BitLocker

Aurélien Bordes

aurelien26@free.fr

Résumé BitLocker est une solution de chiffrement de disque dur apparue avec Windows Vista et disponible sur les éditions haut de gamme de Windows (professionnelle, intégrale et entreprise pour Vista, mais uniquement intégrale et entreprise pour Windows 7). BitLocker est également disponible sur les versions serveur du système (2008 et 2008R2). Fonctionnellement, BitLocker permet le chiffrement ¡¡ intégral¿¿ de volumes, qu'il s'agisse du volume système d'exploitation ou d'un volume de données.

Cet article propose d'étudier en détail le fonctionnement de BitLocker. L'étude sera menée sur un système Windows 7 32 bits édition intégrale. Même si un certain nombre de points peut différer par rapport à un système Vista ou 64 bits, le fonctionnement global reste identique.

1 Introduction

L'article est articulé en trois parties. La première partie décrit le fonctionnement de BitLocker et son intégration au sein du système d'exploitation. La seconde partie détaille le processus de chargement du système d'exploitation, en particulier l'initialisation des clés de chiffrement, ainsi que la vérification de l'intégrité de la séquence de démarrage du système avec ou sans le support d'un module TPM. Enfin, la dernière partie décrit différentes menaces et scénarios d'attaques sur un système protégé par BitLocker et présente les bonnes pratiques de sécurité à adopter.

En revanche, la solution BitLocker To Go apparue avec Windows 7 ne sera pas abordée. Même si le nom semble faire penser qu'il existe un lien direct avec BitLocker, dans les faits, il s'agit d'une solution technique avec quelques différences.

2 Éléments du système liés à BitLocker

2.1 Préparation du système

Lorsqu'un système Windows 7 est installé sur un disque vierge, celui-ci est ¡¡ préparé ¿¿ implicitement par le programme d'installation pour une future utilisation éventuelle de BitLocker. Dans les faits, cette opération consiste à créer deux partitions afin d'y séparer les éléments de la chaîne de démarrage.

La première partition, d'environ 100 Mo, contient le gestionnaire de chargement de Windows (`bootmgr`), la base de configuration de démarrage (BCD) et un logiciel de test de la mémoire (`memtest.exe`). Par défaut, aucune lettre de lecteur n'est affectée à

cette partition, ce qui la rend invisible par l'explorateur de fichier de Windows. Cette partition est marquée active dans la table des partitions et ne pourra pas être chiffrée par BitLocker. Dans la suite de l'article, cette partition sera dénommée la **partition de démarrage**. La seconde partition créée contient tous les autres éléments, soit la suite de la chaîne de démarrage, le système d'exploitation (répertoire Windows) et les divers programmes installés. Cette partition sera dénommée la **partition système** et elle pourra être intégralement chiffrée par BitLocker.

Il est possible par la suite de créer des partitions supplémentaires sur le même disque, ou sur un autre disque. Ces partitions seront dénommées les **partitions de données** et pourront être également intégralement chiffrées par BitLocker.

2.2 Boot Configuration Database

Avec le passage à Windows Vista, Microsoft a remanié en profondeur le processus de démarrage d'un système Windows. Le traditionnel chargeur ntldr et son fichier de configuration boot.ini ont disparu et ont été remplacés par un nouveau gestionnaire de démarrage, `bootmgr`, qui repose sur la BCD (Boot Configuration Database) pour stocker ses paramètres de configuration.

Les données de la BCD sont stockées dans un fichier de type ruche de base de registre, ce qui permet de réutiliser les API de la base de registre pour les manipuler. Le fichier est stocké dans le fichier `\boot\BCD` sur la partition de démarrage. Afin d'être manipulé par le système lorsque celui-ci est en service, la ruche est montée sous la clé `HKLM\BCD00000000` dans la base de registre.

Le fonctionnement de la BCD ainsi que ses structures sont décrits dans un document de Microsoft [1] ainsi que dans la MSDN [2]. Les principaux éléments utiles pour BitLocker sont présentés ci-dessous.

Une BCD est constituée d'entrées dénommées **Objets BCD**. Chaque objet est associé à un GUID et à un identifiant de type (le tableau 1 énumère les principaux types d'objet). Certaines entrées par défaut possèdent un GUID fixe. À chaque objet sont associés plusieurs attributs dénommés **Éléments BCD** qui configurent tel ou tel paramètre de l'objet (le tableau 2 énumère les principaux éléments pour un objet de type chargeur de système Windows). Sur un système en cours d'exécution, la BCD peut être manipulée localement ou à distance via WMI ou au moyen de l'utilitaire `bcdedit` [3]. Par exemple, la commande `bcdedit /enum all` affiche tous les objets de la base et leurs éléments associés (l'argument `/v` permet d'afficher le GUID en lieu et place du nom connu). Sachant que la BCD est une ruche de base de registre, les outils manipulant des fichiers de base de registre permettent d'en modifier le contenu en mode ¡¡ hors-ligne ¿¿ (*off-line*).

ID d'objet	Description	Objet par défaut
10100002	Application / Firmware application Windows boot manager	Windows Boot Manager 9dea862c-5cdd-4e70-acc1-f32b344d4795 {bootmgr}
10200003	Application / Boot application Windows boot loader	Windows 7 (\Windows\system32\winload.exe)
10200004	Application / Boot application Windows resume application	Windows Resume Application (\Windows\system32\winresume.exe)
10200005	Application / Boot application Windows memory tester	Diagnostics mémoire Windows {memdiag} (\boot\memtest.exe)
10400008	Application / Real-mode application Boot sector	

Table 1. Identifiant des objets BCD

ID	Type	Format	Nom	Description
11000001	BcdLibrary	Device	ApplicationDevice	Volume où l'application est située
12000002	BcdLibrary	String	ApplicationPath	Chemin d'accès à l'application
12000004	BcdLibrary	String	Description	Description affichée dans le menu de sélection
12000005	BcdLibrary	String	PreferredLocale	Identifiant de langue
14000006	BcdLibrary	ObjectList	InheritedObjects	Objets hérités
16000009	BcdLibrary	Boolean	AutoRecoveryEnabled	
21000001	BcdOSLoader	Device	OSDevice	Volume où le noyau est situé
22000002	BcdOSLoader	String	SystemRoot	Répertoire de base d'un système Windows
23000003	BcdOSLoader	Object	AssociatedResumeObject	
22000011	BcdOSLoader	String	KernelPath	Chemin d'accès au noyau
25000020	BcdOSLoader	Integer	NxPolicy	
260000a0	BcdOSLoader	Boolean	KernelDebuggerEnabled	

Table 2. Identifiant des éléments BCD

L'objet BCD le plus important est `{bootmgr}` [1] qui définit les paramètres du gestionnaire de démarrage. Le tableau 3 énumère les principaux attributs, dont une liste exhaustive est disponible sur [4].

ID	Type	Format	Nom	Description
24000001	BcdBootMgr	ObjectList	DisplayOrder	Énumère les systèmes d'exploitation pouvant être chargés
23000003	BcdBootMgr	Object	DefaultObject	Indique le système par défaut à charger
25000004	BcdBootMgr	Integer	Timeout	Valeur du délai d'expiration avant de charger le système par défaut
24000010	BcdBootMgr	ObjectList	ToolsDisplayOrder	Énumère les utilitaires pouvant être chargés

Table 3. Identifiant des éléments du bootmanager

La BCD joue un rôle important sur la sécurité d'un système. En effet, certains paramètres ont une incidence majeure sur l'intégrité de la chaîne de démarrage ou la sécurité du système d'exploitation devant être chargé. Or, la BCD étant située sur la partition de démarrage qui ne peut pas être chiffrée par BitLocker, elle peut donc être aisément modifiée de manière hors-ligne. Parmi les modifications qui sont les plus critiques, on peut citer :

- la modification de l'élément `KernelDebuggerEnabled` de l'objet du système d'exploitation chargé, qui permet d'activer ou non le débogueur du système chargé (s'il est activé, il devient aisé de contourner certains des dispositifs de sécurité d'un système) ;
- l'ajout d'une nouvelle entrée de type application, configurée pour être démarrée par défaut, permettant de démarrer un autre système d'exploitation que celui installé sur la partition système. La partie 6.2 détaillera un tel scénario.

Ainsi, lors du processus de chargement d'un système, il sera nécessaire, dans la validation de la chaîne de démarrage, de valider tout ou partie de l'intégrité de la BCD. Ce point est décrit dans la partie 4.2.

2.3 Chaîne de démarrage

Lorsque BitLocker n'est pas activé, les principales étapes de chargement du système sont les suivantes :

1. Dont le GUID est fixe et vaut 9dea862c-5cdd-4e70-acc1-f32b344d4795.

1. le BIOS effectue les tâches d'initialisation de la machine et choisit de poursuivre le démarrage sur un périphérique, généralement le disque dur ;
2. le MBR du disque est chargé puis exécuté. Celui-ci lit la table des partitions afin de déterminer la partition active. Lors de l'installation du système, c'est la partition de démarrage qui a été marquée comme active ;
3. le VBR (*Volume Boot Record*) de cette partition, dénommé également 1er étage de chargement ou *NTFS Boot Sector*, est chargé puis exécuté. Il s'agit des 512 premiers octets de la partition. Le VBR charge à son tour un code plus conséquent (environ 8 Ko) situé immédiatement à la suite du VBR. Il s'agit du 2e étage de chargement ou *NTFS Boot Block* ;
4. le 2e étage charge alors le gestionnaire de démarrage de Windows, le *Boot Manager*, c'est-à-dire le programme `bootmgr`, situé à la racine de cette même partition ;
5. le gestionnaire de démarrage lit sa configuration depuis la BCD toujours située sur la même partition (`\boot\BCD`).

Le gestionnaire de démarrage est un programme largement plus volumineux que les précédents éléments de la chaîne de démarrage (sa taille étant de 375 Ko). Il possède de nombreuses fonctionnalités parmi lesquelles la présentation à l'utilisateur d'une interface de configuration de la suite du processus de démarrage. Cette interface présente deux listes à l'utilisateur lui permettant de choisir la prochaine application à démarrer. La première liste concerne les systèmes d'exploitation (la liste est donnée par l'élément 24000001) et la seconde les utilitaires de diagnostic (la liste est donnée par l'élément 24000010). L'interface permet également de modifier des options de démarrage via les touches F8 (activation des modes sans échec ou mode debug) ou F10 (passage de paramètres en ligne de commande au noyau de Windows).

Lorsque l'utilisateur a choisi une application, ou que celle par défaut est automatiquement sélectionnée après expiration du délai prédéfini, les attributs associés à l'objet sélectionné sont lus dans la base BCD. Ceux-ci indiquent tous les éléments nécessaires pour poursuivre le chargement. Les principaux paramètres sont le périphérique (`ApplicationDevice`) et le chemin (`ApplicationPath`) de l'application à lancer. Dans le cas d'un système Windows, l'application est le chargeur de noyau situé sur la partition système (`\Windows\system32\winload.exe`). Celui-ci charge alors le noyau (indiqué par les paramètres `OSDevice`, `SystemRoot`, `KernelPath`), lui passe divers paramètres s'ils sont configurés (`NxPolicy`, `PAEPolicy`, `KernelDebuggerEnabled`) ou spécifiés explicitement via la touche F10.

Enfin, l'exécution est transférée au noyau et celui-ci continue le processus de chargement.

3 Architecture de BitLocker

Cette partie décrit l'intégration de BitLocker au sein d'un système et l'architecture des divers composants. S'il existe de nombreux articles de Microsoft sur ce sujet, le document [17] (partie *Full volume encryption*) présente une vision complète et technique sur le sujet.

3.1 Intégration au sein du système d'exploitation

Sous Windows, toute la gestion des périphériques (dont les disques) est sous la responsabilité du gestionnaire d'entrées/sorties du noyau (*IO manager*) qui s'appuie sur des pilotes (*drivers*) pour la gestion de périphériques (*devices*). Lorsque qu'un pilote détecte un périphérique qu'il prend en charge, il crée généralement un objet noyau nommé de type *Device* dans l'espace des objets du noyau (par convention dans le chemin `\Device\`).

Les volumes de disques sont gérés par le gestionnaire de volumes (`volmgr`). Lors du démarrage du système, celui-ci crée pour chaque partition reconnue un volume représenté par un objet *Device* nommé `HarddiskVolumeX` (où X commence à 1). Ces objets étant créés dans le répertoire `\Device\` de l'espace des objets du noyau et ne sont, théoriquement, accessibles qu'au noyau. Afin de les rendre accessibles aux applications en espace utilisateur, des liens symboliques sont créés dans l'espace `\GLOBAL??\`. Les volumes sont alors accessibles au moyen d'une lettre de lecteur (par exemple le lien `\GLOBAL??\C:` redirige vers `\Device\HarddiskVolume2`).

Sur un système avec trois partitions (démarrage, système et données) l'organisation est la suivante : le pilote (`DRV`) `volmgr` créé trois périphériques (`DEV`), un pour chaque partition, auxquels s'ajoute un périphérique destiné à la gestion des partitions (`VolMgrControl`). L'organisation hiérarchique est la suivante :

```
DRV - \Driver\volmgr
   DEV - \Device\HarddiskVolume3
   DEV - \Device\HarddiskVolume2
   DEV - \Device\HarddiskVolume1
   DEV - \Device\VolMgrControl
```

Le composant noyau principal de BitLocker est un pilote de périphérique dénommé `fvevol.sys`. Ce pilote est de type *filter driver* [5] [6]. Ce type de pilote vient se positionner ¡¡ au-dessus ¿¿ d'un périphérique ce qui lui permet d'intercepter toutes les demandes adressées au périphérique inférieur (ouverture, fermeture, lecture, écriture et IOControls). Il est d'ailleurs possible d'avoir plusieurs *filter drivers* chaînés en série au-dessus d'un même périphérique, chacun interceptant séquentiellement les demandes.

L'analyse de la chaîne de pilotes montre que, dans une installation de base d'un système Windows, il existe, au-dessus du périphérique d'un volume créé par `volmgr`, trois filtres attachés (`ATT`) :

- `fvevol` en charge du chiffrement BitLocker ;
- `rdyboost` utilisé par le système ReadyBoost ;
- `volsnap` (*Volume Shadow Copy Driver*) responsable de la partie noyau du mécanisme des clichés instantanés de volume VSS (*Volume Shadow Copy Service*).

En outre, un pilote de type système de fichiers (`FS`) pour la prise en charge du système NTFS (`Ntfs`) est également associé au périphérique, lui-même ayant un filtre attaché (`FltMgr`) en charge des pilotes de type *File System minifilters* [7]. Pour un volume, l'organisation hiérarchique est la suivante :

```
DRV - \Driver\volmgr
   DEV - \Device\HarddiskVolume1
      ATT - unamed: \Driver\fvevol
      ATT - unamed: \Driver\rdyboost
      ATT - unamed: \Driver\volsnap
   FS - unamed: \FileSystem\Ntfs
      ATT - unamed: \FileSystem\FltMgr
```

Ainsi, toutes les requêtes adressées au volume via le périphérique `HarddiskVolume1` seront d'abord traitées par les filtres `volsnap`, `rdyboost` et `fvevol`, puis par le pilote `volmgr`. L'ordre de chargement des filtres est strictement défini par Microsoft [8] et bien évidemment, `fvevol` doit être un filtre chargé au plus proche du périphérique. Ceci garantit, lors d'une opération d'écriture, que le filtre soit le dernier sollicité, permettant ainsi l'ultime opération de chiffrement avant l'écriture sur le volume. Inversement, lors d'une opération de lecture, le déchiffrement intervient en premier.

3.2 Structure du chiffrement

Les possibilités de chiffrement et les modes de fonctionnement de BitLocker sont largement décrits dans les divers documents de Microsoft ou dans des articles relatifs à BitLocker [19] [9] et [10]. Les grands principes sont décrits ci-dessous et Windows 7 n'apporte pas de nouveauté particulière.

Le filtre BitLocker permet, sur un système de fichiers reconnu par Windows, le chiffrement des secteurs dont la taille peut aller de 512 à 8192 octets. L'algorithme par défaut utilisé est AES en mode CBC avec une taille de clé de 128 bits ou de 256 bits. Si AES-CBC était utilisé tel quel, un certain nombre d'attaques cryptographiques serait possible (en particulier les attaques à clairs connus). Pour éviter de telles attaques, il est possible d'utiliser un mécanisme cryptographique supplémentaire de type diffuseur baptisé *Elephant* dont l'auteur, Niels Ferguson, détaille dans un article

[18] le fonctionnement et les bénéfices apportés en matière de sécurité cryptographique. Bien qu'il n'existe pas d'étude cryptographique avancée sur *Elephant*, son utilisation ne peut pas introduire de faiblesse sur la sécurité d'AES alors qu'il est supposé réduire les attaques cryptographiques. Son utilisation est donc fortement recommandée.

Quatre modes de chiffrement sont proposés pour le chiffrement d'un volume par BitLocker : `aes128`, `aes256`, `aes128_diffuser` ou `aes256_diffuser`. Le mode choisi détermine la taille de la clé AES ainsi que l'utilisation ou non du diffuseur. Le mode par défaut est `aes128_diffuser` (il n'est pas possible de choisir un autre mode via l'interface graphique).

Outre la clé utilisée pour le chiffrement par AES (128 ou 256 bits), le diffuseur nécessite sa propre clé baptisée ¡¡ clé de secteur ¿¿, dont la taille est identique à celle de chiffrement. Ainsi, suivant le mode de chiffrement utilisé, la taille de la clé varie en réalité de 128 bits (`aes128`) jusqu'à 512 bits (`aes256_diffuser`).

Lors de la mise en place de BitLocker pour un volume, la clé de chiffrement générée[2] aura toujours une taille fixe de 512 bits. Cette taille correspond à la plus grande taille possible, c'est-à-dire dans le mode `aes256_diffuser` ou 256 bits sont utilisés pour la clé de chiffrement et 256 bits pour la clé de secteur. Sur ces 512 bits, une partie ou l'intégralité est utilisée suivant le mode de chiffrement retenu.

Cette clé est baptisée la FVEK (*Full Volume Encryption Key*) et sert directement dans les opérations de chiffrement du disque. La connaissance de cette clé permet d'accéder au contenu du disque chiffré, y compris en mode hors-ligne. À titre d'exemple, il existe des outils (tels qu'une implémentation d'un pilote FUSE sous Linux) offrant la possibilité, via la fourniture d'une clé FVEK, d'accéder à un volume protégé par BitLocker en déchiffrant les secteurs du disque.

Si une clé FVEK est compromise, elle ne peut être changée qu'en désactivant puis réactivant BitLocker, ce qui entraîne le déchiffrement intégral du volume puis son chiffrement avec une nouvelle clé. Or, les opérations de chiffrement ou de déchiffrement sont extrêmement longues et peuvent durer plusieurs heures (la durée de chiffrement dépendant de la taille du volume). Ainsi, il est difficile de changer la FVEK.

Afin de permettre une certaine souplesse et de protéger au maximum la FVEK, celle-ci n'est jamais exposée à l'utilisateur. La FVEK est chiffrée, via AES-256 en mode CCM, par une clé de 256 bits baptisée VMK (*Volume Master Key*). La possession de la VMK permet le déchiffrement de la FVEK qui permet à son tour le déchiffrement des secteurs du volume. Cette solution permet de changer la VMK sans qu'il soit nécessaire de changer la FVEK.

2. Pour toutes les générations de clés de chiffrement, BitLocker utilise la fonction BCryptGenRandom de la CryptoNG.

Le mécanisme de récupération de la VMK par le système afin que le volume soit chiffré/déchiffré à la volée est appelé déverrouillage du volume (*unlock*). De même, l'opération inverse (arrêt des opérations et effacement de la VMK en mémoire) est appelée verrouillage du volume (*lock*).

3.3 Protection de la VMK

Afin de protéger la VMK, BitLocker propose différents mécanismes. Une VMK chiffrée par tel ou tel mécanisme est appelé un protecteur (*protector*) et il est possible d'avoir plusieurs protecteurs différents pour un même volume.

Trois protecteurs sont basés sur un secret (`External key`, `Numerical password` et `Passphrase`), un protecteur est basé sur un bi-clé (*Public Key*), un protecteur repose uniquement sur l'utilisation d'un module TPM et trois protecteurs combinent le module TPM et un secret (`TPMAndPIN`, `TPMAndStartupKey` et `TPMAndPINAndStartupKey`). Il est préférable d'utiliser un protecteur utilisant le module TPM car celui-ci assure l'intégrité de la chaîne de démarrage. En revanche, l'utilisation d'au moins un protecteur non basé sur le module TPM est fortement souhaitable afin de disposer d'un mode de récupération.

Les protecteurs basés sur le module TPM ne peuvent être utilisés que pour un volume de type ¡¡ lecteur du système d'exploitation ¿¿, c'est-à-dire un système démarré par le gestionnaire de démarrage de Windows. De plus, il ne peut y avoir qu'un seul protecteur de ce type par volume. Inversement, le protecteur de type `Password` ne peut être utilisé que sur les volumes de ¡¡ lecteurs de données ¿¿.

Dans la description des protecteurs qui suit, le numéro entre parenthèses correspond au type retourné par la fonction `GetKeyProtectorType` du fournisseur WMI de BitLocker. Toutes les opérations de chiffrement de la VMK sont basées sur AES avec une clé de 256 bits.

TPM (1) Avec ce type de protection, la VMK est protégée directement par le module TPM. Le seul élément de sécurité conditionnant sa déprotection par le TPM est l'intégrité du système mesuré via les registres PCR (voir partie 4.1).

Clé externe (2) Ce mode est également appelé `External key`, `Startup key` ou `Recovery key`.

Une clé de 256 bits est générée et chiffre directement la VMK. Cette clé est sauvée dans un fichier, dont le nom est celui-ci de l'identifiant unique (GUID) du protecteur avec l'extension BEK. Ce fichier doit être mis à la racine d'un disque ou d'une clé USB afin d'être présenté lors du déverrouillage d'un volume.

Mot de passe numérique (3) Ce mode est également appelé `Numerical password` ou `Recovery password`.

Une clé de 128 bits est générée ou peut être saisie par un utilisateur. Cette clé est généralement convertie sous forme de mot de passe numérique lorsqu'elle est présentée à l'utilisateur (voir la partie 3.4 pour le détail de l'opération de conversion).

Pour être utilisée pour le chiffrement de la VMK, cette clé de 128 bits doit être étirée (*stretch*) (voir la partie 3.4 pour le détail de l'opération d'étirement) en vue d'obtenir une clé de 256 bits.

TPMAndPIN (4) Ce mode à sensiblement évolué avec Windows 7 suite à des attaques présentées à Black Hat [21] sur le module TPM.

Dans le mode initial, la VMK est protégée de la même manière que le mode TPM. Cependant, lors de l'opération de protection (via la commande `TPM_Seal`), un ¡¡ code ¿¿ est demandé à l'utilisateur. Celui-ci est utilisé dans le paramètre d'authentification de l'opération de protection. Lorsque la VMK est déprotégée (commande `TPM_Unseal`), le même code doit obligatoirement être fourni dans le paramètre d'authentification pour que le TPM autorise l'opération (en plus de la validité des registres PCR).

Par défaut, uniquement des chiffres sont autorisés pour la génération de ce code, mais il est possible, depuis Windows 7 et via une stratégie de groupe, d'autoriser n'importe quel type de caractères : on parle alors de ¡¡ code confidentiel de démarrage amélioré ¿¿. Mais l'utilisation du code confidentiel renforce également la protection de la VMK. En effet, le code PIN est étiré afin de générer une clé temporaire de 256 bits. Ce n'est alors plus directement la VMK qui est protégée par le module TPM, mais la VMK combinée à la clé temporaire. Ainsi, même s'il devient possible d'extraire l'élément protégé du module TPM, il faut encore réaliser une attaque en force brute sur le code PIN afin d'obtenir la VMK.

TPMAndStartupKey (5) Dans ce mode, deux clés de 256 bits sont générées et c'est la combinaison des deux (via un XOR) qui chiffre la VMK. La première clé est protégée via le TPM et la seconde clé est traitée de la même manière qu'une clé externe. Ainsi, pour pouvoir déprotéger la VMK, il faut, en plus de la validation de l'intégrité de la machine mesurée par les registres PCR, fournir la clé externe sous forme de fichier sur un disque ou une clé USB.

TPMAndPINAndStartupKey (6) Ce mode combine à la fois l'utilisation du code d'authentification et de la clé externe. Il s'agit du mode de protection le plus sécurisé (car combinant intégrité, authentification et secret externe), mais également le plus contraignant.

Clé publique (7) La VMK est protégée via une clé de 256 bits chiffrée à son tour par une clé publique. Cette solution est mise en œuvre via l'utilisation d'un certificat numérique ou d'une carte à puce.

Mot de passe (8) Ce mode est également appelé `Passphrase` ou `Password`. L'utilisateur doit entrer un mot de passe. Celui-ci doit être étiré afin de le convertir une clé de 256 bits et d'éviter les attaques de type force brute sur le mot de passe. C'est la clé générée de 256 bits dérivés du mot de passe qui chiffre la VMK.

3.4 Modes particuliers et fonctions annexes

Suspension de BitLocker Pour un volume donné, il est possible de suspendre la protection BitLocker (*disable*) afin de permettre son libre accès, y compris sur un autre système, sans qu'il soit nécessaire de déchiffrer tout le volume. L'opération de suspension consiste à créer un protecteur particulier composé d'une clé, en clair, et de la VMK du volume chiffrée par cette même clé. Lorsque ce type de protecteur est présent dans les métadonnées d'un volume, la VMK est toujours accessible et le volume est systématiquement déverrouillé.

Déverrouillage automatique de volume Lorsque le système démarre, le gestionnaire de démarrage de Windows ne déverrouille que le volume du système d'exploitation. Or s'il existe des volumes de données supplémentaires, ces derniers doivent également être déverrouillés pour être accédés. Cette opération peut être effectuée manuellement par l'utilisateur (il doit à chaque fois fournir un mot de passe ou une clé externe) ou être effectuée automatiquement par le système (on parle alors de déverrouillage automatique).

Lorsque le déverrouillage automatique est activé pour un volume, les opérations suivantes sont réalisées :

- un protecteur de type `clé externe` est créé pour le volume donné. Pour rappel, la clé de ce protecteur (`Ke`) chiffre la VMK du volume de donnée (`VMKd`) ;
- si elle n'existe pas encore, une clé de déverrouillage automatique (`Ka`) associé au volume est générée, puis est stockée chiffrée par la VMK du système d'exploitation (`VMKs`) dans les métadonnées du système d'exploitation ;
- la clé externe (`Ke`) est chiffrée par la clé de déverrouillage automatique (`Ka`) et est stockée dans la base de registre `HKLM\SYSTEM\CurrentControlSet\Control\FVEAutoUnlock\{GUID volume}`[3].

3. Le pilote BitLocker impose que cet emplacement ne soit accessible qu'à l'entité SYSTEM. Il ne peut donc pas être consulté au moyen de `regedit`, même dans un contexte d'administrateur.

Ainsi, nous avons (la notation `K[D]` signifiant donnée `D` chiffrée par clé `K`) :

- dans les métadonnées du système d'exploitation : `VMKs[Ka]`
- dans la base de registre : `Ka[Ke]`
- dans les métadonnées du disque de données : `Ke[VMKd]`

Lorsque le système d'exploitation est déverrouillé et que le pilote BitLocker dispose de la VMK du volume système (`VMKs`), il peut donc accéder à la VMK de ce volume (`VMKd`) et ainsi déverrouiller automatiquement le volume de données.

Étirement de clés Les clés utilisées en interne par BitLocker sont des clés de 256 bits. Cependant, certaines clés ou mots de passe fournis par l'utilisateur n'ont pas une telle taille (code PIN amélioré, mot de passe, mot de passe numérique). Afin de renforcer ces secrets, une fonction d'étirement (*stretch*) est utilisée (la description de cette fonction, ainsi qu'un pseudo-code sont disponibles dans [9]). Le prototype de cette fonction est le suivant :

`Clé 256 bits = Stretch(`Data de taille quelconque, Sel de 256 bits`)`

Cette fonction prend en entrée une donnée de taille arbitraire et un sel et retourne, en sortie, une clé de 256 bits. En interne, la fonction utilise la fonction SHA-256 avec 2^{20} tours, ce qui limite considérablement les attaques de type force brute sur la donnée d'entrée (la conversion durant plus d'une seconde).

Conversion d'un mot de passe numérique Le mot de passe numérique est une clé de 128 bits. Or celle-ci peut être rentrée par l'utilisateur au démarrage du système [15]. Afin de faciliter la saisie, la clé n'est pas affichée sous forme hexadécimale, mais sous forme de 6 groupes de 6 chiffres. Cette forme permet de n'avoir que des chiffres (ce qui permet la saisie avec les touches F1 à F10) et permet un contrôle de parité sur chaque groupe de chiffre (chacun doit être divisible par 11). Après la saisie, le mot de passe numérique est reconverti en clé de 256 bits via la fonction d'étirement. La fonction `IsNumericalPasswordValid` du fournisseur WMI de BitLocker (partie 5.1) permet de valider qu'un mot de passe numérique soit correct (au sens de la parité de tous les groupes de chiffre).

Exemple de mot de passe numérique :

Format d'affichage	099550	445236	615868	677281	630102	546612	392150	533742
Blocs divisés par 11	9050	40476	55988	61571	57282	49692	35650	48522
Clé de 128 bits	5a 23	1c 9e	b4 da	83 f0	c2 df	1c c2	42 8b	8a bd

3.5 Mise en place de BitLocker

Lorsque BitLocker est activé pour un volume donné, les transformations suivantes sont appliquées :

- le VBR (*Volume Boot Record*) contenu dans les 512 premiers octets du volume est remplacé par un VBR de type BitLocker. La signature d'un tel VBR est `-FVE-FS-` ;
- l'intégralité du volume est chiffrée à l'exception de trois zones de 65 Ko utilisées pour le stockage des métadonnées. L'emplacement de ces trois zones est indiqué dans le VBR du volume.

Sur chaque volume chiffré, BitLocker doit stocker des informations nécessaires au déchiffrement du volume (en particulier les protecteurs de VMK), à la vérification de l'intégrité de ces données et éventuellement à la vérification de l'intégrité du système. Ces données, baptisées métadonnées, sont écrites sur une des trois zones non chiffrées du volume. Cette zone est ensuite répliquée sur les deux autres afin d'assurer une redondance de ces métadonnées, car leur perte entraînerait l'impossibilité définitive de déchiffrer le volume.

Ces métadonnées sont composées de deux blocs. Le premier bloc (`INFORMATION`) contient les éléments ci-dessous (leur composition est détaillée dans les parties suivantes) :

- un identifiant unique de type GUID et le mode de chiffrement appliqué au volume ;
- une chaîne de caractères décrivant le nom du volume ainsi que la date de chiffrement initiale ;
- la FVEK du volume chiffrée par la VMK ;
- pour chaque méthode de protection de la VMK choisie, un protecteur contenant la VMK protégée par cette méthode ;
- pour le volume du système d'exploitation, une structure servant à la validation des éléments de la BCD (voir partie 4.4).

Le second bloc (`VALIDATION`) permet la validation du bloc précédent. Sa composition est détaillée dans la partie 4.3.

3.6 Structures de données BitLocker

Le pilote BitLocker utilise pour son fonctionnement des structures de données baptisées *datum*. Ces structures sont utilisées par exemple en mémoire pour le stockage des clés ou dans la communication avec le pilote, mais également sur un disque dans le stockage de métadonnées sur le volume.

Le document [9] esquisse une analyse de la structure des *datums* et constitue ainsi un document de référence. L'analyse est d'autant plus complexe que les *datums* peuvent se suivre ou s'imbriquer à plusieurs niveaux à l'image de structures de protocole réseau. Cependant, l'écriture d'un parseur de *datums* est une étape indispensable pour la compréhension des données de BitLocker. Les principaux *datums* sont présentés ci-dessous et les types associés sont donnés par le tableau 4.

Type	Type de Datum (usType)
1	Datum Unicode
2	Datum Key
3	Datum StretchKey
4	Datum UseKey
5	Datum AES256-CCM
8	Datum VMK

Table 4. Types de Datum

S'il existe différents types de *datum*, ils possèdent tous un en-tête commun de 8 octets défini par la structure suivante :

```
typedef struct _DATUM_HEADER
{
   USHORT usSize;       // Taille en octets du datum
   USHORT usContext;    // Contexte d'utilisation du datum
   USHORT usType;       // Type de structure de donnees du datum
                        // apres l'en-tete
   USHORT usUnknown;
} DATUM_HEADER, *PDATUM_HEADER;
```

Datum Unicode C'est le *datum* le plus simple : il est utilisé pour stocker une chaîne de caractères, par exemple pour indiquer le nom d'un volume de disque ou un type de protection. Sa définition est la suivante :

```
typedef struct _DATUM_UNICODE
{
```

```
    DATUM_HEADER dhHeader;
    WCHAR        szText[1];
} DATUM_UNICODE, *PDATUM_UNICODE;
```

Datum Key Le *datum* de type clé sert à stocker une clé en clair (les principaux types de clé sont indiqués dans le tableau 5). Il n'est pas vraiment plus complexe que le type précédent et sa définition est la suivante :

```
typedef struct _DATUM_KEY
{
    DATUM_HEADER dhHeader;
    USHORT       usType;
    USHORT       usUnknown;
    BYTE         bKey[1];
} DATUM_KEY, *PDATUM_ KEY;
```

Type	Description
0x2002	Clé externe
0x2003	VMK
0x2005	HASH256 de validation
0x8000	FVEK AES128 avec Elephant
0x8001	FVEK AES256 avec Elephant
0x8002	FVEK AES128
0x8003	FVEK AES256

Table 5. Types de clés

Le rôle des types ¡¡ Clé externe ¿¿ et ¡¡ HASH256 ¿¿ et leur cadre d'utilisation sont détaillés dans les parties 3.6 et 4.3.

Datum AES256-CCM Ce *datum* est déjà plus complexe que les précédents. Il permet de stocker un bloc de données en le protégeant via un chiffrement AES en mode CCM avec une clé de 256 bits. Le mode CCM (*Counter with CBC-MAC*) permet d'apporter un code d'authentification supplémentaire en plus de l'opération de chiffrement via AES. Dans la quasi-totalité des cas, un *datum* AES256-CCM sert à chiffrer un *datum* Key, par exemple pour protéger une FVEK ou une VMK. La définition de ce *datum* est la suivante :

```
typedef struct _DATUM_AES256_CCM
{
    DATUM_HEADER dhHeader;
```

```
    FILETIME     ftFileTime;
    ULONG        ulCounter;
    BYTE         bMac[16];
    BYTE         bEncryptedData[1];
} DATUM_AES256_CCM, *PDATUM_AES256_CCM;
```

Les champs ftFileTime et ulCounter servent de nonce pour le chiffrement AES. En interne, BitLocker implémente le chiffrement AES-CCM. Cependant, depuis la version 7.1 du SDK, ce mode est disponible dans la Crypto NG. L'annexe 8 décrit un pseudo-code C de déchiffrement d'un *datum* AES256-CCM en utilisant les fonctions de la Crypto NG qui ne pourra fonctionner que sous Windows Vista SP1 ou Windows 7.

Il est important de noter que, dans le code d'exemple, le second appel à la fonction `BCryptDecrypt` échoue si le code MAC spécifié via le paramètre bMac n'est pas valide. Ceci permet de s'assurer que la clé fournie est correcte.

Datum VMK Le *datum* de type VMK est utilisé pour un protecteur d'une VMK. Il existe donc une structure de ce type dans les métadonnées du volume pour chaque protecteur de VMK ajoutés. La définition de ce *datum* est la suivante :

```
typedef struct _DATUM_VMK
{
    DATUM_HEADER dhHeader;
    GUID         guid;
    FILETIME     CreationDate;
    USHORT       usUnknown1;
    USHORT       usUnknown2;
} DATUM_VMK, *PDATUM_VMK;
```

Le champ guid sert à identifier de manière unique chaque protecteur de VMK. Ce *datum* inclut un certain nombre d'autres *datums* suivant le type de protection mis en œuvre sur la VMK :

- un *datum* de type Unicode (sauf à de rares exceptions) qui reprend dans sa chaîne de caractères la description du type de protection de la VMK (`DiskPassword`, `Passphrase`, `TPMAndPin`, etc.) ;
- un *datum* de type AES256-CCM qui sert à stocker la VMK chiffrée. La clé de chiffrement découle du mécanisme de protection de la VMK (protection directe) ;
- un *datum* de type UseKey ou StretchKey qui sert, inversement, à chiffrer l'élément de protection de la VMK. Ces *datums* contiennent à leur tour un *datum* type AES256-CCM dont la clé est la VMK et l'élément protégé le mécanisme de protection de la VMK (protection inverse[4]). La différence entre

4. L'utilité de la protection inverse est détaillée à la fin de cette partie.

le *datum* de type `UseKey` et `StretchKey` est la présence, dans le second, d'un sel servant à étirer (*stretch*) le secret.

Afin d'illustrer ces enchaînements complexes, prenons un exemple dans le cas d'un protecteur de type `Mot de passe numérique` (`RecoveryPassword`). Le *datum* VMK contient alors :

```
[DATUM_VMK]
   Guid: {F4691499-E34B-44FA-973B-CE25D5F7B966}
   Date: 04/10/2010 08:35:07
   [DATUM_UNICODE]
      Value: DiskPassword
   [DATUM_STRETCH_KEY]
      Type: Password
      Salt: 84b9c7b477910140d9bd5f6554eb1ae9
      [DATUM_AES256_CCM] - AES256CCM Decrypted with key:
                           4218b0...VMK DU VOLUME...94da86
         [DATUM_KEY]
            Size: 128bit
            Type: Password
            Value: 5118d6e2cd08fc64741c9124bb2d6cbf
            Formatted value:
      068475-638770-024783-284372-080124-102971-128777-539044
   [DATUM_AES256_CCM] - AES256CCM Decrypted with key: C12C57...18F151
      [DATUM_KEY]
         Size: 256bit
         Type: VMK
         Value: 4218b0...VMK DU VOLUME...94da86

Pour info :
   C12C57...18F151 = Stretch(Data=5118d6...2d6cbf, Salt=84b9c7...eb1ae9)
```

On constate que, dans le troisième sous *datum*, la VMK est chiffrée par la clé de récupération étirée avec le sel fourni dans le *datum* `StretchKey`. Inversement, cette même clé de récupération est chiffrée par la VMK dans le second sous *datum*.

Autre exemple dans le cas d'un protecteur de type clé externe (`RecoveryKey`), la structure du protecteur est la suivante :

```
[DATUM_VMK]
   Guid: {127FFBCD-9766-4267-A581-1D7C3C1BF24D}
   Date: 28/09/2010 08:41:42
   [DATUM_UNICODE]
      Value: ExternalKey
   [DATUM_USE_KEY]
      Type: External
      [DATUM_AES256_CCM] - AES256CCM Decrypted with key:
               4218b0...VMK DU VOLUME...94da86
         [DATUM_KEY]
            Size: 256bit
            Type: External
            Value: c67b5b...6c5f74
   [DATUM_AES256_CCM] - AES256CCM Decrypted with key: C67B5B...6C5F74
      [DATUM_KEY]
         Size: 256bit
```

```
      Type: VMK
      Value: 4218b0...CLÉ VMK DU VOLUME...94da86
```

Ce cas est plus simple : la VMK est chiffrée directement par la clé externe et, inversement, la clé externe est chiffrée directement par la VMK sans l'utilisation de fonction d'étirement.

Le fichier de la clé externe est également structuré sous forme de *datums*. Il contient généralement une chaîne de description et la clé :

```
[DATUM_EXTERNAL_KEY]
   Date: 28/09/2010 08:41:42
   Guid: {127FFBCD-9766-4267-A581-1D7C3C1BF24D }
   [DATUM_UNICODE]
      Value: ExternalKey
   [DATUM_KEY]
      Size: 256bit
      Type: External
      Value: c67b5b...6c5f74
```

Le chiffrement inverse offre la possibilité, à condition d'avoir la VMK d'un volume et si cela est applicable, la récupération des éléments de protection de la VMK :

- la clé de récupération dans le cas d'un protecteur de type `Mot de passe numérique` ;
- la clé externe dans le cas de protecteurs de type `Clé externe`, `TPMAndStartupKey` et `TPMAndPINAndStartupKey`.

Cela permet de les imprimer ou de les sauvegarder à nouveau ou de changer la VMK sans redemander à l'utilisateur tous les secrets des protecteurs.

3.7 Communication avec le pilote

L'analyse de la communication entre les applications et un pilote est toujours intéressante. En effet, il s'agit d'un point délicat souvent source d'élévation de privilège si un contrôle d'accès n'est pas mis en place. En effet, si les applications s'exécutent dans le contexte de sécurité d'un utilisateur, le code du pilote s'exécute, quant à lui, dans le contexte extrêmement privilégié du noyau.

Ouverture du périphérique Comme vu dans la partie 3.1, le pilote `fvevol` est un pilote noyau. Celui-ci créé un périphérique qui n'est pas nommé, ce qui rend son ouverture en espace utilisateur impossible.

Or ce pilote agit également comme filtre sur tous les périphériques de volume et intercepte, à ce titre, les IOControl envoyés au volume. Ainsi, pour communiquer avec le pilote BitLocker, il suffit, après avoir retrouvé les numéros d'IOControl interceptés

que le pilote prend en charge, d'ouvrir le périphérique d'un volume et d'y envoyer des IOControl.

Cependant, avant de transmettre l'appel, le gestionnaire d'entrée/sortie effectue un contrôle d'accès afin de déterminer si l'application est autorisée à effectuer l'IOControl demandé. Pour cela, il confronte le jeton d'accès du processus initiateur avec les droits accordés à l'ouverture du périphérique associé au volume. Le ou les droit(s) vérifié(s) est(sont) donné(s) par le numéro d'IOControl (la description complète du mécanisme est disponible dans la MSDN [14]). Ces droits peuvent être `FILE_ANY_ACCESS (ANY)`, `FILE_READ_ACCESS (READ)` ou `FILE_WRITE_ACCESS (WRITE)`.

À l'aide de WinDBG, il est possible d'obtenir le descripteur de sécurité d'un volume :

```
!devobj \device\HarddiskVolume2
...
Device (ACL):
...
ACL is: ->Ace[0]: ->AceType: ACCESS_ALLOWED_ACE_TYPE
ACL is: ->Ace[0]: ->AceFlags: 0x0
ACL is: ->Ace[0]: ->AceSize: 0x14
ACL is: ->Ace[0]: ->Mask : 0x001200a0
ACL is: ->Ace[0]: ->SID: S-1-1-0 (Tout le monde)

ACL is: ->Ace[1]: ->AceType: ACCESS_ALLOWED_ACE_TYPE
ACL is: ->Ace[1]: ->AceFlags: 0x0
ACL is: ->Ace[1]: ->AceSize: 0x14
ACL is: ->Ace[1]: ->Mask : 0x001f01ff
ACL is: ->Ace[1]: ->SID: S-1-5-18 (AUTORITE NT\Système)

ACL is: ->Ace[2]: ->AceType: ACCESS_ALLOWED_ACE_TYPE
ACL is: ->Ace[2]: ->AceFlags: 0x0
ACL is: ->Ace[2]: ->AceSize: 0x18
ACL is: ->Ace[2]: ->Mask : 0x001f01ff
ACL is: ->Ace[2]: ->SID: S-1-5-32-544 (BUILTIN\Administrateurs)

ACL is: ->Ace[3]: ->AceType: ACCESS_ALLOWED_ACE_TYPE
ACL is: ->Ace[3]: ->AceFlags: 0x0
ACL is: ->Ace[3]: ->AceSize: 0x14
ACL is: ->Ace[3]: ->Mask : 0x001200a0
ACL is: ->Ace[3]: ->SID: S-1-5-12 (BUILTIN\Administrateurs)
```

On constate donc que l'entité SYSTEM et les membres du groupe Administrateurs disposent d'un accès en contrôle total au volume d'un disque (masque de `0x001f01ff`).

Le tableau 6 liste les numéros d'IOControl pris en charge par le pilote BitLocker. Le décodage de ces numéros peut-être effectué par un outil [5] afin de déterminer les différentes parties (Device Type, Required Access, Function Code, Transfert Type).

Par rapport aux droits du descripteur de sécurité du volume, seuls SYSTEM et Administrateurs sont donc autorisés à appeler les IOControl qui requièrent un accès

5. Par exemple IoctlDecoder d'OSR Online.

`FILE_READ_ACCESS (READ)` ou `FILE_WRITE_ACCESS (WRITE)`. En revanche, tous les autres utilisateurs ne peuvent appeler que les IOControl qui requièrent un accès `FILE_ANY_ACCESS (ANY)`.

IOCtl	Fonction associée	Numéro	Accès
0x45561083	IoctlFveGetDataset	0x420	ANY
0x4556D087	IoctlFveSetDataset	0x421	READ + WRITE
0x45561088	IoctlFveGetStatus	0x422	ANY
0x4556508F	IoctlFveGetKey	0x423	READ
0x45561097	IoctlFveProvideVmk	0x425	ANY
0x4556D0A0	IoctlFveAction (type RW)	0x428	READ + WRITE
0x4556D0A7	IoctlFveBindDataVolume	0x429	READ + WRITE
0x4556D0AB	IoctlFveUnbindDataVolume	0x42A	READ + WRITE
0x455650AC	IoctlFveVerifyBindDataVolume	0x42B	READ
0x455650B0	IoctlFveCheckUnlockInfoDataVolume	0x42C	READ
0x4556D0B7	IoctlFveUnbindAllDataVolume	0x42D	READ + WRITE
0x4556D0BB	IoctlFvePrepareHibernate	0x42E	READ + WRITE
0x4556D0BF	IoctlFveCancelHibernate	0x42F	READ + WRITE
0x4556D0C3	IoctlFveRegisterContext	0x430	READ + WRITE
0x4556D0C7	IoctlFveUnregisterContext	0x431	READ + WRITE
0x455610C8	IoctlFveAction (type ANY)	0x432	ANY

Table 6. IOCtl du pilote bitlocker

Les droits sur le pilote sont donc correctement positionnés et la majorité des appels nécessitent des droits élevés. Avec de tels droits, il est en revanche très facile de récupérer la VMK d'un volume via l'IOControl `0x4556508F` qui appelle la fonction `IoctlFveGetKey` (le listing ci-dessous fournit un exemple de code pour réaliser cet appel).

En tant que simple utilisateur, peu de fonctions sont autorisées, mais il est possible, pour un volume donné, de connaître l'état de BitLocker ou de récupérer les métadonnées. Qu'un simple utilisateur puisse récupérer les métadonnées d'un volume n'est en soit pas un problème de sécurité, puisque ces données ne sont pas chiffrées sur le disque et les clés contenues sont chiffrées (VMK par les protecteurs, FVEK par la VMK).

```
#define IOCTL_FVE_PROVIDE_VMK 0x45561097

HANDLE hVolume;
DWORD  dwSizeBufferOut, dwSizeReturned;
PBYTE  pbBufferOut;
LPTSTR szVolumeName = TEXT("\\\\?\\GLOBALROOT\\Device\\HarddiskVolume2");

// Ouverture du peripherique avec des droits GENERIC_READ et GENERIC_WRITE
```

```
// ce qui necessite d'etre membre du groupe Administrateurs
hVolume = CreateFile(szVolumeName, GENERIC_READ | GENERIC_WRITE | SYNCHRONIZE,
    \
   FILE_SHARE_READ | FILE_SHARE_WRITE | FILE_SHARE_DELETE, \
   NULL, OPEN_EXISTING, 0, NULL);

dwSizeBufferOut = 100000;
pbBufferOut = (PBYTE)malloc(dwSizeBufferOut);

// Appel de l'IOControl IOCTL_FVE_PROVIDE_VMK
// hVolume doit etre ouvert avec des droits GENERIC_READ
DeviceIoControl(hVolume, IOCTL_FVE_PROVIDE_VMK, NULL, 0, \
   pbTotoOut, dwSizeBufferOut, &dwSizeReturned, 0);
```

3.8 Gestion du module TPM

Plusieurs composants sous Windows permettent d'interagir avec un module TPM. Il y a tout d'abord le pilote TPM (celui intégré ou fourni par le fabricant) qui assure la prise en charge matérielle du module. Le pilote crée généralement un périphérique non accessible aux utilisateurs du système. Second composant, le service TBS [6] assure l'interface entre le périphérique et l'API du TPM [11]. Cette API permet d'envoyer des commandes au module TPM via la fonction `Tbsip_Submit_Command`. Le jeu des commandes autorisées ou bloquées est défini dans la base de registre ou par une stratégie de groupe. Enfin, il existe un fournisseur WMI [12] permettant l'administration d'un module TPM localement ou à distance via l'infrastructure WMI.

Toutes les commandes et les structures de données sont définies dans la norme du Trusted Computing Group à laquelle le module TPM doit se conformer. Par exemple, le code donné en annexe 9 permet de lire la valeur actuelle des registres PCR via la commande `TPM_PCRRead`.

4 Processus de chargement et validation

Avant d'autoriser l'exécution d'un système, `bootmgr` effectue plusieurs vérifications d'intégrité.

La première validation concerne la chaîne de démarrage. Les éléments de cette chaîne sont exécutés les uns après les autres et un élément exécuté est mesuré [7] par son prédécesseur et doit mesurer son successeur. Cette chaîne va du BIOS, supposé non détourné [8] et initiant la chaîne, jusqu'au `bootmgr`, dernier élément exécuté et en

6. Dont le nom complet est Services de base de module de plateforme sécurisée.
7. La mesure consiste à calculer l'empreinte cryptographique de l'élément.
8. Selon les spécifications, le BIOS est supposé, au moins en partie, être immuable, mais cette spécification n'est pas toujours respectée sur certaines plateformes.

charge de la validation. Pour valider cette chaîne, BitLocker repose sur l'utilisation d'un module TPM.

La seconde validation concerne les données qui ne peuvent pas être chiffrées par BitLocker. Il s'agit, d'une part, des métadonnées du volume et, d'autre part, de la base de configuration de démarrage (BCD). Ces validations d'intégrité ne sont effectuées que lorsqu'un protecteur reposant sur le module TPM est utilisé (`TPM`, `TPMAndPIN`, `TPMAndStartupKey` et `TPMAndPINAndStartupKey`). Dans les autres cas, les validations d'intégrité ne sont pas vérifiées, ce qui permet un mode de recouvrement, évidemment moins sécurisé.

4.1 Utilisation du module TPM

Cette partie n'a pas pour vocation d'expliquer en détail le fonctionnement d'un module TPM mais plutôt son utilisation par BitLocker. Il existe de nombreux articles (par exemple [24]) décrivant l'architecture TPM ainsi que les différents modes d'utilisation possibles, mais BitLocker n'utilise que le mode SRTM (*Static Root of Trust Measurement*). Ce mode permet de protéger un secret et de vérifier l'intégrité de certains éléments du système. Un secret protégé par le module TPM ne pourra être restitué que si les empreintes cryptographiques des éléments mesurés sont identiques à celles indiquées pendant la phase de protection.

Pour ce faire, le module TPM dispose de PCR (*Platform Configuration Registers*), registres dont la valeur ne peut pas être fixée arbitrairement, mais uniquement ¡¡ étendue ¿¿, c'est-à-dire que la nouvelle valeur du PCR dépend de l'ancienne valeur et de la donnée à étendre suivant la formule $PCR_{n+1} = SHA1(PCR_n|M)$, où $M = SHA1(Data)$. Tous les registres PCR sont fixés à 0 lors de l'initialisation du module TPM au démarrage de la machine [9].

Lorsqu'une information est scellée par le TPM (*seal*) (opération `TPM_Seal`), une liste de registres PCR et leur valeur souhaitée est spécifiée comme paramètre de l'opération. Lorsque l'information doit être déprotégée (*unseal*) (opération `TPM_Unseal`), le TPM autorise l'opération uniquement si les registres PCR courants du TPM sont identiques à ceux spécifiés lors de la protection.

Un module TPM dispose d'au moins 16 registres PCR dont les spécifications du TCG (*Trusted Computing Group*) indiquent la façon dont ils doivent être utilisés sur une plate-forme conforme (voir tableau 7).

Les données destinées à être stockées dans les registres 0 à 7 sont mesurées par le CRTM [10] et le BIOS lors de l'initiation de la machine. En revanche, les registres

9. À quelques exceptions près. Les registres non initialisés à 0 ne sont pas utilisés par BitLocker
10. Le CRTM est le premier code exécuté qui se mesure lui-même et initie la chaîne.

PCR	Mesure	Mesuré par
0	CRTM (*Core Root of Trust of Measurement*) et BIOS	CRTM et BIOS
1	Données de configuration du BIOS	BIOS
2	Codes de ROM supplémentaires	BIOS
3	Données de configuration de codes de ROM supplémentaires	BIOS
4	Code de l'IPL (*Initial Program Load*), généralement le MBR du disque	BIOS
5	Données de l'IPL, soit la table des partitions pour un MBR	BIOS
6	State Transition and Wake Events	BIOS
7	Host Platform Manufacturer Control	BIOS
8-15	Utilisé par le système d'exploitation	Composants du système

Table 7. Utilisation des registres PCR de la norme TCG

d'indice 8 et plus sont à la disposition du système d'exploitation afin de mesurer ses propres éléments de la chaîne de démarrage. Ces éléments sont ceux exécutés postérieurement au MBR.

Mesures initiales effectuées par le BIOS Les mesures effectuées par le BIOS concernent le CRTM lui-même, le BIOS ainsi que divers éléments. Le résultat des mesures est stocké dans le PCR[0]. En matière de sécurité, cette étape soulève deux problèmes :

- il est difficile de vérifier la pertinence des mesures effectuées, car il est complexe de disposer du code du CRTM, du BIOS et des données mesurées. Il faut donc faire confiance au résultat sans pouvoir le vérifier ;
- dans le principe, le code du CRTM se mesure lui-même. Si un attaquant est en mesure de changer le code du CRTM ou du BIOS d'un système, il peut mettre en défaut le début de la chaîne de mesure.

Les autres mesures concernent la configuration du BIOS (PCR[1]), les éventuels codes de ROM supplémentaires (PCR[2]) et les données de configurations de ces ROM (PCR[3]). Là encore, il est très difficile de vérifier si les mesures sont correctement effectuées.

Dans sa configuration par défaut, BitLocker ne vérifie pas les PCR associés aux données de configuration (PCR[1] et [3]) (afin probablement d'éviter les problèmes de comptabilité avec certains BIOS). Toutefois, cette configuration peut être modifiée afin de vérifier également la valeur des PCR[1] et [3] (voir partie 5.3).

Les ultimes mesures effectuées par le BIOS concernent le code et les données de l'IPL (*Initial Program Load*) du média démarré et sont stockées dans les PCR[4] et

[5]. Pour un disque dur ou tout média disposant d'un MBR (*i.e.* une clé USB), le code du MBR (octets 0 à 1B7h du MBR) est mesuré et stocké dans le PCR[4] et la table de partition (octets 1B8 à 1FFh du MBR) est mesurée et stockée dans le PCR[5]. Pour un lecteur de disquette, c'est l'intégralité du MBR (octets 0 à 1FFh) qui est mesurée et stockée dans le PCR[4].

Ces mesures, même si elles sont effectuées par le BIOS, sont, contrairement aux précédentes, vérifiables, car les données mesurées sont accessibles. Il est à ce titre intéressant de comparer la valeur effective des PCR (accessible grâce au code de la partie 9) avec leur valeur théorique.

Suite des mesures La suite des mesures effectuées sur la chaîne de démarrage (récapitulées dans le tableau 8), concerne tous les éléments liés à la partition marquée active dans la table de partition (la partition démarrage de la partie 2.1). Ces mesures sont systématiquement réalisées même si un module TPM n'est pas présent sur le système.

PCR	**Mesure**	**Mesuré par**
8	VBR (1er étage) : NTFS Boot Sector	MBR
9	VBR (1er étage et 2e étage) : NTFS Boot Sector et Boot Block	1er étage du VBR
10	bootmgr	2e étage du VBR

Table 8. Utilistation des registres PCR 8 à 10

La façon dont le MBR communique avec le TPM et l'analyse de la mesure du *NTFS Boot Sector* sont décrites dans le blog de Pascal Sauliere [13]. Pour résumer, l'interruption 1Ah permet la communication avec le module TPM et le MBR charge le premier secteur de démarrage de la partition active (Boot Sector), puis le mesure dans le PCR[8].

Ce travail d'analyse peut être effectué sur les éléments suivants de la chaîne de démarrage :

- le *NTFS Boot Sector* (situé tout au début de la partition) charge à son tour le NTFS Boot Block situé juste après le *Boot Sector*. Pour cela, 8192 octets sont chargés en mémoire à partir du disque ;
- le *NTFS Boot Sector* mesure le *Boot Block* dans le PRC[9]. Cependant, la mesure inclut également les 70 derniers octets du Boot Sector. L'élément mesuré est donc `BootSector[1B9-1FF]|BootBlock[0-FinCode]` ;

– l'exécution est transférée au *Boot Block* qui recherche sur le système NTFS le fichier `bootmgr` puis le charge en mémoire ;
– le `bootmgr` est mesuré dans le PCR[10]. Cependant, ce n'est pas le fichier lui-même qui est mesuré, mais son empreinte calculée par la fonction SHA-1.

À ce stage, le gestionnaire de démarrage `bootmgr` est chargé et a été mesuré. C'est maintenant de sa responsabilité de vérifier l'intégrité du système et de terminer le chargement de Windows.

4.2 Validation de la chaîne de démarrage

Lorsque BitLocker est activé sur un système, le processus initial de chargement décrit dans la partie 2.3 est identique jusqu'à l'étape de chargement de `winload.exe`, le chargeur du noyau. Lorsque BitLocker est activé, le fichier du chargeur se trouve alors sur un volume chiffré et `bootmgr` doit disposer de la VMK pour déverrouiller le volume. C'est à l'étape de récupération de la VMK que le contrôle d'intégrité du système est réalisé.

Sachant qu'il est possible d'avoir plusieurs méthodes de protection de la VMK (protecteurs), `bootmgr` privilégie toujours les méthodes basées sur l'utilisation d'un module TPM censées offrir plus de sécurité.

Dans le mode SRTM, BitLocker protège la VMK via le module TPM [11] et se sert des registres PCR pour mesurer l'intégrité des éléments de la chaîne de démarrage. Le principe est le suivant :

– lorsqu'une clé VMK est protégée via le module TPM, les registres PCR sont dans un état censé correspondre à un état de référence. Les valeurs actuelles [12] des PCR sélectionnés par le profil de validation sont scellées par le TPM conjointement avec la clé ce qui génère un bloc binaire ;
– lorsque `bootmgr` tente de récupérer une VMK d'un protecteur de type `TPM`, l'opération n'est autorisée par le TPM que si les valeurs actuelles des registres PCR sont identiques à celles contenues dans le bloc généré lors de la protection. Ceci garantit que l'état du système au moment de la déprotection est identique à l'état au moment de la protection.

En cas d'échec de la méthode basé sur le module TPM, `bootmgr` entre en mode de récupération et une autre méthode de récupération de VMK est tentée. Dans ce cas, l'intégralité de la chaîne de démarrage n'est plus vérifiée.

11. Ceci est simplifié : en réalité, il peut également s'agir d'une clé chiffrant la VMK qui est protégée (cf. partie 3.3).
12. Sauf la valeur correspondante au PCR[11] qui est explicitement fixé à zéro (voir chapitre 4.6).

4.3 Validation des métadonnées

Par nature, les métadonnées d'un volume ne peuvent pas être chiffrées car elles sont nécessaires à son déchiffrement. Étant en claire sur un disque, elles peuvent donc être accédées et éventuellement modifiées hors-ligne. Il est donc nécessaire d'en assurer l'intégrité. Pour cela, dans les métadonnées d'un volume, à la suite du bloc INFORMATION est présent un bloc de type `VALIDATION` dont le rôle est de valider l'intégrité du premier bloc. Celui-ci contient du *datum* de type AES256-CCM dont la clé est la VMK du volume. Ce *datum* chiffre un *datum* de type Key dont le contenu n'est pas une clé de chiffrement, mais l'empreinte SHA-2 du bloc `INFORMATION`.

```
[VALIDATION]
  [DATUM_AESCCM_ENC] - AES256CCM Decrypt with key: 4218b0...VMK DU VOLUME...94da86
    [DATUM_KEY]
      Size: 256bit
      Type: Hash
      Value: 5c34968320acea52cfd170df610fbe76ba10a8bde8f91f7c4c61b0c7bd7822a2
```

Pour valider les métadonnées et avant de déverrouiller un volume, le pilote BitLocker ou `bootmgr` récupèrent la VMK du volume puis déchiffre le *datum* contenu dans le bloc `VALIDATION` afin d'en extraire l'empreinte numérique. Ils calculent ensuite l'empreinte numérique SHA-2 du bloc `INFORMATION` et comparent les deux empreintes.

4.4 Validation de la BCD

Si le bloc `INFORMATION` est considéré comme intègre, la dernière validation concerne la base de configuration de démarrage (BCD, voir chapitre 2.2). Celle-ci étant stockée sur la partition de démarrage qui n'est pas chiffrée, le fichier de la base peut aisément être modifié par une édition hors-ligne et son intégrité doit être vérifiée.

Pour valider les éléments de la BCD, le bloc `INFORMATION` du volume du système d'exploitation contient un *datum* particulier (de type 7) qui est un tableau d'entrées dont la structure de données est la suivante :

```
typedef struct _VALIDATION_BCD
{
   USHORT usType;
   USHORT usDisable;
   ULONG  ulIdBcdElement;
   BYTE   bHash[32];
} VALIDATION_BCD, *PVALIDATION_BCD;
```

Les entrées de type 1 correspondent à des applications (dont l'empreinte numérique est donnée par le champ Hash) qui, si elles sont lancées par `bootmgr`, sont autorisées à disposer de la VMK du volume du système d'exploitation. Les entrées de type 2 correspondent à des éléments de configuration de la BCD associés à une application

(l'identifiant de l'élément est donné par le champ IdBcdElement et l'empreinte numérique SHA-2 de sa valeur est contenue dans le champ Hash).

Lorsqu'une application est sélectionnée pour être exécutée, `bootmgr` calcule l'empreinte de l'exécutable [13] puis recherche dans le tableau l'entrée correspondante (le type doit être de 1 et l'empreinte numérique doit correspondre). Les éléments de la BCD contrôlés et associés à cette application sont toutes les entrées de type 2 situées à la suite de l'entrée de l'application jusqu'à la prochaine entrée de type application ou la fin de la liste. Si des éléments sont présents dans le tableau et dans la BCD, les empreintes des valeurs des éléments de la BCD doivent correspondre aux empreintes du tableau afin de considérer la configuration de l'application comme intègre.

L'analyse d'un *datum* de validation de la BCD donne le résultat suivant :

```
[VALIDATION]
   1 / 0 / 00000000 /
       120d8eda85d8e2da0f7b5230957c45bd8da3c0ec000000000000000000000000
   2 / 0 / 11000001 /
       515108c20926960fe3bb777978d9f3c6d55280a8c31c74ef4efb41af1abd1c67...
   2 / 0 / 250000f5 /
       79158ec688ebf9c00a3986ec3b6361e34dbdca69ae75dca1bbc1bafe32a534cc
   1 / 0 / 00000000 /
       0be649fd894fce9e901420c858e5105241b7767e000000000000000000000000
   2 / 0 / 11000001 /
       515108c20926960fe3bb777978d9f3c6d55280a8c31c74ef4efb41af1abd1c67...
   2 / 0 / 16000020 /
       47dc540c94ceb704a23875c11273e16bb0b8a87aed84de911f2133568115f254
   1 / 0 / 00000000 /
       6f38c9cd52009860d5e8eda9f3bbe714c40a533300000000000000000000000
   2 / 0 / 11000001 /
       b0b59651bf77b43e72a1bc45081986dcc811491a4d4242c90ff3211936563a70...
   2 / 0 / 16000020 /
       47dc540c94ceb704a23875c11273e16bb0b8a87aed84de911f2133568115f254

Légende: Type / Disable / IdBcdElement / Hash
```

Lorsque BitLocker est activé pour le volume du système d'exploitation, la validation porte sur les trois applications installées par défaut (winload.exe, winresume.exe et `memtest.exe`) ainsi que pour leurs éléments associés (à l'exception quelques éléments particuliers par exemple l'élément Description).

Lorsque la configuration d'une application doit être légitiment modifiée (par exemple lors de l'activation du débogueur noyau), il est nécessaire de supprimer le protecteur de type `TPM` puis de le recréer afin de régénérer un *datum* de validation correct.

13. Il ne s'agit pas de l'empreinte SHA-2, mais SHA-1, ce qui explique les 12 octets nuls de bourrage à la fin.

4.5 Transfert de l'exécution et de la VMK

Si les diverses validations sont réussies, `bootmgr` transfère l'exécution à l'application. Pour cela, il déverrouille le volume du système d'exploitation en déchiffrant la FVEK à l'aide de la VMK. À partir de cet instant, il lui est possible d'accéder aux parties chiffrées du volume. Pour le démarrage d'un système d'exploitation, `bootmgr` charge l'application (le chargeur de noyau `winload.exe`), le noyau (`ntoskrnl.exe`) ainsi que les pilotes de démarrage (dont le pilote BitLocker `fvevol.sys`). Afin que ce dernier soit en mesure de procéder aux opérations de déchiffrement à la volée, `bootmgr` doit lui transmettre la VMK. Pour ce faire, une zone de mémoire physique baptisée *Key Ring* est allouée et `bootmgr` y dépose la VMK du volume. L'adresse physique de cette zone est transmise au pilote via un paramètre noyau [14].

Lors de son initialisation, le pilote BitLocker récupère dans le *Key Ring* la VMK et procède à son tour au déchiffrement de la FVEK. Il met alors en place les opérations de chiffrement et déchiffrement à la volée permettant ainsi au noyau et à tout le système d'accéder de manière transparente au volume chiffré.

4.6 Blocage du TPM

Dans tous les cas, avant de se terminer ou de transférer l'exécution à un programme tiers, `bootmgr` ¡¡ bloque le TPM ¿¿. Cette opération consiste à écrire une valeur arbitraire dans le PCR[11]. Ainsi, tout système ou application exécuté après `bootmgr` sera dans l'incapacité d'utiliser le module TPM pour récupérer une VMK d'un protecteur de type `TPM`.

En effet, un protecteur de ce type, lorsqu'il est généré, spécifie que le PCR[11] doit être à sa valeur par défaut (soit que des zéros) pour autoriser la déprotection de la clé par le module TPM. Or si une valeur arbitraire est écrite dans le registre, il ne sera plus possible de revenir à la valeur par défaut et l'opération de déproctection ne sera pas autorisée (tableau 9).

PCR	Mesure	Mesuré par
11	Blocage du TPM	bootmgr

Table 9. Utilisation du registre PCR 11

14. Ce paramètre est visible dans l'attribut `SystemStartOptions` de la clé `HKLM\SYSTEM\CurrentControlSet\Control`.

5 Administration de BitLocker

5.1 Configuration à distance via WMI

Un élément important de l'administration de BitLocker est l'existence d'un fournisseur WMI (`Win32_EncryptableVolume`) qui permet de réaliser toutes les opérations liées à BitLocker (activation du chiffrement, déverrouillage d'un volume, gestion des protecteurs, etc.) sur un volume.

L'extrait de script VBS ci-dessous montre un exemple de récupération de l'état de tous les volumes d'un système et énumère des protecteurs associés :

```
Set colItems = objWMIService.ExecQuery("SELECT * FROM Win32_EncryptableVolume",
     , 48)
For Each objItem in colItems
   objItem.GetEncryptionMethod EncryptionMethod
   objItem.GetProtectionStatus ProtectionStatus
   objItem.GetLockStatus LockStatus

   Wscript.Echo "EncryptionMethod: " & EncryptionMethod
   Wscript.Echo "ProtectionStatus: " & ProtectionStatus
   Wscript.Echo "LockStatus: " & LockStatus

   objItem.GetKeyProtectors 0, VolumeProtectorID
   For Each protectorId in VolumeProtectorID
      objItem.GetKeyProtectorType objId, VolumeKeyProtectorType

      Wscript.Echo "   Protector Guid: " & protectorId
      Wscript.Echo "   Protector Type: " & VolumeKeyProtectorType
   Next
Next
```

Évidemment, il est nécessaire de s'authentifier après du fournisseur avec des droits adéquats. Dans l'espace de nom WMI, le fournisseur est situé à l'emplacement `root\CIMv2\Security\MicrosoftVolumeEncryption` où le descripteur de sécurité de l'emplacement n'autorise que les membres du groupe Administrateurs à effectuer des requêtes.

Un grand intérêt de l'utilisation de WMI est la possibilité de pouvoir effectuer les requêtes sur un système distant. Ainsi, toute l'administration de BitLocker est possible via le réseau. Cependant, si le fournisseur WMI est appelé à distance, il impose le chiffrement des échanges réseau.

Il faut également noter que les deux autres composants liés à BitLocker, le module TPM et la base BCD disposent également de fournisseurs WMI.

5.2 Outils d'administration

Outre les diverses opérations réalisables au moyen des interfaces graphiques, Microsoft propose des outils afin de gérer plus finement la configuration de BitLocker.

Le plus important est `manage-bde.exe` (précédemment un script VBS sous Vista) qui offre la possibilité de réaliser les principales opérations en ligne de commande. Basé sur WMI, `manage-bde` peut également effectuer toutes les opérations demandées sur un système distant. L'outil prend également en charge quelques opérations basiques sur le module TPM.

Deux autres outils sont désormais intégrés de base dans Windows 7 :

- `bdehdcfg.exe` qui prépare un système pour BitLocker. Il s'agit principalement de créer, si elle n'existe pas, la partition de démarrage et de s'assurer que le gestionnaire de démarrage et la BCD y soient présents ;
- `repair-bde.exe` qui tente de récupérer un volume où BitLocker est corrompue. L'opération consiste alors à rechercher toutes les structures en clair de métadonnées sur le volume et de s'assurer de leur cohérence.

Enfin, en mars 2011, Microsoft a sorti la version bêta de l'outil de gestion MBAM (Microsoft BitLocker Administration and Monitoring) qui permet de simplifier le déploiement de l'administration de BitLocker au sein d'un parc informatique.

5.3 Stratégie de groupe

Les paramètres de configuration de BitLocker et du module TPM sont accessibles au travers d'une stratégie de groupe (Ordinateur / Modèles d'administration / Composants Windows / Chiffrement de lecteur BitLocker pour le premier et Ordinateur / Modèles d'administration / Système / Services de module de plateforme sécurisé pour le second), ce qui les rend déployables dans un Active Directory.

Concernant la configuration de BitLocker, les principaux paramètres portent sur :

- la mise en place de stratégie de récupération (imposer l'utilisation d'une méthode de récupération, sauvegarde des informations de récupération dans l'Active Directory) ;
- le choix de divers paramètres de sécurité (algorithme de chiffrement, profil de validation des registres PCR, utilisation des codes confidentiels améliorés, utilisation de carte à puce) ;
- le type de protecteurs autorisés à être utilisés (avec ou sans utilisation d'un module TPM) ;
- la mise en place d'une politique pour le choix des codes PIN ou des mots de passe.

6 Analyse de la sécurité de BitLocker

On peut considérer deux grands types de menaces sur un ordinateur dont le disque dur est chiffré.

Le premier type de menaces concerne les cas où un attaquant à accès à l'ordinateur dans un état éteint [15]. Deux scénarios d'attaques sont alors possibles :

- les attaques ¡¡ hors-ligne un temps ¿¿ où un attaquant n'a accès à l'ordinateur éteint qu'une seule fois ;
- les attaques ¡¡ hors-ligne deux temps ¿¿ où un attaquant a accès une première fois à l'ordinateur éteint et peut, par la suite, soit avoir accès à nouveau à l'ordinateur (toujours éteint), soit récupérer des données à distance en ayant mis en place un dispositif de fuite sur l'ordinateur lors du premier accès.

Le deuxième type de menaces concerne les cas où l'attaquant accède au système alors que celui-ci est en cours d'exécution ou en veille S3 et que ses volumes chiffrés sont déverrouillés. Dans ce cas, l'attaquant peut soit avoir un accès physique à l'ordinateur, soit communiquer à distance via le réseau. Les attaques effectuées sont alors ¡¡ en ligne ¿¿.

6.1 Attaques hors-ligne un temps

Il s'agit typiquement de l'évènement redouté de la perte ou du vol d'un ordinateur où un attaquant va tenter d'accéder aux données du disque. Microsoft, dans ses présentations, a toujours indiqué que BitLocker n'offrait une protection que contre ce type d'attaque.

Les volumes protégés par BitLocker ne sont, dans ce cas, théoriquement pas accessibles, car les VMK sont chiffrées soit par des clés de 128 bits qui doivent être étirées (Mot de passe numérique et Mot de passe numérique), soit par des clés de 256 bits (`TPMAndStartupKey`, `TPMAndPINAndStartupKey` et `Clé externe`). Évidemment, si la protection BitLocker est suspendue sur un volume (voir partie 3.4), la VMK peut être récupérée directement et toutes les données du volume peuvent être récupérées.

Lorsque les protecteurs de type `TPM` ou `TPMAndPIN` sont mis en œuvre, la robustesse de la protection des VMK (donc des données du disque) repose uniquement sur la robustesse offerte par le module TPM. De plus, lorsque le mode TPM est utilisé, le système d'exploitation peut être démarré et les attaques en ligne peuvent alors être effectuées. Il est important de noter qu'il s'agit de la méthode de protection par défaut.

6.2 Attaques hors-ligne deux temps

Ce type de menace est typiquement celui de la femme de ménage, scénario baptisé également ¡¡ *evil maid* ¿¿ où un utilisateur va laisser pendant un certain temps son

15. Ce type ne couvre pas un portable en veille S3 (suspension en mémoire).

ordinateur éteint sans surveillance. L'attaquant va alors installer un dispositif dont le but sera de récupérer tous les éléments permettant d'accéder a posteriori à la VMK. Il s'agit du type d'attaque sur BitLocker qui a été le plus médiatisée.

Le scénario repose sur l'installation d'un faux système lancé en lieu et place du système légitime. Cette modification est possible, car les éléments de la BCD qui sont vérifiés sont ceux de l'application lancée (le système légitime), mais pas ceux du gestionnaire de démarrage (`bootmgr`). Il est alors possible d'installer dans la partition de démarrage (qui n'est jamais chiffrée) une image ISO d'un faux système puis de modifier la configuration du `bootmgr` (éléments *DisplayOrder* et *DefaultObject*) dans la BCD (qui n'est pas chiffrée) pour démarrer sur ce nouveau système à la place du système légitime.

Lors du démarrage de la machine, le faux système présente alors des interfaces identiques à celle du système légitime demandant la saisie des éléments nécessaires à la déprotection de la VMK (code PIN, clé externe, mot de passe numérique). L'attaquant doit ensuite les récupérer soit par un second accès à l'ordinateur, soit par mécanisme de fuite par la carte réseau filaire ou Wifi opéré par le faux système.

Cependant, si un protecteur basé sur le module TPM est utilisé, ce type d'attaque est détectable. En effet, les protecteurs utilisant un module TPM sont toujours utilisés en priorité. Dans ce cas, lorsque `bootmgr` va transmettre l'exécution au faux système, il va bloquer ce TPM rendant impossible la déprotection de la VMK par la suite (voir partie 4.6). Le faux système n'aura alors pas d'autre choix que de redémarrer le système, ce qui constitue un élément certes (faiblement) détectable.

Une vidéo de démonstration de ce type d'attaque effectuée par des membres du SIT est disponible à [22].

6.3 Attaques en ligne

Les attaques sur un système en ligne sont de natures complètement différentes. En effet, lorsque le système est en cours s'exécution, cela impose la présence en mémoire d'au moins la FVEK de chaque volume déverrouillé. En pratique la FVEK ainsi que la VMK sont présentes dans la mémoire du noyau. Les attaques en ligne vont donc essentiellement chercher à récupérer les clés liées à BitLocker.

Récupération des clés à distance Lorsqu'un système en ligne est connecté à un réseau, il offre une exposition particulièrement importante. Si un attaquant dispose d'un compte membre du groupe Administrateurs, il peut par WMI où par exécution de commande à distance (de type psexec ou winexec) très facilement récupérer la VMK ou les secrets des protecteurs (mot de passe numérique, clé externe).

Clé en mémoire Si l'attaquant a à sa disposition un système en cours de fonctionnement (et évidemment verrouillé par un mot de passe et un économiseur d'écran), certains types d'attaques portent sur les bus de communication. Des présentations aux précédentes éditions de SSTIC ont montré qu'à partir de bus FireWire ou autre, il était possible d'interagir avec la mémoire d'un système [23]. À ce titre, Microsoft recommande d'ailleurs de désactiver dans certains cas le pilote du contrôleur 1394 lorsque BitLocker est utilisé [16].

Enfin, restent les attaques de type *cold boot* qui tentent, par des méthodes plus ou moins contraignantes, d'accéder à la mémoire d'un système après son arrêt dans le but d'y retrouver des éléments secrets.

Toutes ces attaques démontrent qu'il est fortement recommandé de ne jamais laisser sans surveillance un ordinateur où BitLocker est activé lorsque celui-ci est sous tension.

6.4 Activation du Débogueur

Évidemment, si le ¡¡ mode debug ¿¿ d'un système Windows est activé, il devient possible d'accéder à la mémoire du noyau et d'y retrouver les FVEK et VMK des volumes déverrouillés. Il est cependant impossible d'activer illégitimement le mode debug (via par exemple la commande `bcedit /debug {current} on`). En effet, cela entraîne la modification de l'élément `KernelDebuggerEnabled` dans la BCD pour l'application du chargeur de Windows. Les données de validations doivent alors être recalculées (voir chapitre 4.4) sinon l'intégrité de la BCD ne sera plus valide.

En revanche, il est possible d'activer le mode debug du gestionnaire de démarrage (via la commande `bcdedit /bootdebug {bootmgr}on`). Le paramètre concerne alors un élément de l'objet du gestionnaire de démarrage `{bootmgr}` qui n'est pas une application vérifiée : la modification ne peut être détectée. Afin de protéger contre un tel scénario, au démarrage de `bootmgr` et avant la mise en place du canal de communication avec un débogueur, `bootmgr` bloque le TPM afin d'empêcher toute utilisation non conforme si le débogueur est activé (ce comportement est d'ailleurs identique lorsque les touches F8 et F10 sont utilisées). Il est alors obligatoire d'utiliser un mécanisme de recouvrement.

7 Conclusion

Un point non abordé concernant BitLocker dans cet article est sa grande fiabilité et sa parfaite intégration au sein de Windows. Les cas de perte de données non liés à des pertes de clés de chiffrement ou de recouvrement sont extrêmement rares. Pour le

reste, BitLocker est un classique produit de chiffrement de disque, qui, comme tout produit de sécurité, doit être correctement utilisé et configuré afin de fournir une sécurité efficace.

Cependant, il ne faut pas oublier que le premier objectif de BitLocker est d'apporter une solution contre les risques liés à la perte d'ordinateur.

Enfin, pour terminer, on peut proposer quelles recommandations sur l'utilisation de BitLocker :

- chiffrer tous les volumes d'un disque (volume du système d'exploitation et tous les volumes de données supplémentaires) et s'assurer que BitLocker est activé sur tous les volumes ;
- préférer le mode de chiffrement AES256 + diffuseur ;
- préférer la méthode de recouvrement de type clé externe à celle du mot de passe numérique ;
- si un module TPM est présent au sein du système, l'utiliser et choisir un mode de protection avec clé externe (`TPMAndStartupKey` ou `TPMAndPINAndStartupKey`) ;
- activer l'utilisation des codes confidentiels améliorés ;
- ne pas laisser un système allumé sans surveillance, y compris en mode S3 de veille ;
- désactiver la mise en veille prolongée sur le système (`powercfg -h off`). Si celle-ci doit être utilisée, le fichier d'hibernation doit être stocké sur un volume chiffré ;
- dans le profil de validation du TPM, utiliser au minimum les PCR par défaut (0, 2, 4, 5, 8, 9, 10 et 11) et, si possible, les PCR[1] et [3] ;
- en cas d'anomalie lors d'un démarrage (redémarrage anormal), considérer que le système est compromis ;
- en cas de compromission d'un système, déchiffrer puis chiffrer à nouveau tous les volumes afin de renouveler les FVEK ;
- limiter au maximum dans le BIOS, les périphériques de démarrage, et tous les contrôleurs ayant des ports externes accessibles (1394, PCMCIA, CardBus) ;
- activer, si elles sont présentes, d'autres protections de démarrage sur le système (mot de passe disque, mot de passe sur le BIOS) ;
- en cas de système connecté à un réseau, s'assurer qu'il n'est pas possible d'y accéder avec des droits d'administrateur et qu'UAC est activé à son niveau maximal. Utiliser si nécessaire un pare-feu correctement configuré.

Remerciements Merci à Loïc Duflot pour sa relecture et ses conseils avisés tout au long de l'étude ainsi qu'à Victor Vuillard pour sa relecture.

Merci (surtout) à steck qui a assuré, avec plein de malice, le caoutchoutage de l'article.

Références

1. Boot Configuration Data in Windows Vista. http://msdn.microsoft.com/en-us/windows/hardware/gg463059
2. Boot Configuration Data WMI Provider. http://msdn.microsoft.com/en-us/library/bb986746(VS.85).aspx
3. BCDEdit Commands for Boot Environment. http://msdn.microsoft.com/en-us/windows/hardware/gg463064.aspx
4. BcdBootMgrElementTypes Enumeration. http://msdn.microsoft.com/en-us/library/aa362641(VS.85).aspx
5. File System Filter Drivers. http://msdn.microsoft.com/en-us/windows/hardware/gg462968
6. File System Filter Drivers. http://msdn.microsoft.com/en-us/library/ff540382(VS.85).aspx
7. File System Minifilter Drivers. http://msdn.microsoft.com/en-us/library/ff540402(VS.85).aspx
8. Load Order Groups for File System Filter Drivers. http://msdn.microsoft.com/en-us/library/ff549694(VS.85).aspx
9. Implementing BitLocker Drive Encryption for Forensic Analysis. http://jessekornblum.com/publications/di09.html
10. BitLocker Drive Encryption. http://technet.microsoft.com/en-us/library/cc731549(WS.10).aspx
11. TPM Base Services. http://msdn.microsoft.com/en-us/library/aa446796(VS.85).aspx
12. Trusted Platform Module Provider. http://msdn.microsoft.com/en-us/library/aa376480(VS.85).aspx
13. Bitlocker : analyse du MBR de Windows Vista. http://blogs.technet.com/b/pascals/archive/2007/11/27/bitlocker-analyse-du-mbr-de-windows-vista.aspx
14. Defining I/O Control Codes. http://msdn.microsoft.com/en-us/library/ff543023(VS.85).aspx
15. BitLocker recovery password details. http://blogs.msdn.com/b/si_team/archive/2006/08/10/694692.aspx
16. Bloquer le pilote SBP-2 afin de réduire les menaces DMA 1394 pour BitLocker. http://support.microsoft.com/kb/2516445
17. Addressing a Commercial Grade Operating System Security Functional Requirement Set with Windows Vista and Server 2008. http://download.microsoft.com/download/D/7/1/D7158253-CE22-4CB3-B622-E3460AB2B9B1/CommercialOSSecFunReqsPublic.docx
18. AES-CBC + Elephant diffuser - A Disk Encryption Algorithm for Windows Vista. http://www.microsoft.com/downloads/en/details.aspx?familyid=131dae03-39ae-48be-a8d6-8b0034c92555
19. Bitlocker and Windows Vista. http://www.nvlabs.in/archives/1-NVbit-Accessing-Bitlocker-volumes-from-linux.html
20. BitLocker : plongée en eaux profondes. TechDays. http://download.microsoft.com/download/9/5/1/951c5d52-020b-4cda-b000-411864b3e4f2/Jour1-252B-3-BitLocker.pptx
21. Black Hat TPM Hack and BitLocker. http://windowsteamblog.com/windows/b/windowssecurity/archive/2010/02/10/black-hat-tpm-hack-and-bitlocker.aspx
22. Attacking the Bitlocker Boot Process. http://testlab.sit.fraunhofer.de/content/output/project_results/bitlocker_skimming/bitlockervideo.php?s=2

23. Compromission physique par le bus PCI. http://www.sstic.org/2009/presentation/Compromission_physique_par_le_bus_PCI/

24. Trusted Computing : Limitations actuelles et perspectives. http://www.sstic.org/2010/presentation/Trusted_Computing_Limitations_actuelles_et_perspectives/

8 Annexe 1 - Code de déchiffrement AES-CCM

```
// Fonction incomplete de dechiffrement d'un datum DATUM_AES256_CCM
// La cle est specifiee via le parametre pKey et doit faire 256 bits (32 octets
    )
// Le parametre dwEncryptedDataSize indique la taille des donnees chiffres dans
     le datum pDatumAES

PBYTE DecryptDatumAES256CCM(PBYTE pKey, PDATUM_AES256_CCM pDatumAES, DWORD
    dwEncryptedDataSize)
{
   BCRYPT_ALG_HANDLE hAesAlg;
   BCRYPT_KEY_HANDLE hKey;
   BCRYPT_AUTHENTICATED_CIPHER_MODE_INFO AuthCipher;

   DWORD dwPlainTextSize, dwKeyObjectSize, dwDataSize;
   PBYTE pbPlainText, pbKeyObject;

   BCryptOpenAlgorithmProvider(&hAesAlg, BCRYPT_AES_ALGORITHM, NULL, 0);
   BCryptGetProperty(hAesAlg, BCRYPT_OBJECT_LENGTH, (PBYTE)&dwKeyObjectSize,
       sizeof(DWORD), &dwDataSize, 0);
   pbKeyObject = (PBYTE)HeapAlloc(hHeap, HEAP_ZERO_MEMORY, dwKeyObjectSize);
   BCryptSetProperty(hAesAlg, BCRYPT_CHAINING_MODE, (PBYTE)
       BCRYPT_CHAIN_MODE_CCM, sizeof(BCRYPT_CHAIN_MODE_CCM), 0);
   BCryptGenerateSymmetricKey(hAesAlg, &hKey, pbKeyObject, dwKeyObjectSize,
       pKey, 32, 0);

   BCRYPT_INIT_AUTH_MODE_INFO(AuthCipher);
   AuthCipher.pbNonce = (PUCHAR)&pDatumAES->ftFileTime;
   AuthCipher.pbTag = (PUCHAR)&pDatumAES->bMac;
   AuthCipher.cbNonce = 12;
   AuthCipher.cbTag = 16;

   // Premier appel pour avoir la taille du clair
   BCryptDecrypt(hKey, pDatumAES-> bEncryptedData, dwEncryptedDataSize, &
       AuthCipher, NULL, 0 ,NULL, 0, &dwPlainTextSize, 0);
   pbPlainText = (PBYTE)HeapAlloc(hHeap, 0, dwPlainTextSize);
   // Second appel pour le dechiffrement
   BCryptDecrypt(hKey, pDatumAES-> bEncryptedData, dwEncryptedDataSize, &
       AuthCipher, NULL, 0,   pbPlainText, dwPlainTextSize, &dwPlainTextSize,
       0);

   // Liberation des ressources a realiser
   // via BCryptDestroyKey BCryptCloseAlgorithmProvider et HeapFree

   return pbPlainText;
}
```

9 Annexe 2 - Code (partiel) de lecture des registres PCR

```
#include <windows.h>
#include <stdio.h>
#include <tbs.h>
#include <tchar.h>
#include "tpm.h"

#define BUFFER_SIZE    512

// Note : les structures (TPM_PCRRead_In) et constantes (TPM_TAG)
// doivent etre definies conformement a la norme TGS
#define TPM_TAG_RQU_COMMAND      0x00C1

VOID TPMReadPcr(TBS_HCONTEXT hContext)
{
   TBS_RESULT rv;

   TPM_PCRRead_In   Blob_TPM_PCRRead_In;
   PTPM_PCRRead_Out pBlob_TPM_PCRRead_Out;
   BYTE bInputSize;

   BYTE buf[BUFFER_SIZE];
   UINT32 buf_len = BUFFER_SIZE;

   bInputSize = sizeof(Blob_TPM_PCRRead_In);

   for (int i=0; i<16; i++)
   {
      memset(&Blob_TPM_PCRRead_In, 0, bInputSize);
      Blob_TPM_PCRRead_In.tag = htons(TPM_TAG_RQU_COMMAND);
      Blob_TPM_PCRRead_In.paramSize = htonl(bInputSize);
      Blob_TPM_PCRRead_In.ordinal = htonl(TPM_TAG_RQU_COMMAND);
      Blob_TPM_PCRRead_In.pcrIndex = htonl(i);

      buf_len = BUFFER_SIZE;
      rv = Tbsip_Submit_Command(hContext, TBS_COMMAND_LOCALITY_ZERO, \
         TBS_COMMAND_PRIORITY_NORMAL, (PBYTE)&Blob_TPM_PCRRead_In, bInputSize,
             \^
         buf, &buf_len);
      pBlob_TPM_PCRRead_Out = (PTPM_PCRRead_Out)buf;

      printf("PCR: %d, Result: 0x%08x, Value: ", i, pBlob_TPM_PCRRead_Out->u.
          returnCode);
      PrintHex((PBYTE)&pBlob_TPM_PCRRead_Out->outDigest, digestSize);
      printf("\r\n");
   }
}

int main(int argc, char *argv[])
{
   TBS_CONTEXT_PARAMS pContextParams;
   TBS_HCONTEXT hContext;
   TBS_RESULT rv;

   pContextParams.version = TBS_CONTEXT_VERSION_ONE;
   rv = Tbsi_Context_Create(&pContextParams, &hContext);
```

```
        TPMReadPcr(hContext);

        rv = Tbsip_Context_Close(hContext);

        return 0;
    }
```