE-2012-0268	JPEG
ACDSee	Image viewer	14.1	N	CVE-2012-1197	BMP
Opera	Web browser	11.6	N	CVE-2012-1003	HTML
Adobe Flash Player	Web browser plug-in	10.0.42.34	N	CVE-2010-2170	SWF
Adobe Reader	PDF viewer	9.1.3	N	CVE-2009-3459	PDF
RealPlayer	Multimedia player	SP 1.1	N	CVE-2010-3000	FLV
QuickTime Player	Multimedia Player	7.1.3	N	CVE-2007-0714	MPEG-4
Microsoft Linker	Key component of Microsoft Visual Studio	10.00.30319.01	N	N/A	PE
Summary: 10 integer overflows and 10 different input formats.					

Use 10 programs on Windows to evaluate our system.
We obtained the exploit samples from different sources.

Attack Diagnosis Results

Software	Integer Overflow Vulnerability	Module	Offset	# Overflow Sites
Openoffice.org	`imul edi, edx`	vclmi.dll	0x1ad49f	1122
VLC	`lea esi, ptr [ecx*4+0x4]`	libplaylist_plugin.dll	0xfcd9	423
Yahoo Messenger	`imul eax, ebx`	YImage.dll	0x21531	354
ACDSee	`imul ebp, ecx`	IDE_ACDStd.apl	0x59639	288
Opera	`imul eax, dword ptr [esp+0xc]`	opera.dll	0x889f5b	428
Adobe Flash Player	`imul eax, ecx`	Flash10d.ocx	0x9165e	860
Adobe Reader	`lea edx, ptr [ecx*4+0x48]`	AcroRd32.dll	0xa60a5	1082
RealPlayer	`imul ecx, ecx, 0x23`	flvff.dll	0x8bc4	381
QuickTime Player	`add ecx, edi`	QuickTime.qts	0x295a74	567
Microsoft Linker	`lea edi, ptr [eax+eax*8]`	linker.exe	0xa2c10	88

Accurately locate the harmful integer overflows

Patch Generation Results

Table 4: Policy I Patch Evaluation Results

Software	Relevant Checks	Validation Checks	# Final Patches
Openoffice.org	17	9	8
Yahoo Messenger	14	4	4
ACDSee	10	10	3
Opera	1	1	1
VLC	2	1	1
Adobe Reader	23	8	8
Microsoft Linker	1	1	1
Summary: successfully fixed these 7 vulnerabilities by using Policy I.			

Table 5: Policy II and III Patches

Policy Type	Software	Fixed
II	Adobe Flash Player	Y
	Quicktime	Y
III	RealPlayer	Y

Successfully generated patches for the ten integer overflows using different policies

Performance

Table 6: System Performance Results

Software	Diagnosis(s)	Tracing(s)	Slicing(s)	Patching(s)
Yahoo Messenger	57	164	16	6.3
OpenOffice.org	181	210	53	10.2
ACDSee	123	206	18	8.8
Opera	105	332	49	6.5
VLC	112	134	28	8.6
Adobe Reader	99	361	71	21.5
Adobe Flash Player	144	344	52	N/A
QuickTime	78	217	73	N/A
RealPlayer	93	228	31	N/A
Microsoft Linker	37	66	27	12.1
Summary: diagnosing and patching were completed in minutes.				

Table 7: Patch Overhead

Software	Normal (μs)	Malicious (μs)
Yahoo Messenger	3190	4503
OpenOffice.org	5028	6572
ACDSee	1241	2442
Opera	727	761
Adobe Reader	597	1524
VLC	306	509
MS linker	1660	1819

Generate patches in minutes. And Patches introduce trivial performance overhead

Conclusion

- Propose a taint-based approach to pinpoint integer overflow vulnerabilities from an exploit instance
- Propose local error virtualization to generate emergency patches for integer overflow vulnerabilities
- Evaluation with 10 real world programs shows the effectiveness and the efficiency of our prototype system

Limitations

- Completeness and Soundness
  - Detection
  - Patch
- It is a heuristic-based system, and it's purpose is to provide *a temporary protection when there is no official patch available*