

# « La technique de l'attaque Shatter par l'exemple »

This is a personal translation of : / Traduction personnelle du document :

« Shattering By Example »

[http://www.security-assessment.com/Papers/Shattering\\_By\\_Example-V1\\_03102003.pdf](http://www.security-assessment.com/Papers/Shattering_By_Example-V1_03102003.pdf)

Brett MOORE, Network Intrusion Specialist, [Security-Assessment.com](http://www.security-assessment.com)

Par Jérôme ATHIAS

[Jerome@ATHIAS.fr](mailto:Jerome@ATHIAS.fr)

<http://www.athias.fr/jerome/DOC/>

Done and published with the agreement of [Security-Assessment.com](http://www.security-assessment.com) – All rights reserved/

Réalisée et publiée avec l'accord de [Security-Assessment.com](http://www.security-assessment.com) – Tous droits réservés

Dernière mise à jour : 23/06/2004

## **Introduction:**

« L'attaque par Shatter » est un terme employé pour définir des attaques de l'environnement GUI de Windows qui permettent à un utilisateur d'injecter et de faire exécuter du code à un processus ne lui appartenant pas (lancé par un autre utilisateur, avec les droits de ce même utilisateur tiers) en utilisant les messages Windows.

Ce document (sous sa forme originale) contient des exemples de codes source écrits en langage C et n'a pas pour but de décrire précisément les bases de ces attaques. Il vous est recommandé la lecture des documents suivants pour bien comprendre le mécanisme de ces attaques :

- Shatter Attacks – How to break Windows – Chris PAGET  
<http://security.tombom.co.uk/shatter.html>

*Traduction française par Damien :*

<http://www.katabatik.com/ktk/sections.php?op=viewarticle&artid=22>

- Win32 Message Vulnerabilities Redux – Olivier LAVERY  
[http://www.idefense.com/idpapers/Shatter\\_Redux.pdf](http://www.idefense.com/idpapers/Shatter_Redux.pdf)

Document original:

[http://www.security-assessment.com/Papers/Shattering\\_By\\_Example-V1\\_03102003.pdf](http://www.security-assessment.com/Papers/Shattering_By_Example-V1_03102003.pdf)

## **Résumé:**

Les attaques par shatter précédentes se basaient sur l'utilisation de messages qui acceptent un pointeur comme paramètre. Ce pointeur dirige vers un flot de données passé par l'attaquant, permettant à ce même attaquant de faire exécuter le code de son choix au processus.

Plusieurs messages Windows acceptent un pointeur sur une fonction callback comme paramètre de l'API SendMessage. L'un d'entre eux est le LVM\_SORTITEMS :

Message	LVM_SORTITEMS
Description	Utilise une fonction de comparaison définie par l'application pour effectuer un tri d'une liste.
Appel	SendMessage( (HWND) hWndControl, //handle du contrôle (UINT) LVM_SORTITEMS, //ID du message wParam = (WPARAM) (LPARAM) lParamSort; lParam = (LPARAM) (PFNLVCOMPARE) pfnCompare;
Paramètres	lParamSort Valeur définie par l'application qui est passée à la fonction de comparaison  pfnCompare Adresse de la fonction de comparaison définie par l'application. La fonction de comparaison est appelée durant le processus de tri à chaque fois que l'ordre doit être réévalué.

La méthode d'attaque décrite dans ce document utilise des messages qui, à première vue, semblent inoffensifs, mais comme nous allons le voir, peuvent être utilisés pour écrire des valeurs arbitraires dans l'espace mémoire d'un processus, conduisant à l'exécution de certaines commandes. Cette technique permet à un utilisateur avec des droits restreints de redéfinir des portions importantes dans l'espace mémoire utilisé par un processus SYSTEM comme des structures de données et des handlers structurés d'exceptions.

**(Rect\*) Overwrite :**  
**Réécriture (Rect\*) :**

Différents messages Windows acceptent un pointeur sur une structure POINT ou RECT utilisés pour retrouver des informations GDI de Windows. Ces pointeurs semblent ne jamais être vérifiés d'aucune manière.

Nous allons nous pencher sur le message HDM\_GETITEMRECT

Message	HDM_GETITEMRECT
Description	?
Appel	SendMessage( (HWND) hWndControl, //handle du contrôle (UINT) HDM_GETITEMRECT, //ID du message (WPARAM) wParam, //(WPARAM) (int) iIndex; (LPARAM) lParam ); //(LPARAM) (RECT*)
Paramètres	wParam Index basé sur 0 du contrôle d'entête (control header) de l'objet pour lequel on souhaite retrouver le rectangle borné lParam Pointeur vers une structure RECT qui va recevoir les informations sur les dimensions/bornes du rectangle

En passant une valeur arbitraire comme paramètre lParam, le processus receveur écrira les données résultats du RECT dans un emplacement mémoire de notre choix.

Par exemple, si nous voulions redéfinir le filtre d'exception (Unhandled Exception Filter) en 0x77EDXXXX nous appellerions :

```
SendMessage(hwnd,HDM_GETITEMRECT,0, 0x77EDXXXX)
```

Pour contrôler ce qui sera écrit à cette adresse, nous allons étudier le format de la structure reçue. Pour le message HDM\_GETITEMRECT, un pointeur sur la structure RECT est employé.

Structure	RECT
Définition	<pre>typedef struct_RECT {     LONG left;     LONG top;     LONG right;     LONG bottom; } RECT, *PRECT;</pre>

La structure RECT est composée de 4 valeurs de type LONG consécutives. Si nous passons l'adresse 0x00024030, l'écriture résultante ressemblera à :

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

A = Gauche, B = Haut, C = Droite, D = Bas  
A = Left, B = Top, C = Right, D = Bottom

En définissant la largeur de la première colonne d'un contrôle Listview, nous prenons le contrôle sur la valeur Gauche (Left) de la deuxième colonne. Nous pouvons alors utiliser le bit le moins significatif de la valeur de retour Gauche pour réécrire l'espace mémoire octet par octet.

Si nous voulions écrire la valeur 0x58, nous devrions donc définir la largeur de la première colonne à 0x58, puis envoyer le message HDM\_GETITEMRECT. L'adresse spécifiée serait alors réécrite comme suit :

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

En réalisant une écriture, puis en incrémentant notre adresse d'écriture, nous avons la possibilité d'écrire une chaîne d'octets donnée dans un emplacement mémoire donné.

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

Cet emplacement pourrait être un programme de lecture/écriture d'un espace de données, ou l'espace d'une valeur globale d'une application comme un TEB/TEP.

Cette méthode peut être utilisée pour écrire un shellcode dans une adresse mémoire en écriture.

Après cela, le flot d'exécution peut être redirigé en écrasant le handler SHE avec l'adresse mémoire, déclenchant une exception.

Nous avons la possibilité d'automatiser la définition de la taille des colonnes du Listview en envoyant le message LVM\_SETCOLUMNWIDTH.

Message	LVM_SETCOLUMNWIDTH
Description	Change la largeur d'une colonne en mode rapport ("report-view") ou la largeur de toutes les colonnes en mode liste (« list-view »)
Appel	SendMessage( (HWND) hWndControl, //handle du contrôle (UINT) LVM_SETCOLUMNWIDTH, //ID du message (WPARAM) wParam, // = (WPARAM) (int) iCol (LPARAM) lParam); // = MAKELPARAM ((int) cx, 0);
Paramètres	WParam Index basé sur 0 d'une colonne valide LPARAM Nouvelle largeur de la colonne en pixels

En passant l'octet que nous désirons écrire comme paramètre lParam pour définir la largeur, lorsque le HDM\_GETITEMRECT est appelé, notre octet sera écrit à notre adresse mémoire spécifiée.

Cette méthode a été démontrée comme fonctionnelle contre les contrôles Tab (Tab controls), tout comme en utilisant la paire de message :

TCM\_SETITEMSIZE  
TCM\_GETITEMRECT

### Exemple de réécriture (Rect\*) contre le contrôle Listview :

#### Shatterseh2.c

Illustre l'utilisation des messages du contrôle Listview pour :

Injecter un shellcode à un emplacement prédéfini  
Réécrire 4 octets d'une adresse mémoire critique

3 variables doivent être définies pour une exécution correcte :

tWindow est le nom de la fenêtre principale du programme cible  
sehHandler représente l'adresse mémoire critique  
shellcodeaddr représente l'emplacement mémoire où injecter le shellcode

L'option de recherche automatique ("autofind") peut ne pas être fonctionnelle contre tous les programmes.

Testez la contre tout les programmes contenant un Listview (Ex: l'explorateur, IE, tout dialogue d'ouverture de fichier)

**CODE SOURCE : Se reporter au document original**

**(PBRange\*) Overwrite :**  
**Réécriture (PBRange\*) :**

Le contrôle “Barre de progression” (progress bar) permet l’utilisation du message PBM\_GETRANGE pour retrouver les bornes minimum et maximum.

Message	PBM_GETRANGE
Description	Fournit les informations sur les limites courantes minimum et maximum d’un contrôle progress bar donné
Appel	SendMessage( (HWND) hWndControl, //handle du contrôle (UINT) LVM_SETCOLUMNWIDTH, //ID du message (WPARAM) wParam, // (WPARAM) (LPARAM) lParam); // (PBRANGE) ppPBRange;
Paramètres	lParam Pointeur sur une structure PBRange qui doit recevoir les valeurs des limites minimum et maximum d’un contrôle progress bar.

Le paramètre lParam de ce message n’est pas contrôlé avant d’être écrit, nous permettant de réécrire des adresses mémoires d’une manière similaire à celle décrite dans le précédent chapitre.

Le message utilisé pour fixer notre octet est PBM\_SETRANGE.

Message	PBM_SETRANGE
Description	Fixe les valeurs minimum et maximum d’une barre de progression et redessine la barre en tenant compte de ces nouvelles valeurs.
Appel	SendMessage( (HWND) hWndControl, //handle du contrôle (UINT) PBM_SETRANGE, //ID du message (WPARAM) wParam, // (WPARAM) (LPARAM) lParam); // MAKELPARAM (nMinRange,MaxRange);
Paramètres	lParam Bornes minimum et maximum de la barre de progression.

Dans certaines circonstances, il est possible d’utiliser cette méthode contre le service d’installation de Windows (Windows installer service), pour élever ses privilèges.

Dans beaucoup de cas, cela utilise les droits SYSTEM et s’exécute en tant qu’utilisateur, mais cela s’exécutera relativement souvent en SYSTEM. Par exemple avec les utilitaires de déploiement de la stratégie de groupe (group policy), ou encore si l’installation avec des privilèges élevés est activée.

Vous pourriez probablement forcer l’affichage d’une barre de progression au niveau système en installant une application publicitaire. Dans le pire des cas, en tentant de réparer un composant préalablement installé par un administrateur.

### **Exemple de réécriture (Pbrange\*) contre des barres de progression :**

Illustre l'utilisation des messages du contrôle de progression (progress Control) pour :

- Injecter un shellcode à un emplacement prédéfini
- Réécrire 4 octets d'une adresse mémoire critique

3 variables doivent être définies pour une exécution correcte :

- tWindow est le nom de la fenêtre principale du programme cible
- sehHandler représente l'adresse mémoire critique
- shellcodeaddr représente l'emplacement mémoire où injecter le shellcode

Un shellcode local lance des adresses appropriées.

Tester la contre n'importe quel programme avec une barre de progression

**CODE SOURCE : Se reporter au document original**

## **Jumelage des messages**

Comme l'illustre les exemples précédents, la méthode d'exploitation tient en l'utilisation d'une paire de messages. Le premier message sert à définir la taille ou une autre valeur à l'octet que nous désirons écrire. Le second message est employé pour récupérer la valeur définie par le précédent message dans une adresse mémoire dans laquelle nous souhaitons écrire.

Cette méthode d'exploitation se confère dans l'utilisation d'un couple de messages respectivement de type T-2 et T-3.

Pour l'explication de ce document nous allons utiliser les termes suivants pour décrire comment les paramètres des messages sont manipulés.

### **T-1**

Les paramètres du message sont passés correctement. Un exemple est : WM\_SETTEXT. Un pointeur est passé vers une valeur chaîne qui est ajustée et passée sans problème par le système de gestion des messages. La chaîne est copiée dans un espace mémoire disponible au processus receveur et le pointeur ajusté en conséquence.

### **T-2**

Les paramètres du message sont passés directement. Un exemple pour cela est : LVM\_SETCOLUMNWIDTH où une valeur longue est passée avec le message. Aucun pointeur n'intervient.

### **T-3**

Les paramètres du message sont passés de manière incorrecte. Un exemple : PBM\_GETTRANGE. Un pointeur sur une structure est passé pour soit définir soit recevoir des données. Ce pointeur est utilisé pour accéder localement à l'espace mémoire du processus, permettant de définir ou retrouver des espaces mémoires arbitraires.



## Attaque Shatter de la barre d'état :

Les chapitres suivants vont se focaliser sur l'utilisation de divers messages pour arriver au même résultat que vu précédemment. Cet exploit est mené à bien contre la barre d'état par l'utilisation des messages suivants :

WM\_SETTEXT  
SB\_SETTEXT  
SB\_GETTEXTLENGTH  
SB\_SETPARTS  
SB\_GETPARTS

Son explication est décomposée en deux parties :

La paire de messages  
Le brute force du heap

### La paire de messages :

La barre d'état accepte de recevoir un message SB\_GETPARTS qui utilise un pointeur sur un tableau d'entiers comme paramètre.

Message	SB_GETPARTS
Description	Renvoie le nombre de parties d'une fenêtre de statut. Le message renvoie également les coordonnées du bord droit du nombre de parties spécifiées.
Appel	SendMessage( (HWND) hWndControl, //handle du contrôle (UINT) SB_GETPARTS, //ID du message (WPARAM) wParam, //=(WPARAM) (int) nParts; (LPARAM) lParam); //=(LPARAM) (LPINT) aRightCoord; );
Paramètres	nParts Nombre de parties pour lesquelles retrouvées les coordonnées. Si ce paramètre est plus grand que le nombre de parties de la fenêtre, le message renvoie uniquement les coordonnées des parties existantes.  aRightCoord Pointeur sur un tableau d'entiers qui a le même nombre d'éléments que de parties spécifiées par nParts. Chaque élément du tableau reçoit les coordonnées clientes du bord droit de la partie correspondante. Si un élément est défini à -1, la position du bord droit de cette partie est étendue au bord droit de la fenêtre. Pour renvoyer le nombre courant de parties, définir ce paramètre à 0.

Suivant la tendance décrite ci-dessus, le paramètre lParam n'est pas validé avant d'être écrit, nous permettant de l'employer pour réécrire des adresses mémoires arbitraires. Ce message est de type T-3.

Le message couplé à ce dernier utilisé pour définir la longueur des parties est décrit comme suit :

Message	SB_SETPARTS
Description	Fixe le nombre de parties dans une fenêtre de statut et les coordonnées du bord droit de chaque partie.
Appel	SendMessage( (HWND) hWndControl, //handle du contrôle (UINT) SB_SETPARTS, //ID du message (WPARAM) wParam, // = (WPARAM) (int) nParts; (LPARAM) lParam); // = (LPARAM) (LPINT) aWidths; );
Paramètres	nParts Nombre de parties à définir (256 maximums)  aWidths Pointeur vers un tableau d'entiers. Le nombre d'éléments est défini par nParts. Chaque élément spécifie la position, dans les coordonnées clientes, du bord droit de la partie correspondante. Si un élément à la valeur -1, le bord droit de la partie correspondante s'étend jusqu'au bord de la fenêtre.

Ce message accepte un pointeur sur un tableau d'entiers pour fixer la taille du nombre de parties spécifiées. Ce message est également de type T-3.

Afin d'exploiter la combinaison des messages SB\_GETPARTS/SB\_SETPARTS, nous devons en premier lieu être en mesure d'écrire un nombre suffisant de données dans l'espace mémoire d'un processus afin de constituer un tableau d'entiers.

Pour nos démonstrations, ce tableau ne nécessite qu'un élément pour que nous puissions définir la largeur de la première colonne, nous pouvons ainsi écrire la valeur de la largeur du bord droit de la première colonne dans notre espace mémoire arbitraire.

### Le brute force du heap :

Retrouver une donnée arbitraire dans l'espace mémoire d'un processus peut s'opérer de différentes manières qui ont été présentées dans les documents précédents traitants de l'attaque shatter. Pour cet exemple, nous emploierons le message WM\_SETTEXT.

Message	WM_SETTEXT
Description	Un programme envoie un message WM_SETTEXT pour définir le texte d'une fenêtre.
Appel	SendMessage( (HWND) hWndControl, //handle du contrôle (UINT) WM_SETTEXT, //ID du message wParam=0; //Non utilisé, doit être nul lParam = (LPARAM) (LPCTSTR)lpsz; //adresse d'une chaîne texte de la fenêtre
Paramètres	lpsz Valeur de lParam. Pointeur sur une chaîne terminée par null qui représente le texte de la fenêtre.

Nous allons utiliser ce message pour définir les données de la barre de titre des applications vulnérables avec la valeur de notre choix. Eventuellement, nous utiliserons ce message pour envoyer les octets que nous désirons écrire, octet par octet, comme la taille de l'entier du tableau attendu par le message SB\_SETPARTS.

Avant de pouvoir utiliser les données avec le SB\_SETPARTS, nous devons connaître l'emplacement où est stocké le bord.

Nous pouvons réaliser un brute force de cet emplacement par l'intermédiaire des messages SB\_SETTEXT et SB\_GETTEXTLENGTH.

Message	SB_SETTEXT
Description	Le message SB_SETTEXT définit le texte d'une partie spécifiée d'une fenêtre d'état.
Appel	SendMessage( (HWND) hWndControl, //handle du contrôle (UINT) SB_SETTEXT, //ID du message (WPARAM) wParam, // = (WPARAM) (UINT) lpart (LPARAM) lParam // = (LPARAM) (LPSTR) szText );
Paramètres	lPart Index basé sur 0 de la partie à définir. Si ce paramètre est défini avec la valeur SB_SIMPLEID, la fenêtre d'état est traitée comme étant une fenêtre simple avec une seule partie.  szText Pointeur sur une chaîne terminée par null qui représente le texte à définir.

Le texte de la barre de titre est stocké en Unicode, ainsi si l'on envoie le message WM\_SETTEXT avec une grande quantité de X cela apparaîtra dans la mémoire du processus receveur comme ceci :

0002409B	5800 5800 5800 5800	X.X.X.X.
000240A0	5800 5800 5800 5800	X.X.X.X.
000240AB	5800 5800 5800 5800	X.X.X.X.

Si nous envoyons plusieurs messages SB\_SETTEXT en spécifiant l'emplacement de notre capture du heap (« heap guess ») comme le paramètre szText, le texte de la première partie sera défini à la valeur X quand nous avons récupéré l'adresse mémoire correcte du heap.

Nous ne pouvons pas utiliser le message SB\_GETTEXT pour vérifier le texte de la première partie, car il s'agit également d'un message de type T-3. Nous pouvons par contre employer le message SB\_GETTEXTLENGTH, qui quant à lui est de type T-2.

Message	SB_GETTEXTLENGTH
Description	Le message SB_GETTEXTLENGTH renvoie la longueur, en caractères du texte d'une partie donnée d'une fenêtre d'état. (status window)
Appel	SendMessage( (HWND) hWndControl, //handle du contrôle (UINT) SB_GETTEXTLENGTH, //ID du message (WPARAM) wParam, // (WPARAM) (INT) iPart; (LPARAM) lParam // = 0; non utilisé, doit être à 0);
Paramètres	iPart Index basé sur 0 de la partie dont on souhaite retrouver le texte. lParam Doit être à 0.

Ce message renvoie la longueur du texte de la partie spécifiée. Ainsi, lorsque nous aurons trouvé l'adresse heap correcte et que la première partie aura été définie à X, ce message renverra 1.

Ce n'est pas encore suffisant car plusieurs adresses mémoires définiront le texte de la première partie avec une chaîne d'un caractère de long. Ainsi, après avoir trouvé une adresse qui retourne 1 à ce message, nous retournons dans la procédure, définissant la barre de titre avec une valeur de 0x80. Ce qui sera convertit en \xAC\x20 en Unicode, puis si nous avons l'adresse correcte, le prochain appel à SB\_GETTEXTLENGTH retournera une valeur supérieure à 1. Si nous n'avons pas l'adresse correcte, cela retournera 1 à nouveau.

**Statusbar Overwrite Example :**  
**Exemple de réécriture de la barre d'état :**

Illustre l'utilisation de plusieurs messages Windows pour :  
Brute forcer une adresse heap utilisable  
Placer les informations d'une structure dans un processus  
Injecter un shellcode à un emplacement prédéfini  
Réécrire 4 octets d'une adresse mémoire critique

4 Variables doivent être définies pour une exécution correcte :  
tWindow est le nom de la fenêtre principale du programme cible  
sehHandler représente l'adresse mémoire critique  
shellcodeaddr représente l'emplacement mémoire où injecter le shellcode  
heapaddr représente l'adresse heap de base pour commencer le brute force

Le shellcode local est codé en dur pour Windows 2000 SP4 ENG à cause des implications UNICODE.

Testez le contre tous les programmes avec une barre d'état.

**CODE SOURCE : Se reporter au document original**

## **Conclusions :**

L'exploitation des attaques shatter a bien évolué depuis l'annonce de cette vulnérabilité. Comme nous l'avons démontré dans ce document, même le plus obscur des messages peut être utilisé pour déclencher l'exécution par un processus de code qu'il n'est pas censé exécuter.

Depuis il y eu de nombreuses discussions sur le filtrage des messages pour protéger les applications interactives qui sont lancées dans un contexte avec des privilèges plus élevés. Il devient clair que la seule solution sûre est de s'assurer que de telles applications ne soient pas exécutées sur le bureau d'utilisateurs non habilités.

Les concepteurs d'applications et les administrateurs systèmes se doivent d'être conscients et vigilants vis-à-vis des dangers potentiels dans le cas où des programmes sont exécutés avec des privilèges plus élevés sur le bureau des utilisateurs, et prendre des mesures pour prévenir leur exploitation.

Les exemples inclus dans ce document peuvent être utilisés contre toute application interactive tournant avec des privilèges plus élevés, en spécifiant simplement les paramètres comme le titre de la fenêtre. Une exploitation réussie permettra à un utilisateur d'exécuter des commandes dans un contexte avec des privilèges de sécurité plus élevés.

## **Messages Callback :**

Les messages suivants utilisent des callbacks comme un paramètre et sont connues comme vulnérables à ce type d'exploitation.

WM\_TIMER (Un patch a été diffusé pour ce cas)  
LVM\_SORTITEMS  
LVM\_SORTITEMSEX  
EM\_SETWORDBREAKPROC

Les messages suivants utilisent des callbacks comme un paramètre à travers un pointeur sur une structure. Ils sont potentiellement vulnérables à ce type d'exploitation.

EM\_STREAMOUT  
EM\_STREAMIN  
EM\_SETHYPHENATEINFO  
TVM\_SORTCHILDRENCB

**Messages de réécriture :**  
**Overwrite Messages :**

Les messages suivants utilisent un pointeur sur une structure comme paramètre et sont connues comme permettant la réécriture d'emplacements mémoire arbitraires.

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  
TCM\_GETITEMRECT



## Références

<http://security.tombom.co.uk/shatter.html>

*Traduction française par Damien :*

<http://www.katabatik.com/ktk/sections.php?op=viewarticle&artid=22>

[http://www.idefense.com/idpapers/Shatter\\_Redux.pdf](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://nextgenss.com/advisories/utilitymanager.txt>

<http://www.securityfocus.com/bid/5408/exploit/>

<http://www.securityfocus.com/data/vulnerabilities/exploits/mcafee-shatterseh2.c>

<http://security-assessment.com>