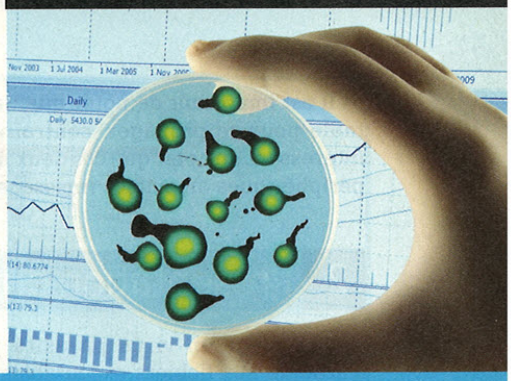

ANALYSE DU VIRUS.WIN32.SALITY

Nicolas Brulez – nicolas.brulez@kaspersky.fr

Senior Malware Researcher – Global Research and Analysis Team

Kaspersky Lab France

mots-clés : CODES MALICIEUX / REVERSE ENGINEERING / VIRUS / ANALYSE DE CODE / FILE INFECTOR / SALITY

Les vrais virus (infecteurs d'exécutables) se font rares de nos jours. Ils ont été remplacés par les chevaux de Troie et autres vers réseau tels que Conficker. Deux familles de virus parasites se démarquent pourtant : Virus.Win32.Sality et Virus.Win32.Virut. Cet article traite de la première famille et présente l'analyse de la variante la plus répandue de ce virus polymorphique. Tout le monde le sait, il est important de bien configurer les partages de fichiers pour ne pas autoriser l'écriture lorsque cela n'est pas nécessaire. Malheureusement, nombreuses sont les entreprises avec des applications internes sur les partages réseaux avec accès complet en écriture. Les virus comme Sality ne se privent pas pour les infecter et c'est ensuite l'épidémie dans l'entreprise.

L'exécution automatique des *autorun* sur les clés USB et disques réseaux ? Dangereux ? Tout le monde le sait. Pourtant, les entreprises qui bloquent l'exécution automatique sont encore trop rares et l'infection par périphériques amovibles fonctionne encore très (trop !) bien à l'heure d'aujourd'hui. Je vous invite à regarder quelques statistiques pour vous en convaincre.

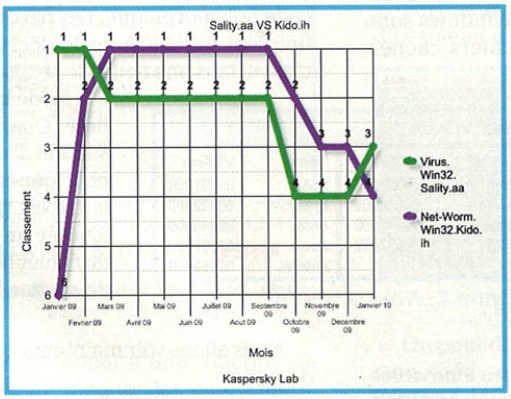

Figure 1 : Classement de Sality.aa et Kido.ih depuis les 13 derniers mois

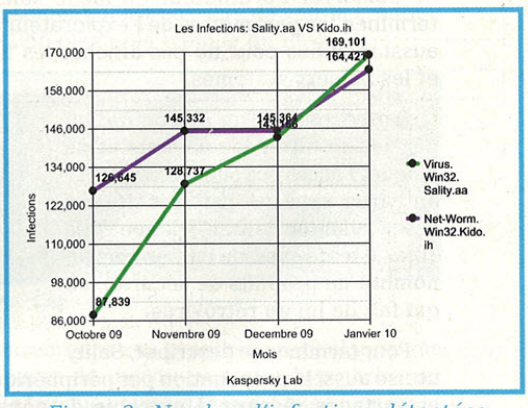

Figure 2 : Nombre d'infections détectées depuis octobre 2009

1 Statistiques

Le choix de ce virus pour le *Malware Corner* n'est pas un hasard. Depuis 13 mois, il est placé dans le TOP 4 des infections que nous détectons. Comme vous allez le voir, il a même détrôné une des variantes de Conficker (Kido) les plus présentes depuis maintenant 1 an : voir Figure 1.

Le graphique ci-dessus nous montre la position de Sality.aa et de Kido.ih dans le TOP des infections détectées et nettoyées.

Nous allons maintenant voir le nombre d'infections référencées depuis le mois d'octobre 2009 pour ces deux codes malicieux : voir Figure 2.

Le virus Sality est en progression constante et a même détrôné le nombre d'infections de Kido.ih au mois de janvier 2010.

Il est important de noter que Sality n'utilise aucune faille pour se propager, contrairement à Kido. Il est intéressant de noter que ces deux *malwares* utilisent les périphériques amovibles (message subliminal).

2 Présentation du virus Sality

Sality est un virus parasite polymorphique qui s'attaque aux fichiers exécutables Windows (PE). Lors d'une infection, le virus écrase certaines parties du code du fichier hôte avec du code polymorphique. Le code du virus peut ensuite être inséré de deux façons différentes dans le fichier hôte : soit en agrandissant la dernière section, soit en ajoutant une nouvelle section dans le fichier PE. Le corps du virus est chiffré à l'aide de l'algorithme RC4.

Ce parasite est aussi résident mémoire. Il injecte des processus mais n'utilise aucun *hook* pour se propager plus rapidement, contrairement à Virut. Le virus utilise un système d'autoprotection pour empêcher que tous les processus infectés ne soient terminés. Les processus infectés se surveillent entre eux pour assurer la présence du malware en mémoire.

Pour éviter de tenter l'utilisateur infecté, le gestionnaire de tâches est désactivé via modification de la base de registre. L'éditeur de registre est lui aussi désactivé par la même occasion. Lors de ces diverses modifications du registre, Sality efface aussi les clés de registre permettant de démarrer l'ordinateur en mode sans échec. Pour terminer, les paramètres de l'explorateur Windows sont aussi modifiés pour ne pas afficher les fichiers cachés et les fichiers systèmes.

Sality contient un *driver*, qui lui permet de filtrer les paquets et de bloquer l'accès aux sites des éditeurs antivirus, ainsi que certains sites de désinfection, ou de scan, tels que Virus Total. Il tente aussi de tuer un certain nombre de produits de sécurité, ce qui fait de lui un rétrovirus.

[Section Table]

Name	VOffset	Name	VOffset
.text	00001000	.text	00001000
.rdata	00032000	.rdata	00032000
.data	00048000	.data	00048000
.rsrc	00057000	.rsrc	00057000
		.nrdata	00060000

AVANT APRÈS

Figure 4 : Nom de section du virus

Pour terminer ce descriptif, Sality utilise aussi la propagation par périphériques amovibles et partages réseaux. Une copie de `notepad.exe` est déposée sur le périphérique, renommée et infectée. Un fichier `autorun.inf` est aussi créé pour l'auto exécution sur les machines qui l'autorisent. Un `autorun.inf` est aussi créé sur les partages réseaux.

3 Analyse technique

Dans cette partie, nous allons voir l'analyse du virus par *reverse engineering*. Toutes les fonctionnalités du virus ne seront pas détaillées par manque de place, mais les plus importantes seront présentées.

3.1 L'infection d'exécutables

L'infection de Sality est assez classique. Le point d'entrée dans le PE *header* n'est pas modifié et l'exécution commence dans la section code, pour éviter les détections heuristiques. Il s'agit d'un virus *appender*, le corps du virus est donc placé à la fin du fichier.

Pour comprendre l'infection des fichiers PE, voici un schéma qui représente la structure des fichiers avant et après l'infection :

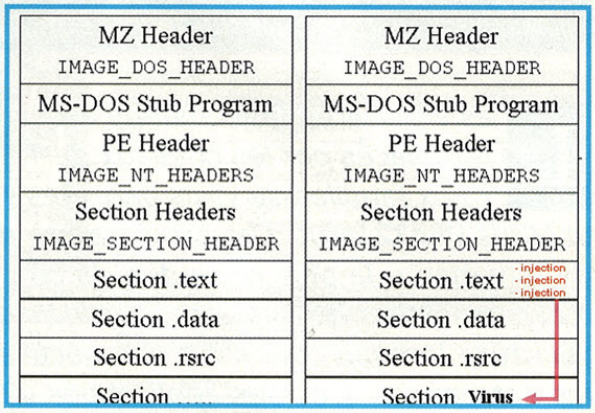

Figure 3 : Modification du virus sur la structure d'un fichier PE

Sality écrase plusieurs parties du programme hôte (le code originel est sauvegardé au préalable) avec son propre code polymorphique. Les parties sont reliées entre elles et finissent par exécuter le décrypteur du virus. Celui-ci se trouve dans une nouvelle section (comme sur le schéma) ou à la fin de la dernière section du fichier hôte. Quand Sality crée une nouvelle section, il utilise un algorithme simple pour générer un nom de section. Une lettre est générée aléatoirement puis ajoutée au nom de la seconde section du fichier hôte. Voici un exemple d'ajout de section : voir Figure 4.

Nous allons voir maintenant des exemples d'écrasement de code.

Avant infection : voir Figure 5.

Le code précédent est le point d'entrée d'une application saine. Voici maintenant le point d'entrée du fichier après infection : voir Figure 6.

On constate une différence importante entre les deux routines, notamment l'appel à la fonction `UnmapViewOfFile`, qui n'est pas présente dans l'application saine. En effet, Sality utilise des appels à des fonctions de l'API Windows avec des paramètres erronés.

Le but étant de bloquer l'analyse par émulation et, par conséquent, la détection du virus si le moteur tente d'émuler la routine de décryptage.

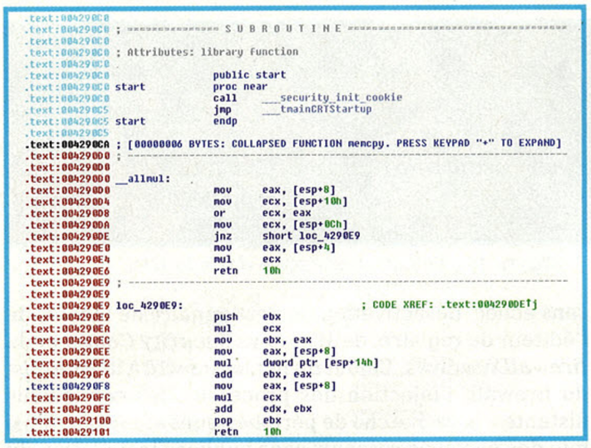

Figure 5 : Point d'entrée de l'application avant infection

Le virus injecte plusieurs routines du genre dans le fichier hôte. La première routine redirige vers la seconde, qui remplacera le code suivant dans notre fichier cible. Pour montrer la taille du code polymorphique générée par routine, voici la suite de la première routine avant l'infection :

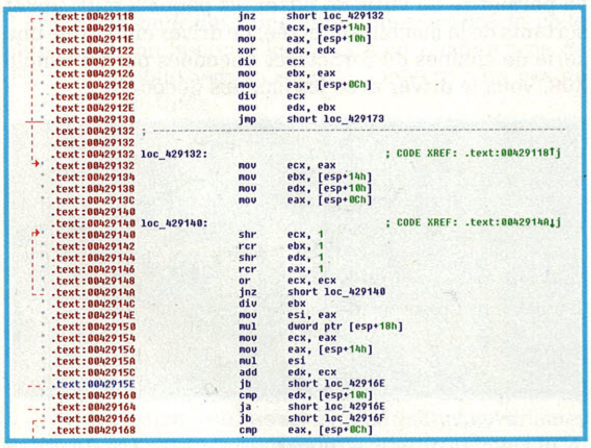

Figure 7 : Point d'entrée de l'application avant infection (suite)

Maintenant, cette même suite écrasée par notre infecteur : voir Figure 8.

Encore une fois, Sality fait appel à une fonction de l'API Windows pour tenter de bloquer les émulateurs. On notera aussi que la fonction **FindClose** crée une exception si le *handle* passé en paramètre n'est pas valide lorsque le programme est débogué. Cette routine fait ainsi office d'anti-debug (probablement par hasard, puisque les appels de fonctions ne sont jamais les mêmes).

3.2 Le déchiffrement du virus

Une fois les sauts intra-routine terminés, le code du décrypteur est exécuté. Sality.aa utilise l'algorithme RC4. Celui-ci est noyé dans le code inutile créé par le moteur polymorphique. Un *virus analyst* non expérimenté

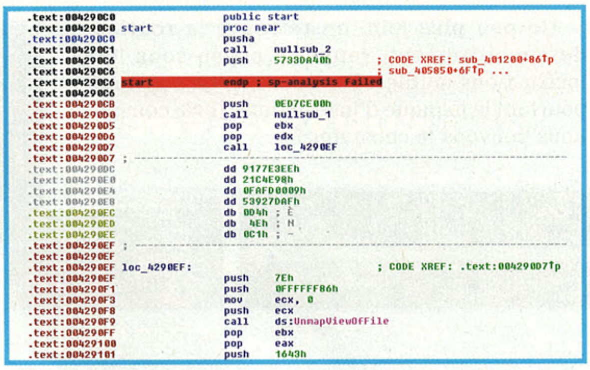

Figure 6 : Point d'entrée de l'application après infection

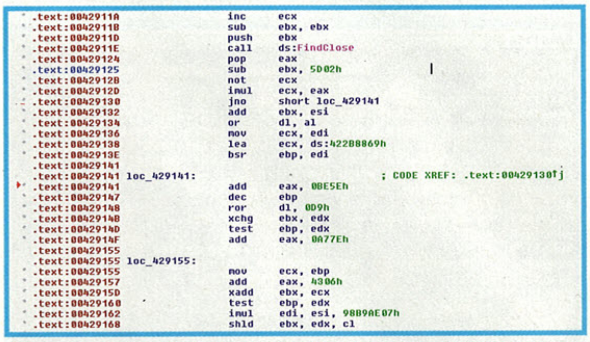

Figure 8 : Point d'entrée de l'application après infection (suite)

pourrait ne pas identifier l'utilisation de RC4, du moins dans un premier temps.

Fort heureusement, il est possible de voir la création du tableau de 256 éléments typique du RC4 :

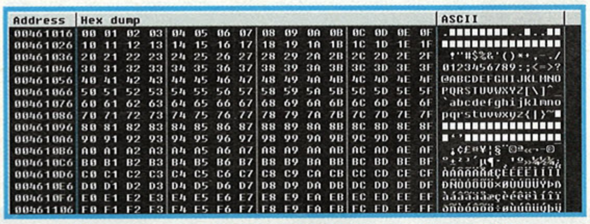

Figure 9 : Table de 256 éléments pour RC4

On peut donc émettre une hypothèse sur l'algorithme sans trop se poser de questions. Cette hypothèse est confortée par le fait que notre tableau est mis à jour avant le déchiffrement du virus. En effet, la clé joue un rôle dans la mise à jour du tableau. Voici donc notre tableau modifié :

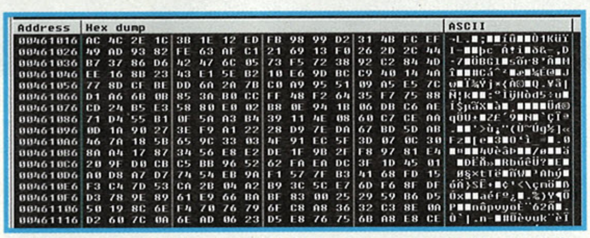

Figure 10 : Table de 256 éléments pour RC4 mise à jour avec la clé

Un peu plus loin, on retrouve la routine de déchiffrement, toujours noyée sous les instructions inutiles. Par endroits, on constate pourtant le manque d'instructions **junk**, comme nous pouvons le constater ici :

Figure 11 : Partie de l'algorithme RC4

Pour finir, voici le début du virus déchiffré, sans aucune instruction polymorphique, « simple » à analyser :

```
00461116  E8 00000000      CALL VMwareHo.0046111B
0046111B  5D               POP EBP
0046111C  81ED 05104000    SUB EBP,VMwareHo.00401005
00461122  58               POP EAX
00461123  2D 406F0300      SUB EAX,36F40
00461128  8985 43124000    MOV DWORD PTR SS:[EBP+401243],EAX
0046112E  80BD 73274000 0  CMP BYTE PTR SS:[EBP+402773],0
00461135  75 19            JNZ SHORT VMwareHo.00461150
00461137  C785 3A144000 2  MOV DWORD PTR SS:[EBP+40143A],22222222
00461141  C785 29144000 3  MOV DWORD PTR SS:[EBP+401429],33333333
0046114B  E9 82000000      JMP VMwareHo.004611D2
00461150  33DB             XOR EBX,EBX
00461152  64:67:8B1E 3000  MOV EBX,DWORD PTR FS:[30]
00461158  85DB             TEST EBX,EBX
0046115A  78 0E            JS SHORT VMwareHo.0046116A
0046115C  8B5B 0C          MOV EBX,DWORD PTR DS:[EBX+C]
0046115F  8B5B 1C          MOV EBX,DWORD PTR DS:[EBX+1C]
00461162  8B1B             MOV EBX,DWORD PTR DS:[EBX]
00461164  8B5B 08          MOV EBX,DWORD PTR DS:[EBX+8]
00461167  F8               CLC
00461168  EB 0A            JMP SHORT VMwareHo.00461174
0046116A  8B5B 34          MOV EBX,DWORD PTR DS:[EBX+34]
0046116D  8D5B 7C          LEA EBX,DWORD PTR DS:[EBX+7C]
00461170  8B5B 3C          MOV EBX,DWORD PTR DS:[EBX+3C]
00461173  F8               CLC
00461174  66:813B 4D5A     CMP WORD PTR DS:[EBX],5A4D
00461179  74 05            JE SHORT VMwareHo.00461180
```

Figure 12 : Virus déchiffré

3.3 Poupées russes

Une fois décrypté, on s'aperçoit que notre virus embarque un autre exécutable PE. Malgré le fait que cet exécutable PE ne contienne pas le flag DLL, celui-ci sera traité comme tel. Notre DLL est aussi compressée à l'aide d'UPX :

Figure 13 : DLL compressée et embarquée dans le virus

Une fois notre DLL décompressée, celle-ci embarque un autre fichier PE. Il s'agit cette fois d'un driver : voir Figure 14.

Cette DLL commence par générer des *threads* pour effectuer diverses opérations : effacement du mode

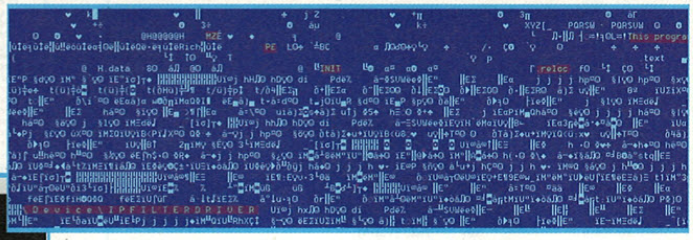

Figure 14 : Driver embarqué dans la DLL

sans échec, désactivation du gestionnaire de tâches, de l'éditeur de registre, de *Windows Security Center* et du *firewall* Windows. L'ajout du fichier infecté à la *whitelist* du firewall, l'injection des processus, les connexions distantes, la recherche de périphériques amovibles ainsi que des partages réseaux, sans oublier l'installation du driver, sont aussi des actions effectuées par notre DLL.

3.4 Le driver

Le driver embarqué dans notre DLL est un driver qui va permettre au virus de filtrer les paquets entrants et sortants de la machine infectée. Le driver embarque une série de chaînes de caractères encodées par un simple **XOR**. Voici le driver avec les chaînes décodées :

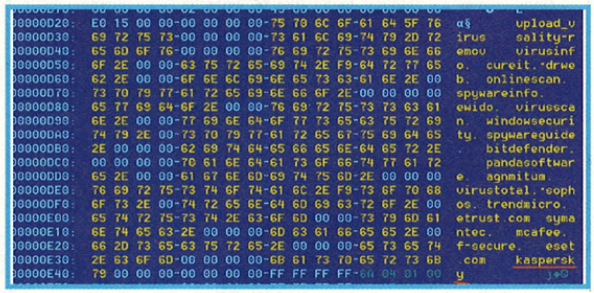

Figure 15 : Déchiffrement des chaînes de caractères du driver

Une fois le driver installé, il est impossible de contacter les sites contenant les chaînes précédentes. Le driver bloque donc l'accès aux sites des éditeurs antivirus, à Virus Total, etc.

3.5 L'infection des périphériques amovibles et réseaux

La DLL modifie les paramètres de l'explorateur Windows pour ne pas afficher les fichiers cachés et systèmes. Lors de l'insertion d'une clé USB, le fichier originel **notepad.exe** est copié sur la clé avec un nom aléatoire, puis infecté par Sality. On retrouve l'habituel **autorun.inf** sur cette même clé USB. L'infection est similaire pour les périphériques réseaux. Voici la routine responsable de l'infection : voir Figure 16.

```
                mov     eax, [ebp+var_1258]
                add     eax, 41h
                mov     [ebp+String1], al
                mov     [ebp+var_124B], 3Ah
                mov     [ebp+var_124A], 5Ch
                mov     [ebp+var_1249], 0
                lea     ecx, [ebp+String1]
                push    ecx             ; lpRootPathName
                call    GetDriveTypeA   ; Récupère le type de lecteur
                mov     [ebp+NumberOfBytesWritten], eax
                cmp     [ebp+NumberOfBytesWritten], DRIVE_REMOVABLE
                jz      short infecte
                cmp     [ebp+NumberOfBytesWritten], DRIVE_REMOTE
                jz      short infecte
                cmp     [ebp+NumberOfBytesWritten], DRIVE_UNKNOWN
                jz      short infecte
                cmp     [ebp+NumberOfBytesWritten], DRIVE_NO_ROOT_DIR
                jnz     loc_40CE7B

infecte:                                ; CODE XREF: sub_40C701+2C5↑j
                                        ; sub_40C701+2CE↑j ...
                mov     edx, offset_41E1BC
                push    edx             ; lpString2
                lea     eax, [ebp+String1]
                push    eax             ; lpString1
                call    lstrcatA
                push    0               ; hTemplateFile
                push    20h             ; dwFlagsAndAttributes
                push    3               ; dwCreationDisposition
                push    0               ; lpSecurityAttributes
                push    1               ; dwShareMode
                push    80000000h       ; dwDesiredAccess
                lea     ecx, [ebp+String1]
                push    ecx             ; lpFileName
                call    CreateFileA
```

Figure 16 : Routine d'infection des périphériques amovibles et réseaux

On peut voir sur la capture d'écran précédente l'appel à la fonction **GetDriveTypeA**, utilisée pour récupérer le type du lecteur en cours d'énumération. IDA permet d'utiliser le nom des constantes pour rendre le code plus lisible, on aperçoit les tests pour chaque type de disque et le saut vers la routine d'infection en cas de disques infectables.

Conclusion

Il reste encore beaucoup à dire sur Sality, mais il faudrait un dossier complet pour pouvoir traiter tous les aspects de cette menace. Pensez donc à bien revoir les accès sur vos partages réseaux et à bien désactiver les exécutions des **autorun.inf** pour éviter les épidémies.

Il existe de nombreuses variantes de Sality. Les variantes Sality.AE et Sality.AF, par exemple, n'utilisent plus RC4, mais de nouvelles techniques pour dissimuler le début du virus : point d'entrée obscur, par exemple, et insertion d'un nombre aléatoire d'octets (aléatoires eux aussi) à la fin du fichier pour que le corps du virus ne soit jamais à la même place par rapport au début de la section.

Sality n'est que la première famille de virus parasites active à l'heure actuelle. La seconde famille de virus parasites « Virut » est très active aussi (nouvelles modifications tous les 3 jours environ) et est en pleine progression au niveau des infections virales. Virut a la particularité d'avoir des partenariats avec d'autres distributeurs de malwares et installent des menaces telles que Zeus (vol d'information) sur vos machines. ■

■ REMERCIEMENTS

Merci à Vyacheslav Zakorzhevsky pour les longues discussions techniques.

AUTOUR DE L'ARTICLE...

■ LE SAVIEZ-VOUS ?

D'après les statistiques de Kaspersky Lab sur l'évolution des *malwares* en 2009...

TENDANCES :

La création de malwares non commerciaux a pratiquement stoppé en 2007/2008.

La majorité des menaces étaient des chevaux de Troie ayant pour but le vol de données, en particulier celles des joueurs en ligne (mots de passe, personnages et leurs objets/argent).

Depuis les 3-4 dernières années, la Chine est devenue la source principale de malwares.

En 2009, le nombre de programmes malicieux dans la collection était de 33,9 millions. Plus du double de l'année 2008.

Kaspersky a identifié environ 15 millions de nouvelles menaces en 2009.

EPIDÉMIES :

En 2009, ont été relevées plusieurs épidémies globales :

Kido (alias Conficker - ver), Sality (virus/ver), Brontok (ver), Mazebat (ver), Parite.b (virus), Virut (virus/bot), Sohanad (ver) et TDSS (rootkit).

L'épidémie la plus importante est sans conteste celle de Kido.ih, avec 3 millions de machines uniques infectées et 1,4 millions de machines infectées par Sality.aa.

MALWARES POUR PLATES-FORMES ALTERNATIVES :

39 nouvelles familles de malwares pour mobiles et 257 nouvelles variantes ont été découvertes en 2009. En comparaison, 30 nouvelles familles et 143 nouvelles variantes furent découvertes en 2008.

2009 a aussi vu l'apparition du premier malware pour Symbian S60 3ème Edition à utiliser un certificat valide, et un autre malware ciblant les distributeurs automatiques de banques, permettant de surveiller les cartes bancaires.