Take a Walk on the Wild Side

Guillaume Arcas[1]

[1] Consultant indépendant - guillaume.arcas@retiaire.org

Résumé. L'année 2007 a été marquée par l'avènement d'une cyber-délinquance d'un genre nouveau : professionnelle par la qualité des ses « outils de production », efficace par le montant de son « chiffre d'affaires », globalisée par son mode de fonctionnement et sa « division du travail ».Au service de cette cyber-délinquance, un botnet d'un genre nouveau : Storm Worm.

Mots clefs : storm worm, botnets, zhelatin, nugache, cybercriminalité.

1 Introduction

En décembre 2006, des violentes tempêtes s'abattirent sur le Royaume-Uni, provoquant de gigantesques inondations, faisant de nombreuses victimes et causant des dégats matériels considérables.

Ces évènements dramatiques donnèrent lieu à des appels aux dons pour venir en aide aux sinistrés. Ils servirent également de support à une campagne virale qui vit se propager, à partir du mois de janvier 2007, ce qui aurait pu n'être que le *énième* maliciel exploitant la générosité des internautes. Ce code malicieux fut “baptisé” Storm Worm.

Comme bon nombre de ces “prédécesseurs”, il enrôlait le PC sur lequel il s'installait dans un *botnet* dont la charge utile consistait à envoyer du Spam en quantités industrielles.

Son analyse devait cependant révéler plusieurs caractéristiques tant techniques qu'organisationnelles qui le distinguèrent rapidement de la grande famille des *bots* de l'époque.

Après quelques mois d'accalmie, Storm Worm – sous la forme d'une variante nommée Zhelatin - refit surface à l'occasion de deux offensives menées respectivement durant l'été et l'hiver 2007.

Parallèlement à ces faits, l'année 2007 fut également marquée par une évolution rapide du paysage de la cybercriminalité informatique. Le “réseau” R.B.N. a ainsi focalisé l'attention des chercheurs et des médias durant cette année.

Cet article retrace l'histoire de Storm Worm / Zhelatin depuis son apparition jusqu'à l'hiver 2007/2008, en présente les principales caractéristiques techniques et explore les liens que l'on peut faire avec la cybercriminalité organisée.

2 Rappels sur les botnets

Le phénomène “botnets” n'est, en cette année 2007, pas nouveau. Sans vouloir retracer leur histoire depuis leur apparition sur les serveurs IRC, une piqure de rappel ne sera pas inutile.

Sur un plan technique, un botnet (ro*bot*s *net*work) peut être vu comme un système distribué tel que le définissent Andrew Tanenbaum et Maarten van Steen dans [1], à savoir “un ensemble d'ordinateurs indépendants donnant l'apparence à leurs utilisateurs de ne former qu'un seul système cohérent”.

La Google Code University [2] précise cette définition : “un système distribué est une application qui exécute un ensemble de protocoles pour coordonner les actions des multiples processus sur un réseau, de telle manière que tous les composants de cette application coopèrent pour réaliser une seule ou une petite série de tâches connexes”.

Un botnet peut ainsi être vu comme un système distribué dont les objectifs sont, au mieux, malicieux, au pire, criminels. Ils se distingue en cela d'autres systèmes distribués plus conventionnels tels que SETI@Home pour ne citer que le plus connu.

Il partage avec ces systèmes distribués traditionnels certains avantages, notamment celui d'être ouvert et évolutif. Ouvert, car chaque composant peut interagir avec ses pairs, évolutif car le système peut être dynamiquement modifié ou reconfiguré en fonction de l'évolution de ses ressources (nombre de composants), de ses besoins et de ses objectifs.

Un botnet se compose ainsi - au moins - des trois sous-systèmes suivants :

- Noeud(s) maître(s) : ce sont les super-utilisateurs du botnet. On classe dans cette catégorie les utilisateurs physiques mais aussi les composants logiques (machines et logiciels) utilisés pour administrer le botnet ;
- Noeuds esclaves : ce sont les composants qui rendent le service et distribuent la charge utile du botnet ;
- Un canal de commande et de contrôle (C&C) utilisé pour gérer le botnet.

  On appellera bot le logiciel installé sur chaque machine du botnet. Cette machine pourra être appelée zombie, agent ou drone. Le botnet est contrôlé par un super-utilisateur appelé botmaster ou bien encore botherder à l'aide d'un canal C&C.

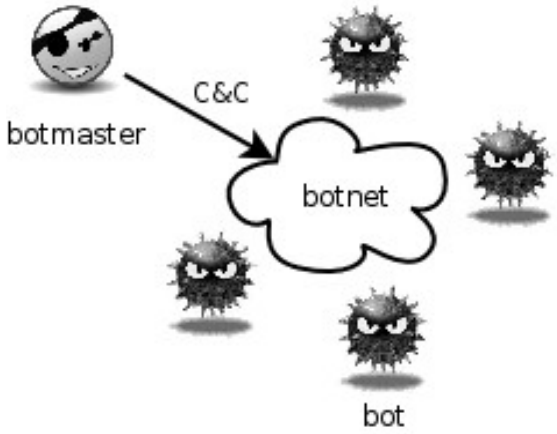

Fig. 1. Schéma fonctionnel simplifié.

3 Le calme avant la tempête, botnets 1.0

Les premiers *bots* furent programmés pour automatiser d'ingrates tâches d'administration sur des gros serveurs publics IRC, comme faire la police parmi les utilisateurs, notamment les moins respectueux de la Netiquette. EggDrop (1993) devint ainsi le bot par excellence. Le terme botnet désigna par la suite tout simplement un réseau de bots (ro*bot net*work) EggDrop oeuvrant de concert.

Les premières versions malicieuses de ces bots furent souvent construites sur la base d'EggDrop ou de clients IRC détournés de leur utilisation première.

Quant à leur charge utile, elle se résumait, la plupart du temps, à lancer contre un serveur IRC une attaque en déni de service pour venger une expulsion mal vécue.

Il fallut attendre 6 ans à partir de la sortie d'EggDrop (soit une éternité à l'échelle d'Internet !) pour que le premier bot à utiliser le protocole IRC comme C&C fasse son apparition (Preety.Park, 1999).

Au même moment des attaques DdoS, dont certaines de grande envergure, frappaient les esprits et certains sites Web, plus connus du grand public que les serveurs IRC considérés comme "*underground*" mais qui n'échappaient pas au phénomène.

Les botnets faisaient ainsi leur entrée dans le paysage de l'insécurité Internet et n'allaient plus en sortir.

GTBot est un bot qui fut lâché dans la nature durant l'année 2000 et rencontra un certain succès. Un bon nombre d'apprentis botmaster se firent la main avec ce bot. GTBot fournissait principalement une version modifiée de mIRC pour Windows et un fichier de configuration dans lequel le botherder n'avait qu'à renseigner l'adresse du serveur IRC et le nom du canal qui hébergeaient le C&C. Un botnet GTBot distribuait ensuite, au choix, du SPAM ou des attaques en déni de service.

Cette architecture ne devait guère changer par la suite, les nouveaux bots se contentant d'améliorer ou d'ajouter des fonctionnalités.

L'année 2002 marqua un premier tournant avec l'arrivée sur le “marché” d'AgoBot puis de SDBot et de leur nombreuse descendance.

AgoBot fut en effet mis à disposition avec son code source et une interface graphique d'aide à la configuration/compilation. Le botnet était ainsi au bout du clic. Autre caractéristique importante d'AgoBot : un code modulaire, bien commenté, de bonne qualité, qui inspira d'autres développeurs et notamment ceux de PhatBot, qui fut le premier bot à utiliser un protocole autre qu'IRC comme C&C.

Moins de deux ans après la sortie d'AgoBot, le phénomène botnet dépassa - et de loin – les virus classiques de par son ampleur et ses effets : les botnets sont à l'origine de plus de 80% du SPAM qui transite quotidiennement sur Internet et il y a derrière chaque attaque DdoS de grande envergure un ou plusieurs botnets.

En quelques années, les botnets s'imposèrent grâce à des technologies éprouvées et à des principes simples : “l'union fait la force” d'une part, un partage intelligent des tâches de l'autre.

Chaque jour apportait son lot de nouvelles variantes de tel ou tel bot, chaque semaine voyait se constituer un nouveau botnet. La riposte consistait alors à identifier les C&C et les neutraliser. Ce n'était pas une chose évidente ni toujours facile, mais on s'en sortait encore et il était somme toute aisé de couper tout ou partie d'un botnet de son botmaster, que ce soit en fermant ses canaux IRC quand ceux-ci transitaient sur des serveurs publics ou en “blacklistant” les machines – souvent elles-mêmes compromises – qui hébergeaient le C&C.

A côté de quelques groupes criminels plus aguerris que les autres, le gros de la troupe des botmasters était constitué d'amateurs plus ou moins éclairés et de francs-tireurs guère organisés. Cela permettait aux services de police de prendre quelques gros poissons au milieu de beaucoup de menu fretin dans leurs filets une à deux fois par an.

Tout aurait pu durer ainsi et longtemps dans le “moins pire” des mondes.

Storm Worm allait, de nouveau, tout bousculer et changer radicalement la donne.

4 Storm Worm

Tout a donc commencé en janvier 2007, peu après le passage sur la Grande-Bretagne de violentes tempêtes.

Une première vague de SPAM se produisit le 18 janvier, suivie d'une seconde le 21, puis de nouvelles les semaines suivantes.

Des millions de courriels porteurs de pièces jointes infectées furent envoyés durant tout le mois de janvier.

Chaque machine infectée rejoignait un botnet, puis se mettait à son tour à envoyer du SPAM en quantités industrielles. A première vue, rien de bien nouveau sous le soleil. A première vue seulement.

Storm Worm – puisqu'il fut baptisé ainsi – est un des premiers bots à utiliser un protocole P2P comme C&C. Tout du moins de manière efficace.

Il y eut bien quelques précédents en la matière : PhatBot, qui proposait un module s'appuyant sur WASTE. Puis Nugache, mais surtout X-Worm, bot diffusé avec son code source comme démonstration qu'un réseau P2P pouvait remplacer très bénéfiquement les canaux IRC. Pourtant les botmasters ne se précipitèrent pas pour autant sur ce protocole. Il faut dire que certaines implémentations se révélèrent défaillantes (Nugache) ou mal adaptée à un grand nombre de bots (PhatBot).

Pour le botmaster, l'utilisation d'un protocole P2P pour le C&C présente un intérêt majeur. Le passage d'une architecture C&C fortement centralisée, ce qui était le cas pour l'IRC, à une architecture distribuée élimine la principale vulnérabilité du botnet : la perte ou la neutralisation du serveur central. Avec la perte des zombies pour cause de désinfection ou de déconnexion, cela constitue en effet la seconde cause la plus fréquente de mortalité pour un botnet. Avec pour conséquence, au mieux l'impossibilité de continuer à vendre sa force de frappe, au pire, l'identification et l'arrestation du pirate.

Pour autant, il ne fallait pas que le remplacement d'un protocole centralisé par un autre se traduise par une diminution de la taille maximale du botnet, taille dont dépend très fortement sa rentabilité et sa valeur financière.

Storm Worm apporta une solution technique à ce problème. Ce ne fut cependant pas sa seule innovation.

5 Zhelatin

Juillet 2007. Quelques mois se sont écoulés depuis les premières campagnes Storm Worm de l'hiver. Si janvier et février furent marqués par l'apparition de ce nouveau maliciel, les mois qui suivirent furent plutôt calmes et l'on aurait pu penser que Storm Worm, comme le furent Nugache et PhatBot/WASTE avant lui, n'avait été qu'une expérimentation. Grossière erreur.

Depuis quelques semaines on notait une recrudescence d'envoi de SPAM d'un type nouveau. Au lieu de vanter les mérites de petites pilules bleues, les pourriels qui commencèrent à inonder les boites aux lettres véhiculaient, en pièces jointes de formats aussi divers que variés (PDF, ZIP, DOC, MP3), des faux conseils d'achat ou de vente d'actions de petites sociétés.

Début août, cette campagne de SPAM connut une accélération importante et les volumes de courriels envoyés explosèrent. De toute évidence un botnet se cachait derrière cette nouvelle campagne.

Storm Worm venait de donner naissance à Zhelatin.

5.1 Architecture globale

Vu de haut, un botnet Zhelatin ressemble à ceci :

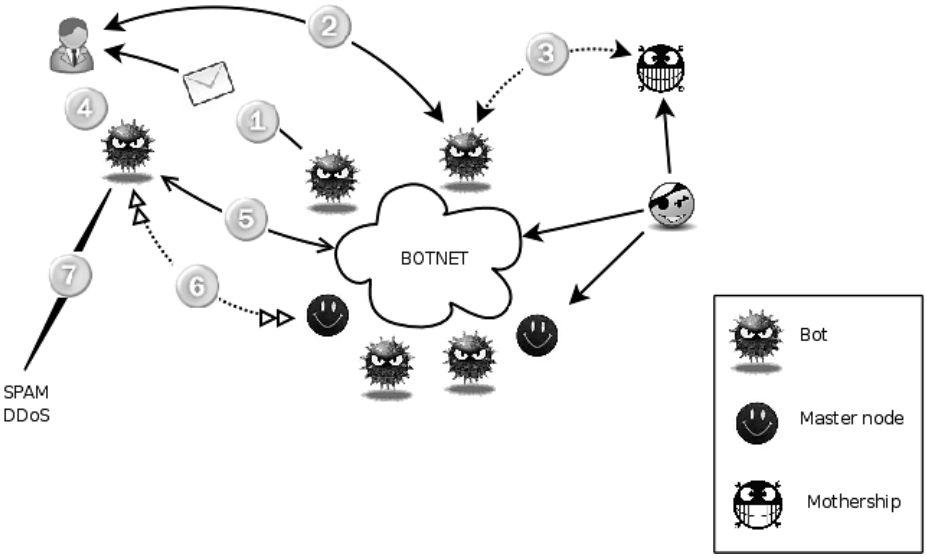

Fig. 2 Architecture globale d'un botnet Storm Worm/Zhelatin.

Nous allons passer en revue chacune des étapes illustrées sur ce schéma.

5.2 Propagation

Commençons donc, logiquement, par le commencement (Etape n°1).

L'appât

Contrairement à la précédente moulure de Storm Worm, Zhelatin fut propagé non en pièce jointe à des courriels mais depuis des sites Web préalablement infectés. Cette méthode présente un premier avantage : éviter le filtrage antivirus, devenu monnaie courante sur les flux SMTP mais - pas encore - sur les flux HTTP.

L'internaute visé par l'attaque reçoit un courriel d'apparence anodine et même assez rustique :

```
Greetings,
We are so happy you joined Cat Lovers.
User Number: 77367718167
Login ID: user3474
Your Temp. Password ID: fj878

Be Secure. Change your Login ID and Password.

Click here to enter our secure server:
http://xxx.yyy.zzz.216/
```

```
Welcome,
New Member Technical Support
Cat Lovers
```

La forme de ce courriel et l'utilisation en clair d'une adresse IP auraient dû à elles-seules attirer l'attention de l'internaute même moyennement sensibilisé à la sécurité. Au regard du nombre de machines qui furent infectées dans les heures et les jours qui suivirent les premiers envois de ces mails piégés, force fut de constater que la curiosité fut la plus forte.

Quelque jours plus tard, une variante du pourriel faisait son apparition, à peine plus travaillée. Tout au plus l'adresse IP était-elle remplacée ou plutôt masquée par un nom de domaine :

```
<html>
<body>
brookmn1@xxxxx.au wants to send you a greeting from
funnygreetings.net.<br>
<br>
To view your card, follow the link below:<br>
<a href="http://w.x.z.y.z/">funnygreetings.net</a><br>
<br>
Sincerly,<br>
funnygreetings.net<br>
</body>
</html>
```

Les sujets varièrent ainsi au fil des jours. La rapidité de changement des courriels devint une caractéristique constante des campagnes de recrutement Zhelatin au point que l'on pourrait presque parler de polymorphisme dans leur contenu.

Après quelques essais de textes plus complexes, de mails plus ou moins longs, on nota une stabilisation autour d'un schéma simple : “une accroche aguicheuse, un texte très court, un lien” sur le modèle Funny Greetings. Furent ainsi “mis à contribution” la curiosité du public pour les vidéos plus ou moins scnadaleuses ainsi que les “grands” noms du Web 2.0 dont YouTube :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
<html>
<body>
If your mom sees this she this video of you she is
gonna freak. go look at it...
<a href="http://a.b.c.d/">http://www.youtube.com/watch?
v=YpgxJEN1PbY</a>
</body>
</html>
```

Tous ces messages poursuivent le même et unique objectif : inciter au clic.

A partir du mois de décembre 2007, Zhelatin met à profit chaque évènement du calendrier – Noël, jour de l'An, fête du travail, Premier avril – pour lancer une nouvelle offensive.

Hameçonnage

Si l'internaute cède à la curiosité et mord à l'hameçon, son navigateur est dirigé vers une page piégée (Etape n°2).

Ce piège prend la forme d'une page d'apparences diverses et variées et faussement anodines, comme dans cet exemple :

```
<br>Your Download Should Begin Shortly.
If your download does not start in approximately 15
seconds,

you can <a href="/video.exe">click here</a> to launch
the download and then press Run.
```

Si l'on creuse un peu, cependant, on découvre la partie immergée de l'iceberg :

```
<Script Language='JavaScript'>

function xor_str(plain_str, xor_key){
          var xored_str = "";
          for (var i = 0 ; i < plain_str.length; ++i)
          xored_str += String.fromCharCode(xor_key ^
          plain_str.charCodeAt(i));
          return xored_str;
}

var plain_str =
"\x49\x64\x63\x64\x63\x64\x63\x64\x63\x64\x63\x64\x63

                    <snipped>

\x49\x64\x63\x64\x63\x64\x63\x64\x63\x64\x63\x64\x63" ;

var xored_str = xor_str(plain_str, 77);
document.write(xored_str);

</script>
```

On se trouve en présence de code JavaScript obfusqué, la protection offerte par cette obfuscation n'étant pas suffisamment solide pour que l'on puisse parler de blindage.

Ce script décode la chaîne de caractères plain_str en lui appliquant un simple XOR à l'aide de la fonction xor_str(). Cette fonction prend comme arguments cette chaine et une clef de déchiffrement. La chaine décodée est ensuite stockée dans la variable xored_str puis affichée dans le document courant.

Cette technique présente un double avantage : tromper les éventuels filtres qui ne verront qu'un script contenant une chaine hexadécimale et leurrer l'internaute qui aurait eu la curiosité d'afficher le code source de la page du site qu'il visite.

Autre point : la clef de déchiffrement est dynamiquement changée à chaque affichage de la page. Il est ainsi vain d'espérer filtrer le script sur une empreinte ou une somme de contrôle.

Si l'on exécute ce script et qu'on affiche le résultat, on obtient un script dont l'analyse mériterait à elle-seule un article complet. Nous allons donc résumer et n'examiner que les poinst marquants.

Le script comporte deux fonctions dF() et cf() qui sont des décodeurs. On leur passe un chaîne obfusquée et on en ressort... du code JavaScript ! On a donc un script obfusqué qui lui-même contient du code obfusqué ainsi que ses propres fonctions de décodage.

Le script reconstitué (c'est-à-dire celui qui s'éxécute dans le navigateur) teste la présence et tente d'exploiter au moins trois failles : une faille WMF, une faille WinZIP et une faille QuickTime.

Des fonctions spécifiques à chacune de ces vulnérabilités sont présentes dans le script. L'exploitation de ces failles permet l'exécution arbitraire de code, en l'occurence le téléchargement et le lancement d'un binaire.

Dans un cas – exploitation de la faille QuickTime – il y a une étape intermédiaire qui consiste à télécharger un fichier QTL (QuickTime PlayList) malformé. Ce fichier est chargé depuis une page PHP appelée dans le script puis lancée dans un objet ActiveX :

```
<param name="src" value="qt.php">
 var qt = new ActiveXObject ('Quick' + 'Time.Qu' +
'ickTime');
```

Certaines fonctions “dangereuses” - au sens où elles pourraient attirer l'attention ou être filtrées – sont découpées puis reconcaténées :

```
    new ActiveXObject ('Quick' + 'Time.Qu' +
'ickTime');
    new ActiveXObject ('WebVi' + 'ewFol' + 'de' + 'rIc'
+ 'on.WebVi' +  'ewFol' + 'derI' + 'con.1');

    CreateObject (a, "Microso" + "ft.XM" + "LHT" +
"TP");

    CreateObject (a, "MSX" + "ML2.Se" + "rverXM" +
"LHT" + "TP");

    CreateObject (a, "ADODB.Str" + "eam");

    CreateObject (a, "WSc" + "ript.Sh" + "ell");

    CreateObject (a, "Shel" + "l.Ap" + "pl" + "icati" +
"on
```

En conclusion, si le navigateur utilisé est vulnérable à l'une de ces failles, l'infection du poste de l'internaute est automatique.

Si ce n'est pas le cas, il faut alors miser sur la naïveté de l'internaute qui chargera et lancera le binaire pointé par le lien.

Il faut d'ailleurs croire que cette dernière méthode s'est révélée suffisamment efficace car l'utilisation de codes JavaScript fut abandonnée au bout de quelques semaines. La structure de la page de téléchargement se résuma à un simple lien vers le binaire.

Pour “légitimer” son installation, ce dernier fut présenté comme un utilitaire permettant de suivre les résultats de la Ligue de football américaine, de partager des fichiers ou encore comme une carte de voeux électronique.

Hébergement des binaires, Fast-Flux avec et sans DNS

Pour distribuer les binaires infectés, le botnet utilise une architecture d'hébergement composée de proxies derrière lesquels se cachent des “motherships” (Etape n°3).

Ce mode de fonctionnement est très fortement inspiré des architectures de type Fast-Flux utilisées pour porter des attaques contre les navigateurs Web. Une telle architecture s'appuie sur des noms de domaines associés à des TTL très court ou égal à 0.

Cela garantit qu'aucune requête n'est cachée par les resolvers DNS utilisés, ce qui permet de changer à chaque résolution l'adresse IP attribuée à un enregistrement DNS.

Le schéma suivant illustre ce mode de fonctionnement.

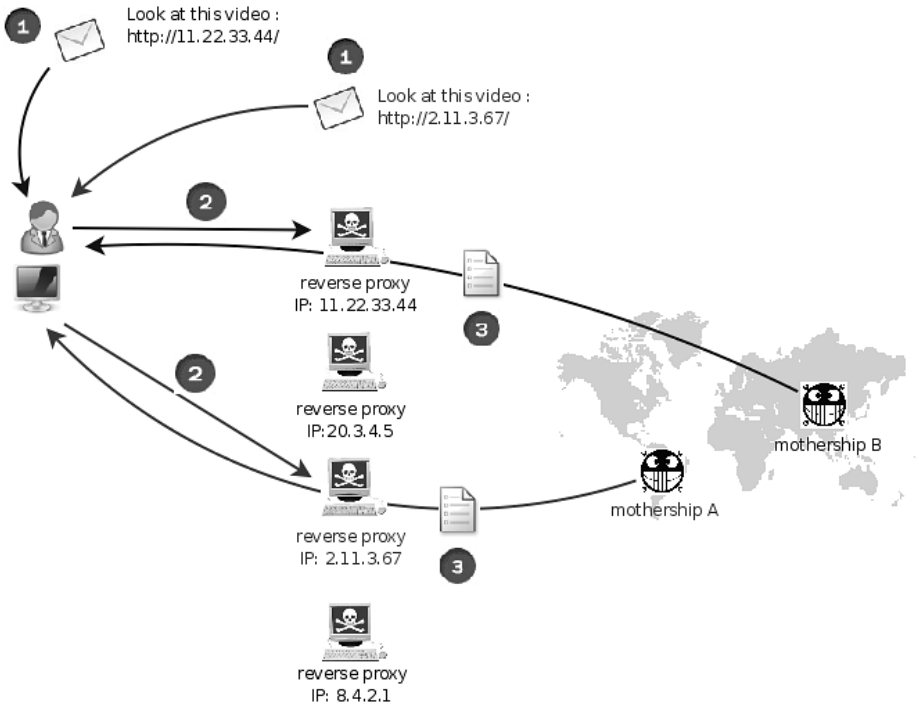

Fig. 3 Architecture d'hébergement des binaires.

On retrouve ce mode de fonctionnement, mais sans utilisation du DNS, dans les courriels d'hameçonnage : les adresses IP qu'ils contiennent changent très fréquemment d'un courriel à l'autre, ce qui rend difficile, sinon illusoire, le filtrage de ces adresses.

Inutile de préciser que les proxies sont des machines elles-mêmes préalablement compromises et membres du botnet. Dans de très nombreux cas, pour ne pas dire tous, le logiciel utilisé pour proxifier les flux est Nginx.

Le botnet se mit à fonctionner mode Fast-Flux DNS à partir du mois de décembre 2007. En lieu et place d'adresses IP les courriels contenaient alors des liens vers des domaines préalablement déposés par les attaquants auprès de registrars peu scrupuleux ou peu réactifs.

Infection

Le binaire chargé et exécuté installe (Etape n°4) sur le poste compromis un client P2P qui assure, entre autres, la connexion au canal C&C du botnet. Ce client a des capacités de téléchargement de fichiers. Il essaie, autant que faire ce peut, de désactiver le pare-feu Windows mais aussi des pare-feux personne comme Zone Alarm et d'antivirus, afin de se protéger contre la désinfection d'une part, et s'assurer une connectivité sans entraves de l'autre.

Toujours pour sa propre sécurité, le processus n'apparait pas dans la liste des processus ni depuis le Gestionnaire de processus Windows, ni depuis un outil tel que What's Running. Par contre, le nom du process apparait lorsque l'on utilise la commande netstat. Cependant, une fois activé, le "shell" Windows devient impossible à lancer.

Parmi les fichiers installé se trouve un fichier de démarrage .ini qui contient une liste d'adresses IP (plusieurs centaines). Ce sont des points d'entrée dans le réseau P2P, que le bot va contacter à son lancement pour découvrir son canal de C&C.

5.3 Fonctionnement du canal de C&C

Le canal de C&C de Storm Worm/Zhelatin est une des caractéristiques les plus marquantes de ce bot.

Il repose sur un réseau P2P et utilise le protocole OverNet/eDonkey.

On distingue deux réseaux, en fait : un réseau utilisé pour localiser les ressources dont le bot a besoin, et un réseau utilisé pour prendre ses ordres. Le premier fonctionne en mode P2P, le second, plus classiquement, en mode client/serveur.

Réseau de localisation

Dans sa version originale, le protocole OverNet permet de localiser des ressources (c'est-à-dire des fichiers et les machines où les trouver) puis d'y accéder. La localisation des ressources se fait à l'aide de messages UDP et le téléchargement sur des flux TCP.

OverNet à la sauce Zhelatin reproduit peu ou prou ce mode de fonctionnement (Etape n°5).

En premier lieu, le bot s'annonce sur le réseau. Il contacte, de manière aléatoire, une dizaine de pairs dont les adresses IP sont contenues dans le fichier de démarrage.

Le client se déclare auprès de ces noeuds à l'aide d'un message UDP/OverNet de type

Publicize. La partie Payload de ce message contient un Hash qui l'identifie de manière unique sur le réseau, son adresse IP publique et le port source sur lequel il écoute à cette adresse.

Les noeuds disponibles renvoient au client un message Publicize ACK. Le client construit ensuite son réseau en contactant de nouveaux

noeuds et en étant contacté par d'autres, et enrichit son fichier de démarrage. Ainsi, en quelques heures, ce fichier peut contenir jusqu'à plusieurs milliers d'adresses IP.

Si le client est incapable de déterminer son IP publique, l'adresse qu'il annonce dans Publicize est 0.0.0.0. Il émet ensuite un message de type IP Query grâce auquel il peut demander à un pair de lui renvoyer via un message IP Query Answer son IP publique. Le bot s'accommode ainsi très bien du filtrage réseau et du NAT !

Les connexions entre pairs se font à l'aides de messages de

type Connect et ConnectReply. Par ces derniers des pairs peuvent s'échanger des listes d'adresses Ips et des Hashes associés.

Lorsque le client obtient une liste de pairs dans un message ConnectReply, il émet des messages Publicize vers ces pairs et s'enregistre auprès d'eux.

Quand un pair souhaite localiser un autre pair (ou un fichier) il émet des messages de type Search, qui contiennent le Hash de l'objet (pair ou fichier) recherché. Il peut aussi contenir le nombre de réponses souhaité (Search Range). Les pairs du réseau qui n'ont pas la réponse envoient en retour un message SearchNext qui contient une liste de pairs susceptibles d'avoir la réponse. Si un pair dispose de la réponse, des messages Search Info et Search End sont échangés. Les résultats sont échangés à l'aide de messages SearchResult. Ces messages contiennent le nom des objets disponibles sur le pair (par exemple: "16832.mpg;size=85739;").

L'activité du bot sur le réseau de localisation est permanente : : tant que le client est actif, les échanges de messages avec ses pairs se font en continu. Cela peut constituer une première signature du ver : sur une heure d'observation d'un poste infecté, la part du protocole UDP dans le trafic réseau constatée s'est en effet élevée à plus de 50% !

Réseau de contrôle

Le réseau de contrôle est composé de machines qui ne sont contactées par le bot qu'en TCP à l'excpetion de tout autre protocole, même si ces machines font partie du botnet (Etape n°6).

Le bot localise ces machines grâce au réseau P2P. Puis il s'authentifie auprès de l'une d'elles comme illustré ci-dessous.

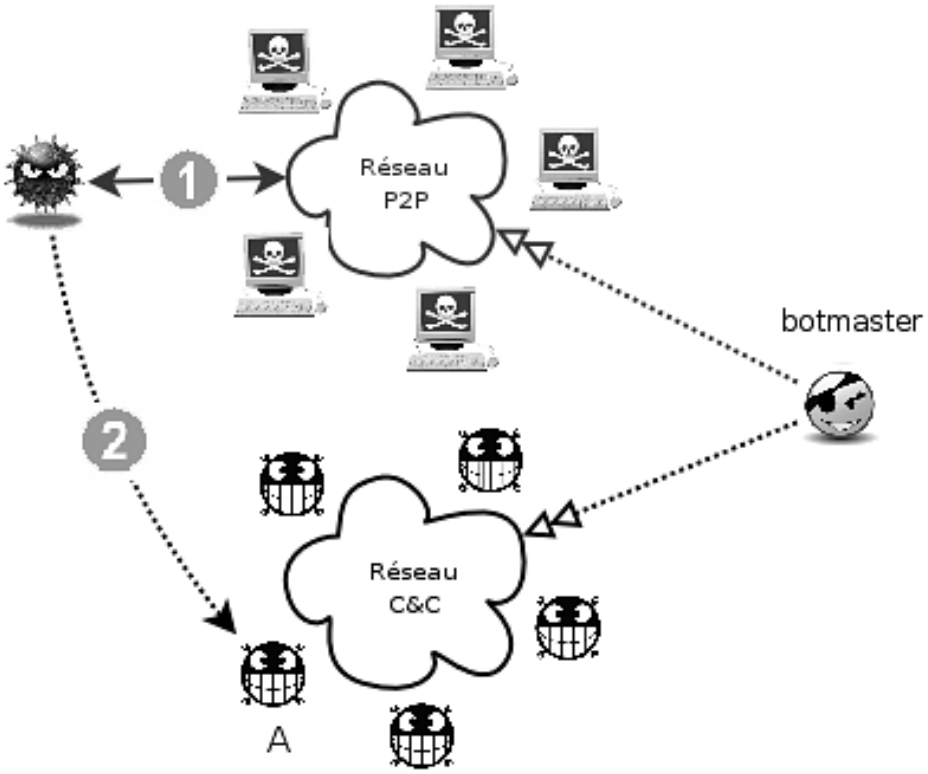

Fig. 4 Localisation d'un contrôleur.

On note qu'à un instant donné, un bot ne dépend que d'un contrôleur, et qu'il y a donc tout lieu de penser qu'un contrôleur ne commande qu'une partie du botnet. Ainsi, la perte d'un contrôleur ne met pas en péril l'ensemble du botnet. Cela permet aussi de morceler le botnet et de le “louer” par petits morceaux. A intervalles réguliers, le bot change de contrôleur. Dans une même journée, il ne dépend ainsi que durant quelques heures de la même machine.

C'est depuis cette machine que le bot prend ses ordres.

5.4 Charge utile

Parmi les éléments échangés durant une session entre un bot et son contrôleur on trouve les modèles (template) des courriels que le bot devra envoyer. Ces courriels sont de deux types : hameçonnage, afin d'augmenter et maintenir la taille du botnet, et SPAM, qui constitue la charge véritablement utile du botnet (Etape n°7).

C'est en effet la capacité à envoyer du SPAM qui conditionne et définit la valeur marchande du botnet.

A côté du SPAM le botnet a des capacités d'attaques DdoS. L'importance de cette charge utile est cependant considérablement moindre que celle du SPAM.

6 Zhelatin et la cybercriminalité

En une petite année Storm Worm / Zhelatin est devenu “le” botnet par excellence. Bien que perfectible, il réunit des qualités qui devraient l'inscrire durablement dans le paysage de l'insécurité. Il restera comme celui qui a démontré la faisabilité, la fiabilité et la stabilité d'un canal C&C en mode P2P.

Mais au delà de ces seules considérations techniques, Zhelatin marque aussi un tournant pour la cybercriminalité.On peut identifier trois grandes phases dans l'histoire récente de la virologie et de la cybercriminalité.

Une première époque, marquée par l'existence de développeurs de virus isolés ou bien travaillant en petits groupes informels, essentiellement motivés par une soif de reconnaissance, quitte à ce qu'elle se rapporte à une capacité de nuisance.

En terme d'organisation du travail, chacun de ces développeurs ou groupes travaillaient plus ou moins dans son coin. La recherche d'une faille, puis la traduction de celle-ci en exploit, enfin l'utilisation de ce dernier dans un code opérationnel, toute la chaine de fabrication du maliciels était contrôlée par le même ensemble.

Puis vint le temps des premiers virus développés et dispersés à des fins lucratives, avec l'avènement du courrier électronique. Ce fut le belle époque des mass-mailing worms et autres usines à Spam. Au même moment, des virus commencèrent à être produits indépendamment de leur exploitation.

Les botnets s'inscrivent dans cette évolution. Leur disponibilité sous forme de code source donna naissance à de nombreuses vocations. Dans le même temps la division du travail se poursuivit, de même que se créèrent des marchés virtuels pour des biens immatériels d'un type un peu spécial : failles, exploits, botnets, etc.

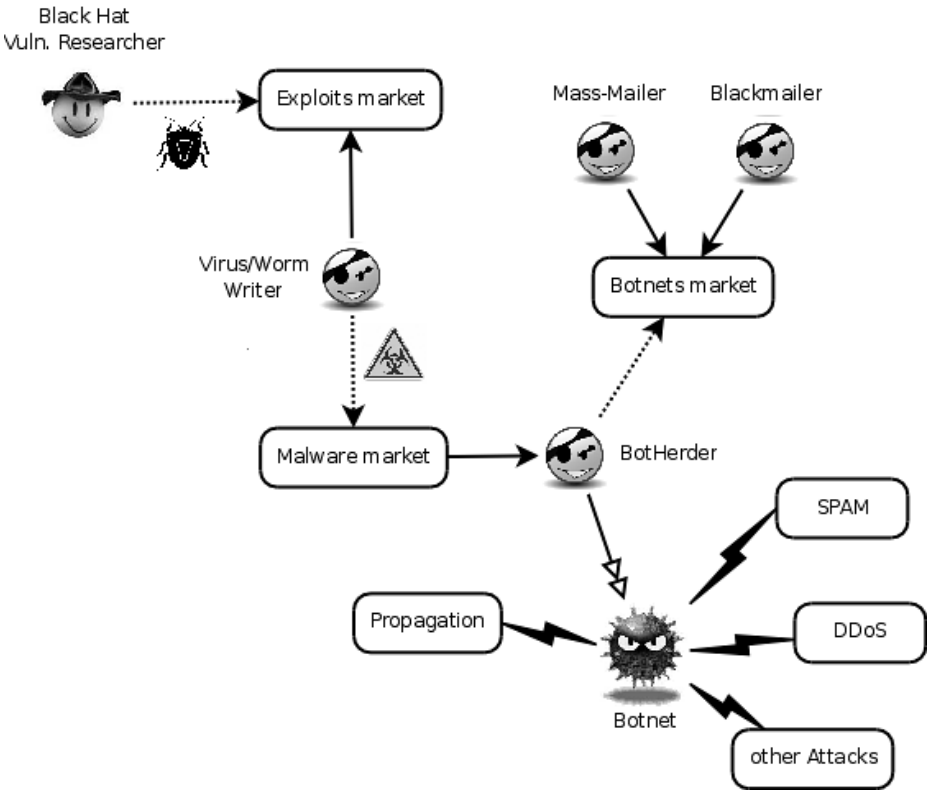

Fig. 5 Les marchés de l'insécurité.

Le marché de l'insécurité pouvait alors paraitre très morcelé. Zhelatin a quelque peu rebattu les cartes en ce sens où, non content d'être un nouveau botnet, on peut presque parler d'écosystème Zhelatin, dont le botnet ne serait qu'un composant.

Cet écosystème trouve ces origines très vraisemblablement à l'Est.

Plusieurs indices le laissent tout du moins penser.

Tout d'abord, certains codes JavaScript utilisés dans les premières campagnes, rappellent étrangement des parties du kit Mpack, développé par des pirates russes. Ensuite, Nginx, logiciel utilisé comme reverse proxy, a lui aussi des origines russes (sans que cela n'implique ses développeurs !). Les domaines utilisés par Zhelatin durant la campagne de l'hiver 2007 ont été enregistrés auprès d'un registrar russe une fois encore. Enfin, les quelques rares fois où il a été possible de remonter jusqu'aux machines qui hébergeaient les binaires (les “motherships”), on est très souvent tomber sur le désormais fameux réseau R.B.N..

Le nom même de Zhelatin témoigne des origines slaves de ce bot.

Ce maliciel s'est en tout cas taillé une bonne place dans l'Internet durant l'année 2007 et tout porte à croire qu'il ne disparaitra pas de si tôt.

Une chose est tout particulièrement frappante : la qualité du cycle de développement du botnet.

Tout laisse à croire que les équipes derrière Zhelatin s'inspirent, consciemment ou non, de méthodes telles qu'OODA (Observation – Orientation – Decision – Action) ou PDCA (Plan – Do – Check – Act). Cela expliquerait en tout cas la très forte réactivité du botnet, et l'abandon, notamment, des méthodes d'infection par JavaScript. Sans parler de l'adaptation permanente des attaques aux évènements et au calendrier.

Certaines évolutions suscitent des inquiétudes : dans les versions les plus récentes du bot, les reverse-proxy servent également de resolvers DNS. Ils gèrent entre autres les zones des domaines déposés auprès de registrars peu regardants ou peu réactifs, utilisés pour propager les binaires.

Mais ces resolvers, quand on les interroge à l'aide de dig ou de nslookup, se comportent comme n'importe quel resolver DNS. La tentation doit être grande, pour les pirates, de faire pointer chaque membre du bot vers ces machines et modifier leur configuration réseau pour n'utiliser que ces resolvers. Les attaques de phishing, les détournements de flux, en seraient grandement facilités.

References

1. Andrew S. Tanenbaum, Maarten van Steen : Distributed Systems, Principles and Paradigms.
2. Google Code University - http://code.google.com/edu/parallel/index.html