0-Days - Recherche et Exploitation de Vulnérabilités en Environnement Win32

Kostya Kortchinsky

Responsable du CERT RENATER
kostya.kortchinsky@renater.fr

Résumé Si on devait établir une échelle de valeur des outils liés à la sécurité des systèmes d'information, les exploits dits « 0day » auraient sans nul doute une place de choix. Ce terme anglais (dont la traduction littérale est 0jour), désigne une catégorie particulière d'outils d'exploitation de vulnérabilités : ceux qui permettent de compromettre des systèmes à jour. Plusieurs raisons possibles à cela : l'éditeur est au courant de la faille mais n'a pas encore publié de correctif ; l'éditeur n'est pas au courant de la faille, cette dernière n'étant partagée que par un nombre restreint de personnes. Toujours est-il que dans une telle situation, les systèmes et réseaux comprenant le composant vulnérable sont exposés. À cause de cela, les 0days sont très recherchés, aussi bien par les pirates que par les professionnels de la sécurité, qui y trouvent un moyen efficace de réaliser avec succès un test d'intrusion. Mais comment mettre la main sur ces exploits si particuliers ? C'est évidemment là que réside l'essence du problème. Certains envisagent les pots de miel comme une solution, en attendant de capturer les paquets d'un exploit pour une vulnérabilité non répertoriée ; mais le taux de réussite est relativement faible. D'autres tentent de les échanger, contre des ressources diverses, de l'argent (oui, ça s'appelle les payer), d'autres 0days (le serpent se mord la queue). De mon point de vue - qui rejoint je pense celui de bon nombre d'autres - la meilleure façon d'avoir un 0day reste encore de chercher des failles soi-même.

1 Introduction

Parmi la cohorte de dangers potentiels susceptibles de menacer aujourd'hui nos systèmes d'information et de communication, un peut inspirer une crainte particulière : les programmes d'exploitation de vulnérabilités (par la suite nommés « exploits ») dits 0day. Ce néologisme anglais, dont la traduction littérale serait « 0jour », désigne la catégorie très spécifique d'exploits ciblant des failles qui ne sont pas encore connues de l'éditeur ou constructeur en cause, et par conséquent ne disposant pas de correctif approprié. La notion peut s'étendre aux vulnérabilités connues du public et non corrigées, ou encore aux vulnérabilités connues et corrigées mais pour lesquelles il n'existe pas encore d'exploit public.

Est-il pour autant nécessaire de redouter davantage ces fameux 0days que les toutes dernières versions des virus de messagerie, ou la fabuleuse habileté des utilisateurs à télécharger et exécuter tout ce qu'ils trouvent sur la toile ? Rien

n'est moins sûr, chaque cas de figure ayant des implications différentes pour un résultat globalement semblable : la compromission d'un système. Au sein du clan très fermé des 0days, une sous-famille se démarque cependant : ceux visant des services en écoute sur le réseau, et dont l'exploitation ne nécessite pas d'authentification. Permettant l'exécution de code à distance sur une machine sans interaction d'un utilisateur local, leur intérêt ne vous échappera sans doute pas.

Maintenant que nous avons explicité notre champ d'étude, il est grand temps de s'intéresser aux moyens d'obtenir ces objets de tant de convoitises. Il est bien entendu relativement rare que le hasard présente à votre vue un de ces précieux programmes au détour d'un chemin de l'autoroute de l'information. Vous devrez forcer la main à la chance. La façon la plus intuitive d'entrer en possession d'un 0day est de trouver la vulnérabilité soi-même, puis de développer l'exploit adapté. Cependant, cela n'est pas à la portée de tous.

2 Acquérir un 0day

Cette thématique n'est pas vraiment au coeur du sujet de cet article, ainsi je me contenterai de passer rapidement en revue certaines possibilités offertes à toute personne un minimum entreprenante ayant cet objectif. Cette liste ne se veut pas exhaustive.

2.1 La Monnaie

Il n'y a rien que l'argent ne puisse acheter. Cet adage trouvera une justification supplémentaire dans le contexte qui nous intéresse. Même s'il n'existe pas en France (du moins publiquement) de marché financier pour les vulnérabilités, c'est tout l'inverse chez nos amis américains, puisqu'elles assurent à certains fortune et gloire. Au sein d'entreprises de sécurité des systèmes d'informations [VSC], des Vulnerabilities Sharing Clubs (VSC) voient le jour. Dans ces associations aux droits d'entrée très élevés circulent des renseignements concernant les vulnérabilités privées les plus récentes, les tous derniers exploits ; le profile type d'un adhérent étant la grosse entreprise, ou l'agence gouvernementale, américaine.

Des canaux alternatifs moins officiels, et probablement à la légalité douteuse, existent certainement, libre à chacun de tenter de les explorer.

2.2 Les Pots de Miel et les Investigations Forensiques

L'un des objectifs avoués, parmi tant d'autres, de la mise en place de ces fameux honeypots est la capture d'attaques inconnues, une analyse minutieuse de ces dernières pouvant aboutir à la découverte d'une vulnérabité non répertoriée. La théorie est alléchante, la pratique l'est beaucoup moins. En plusieurs années d'exploitation de pots de miel à haute interactivité, tout ce que j'ai pu attraper

étaient de vieilles mouches malades, comprenez des vers / virus courants ou des paquets issus d'exploits publiques.

Même s'ils restent de bons moyens de surveiller l'activité de pirates (de moindre importance) in-vivo, il vous faudra opter pour une alternative si vous désirez obtenir quelque chose de plus excitant.

Les investigations forensiques menées sur des machines de production compromises peuvent permettre de recueillir des éléments plus précieux. Leurs activités quotidiennes sont susceptibles d'en faire des cibles idéales à des attaques délimitées, se fondant sur l'utilisation d'exploits spécifiques.

2.3 Les Échanges

Le troc est aussi d'usage dans les milieux sombres d'Internet. Les valeurs sûres de ce marché parallèle ont des cotes relativement fixes, on y trouvera par exemple : le réseau de machines compromises, la liste de numéros de cartes bancaires, les outils d'exploitation de vulnérabilités - le 0day occupant le haut du panier. Tout cela s'échange via des forums plus ou moins privés [ZONEH]. Mais bien entendu il vous faudra verser dans l'illégalité pour obtenir un programme dont l'efficacité sera toute relative, ou déjà avoir en votre possession un 0day pour arriver à vos fins. Dans cette éventualité, il serait dangereux de le confier à une personne inconnue.

Bref, cette voie, comme les autres se révèlera difficile à emprunter pour une personne aux moyens - financiers - limités et soucieuse de rester en conformité avec la morale et les textes de loi.

3 La Recherche de Vulnérabilités

La méthode la plus éprouvée et la plus sure pour se procurer un 0day reste encore de le concevoir soi-même, en menant à bien les différentes étapes de son développement. Dans le cas où l'on se place dans la définition « stricte » du 0day, cela inclut la recherche d'une vulnérabilité non répertoriée, ainsi que le développement d'un outil permettant d'exploiter cette vulnérabilité. Dans un contexte plus large, il s'agit d'identifier une vulnérabilité connue mais non détaillée ou simplement d'implémenter un exploit pour une vulnérabilité connue.

En environnement Windows, la recherche de vulnérabilités est considérée comme une entreprise fastidieuse et demandant un niveau de compétences élevé. Les raisons en sont simples :

- la taille colossale des sources du système d'exploitation et donc des binaires et de leurs fonctionnalités ;
- l'impossibilité pour tout un chacun d'y avoir accès ;
- malgré la grande quantité de documentation disponible, bien des parties du système restent obscures et spécifiques à Microsoft.

La simple connaissance de langages tels que C ou C++ est un prérequis bien vain dans ce cadre, et il ne vous restera comme options que :

- l'analyse : traiter Windows comme une boîte noire et observer ses réactions face à des situations de stress ;
- la rétro-conception : désassembler les binaires de Windows et rechercher des vulnérabilités dans le code assembleur généré, en s'aidant au besoin d'un débogueur pour suivre le fonctionnement du système en temps réel.

Passons en revue les moyens qui s'offrent à vous pour trouver des failles dans Windows.

3.1 Les Essais Manuels

Il s'agit d'utiliser les clients standards intégrés au système d'exploitation ou leurs équivalents ouverts afin de vérifier le comportement de la partie étudiée face à toutes sortes de paramètres, valides ou non. L'objectif principal de ces essais est bien évidemment de détecter une anomalie survenant suite à une requête particulière. Ceci ne constitue cependant que le point de départ de ce type d'analyse, et il est nécessaire de pousser l'observation à l'examen des protocoles réseaux et des aspects de synchronisation.

Il n'est pas rare qu'une application cliente ou une API quelconque effectue un prétraitement des données soumises pour les assainir avant de les transférer au composant ciblé. Lorsque l'on s'attaque à un service réseau, il est indispensable de faire usage d'un outil de capture de paquets [ETHEREAL] afin d'effectuer une corrélation entre les données soumises par l'utilisateur et celles transmises au service.

A partir des données ainsi collectées, il est possible de reconstituer des sessions et selon un principe de « replay », de réinjecter les paquets en tentant de modifier certaines données jugées pertinentes, ou de contourner les modifications apportées par la partie cliente standard. Il est alors aussi pensable de jouer sur la synchronisation de l'envoi des requêtes pour déterminer si le serveur gère les situations de charge extrême.

Ce type de recherche permet de devenir rapidement familier avec le binaire observé sans nécessiter d'outils ou de compétences extraordinaires, et de découvrir des failles simples et immédiates - de plus en plus rares de nos jours dans des applicatifs de renom. Néanmois c'est un procédé laborieux et très limité car il ne couvre que la partie visible de l'iceberg.

3.2 Le Fuzzing

Le fuzzing étend le concept des essais manuels en y introduisant de l'automatisation. En développant des clients spécifiques pouvant générer de larges éventails de requêtes invalides, on pousse le composant ciblé dans ses retranchements - bien au delà des possibilités offertes par le client originel. Deux possibilités s'offrent au testeur :

- opter pour une version semi automatisée de l'outil, permettant une analyse plus fine des résultats, mais nécessitant des modifications fréquentes ;
- opter pour une version totalement automatisée de l'outil qui ne s'arrêtera qu'avec l'arrêt brutal du serveur.

Le fuzzing présente l'avantage conséquent de minimiser toute intervention humaine dans le procédé une fois qu'il a été initié. Mais attention, la conception et le développement d'un fuzzer convenable implique une compréhension avancée des fonctions et protocoles mis en oeuvre, et l'automatisation de la méthode est loin d'être simple.

Une personne un tant soit peu motivée n'aura cependant pas à réinventer la roue, car des bibliothèques offrant un cadre de développement à cette fin existent [SPIKE] [PEACH]. La base qu'elles apportent ne vous prémunira pas contre de laborieuses recherches sur des aspects souvent peu documentés des systèmes d'exploitation Microsoft.

Un autre élément est à prendre en compte : la possibilité qu'une exception générée par nos soins soit traitée par le programme audité, nous masquant la présence d'une faille. L'utilisation d'un débogueur sera indispensable afin d'examiner en détail les résultats de nos requêtes.

Réalisation d'un fuzzer SMTP. SMTP est un protocole relativement simple qui présente les avantages de fonctionner en mode texte, et grâce à un lexique de mots clés relativement réduit. La réalisation d'un fuzzer SMTP en s'aidant des bibliothèques citées ci-dessus est donc une opération simple : il suffit d'égrainer les différentes commandes en les affublant de paramètres éclectiques. Les serveurs de messagerie connus passeront sans trop de difficultés ces quelques tests, les autres pourraient donner un résultat similaire à celui présenté en figure 1.

3.3 L'Analyse d'Exécution de Processus

Rien ne vaut un bon débogueur pour savoir ce qu'il se trame au cours de l'exécution d'une tache. Exceptions masquées par un gestionnaire trop zélé, appels à des fonctions particulières (voir le tableau 1 plus loin), création de processus ou de threads, autant d'éléments qui peuvent aiguiser votre compréhension des évènements et vous mener à des découvertes dignes d'intérêt.

Si vous ne comptez pas quitter le Ring3, je ne saurais trop vous conseiller d'adopter OllyDbg [OLLYDBG], gratuit, complet, et d'une efficacité redoutable. Toutefois, toute tentative de débogage Ring0 (noyau, drivers) réclamera l'utilisation d'un logiciel tel que SoftIce.

Les compétences requises sont conséquentes, la procédure laborieuse. Il est cependant possible de se limiter à l'utilisation d'un traceur [DUMBBUG] qui vous rapportera les appels effectués au cours de l'exécution. Bien entendu, les éléments importants vous seront aussi communiqués : les paramètres passés à ces fonctions, leur emplacement probable en mémoire (pile, tas, section du binaire) ainsi que la valeur retournée par l'appel.

3.4 L'Analyse Différentielle de Binaires

Parmi les diverses techniques servant à percer à jour des brèches dans des logiciels, celle-ci est à classer dans une catégorie particulière. Contrairement aux

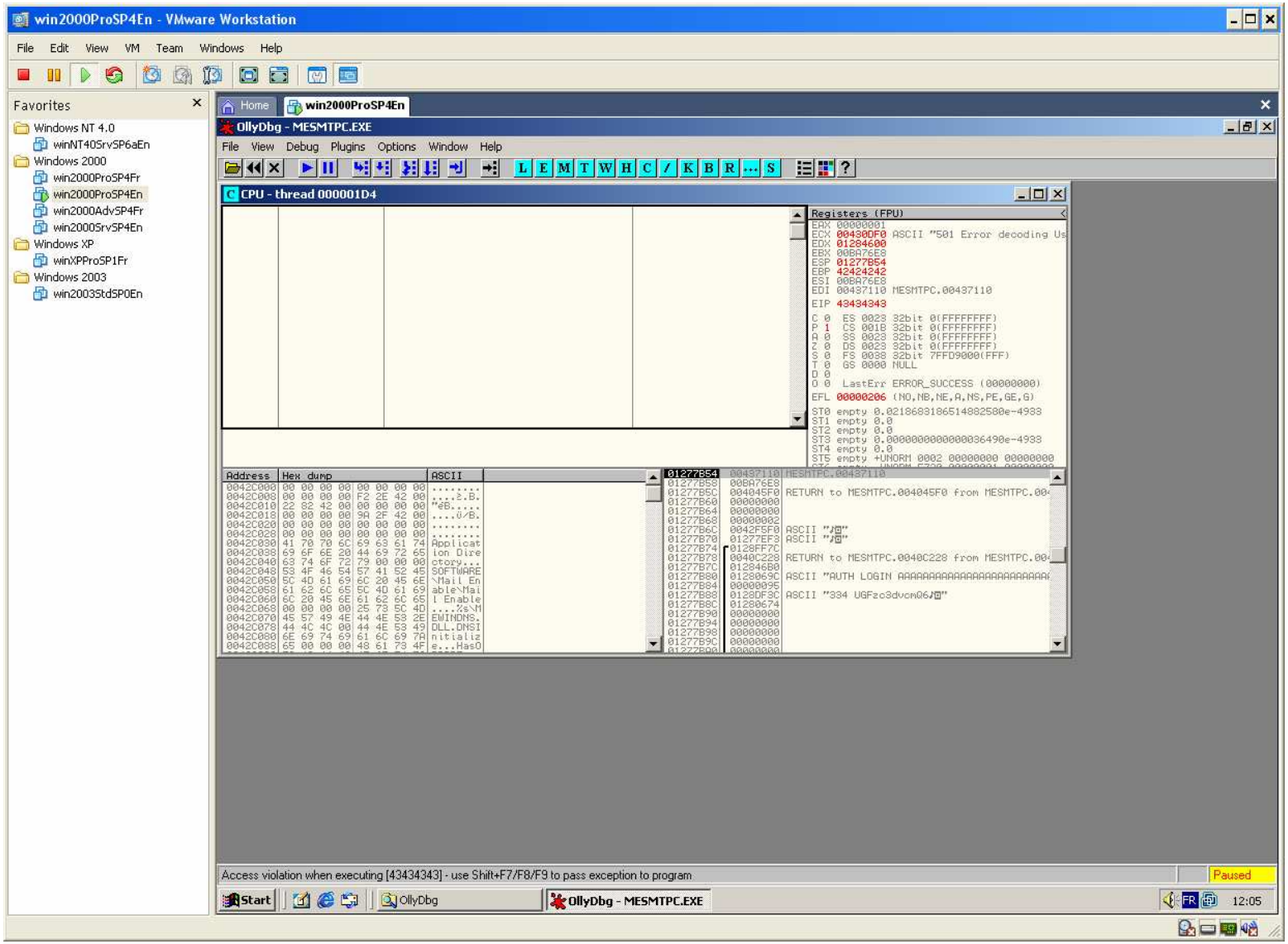

Fig. 1. 0day sur un serveur SMTP trouvé suite à une session de fuzzing

autres méthodes présentées ici, l'analyse différentielle de binaires nécessite la possession d'une version corrigée d'un binaire afin de pouvoir déterminer les modifications qui y ont été apportées et en déduire la nature du problème. Ainsi, la faille est au moins connue de l'éditeur, et n'est pas à strictement parler un « 0day », mais la problématique est d'actualité !

L'analyse différentielle de binaires fait l'objet cette année d'une présentation d'Halvar Flake et je ne m'étendrai pas trop dessus. Toutefois, je donne ci-après une série de fonctions qui peuvent être intégrées à un plugin IDA afin de générer les « signatures » des fonctions présentes dans un binaire.

```
unsigned long *seq_crc_table = NULL;

int init_seq(void)
{
  int i, j, k, n, size, max;
  func_t *f;
  ea_t ea;
  char *seq;

  n = get_func_qty();
  // On alloue l'espace nécessaire au tableau de CRC32
  if ((seq_crc_table =
   (unsigned long *)qalloc(n*sizeof(unsigned long))) == NULL)
    return -1;

  // Pour chaque fonction on construit une séquence d'octets
  // en ne conservant que les éléments indépendants de l'espace
  // d'adressage du processus
  for (i = 0, max = 0; i < n; i++)
  {
    // Dans un premier temps, on calcule la longueur maximale de
    // cette séquence
    size = 0;
    f = getn_func(i);
    ea = f->startEA;
    while (ea != f->endEA)
    {
      ua_ana0(ea);
      // On détermine le nombre d'opérandes de l'instruction
      for (j = 0; j < UA_MAXOP; j++)
        if (cmd.Operands[j].type == o_void)
          break;
      size += 4 + (j << 1);
      ea = next_head(ea, BADADDR);
    }
    if (size > max)
```

```
      max = size;
  }

  // On alloue le nombre d'octets nécessaires
  if ((seq = (char *)qalloc(max * sizeof(char))) == NULL)
    return -1;

  for (i = 0, size = max; i < n; i++)
  {
    memset(seq, 0, size);
    size = 0;
    f = getn_func(i);
    ea = f->startEA;
    while (ea != f->endEA)
    {
      ua_ana0(ea);
      for (k = 0; k < UA_MAXOP; k++)
        if (cmd.Operands[k].type == o_void)
          break;
      // On rajoute à la séquence les informations pertinentes
      seq[size++] = (char)(cmd.itype & 0xff);
      seq[size++] = (char)(cmd.itype >> 8);
      seq[size++] = (char)(cmd.size & 0xff);
      seq[size++] = (char)k;
      for (j = 0; j < k; j++)
      {
        seq[size++] = cmd.Operands[j].type;
        seq[size++] = cmd.Operands[j].dtyp;
      }
      ea = next_head(ea, BADADDR);
    }
    // On renseigne le tableau
    seq_crc_table[i] = get_crc((unsigned char *)seq, size);
  }

  qfree(seq);

  return 0;
}

void term_seq(void)
{
  if (seq_crc_table != NULL)
  {
    qfree(seq_crc_table);
```

```
    seq_crc_table = NULL;
  }
}
```

Ce type de signature est relativement intéressant car il ne prend pas en compte toute information relative à l'espace d'adressage du processus, ainsi que les petites modifications apportées par un compilateur d'une version à une autre que sont, par exemple, les modifications de registres. Le morceau de code précédent s'applique relativement bien aux fonctions « linéaires », moins aux fonctions morcelées que l'on trouve de plus en plus dans les binaires optimisés par les compilateurs Microsoft.

4 L'Analyse Statique de Binaires

L'analyse statique est, de mon point de vue, la manière la plus efficace de faire le tour d'un programme afin d'en trouver les bogues potentiels. Le principe est de désassembler un binaire et le parcourir manuellement ou de façon automatisée afin de délimiter les zones de code à risque, puis de les analyser plus en détail afin de déterminer si le péril est avéré. La méthode est économique en moyens : il ne faudra vous armer que d'un bon désassembleur, de l'exécutable à disséquer, et d'une très bonne connaissance de l'assembleur. En ce qui concerne le désassembleur, je ne saurai trop vous recommander d'utiliser IDA Pro (Interactive Disassembler Pro) [IDA] qui mérite amplement tous les éloges qui peuvent lui être faits.

L'auditeur de code compilé le plus motivé pourra développer ses propres scripts ou plugins afin d'automatiser une partie (plus ou moins importante) de la tâche assommante qu'est la lecture de plusieurs milliers de lignes d'assembleur, même s'il semble qu'on ne puisse se passer de l'homme, chaque faille conservant une part d'unicité.

Malgré cet aspect pesant, l'analyse statique de binaires présente de nombreux avantages indéniables, et un taux de succès intéressant (encore une fois ce dernier est lié à la compréhension de l'assembleur) :

- la couverture de l'analyse est optimale : on ne se contente pas de fonctionnalités documentées comme c'est le cas pour les essais manuels, tout en ne débordant pas sur des fonctionnalités non implémentées, ce qui arrive avec le fuzzing. Le bénéfice principal se situant dans les morceaux de code rarement exécutés, souvent atteignables par des chemins compliqués et peu empruntés, et qui recèlent souvent des trésors ;
- l'assembleur est un langage qui se prête à la recherche de vulnérabilités :
  - il n'y a en général qu'un fichier à étudier, et non une arborescence complexe de fichiers source et en-tête ;
  - l'emplacement des tampons en mémoire est plus aisée à percevoir, qu'ils soient alloués dans la pile, sur le tas ou statiquement dans une section d'un exécutable, ce qui permet de mieux cerner les impacts potentiels d'un débordement ;

– les problèmes de conversion de nombres signés ou non sont plus faciles à détecter ;

Enfin, Microsoft met à disposition de tous quantité d'informations très utiles sous la forme de symboles de débogage, mâchant le travail.

4.1 Les Fonctions Potentiellement Vulnérables

Tout développeur en environnement Windows 32 bits se trouve confronté à une difficulté particulière : pour certaines opérations basiques, il peut exister plus de 5 fonctions répondant à nos attentes. Et bien entendu, lorsque l'une ou l'autre devient facteur d'attaque, ses consoeurs le deviennent aussi. Plusieurs sont déjà identifiées comme à risque, d'autres le deviendront certainement dans un futur proche. Une liste de ces dernières est présentée dans le tableau 1.

Tab. 1. Fonctions périlleuses en environnement Windows

Fonction	Bibliothèque	Cas avéré de vulnérabilité
strcpy, wcscpy, _mbscpy	? ? ?	MS03-001 (Locator Service)
strcat, wcscat, _mbscat	? ? ?	MS05-010 (License Logging Service)
strncpy, wcsncpy, _mbsncpy	? ? ?	? ? ?
strncat, wcsncat, _mbsncat	? ? ?	? ? ?
sprintf, swprintf	? ? ?	MS05-0 ? ? (MSMQ)
vsprintf, vswprint	? ? ?	MS04-011 (LSASS)
_vsnprintf, _vsnwprintf	? ? ?	? ? ?
memcpy, wmemcpy	? ? ?	MS04-045 (WINS)
memmove, wmemmove	? ? ?	? ? ?
MultiByteToWideChar	kernel32.dll	? ? ?
WideCharToMultiByte	kernel32.dll	? ? ?
lstrcpy	kernel32.dll	MS04-022 (Task Scheduler)
lstrcat	kernel32.dll	? ? ?
lstrcpyn	kernel32.dll	? ? ?
ExpandEnvironmentStrings	kernel32.dll	? ? ?
wsprintf	user32.dll	? ? ?
wvsprintf	user32.dll	? ? ?
OemToChar	user32.dll	? ? ?
CharToOem	user32.dll	MS04-034 (Compressed Folders)

Et il ne faut pas perdre de vue le fait que les fonctions exportées dans kernel32.dll et user32.dll peuvent être disponibles en versions ANSI (suffixe A) et en version unicode (suffixe W), ce qui augmente encore le nombre de fonctions à surveiller.

Il est curieux de constater que même si ce type de vulnérabilité est connu depuis plusieurs années, il a persisté dans les sytèmes d'exploitation Microsoft jusqu'à l'avènement de Windows 2003 et Windows XP SP2 qui ont vu la quasi totalité de ces appels remplacés par des alternatives dites de confiance.

La suite n'est qu'une histoire de taille : le tampon destination, où qu'il soit dans l'espace mémoire du processus, est-il assez grand pour accueillir les données qui vont lui être transmises ? Si tel n'est pas le cas, la situation peut dégénérer jusqu'à l'exécution de code arbitraire...

IDA vous favorisera énormément la tâche pour peu que vous ayez quelques connaissances en C ou C++. Vous pourrez alors opter soit pour le développement de scripts (langage IDC très proche du C), soit pour le développement de plugins grâce au SDK mis à disposition par DataRescue. Divers exemples peuvent être trouvés sur le web [PALACE], notamment quelques uns en rapport avec l'audit de code assembleur.

4.2 Les Débordements d'Entiers et Conversions de Signe

Autre catégorie de vulnérabilités, les débordements d'entiers et les conversions de signe ont encore de beaux jours devant eux, car plus compliquée à débusquer que celle précédemment évoquée. Néanmoins, le langage assembleur se révèle bien adapté à la recherche de ces faiblesses spécifiques :

- les opérations sur les entiers sur processeur x86 sont menées sur des registres de 8bits, 16bits, 32bits. Autant les modifications de tailles de variables peuvent être compliquées à suivre dans un langage de haut niveau, autant elles sont simples en assembleur. Il convient de s'intéresser plus particulièrement aux directives movzx, movsx ou encore aux spécifications d'adressage byte, word, dword ;
- les caractéristiques des comparaisons (signées ou non) sont tout aussi complexes à déterminer, et il n'est pas rare que certaines conversions soient faites à l'insu du développeur. Heureusement, le langage assembleur Intel x86 dispose d'instructions de branchements conditionnels spécifiques en fonction du caractère signé ou non des registres (les couples ja*—jb* et jg*—jl*), ainsi qu'une instruction de déplacement et extension signée, movsx.

L'impact de ces failles est varié : contournement de vérification de limites, corruption de tailles de buffer lors d'allocations et de copies ; il n'est pas rare que cela aboutisse à de l'exécution de code.

4.3 En Pratique

Il ne vous reste plus qu'à effectuer la transition entre théorie et pratique. Le logiciel adéquat, de solides connaissances en assembleur, plusieurs versions de Windows (légalement acquises), beaucoup de temps libre, et vous voici prêt à rechercher, et découvrir, des failles dans le système d'exploitation le plus utilisé au monde. Voici quelques exemples concrets (et non diffusés auparavant) de vulnérabilités trouvées par analyse statique de binaires.

MS04-042 : Vulnérabilité du serveur DHCP de Windows NT 4.0. Cette vulnérabilité est une bonne illustration des conséquences de l'utilisation éronnée

d'un type de variable, entrainant un débordement d'entier. L'erreur se situe dans la fonction DhcpMakeClientUID qui peut être atteinte par la plupart des requêtes DHCP. Dans cette fonction se trouve la portion de code suivante :

```
.text:5370FD31        mov     bl, [ebp+arg_4]
                      ; taille de l'adresse "hardware"
.text:5370FD34        add     bl, 5
.text:5370FD37        movzx   ecx, bl
.text:5370FD3A        push    ecx             ; uBytes
.text:5370FD3B        call    _DhcpAllocateMemory@4
```

La valeur passée à la fonction en tant que second argument (arg_4) correspond à la longueur de l'adresse « hardware » codée sur un octet dans une requête DHCP. Elle peut prendre toute valeur comprise entre 0x00 et 0xff. Cependant, pour les valeurs 0xfb à 0xff, on est confronté à un problème particulier : lorsque l'on ajoute 5 à cette valeur, la résultante est supérieure ou égale à 0x100, ce qui sur un octet vaudra 0x00 à 0x04. Un peu plus loin figurent les lignes suivantes :

```
.text:5370FD60        movzx   edx, [ebp+arg_4]
                      ; taille de l'adresse "hardware"
.text:5370FD64        lea     edi, [ecx+1]
                      ; buffer alloué dynamiquement
.text:5370FD67        mov     esi, [ebp+arg_0]
                      ; adresse "hardware"
.text:5370FD6A        mov     ecx, edx
.text:5370FD6C        shr     ecx, 2
.text:5370FD6F        rep movsd
                      ; opération de copie de buffer
.text:5370FD71        mov     ecx, edx
.text:5370FD73        and     ecx, 3
.text:5370FD76        rep movsb
```

Ici, l'adresse « hardware » est copiée en totalité dans le buffer attribué précédemment par DhcpAllocateMemory. Ainsi, pour une taille de 0xff, 0x04 octets seront alloués, et 0xff seront copiés, d'où le débordement sur le tas.

Il est intéressant de noter que cette vulnérabilité a été corrigée dans la branche Windows 2000 en passant la variable utilisée pour l'addition de 8 à 32 bits, sans toutefois que le correctif ne soit appliqué à la branche NT 4.0 jusqu'à une date récente.

Le segment de code précédent a été porté à mon attention par un plugin IDA recherchant dans le code les opérations de copie de buffer dont la source et la taille dépendent d'arguments passés à la fonction. Bien évidemment, ce type de recherche remonte beaucoup d'informations et le traitement reste manuel. Dans ce cas précis, il fallait avant tout localiser la possibilité d'un débordement d'entier sur 8 bits avant l'allocation mémoire.

MS04-045 : Vulnérabilité du serveur WINS de Windows. En 2004, le serveur WINS de Windows a fait couler beaucoup d'encre, ou plutôt d'octets.

En février, une première vulnérabilité était corrigée dans le bulletin MS04-006, néanmoins en Juin, la société ImmunitySec, Inc. annonçait la découverte d'une nouvelle vulnérabilité et son exploitation dans le service de nommage de Windows. Plusieurs rumeurs, copies d'écrans, exploits ont alors circulé sur Internet de façon plus ou moins privée, jusqu'à la publication en Décembre 2004 du correctif MS04-045 corrigeant tout cela.

Lors de requêtes de réplication transitant sur le port 42/TCP, le serveur WINS envoyait un pointeur vers son propre espace d'adressage qui doit lui être renvoyé lors de futures communications afin qu'il puisse faire référence aux bonnes variables. Cela est bien évidemment une très mauvaise idée puisqu'il suffit de modifier ce pointeur dans une requête pour influer sur des opérations de lecture et d'écriture mémoire. Bien peu de personnes savent qu'une autre vulnérabilité a été corrigée avec celle-ci. La fonction RplMsgfUfmUpdVersNoReq contient les instructions suivantes :

```
.text:0290F429        mov     esi, [esp+arg_0] ; source
.text:0290F42D        push    dword ptr [esi]
.text:0290F42F        call    _ntohl@4
.text:0290F434        mov     ecx, [esp+arg_8]
.text:0290F438        add     esi, 4
.text:0290F43B        mov     edi, [esp+arg_4] ; destination
.text:0290F43F        mov     [ecx], eax
.text:0290F441        mov     ecx, eax
.text:0290F443        shr     ecx, 2
.text:0290F446        rep movsd
                      ; opération de copie de buffer
.text:0290F448        mov     ecx, eax
.text:0290F44A        and     ecx, 3
.text:0290F44D        rep movsb
```

Il est primordial de remarquer que la taille du buffer à copier est spécifiée comme un « netlong » occupant les 4 premiers octets du buffer source. Cette fonction n'est appelée qu'une seule fois dans la fonction HandleUpdVersNoReq :

```
.text:0290FF60        lea     ecx, [ebp+var_C]
.text:0290FF63        lea     edx, [ebp+var_138]
.text:0290FF69        push    ecx
.text:0290FF6A        mov     eax, [esi+18h]
.text:0290FF6D        push    edx
.text:0290FF6E        add     eax, 4
.text:0290FF71        push    eax
.text:0290FF72        call    _RplMsgfUfmUpdVersNoReq@12
```

Curieusement le buffer destination (deuxième argument) est défini sur la pile, à seulement 0x13C octets de l'adresse de retour de la fonction, sans qu'aucune vérification sur la longueur de la chaîne à copier ne soit menée. Il suffit donc de contruire une requête particulière en spécifiant une taille supérieure à 0x13C octets et nous avons notre débordement de buffer sur la pile.

Dans la mesure où le protocole WINS est propriétaire et non documenté, il faudra passer par quelques essais manuels afin de déterminer le format de la requête à construire. Il suffit pour cela de mettre en place deux serveurs Windows sur lesquels le composant serveur WINS est installé, de configurer leur réplication, puis de capturer et analyser les paquets transitant sur le port 42/TCP.

Cette vulnérabilité a aussi été trouvée grâce au plugin IDA évoqué ci-dessus.

MS05-010 : Vulnérabilité du serveur de gestion de licenses de Windows. La faille en question est intéressante pour plusieurs raisons. La première réside dans le fait qu'elle est exploitable en environnement 2000 serveur grâce à un double débordement de tampon. Le premier « overflow » a lieu dans un buffer alloué statiquement, dans la section .data du binaire, écrasant par là même d'autres variables déclarées statiquement. S'en suit un second débordement résultant de l'utilisation d'une de ces variables, cette fois-ci dans la pile, aboutissant à l'exécution de code sur la machine. Je ne détaillerai pas plus cette partie du fait de sa sensibilité à l'heure actuelle. La seconde raison est qu'il a permis de souligner un défaut dans l'application du SP4 de Windows 2000. En effet, lorsque celui-ci est mis en place sur une installation existante, l'accès anonyme vers le tube nommé LlsRpc est supprimé, ce qui n'est pas le cas lorsqu'il est intégré à un CD d'installation, et l'exploitation anonyme de la vulnérabilité est alors possible.

Une autre vulnérabilité touche les serveur Windows NT 4.0. La fonction LlsrConnect ressemble à ce qui suit :

```
.text:01833F95        push    20h
.text:01833F97        call    _MIDL_user_allocate@4
.text:01833F9C        mov     esi, eax
.text:01833F9E        mov     eax, [esp+arg_4]
.text:01833FA2        test    eax, eax
.text:01833FA4        jz      short loc_1833FA9
.text:01833FA6        push    eax
.text:01833FA7        jmp     short loc_1833FAE
.text:01833FA9
.text:01833FA9 loc_1833FA9:
.text:01833FA9        push    offset ??_C@_11A@?$AA?$AA?$AA?$AA?$AA@
.text:01833FAE
.text:01833FAE loc_1833FAE:
.text:01833FAE        push    esi
.text:01833FAF        call    ds:__imp__lstrcpyW@8
```

Un buffer de 32 octets est alloué, et si le contenu du second argument est non nul, il est copié dans ce buffer, sans aucune vérification de taille. Dans la mesure où un utilisateur peut influer sur cet argument, cela débouche sur un débordement de tampon dans le tas.

Cette vulnérabilité a été découverte grâce à un plugin IDA recherchant des opérations de manipulation de chaînes de caractères dont la destination est allouée de manière fixe, et dont les sources dérivent d'arguments passés à la fonction (et donc potentiellement sous le contrôle de l'utilisateur).

5 Et Après ?

Trouver une vulnérabilité est une chose, toutefois il ne s'agit que d'une étape, la première, parmi de nombreuses. Apprécier la criticité de la faille, écrire un exploit fonctionnel pour un large éventail de plateformes, prévenir l'éditeur ou le constructeur, écrire un avis, sont autant d'options qui s'offrent au découvreur.

Il est relativement aisé de découvrir une fonction offrant une vulnérabilité potentielle, en faire quelque chose d'exploitable est une autre paire de manches. Ansi, il est absolument nécessaire d'évaluer précisément les potentialités accordées avant toute communication sur le sujet. Dans l'hypothèse où la faille se révèle exploitable par un utilisateur distant ou local, il vous incombe d'estimer la criticité de la faille. Pour cela, plusieurs éléments rentrent en ligne de compte :

- Quel est le type de la vulnérabilité ?
  - Révélation d'information, déni de service, exécution de code arbitraire.
- La vulnérabilité est-elle exploitable localement ou à distance ?
- Nécessite-t-elle un niveau d'authentification quelconque ?
- Quelle est la nature des privilèges que l'on peut obtenir ?
  - SYSTEM, administrateur, utilisateur, invité.
- Les plateformes impactées sont-elles nombreuses ?
- L'exploitation de la faille est-elle fiable ?
  - Débordement sur la pile, sur le tas, écrasement arbitraire de mémoire.

Ces détails vous permettront de vous assurer une compréhension complète des implications de votre découverte. Il n'est pas rare que des erreurs soient faites à ce niveau, y compris par l'éditeur.

La prochaine étape devrait être la prise de contact avec l'éditeur. Il n'est pas dans mon objectif de polémiquer sur le « full disclosure », il s'agit cependant d'une étape que je juge indispensable dans le processus, et surtout avant toute communication destinée au public. L'intervalle entre la première communication et la publication d'un correctif pourra atteindre plusieurs mois pendant lesquels il vous faudra faire preuve d'une grande patience.

Une fois le correctif publié, vous pourrez l'accompagner d'un avis de votre cru, explicitant les aspects techniques de la vulnérabilité, mais gardez en tête que cela arrangera les pirates en tous genres.

5.1 Implémenter un Exploit

Ecrire un outil fiable d'exploitation d'une vulnérabilité en environnement Windows 32 bits est délicat, pourtant c'est à ce prix que vous appréhenderez les tenants et les aboutissants de votre trouvaille. Les shellcodes pour Windows,

ces quelques lignes d'assembleur compilées permettant d'exécuter du code suite à l'exploitation d'une vulnérabilité, sont à eux seuls un monde à part. Et ils ne représentent qu'une seule des briques à maitriser pour écrire un programme fonctionnel.

Ces dernières années, les recherches ont fait des progrès, et les papiers détaillant des aspects méconnus du fonctionnement de Windows fleurissent. En parallèle, des logiciels facilitant grandement le développement de ces morceaux de code si particuliers [METASPLOIT] voient le jour. Les protocoles réseaux utilisés par Microsoft, à l'originie propriétaires, sont de mieux en mieux documentés par des équipes de rétroconcepteurs [SAMBA].

L'implémentation d'exploits pour Windows n'est pas encore une partie de plaisir mais y ressemble de plus en plus ...

5.2 Communiquer et Diffuser

Stade finale de cette belle aventure, la médiatisation de ce que vous avez décelé. Même si j'adhère fermement au principe selon lequel chaque faille appartient à celui qui la trouve, libre à lui d'en disposer comme bon lui semble, ce n'est pas le cas de tout le monde, et en particulier face à la législation française.

Il est indispensable de comprendre toutes les implications de la découverte d'une faille exploitable non corrigée pour la sécurité des sytèmes d'information, d'Internet en général, et la votre en particulier. La divulgation responsable d'informations à l'éditeur ou au constructeur est, de mon point de vue, l'attitude à adopter afin de minimiser les impacts négatifs à tous points de vous. Vous pouvez inclure dans vos communications un organisme coordinateur tel qu'un CERT, qui pourra assoir votre crédibilité.

Quant à la diffusion d'exploits ou de code permettant l'exploitation d'une vulnérabilité, elle n'est pas à prendre à la légère. Un « proof of concept » pourra vous être demandé pour valider une vulnérabilité, et c'est globalement le seul morceau de code qui devrait être transmis à un tiers.

6 Conclusion

La non disponibilité des sources d'un logiciel n'en garantit pas pour autant l'insécurité, de même qu'une gestion obscurantiste des correctifs n'en garantit la sécurité. Si vous devez ne retenir que certains points de cet article, puissent-ils être les suivants :

- l'absence de sources n'a jamais été un frein à l'audit d'un logiciel ou d'un système d'exploitation. Il est regrettable que la recherche française soit tellement en retard dans ce domaine ;
- le désassemblage n'est qu'un des moyens d'arriver à ses fins, les autres nécessitant un panel de compétences différent ;
- tout correctif de sécurité, qu'il soit explicité ou non, permet de déduire la nature des vulnérabilités corrigées. La naissance d'exploit n'est plus alors qu'une question d'heures ;

Le mot de la fin sera consacré au sujet épineux de la sécurité du système d'exploitation de Microsoft. Pour avoir passé un nombre d'heures faramineux dans la multitude de binaires de Windows, je ne peux que constater une amélioration marquée de la sécurité de ces produits. Les progrès les plus marquants ont sans doute été effectués avec Windows XP SP2 et Windows 2003 SP1 qui ont bénéficié de lourdes modifications influant très positivement sur leur sécurité. Les vulnérabilités exploitables par défaut à distance et anonymement sont en voie de d'extinction, les autres se font de plus en plus rare.

Références

[VSC] Immunity, Inc. Vulnerability Sharing Club `http://www.immunitysec.com/services-sharing.shtml`

[ZONEH] Forums Zone-H, `http://forum.zone-h.org/viewforum.php?f=3`

[ETHEREAL] Ethereal, `http://www.ethereal.com/`

[SPIKE] Immunity, Inc. SPIKE, `http://www.immunitysec.com/resources-freesoftware.shtml`

[PEACH] IOActive PEACH, `http://www.ioactive.com/v1.5/tools/index.php`

[OLLYDBG] OllyDbg, `http://home.t-online.de/home/ollydbg/`

[DUMBBUG] Phenoelit dum(b)ug, `http://www.phenoelit.de/dumbbug/`

[IDA] Datarescue IDA Pro, `http://www.datarescue.com/idabase/`

[PALACE] The IDA Palace, `http://home.arcor.de/idapalace/`

[METASPLOIT] The Metasploit Framework, `http://www.metasploit.com/projects/Framework/`

[SAMBA] SAMBA, `http://www.samba.org/`