Recherche de vulnérabilités dans les drivers 802.11 par techniques de fuzzing

Laurent BUTTI et Julien TINNES – France Télécom / Orange Division R&D

firstname dot lastname at orange-ftgroup dot com

Agenda

- Problématiques de sécurité 802.11
- Le fuzzing 802.11
- Conception et développement d’un fuzzer 802.11
- Quelques failles découvertes
- Exemple d’exploitation d’une faille driver 802.11

Ce que l'on savait…

- Le Wi-Fi rend difficile la protection périmétrique de l'entreprise
  - Infrastructures réseau Wi-Fi friables (WEP, WPA mal utilisé)
  - Infrastructures réseau avec point d'accès illégitimes ou mal configurés
- Mais aussi la sécurité du poste client
  - Les points d'accès illégitimes dans les zones publiques (conférences, hot spots…)
  - Les faux points d'accès attaquant les clients [KARMA]
  - L'injection de trafic dans les communications clients [WIFITAP, AIRPWN]
- Et tout ca, bien entendu difficilement détectable…

Ce dont on se doutait…

- Erreurs de programmation dans les drivers 802.11
  - Code développé en C/C++
  - Nombreux constructeurs de chipsets ⇨ Nombreux développeurs ⇨ Hétérogénéité de la qualité des développements
    - Intégrateurs ⇨ Packages de drivers obsolètes
- Erreurs de programmation intéressantes à exploiter
  - Exécution de code arbitraire en mode ring0 (kernel)
    - Contournement des fonctions de sécurité de type PFW, HIPS, AV…
  - Accessibles à distance par la voie radioélectrique
    - Sans (forcément) avoir besoin d'être associé à un point d'accès malveillant
    - Quels que soient les mécanismes de sécurité niveau MAC (WPA/WPA2…)
- Plutôt intéressant, non ?!?

Ce qui arriva…

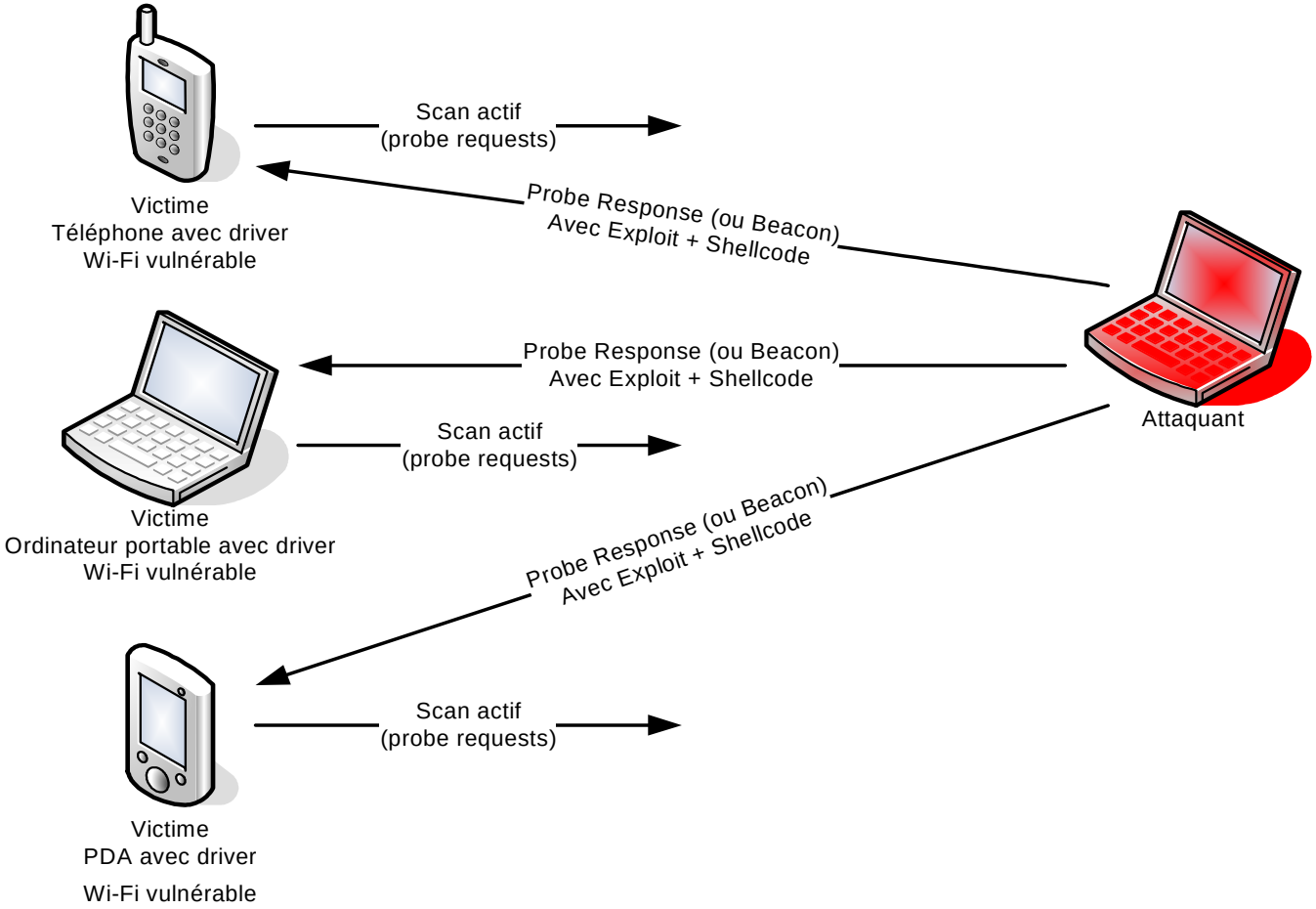
- Premières publications à BlackHat US 2006 par Johnny Cache et David Maynor [CACHE-MAYNOR]

- Month of Kernel Bugs de novembre 2006 [MOKB]
  - Apple Airport 802.11 Probe Response Kernel Memory Corruption (OS X)
  - Broadcom Wireless Driver Probe Response SSID Overflow (Windows)
  - D-Link DWL-G132 Wireless Driver Beacon Rates Overflow (Windows)
  - NetGear WG111v2 Wireless Driver Long Beacon Overflow (Windows)
  - NetGear MA521 Wireless Driver Long Rates Overflow (Windows) (*)
  - NetGear WG311v1 Wireless Driver Long SSID Overflow (Windows) (*)
  - Apple Airport Extreme Beacon Frame Denial of Service (OS X)

- Mais aussi sous Linux…
  - Madwifi stack-based overflow (*)
    - Potentiellement toutes les distributions Linux non patchées avec chipset Atheros

(*) failles découvertes par notre fuzzer

Schéma d'une attaque



- 802.11 exploits a.k.a. 0wn3d par une trame 802.11 ! ;-)

Recherche de vulnérabilités

- Code source fermé
  - Tests en boite noire
  - Reverse engineering
- Code source ouvert (ou disponible)
  - Tests en boite blanche et/ou boite noire
  - Audit du code
- Le reverse engineering peut représenter un travail considérable
  - En particulier lorsque l'on ne sait pas où chercher
- L'audit de code est assujetti à la disponibilité du code source !
- Les tests en boite noire sont pertinents que l'on ait le code source ou pas…

Fuzzing ? (1/2)

- Terme tout aussi difficile à traduire qu'à définir !

- Définitions
  - Le fuzzing est une technique pour tester des logiciels. L'idée est d'injecter des données aléatoires dans les entrées d'un programme. Si le programme échoue (par exemple, en crashant ou en générant une erreur), alors il y a des défauts à corriger. [WIKIPEDIA]
  - Fuzz Testing or Fuzzing is a Black Box software testing technique, which basically consists in finding implementation bugs using malformed or semi malformed data injection in a automated fashion. [OWASP]

- Point commun
  - Technique de tests logiciels pour découvrir des erreurs de programmation

Fuzzing ? (2/2)

- Repose sur l'approche meilleur rapport qualité / prix !
  - Coûteux en temps de trouver de nouvelles vulnérabilités par reverse engineering
  - Peu complexe de développer un fuzzer « basique »
  - La plupart des vulnérabilités découvertes sont des erreurs de programmation triviales à éviter
- Mais le fuzzing ne pourra pas aider dans la découverte d'erreurs de programmation complexes
  - Manque de temps par rapport à l'espace des tests possibles
  - Gestion des protocoles à états (et états dans les implantations logicielles)

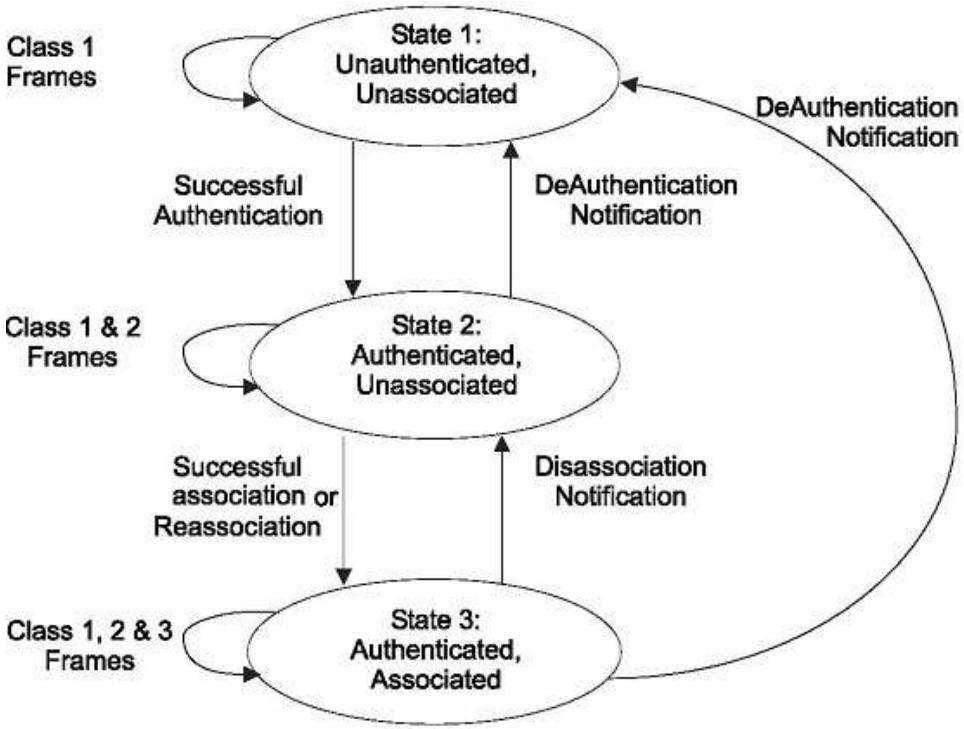
Quelques exemples de fuzzing…

- Article MISC 23
  - « Recherche de vulnérabilités à l'aide du fuzzing »
- Month of Browser Bugs et Month of Kernel Bugs
  - La plupart des failles découvertes par fuzzing
- En particulier le **`fsfuzzer`** de LMH [FSFUZZER]
  - Basique mais s'est révélé très efficace !
- Quelques fuzzers Open Source
  - SPIKE (Immunity): multi-purpose fuzzer [SPIKE]
  - PROTOS suite (Oulu University): SIP, SNMP… [PROTOS]
- Une liste de fuzzers est disponible sur
  - http://www.infosecinstitute.com/blog/2005/12/fuzzers-ultimate-list.html

Fuzzing 802.11 (1/3)

- La norme 802.11 est vaste
  - Plusieurs types de trames (management, data, control)
  - Transport de nombreuses informations de « signalisation »
    - Débits, canal, nom de réseau, capacités cryptographiques, capacités propriétaires…
- Ses extensions le sont (seront) aussi
  - 802.11i pour la sécurité, 802.11e pour la QoS…
  - 802.11w, 802.11r, 802.11k…
- Complexité ++ ⇨ Lignes de code ++ ⇨ Erreurs ++

Fuzzing 802.11 (2/3)



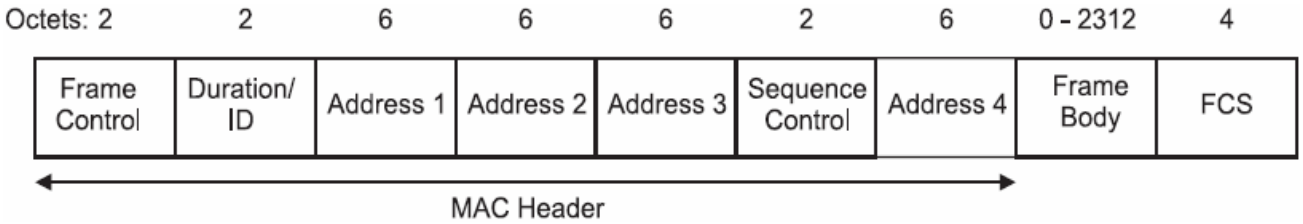
- Les différents états 802.11 sont fuzzables
  - Etat 1 : non authentifié, non associé
  - Etat 2 : authentifié, non associé
  - Etat 3 : authentifié, associé

Fuzzing 802.11 (3/3)

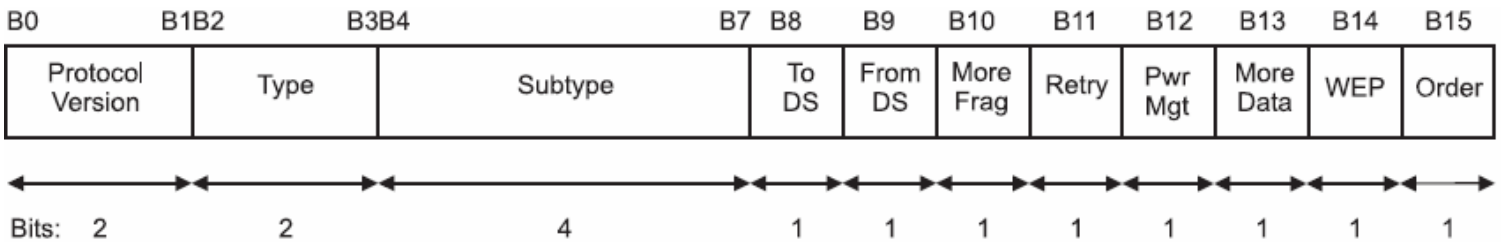
- Choix de la technique d'envoi des trames
  - Mode monitor ou master ?
    - Si fuzzing uniquement de l'état 1 alors mode monitor suffisant
- Choix du fuzzing de l'état 1
  - Le plus simple… mais aussi le plus prometteur !
- Développement des fonctions de pré-calcul des trames
  - En fonction du standard 802.11

Aperçu de trames 802.11

- MAC frame format

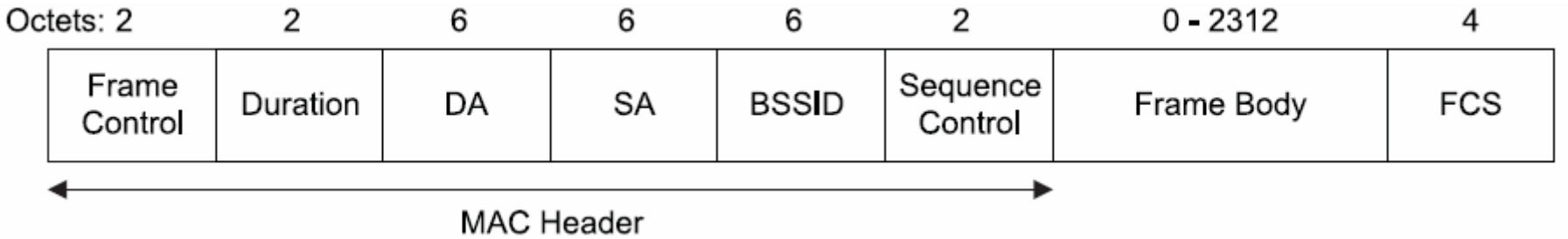


- Frame Control définit les couches supérieures (frame body)



Aperçu de trames 802.11

- Management frame

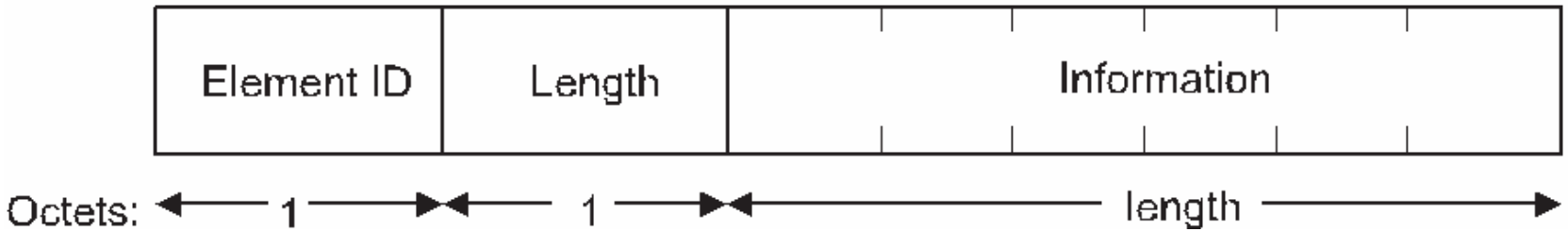


Aperçu de trames 802.11

- Beacon / Probe Response

Order	Information	Notes
1	Timestamp	
2	Beacon interval	
3	Capability information	
4	SSID	
5	Supported rates	
6	FH Parameter Set	The FH Parameter Set information element is present within Beacon frames generated by STAs using frequency-hopping PHYs.
7	DS Parameter Set	The DS Parameter Set information element is present within Beacon frames generated by STAs using direct sequence PHYs.
8	CF Parameter Set	The CF Parameter Set information element is only present within Beacon frames generated by APs supporting a PCF.
9	IBSS Parameter Set	The IBSS Parameter Set information element is only present within Beacon frames generated by STAs in an IBSS.
10	TIM	The TIM information element is only present within Beacon frames generated by APs.

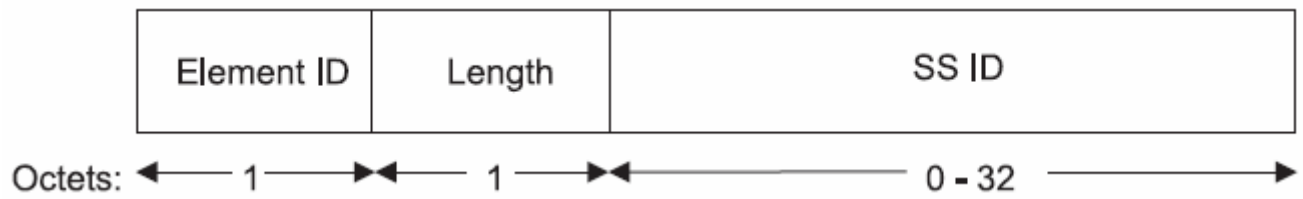
Aperçu de trames 802.11

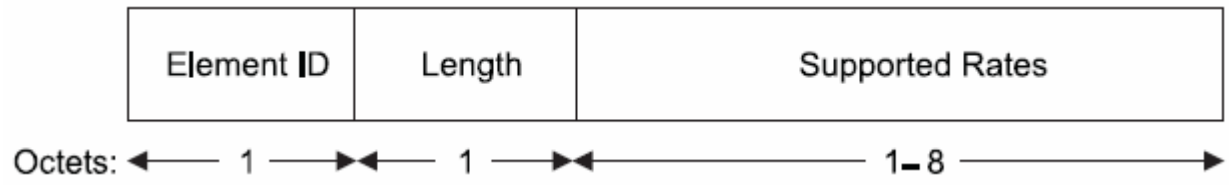


Information element	Element ID
SSID	0
Supported rates	1
FH Parameter Set	2
DS Parameter Set	3
CF Parameter Set	4
TIM	5
IBSS Parameter Set	6
Reserved	7–15
Challenge text	16
Reserved for challenge text extension	17–31
Reserved	32–255

Aperçu de trames 802.11

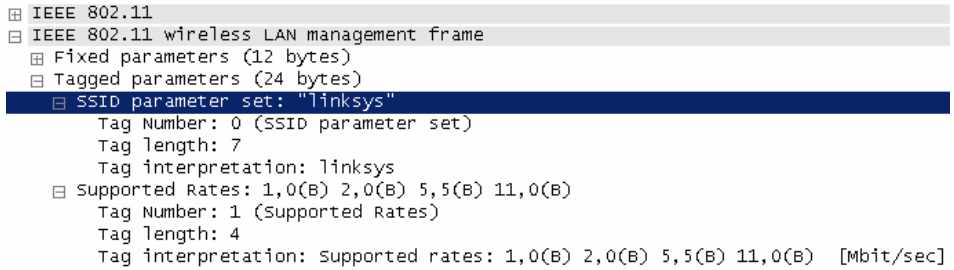
- Quelques Information Elements





Fuzzer l'Information Element

- Information Element
  - Champ optionnel des trames de management
  - De la forme : Type, Length, Value



- On sent alors tout l'intérêt de fuzzer l'Information Element !
  - Certains IE sont souvent de longueur fixe ou maximale
  - On imagine bien alors les erreurs de programmation
    - On attribue un buffer statique d'une longueur fixe prédéfinie (en fonction de ce qui est décrit dans la norme ou de façon arbitraire)
    - On lit sur la trame 802.11 le champ « length »
    - Puis on copie le contenu de « value » dans le buffer statique
    - On déborde alors sur la pile ou le tas

Quelques mots sur le fuzzer 802.11…

- « Tentative » d'optimisation des tests à lancer
  - Sélection des tests en fonction de l'état de l'art des vulnérabilités découvertes
  - Sélection des tests en fonction des Information Elements les plus populaires (SSID, RATES…)
  - Sélection des tests en fonction des bordures à tester (longueur maximale +/- 1, etc…)
- Trames « invalides » vs. trames « valides »
  - De quel point de vue ? De la norme ? Du driver ?
  - Les trames valides au sens 802.11 peuvent être invalides pour le driver

Quelques mots sur le fuzzer 802.11…

- Certains information elements sont complexes
  - WPA/RSN (sécurité), WMM (QoS), IE propriétaires (Atheros, Cisco…)
    - Nécessité d'avoir des générateurs spécifiques
  - Les tester de manière aléatoire ne peut être efficace
    - Matching au minimum sur les premiers octets (OUI)
- Forcer le scan sur l'équipement à fuzzer
  - Obliger le driver à analyser les trames probe responses
    - Netstumbler sous Windows
    - `iwlist` sous Linux (`SIOCSIWSCAN`)

Quelques mots sur le fuzzer 802.11…

- Saturation de la voie radioélectrique avec des Beacons ET des Probe Responses
  - Durant la durée du test et ainsi de suite…
  - But du jeu : être certain que le driver analyse la trame à tester

- Mécanismes de détection de bugs déclenchés
  - Pas de réponse sur les echo requests
    - Détecter un BSOD
  - Analyse des logs kernel
    - Détecter un OOPS
  - Plus d'activité 802.11 de la carte cliente
    - Détecter un dysfonctionnement (BSOD, OOPS, …)

Fuzzing 802.11 avec scapy

- `fuzz()` permet de générer des valeurs aléatoires pour les champs non fournis

- Fuzzer les IEs dans des beacons ?
  - `frame=Dot11(proto=0,FCfield=0,ID=0,addr1=DST,addr2=BSSID, addr3=BSSID,SC=0,addr4=None)/Dot11Beacon(beacon_interval= 100,cap="ESS")/Dot11Elt()`
- Fuzzer les SSIDs dans des beacons ?
  - `frame=Dot11(proto=0,FCfield=0,ID=0,addr1=DST,addr2=BSSID, addr3=BSSID,SC=0,addr4=None)/Dot11Beacon(beacon_interval= 100,cap="ESS")/Dot11Elt(ID=0)`
- Fuzzer de manière aléatoire ?
  - `frame=Dot11(addr1=DST,addr2=BSSID,addr3=BSSID,addr4=None)`

- Puis `sendp(fuzz(frame))`

Etat de l'art : Vulnerabilités 802.11 Driver/Parser

CVE-2007-1218 (PARSER)	Off-by-one buffer overflow in the parse_elements function in the 802.11 printer code (print-802_11.c) for tcpdump 3.9.5 and earlier allows remote attackers to cause a denial of service (crash) via a crafted 802.11 frame. NOTE: this was originally referred to as heap-based, but it might be stack-based.
CVE-2007-0933 (DRIVER/WIN)	Long TIM Overflow
CVE-2007-0686 (DRIVER/WIN)	The Intel 2200BG 802.11 Wireless Mini-PCI driver 9.0.3.9 (w29n51.sys) allows remote attackers to cause a denial of service (system crash) via crafted disassociation packets, which triggers memory corruption of "internal kernel structures," a different vulnerability than CVE-2006-6651. NOTE: this issue might overlap CVE-2006-3992.
CVE-2007-0457 (PARSER)	Unspecified vulnerability in the IEEE 802.11 dissector in Wireshark (formerly Ethereal) 0.10.14 through 0.99.4 allows remote attackers to cause a denial of service (application crash) via unspecified vectors.
CVE-2006-6651 (DRIVER/WIN)	Race condition in W29N51.SYS in the Intel 2200BG wireless driver 9.0.3.9 allows remote attackers to cause memory corruption and execute arbitrary code via a series of crafted beacon frames. NOTE: some details are obtained solely from third party information.
CVE-2006-6332 (DRIVER/LIN)	Stack-based buffer overflow in net80211/ieee80211_wireless.c in MadWifi before 0.9.2.1 allows remote attackers to execute arbitrary code via unspecified vectors, related to the encode_ie and giwscan_cb functions.
CVE-2006-6125 (DRIVER/WIN)	Heap-based buffer overflow in the wireless driver (WG311ND5.SYS) 2.3.1.10 for NetGear WG311v1 wireless adapter allows remote attackers to execute arbitrary code via an 802.11 management frame with a long SSID.
CVE-2006-6059 (DRIVER/WIN)	Buffer overflow in MA521nd5.SYS driver 5.148.724.2003 for NetGear MA521 PCMCIA adapter allows remote attackers to execute arbitrary code via (1) beacon or (2) probe 802.11 frame responses with an long supported rates information element. NOTE: this issue was reported as a "memory corruption" error, but the associated exploit code suggests that it is a buffer overflow.
CVE-2006-6055 (DRIVER/WIN)	Stack-based buffer overflow in A5AGU.SYS 1.0.1.41 for the D-Link DWL-G132 wireless adapter allows remote attackers to execute arbitrary code via a 802.11 beacon request with a long Rates information element (IE).
CVE-2006-5972 (DRIVER/WIN)	Stack-based buffer overflow in WG111v2.SYS in NetGear WG111v2 wireless adapter (USB) allows remote attackers to execute arbitrary code via a long 802.11 beacon request.

Etat de l'art : Vulnerabilités 802.11 Driver/Parser

CVE-2006-5882 (DRIVER/WIN)	Stack-based buffer overflow in the Broadcom BCMWL5.SYS wireless device driver 3.50.21.10, as used in Cisco Linksys WPC300N Wireless-N Notebook Adapter before 4.100.15.5 and other products, allows remote attackers to execute arbitrary code via an 802.11 response frame containing a long SSID field.
CVE-2006-5710 (DRIVER/OSX)	The Airport driver for certain Orinoco based Airport cards in Darwin kernel 8.8.0 in Apple Mac OS X 10.4.8, and possibly other versions, allows remote attackers to execute arbitrary code via an 802.11 probe response frame without any valid information element (IE) fields after the header, which triggers a heap-based buffer overflow.
CVE-2006-3992 (DRIVER/WIN)	Unspecified vulnerability in the Centrino (1) w22n50.sys, (2) w22n51.sys, (3) w29n50.sys, and (4) w29n51.sys Microsoft Windows drivers for Intel 2200BG and 2915ABG PRO/Wireless Network Connection before 10.5 with driver 9.0.4.16 allows remote attackers to execute arbitrary code via certain frames that trigger memory corruption.
CVE-2006-3509 (DRIVER/OSX)	Integer overflow in the API for the AirPort wireless driver on Apple Mac OS X 10.4.7 might allow physically proximate attackers to cause a denial of service (crash) or execute arbitrary code in third-party wireless software that uses the API via crafted frames.
CVE-2006-3508 (DRIVER/OSX)	Heap-based buffer overflow in the AirPort wireless driver on Apple Mac OS X 10.4.7 allows physically proximate attackers to cause a denial of service (crash), gain privileges, and execute arbitrary code via a crafted frame that is not properly handled during scan cache updates.
CVE-2006-3507 (DRIVER/OSX)	Multiple stack-based buffer overflows in the AirPort wireless driver on Apple Mac OS X 10.3.9 and 10.4.7 allow physically proximate attackers to execute arbitrary code by injecting crafted frames into a wireless network.
CVE-2006-1385 (PARSER)	Stack-based buffer overflow in the parseTaggedData function in WavePacket.mm in KisMAC R54 through R73p allows remote attackers to execute arbitrary code via multiple SSIDs in a Cisco vendor tag in a 802.11 management frame.
CVE-2006-0226 (DRIVER/BSD)	Integer overflow in IEEE 802.11 network subsystem (ieee80211_ioctl.c) in FreeBSD before 6.0-STABLE, while scanning for wireless networks, allows remote attackers to execute arbitrary code by broadcasting crafted (1) beacon or (2) probe response frames.

Etat de l'art : Vulnerabilités 802.11 Driver/Parser

- Quelques unes peuvent éventuellement manquer…

- 18 entrées CVE
  - 15 sont relatives à des drivers
    - 9 Windows
    - 4 OS X
    - 1 Linux
    - 1 FreeBSD (plutôt relatif à la couche 802.11 de FreeBSD)
  - 3 sont relatives à des sniffer / parser
    - Ethereal / Wireshark et tcpdump
    - KisMAC

- La 1ère faille était basée sur FreeBSD (début 2006)

Etat de l'art : Vulnerabilités 802.11 Driver/Parser

- Parmi 14 vulnérabilités drivers _différentes_
  - Long SSID (x3), Long Supported Rates (x2), Long TIM (x1)
    - Faciles à découvrir par fuzzing
  - Set of long IEs
    - Faciles à découvrir par fuzzing
  - No valid IE
    - Faciles à découvrir par fuzzing
  - IE WPA/RSN/WMM (madwifi and FreeBSD vulnerabilities)
    - Nécessite un fuzzer générique de OUIs
  - (Flood of) disassociation packets
    - Difficiles à découvrir
  - 3 autres vulnérabilités ne sont pas spécifiées

Failles découvertes

- NetGear MA521 Wireless Driver Long Rates Overflow
  - Utilisation d'une trame avec un IE « Rates » trop long
    - Ce champ est le plus souvent d'une longueur maximale de 8 octets
- NetGear WG311v1 Wireless Driver Long SSID Overflow
  - Utilisation d'une trame avec un IE « SSID » trop long
    - Ce champ est d'une longueur maximale de 32 octets
- D-Link DWL-G650+ (A1) Wireless Driver Long TIM Overflow
  - Utilisation d'une trame avec un IE « TIM » trop long
- Madwifi Driver Remote Buffer Overflow Vulnerability
  - Utilisation d'une trame avec IE WPA/RSN/WMM/ATH trop long
  - Exploitable uniquement lors d'appels à `SIOCGIWSCAN`
    - Par exemple via **`iwlist`** ou **`iwlib.h`**

La faille Madwifi

- Obtention d'un OOPS en utilisant le fuzzer
  - États des registres
  - États de la pile
  - Backtrace
- On remarque immédiatement la valeur de EIP et de EBP
- Le backtrace montre que nous sommes dans un appel système ioctl()
  - C'est à dire en contexte de processus
  - Mais bien sûr en mode noyau

Type de faille

- Nous pouvons conclure rapidement à un kernel stack buffer overflow
  - On retrouve la fonction incriminée à l'aide du backtrace (giwscan_cb)
  - char buf[64 * 2 + 30];
  - memcpy(buf, se->se_wpa_ie, se->se_wpa_ie[1] + 2);
  - Nous contrôlons la taille. Ouch!
- Il est possible de forger une trame 802.11 encore plus malveillante

Conséquences

- Nous avons reporté cette faille avec un patch correctif le 05 décembre 2006
- Madwifi a publié une nouvelle version avec notre correctif le lendemain
- Les distributions ont pu commencer à patcher cette faille rapidement
  - Mais n'ont pas toutes réagit très vite

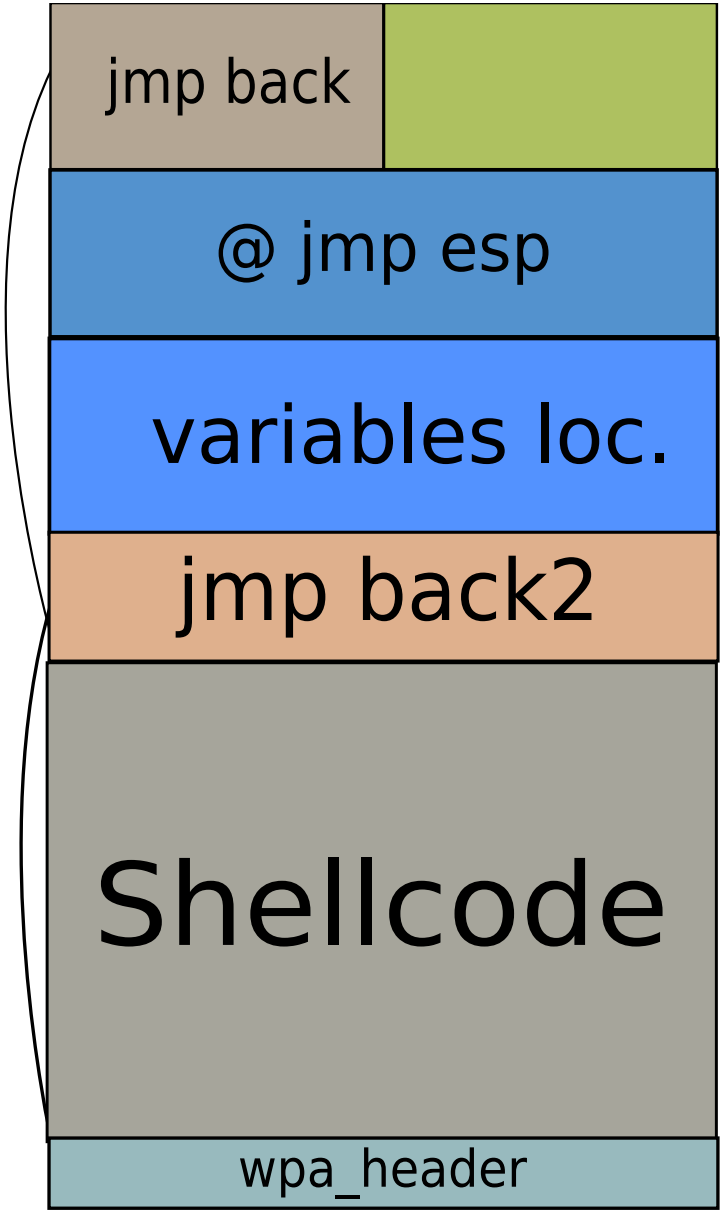
Stratégie d'exploitation

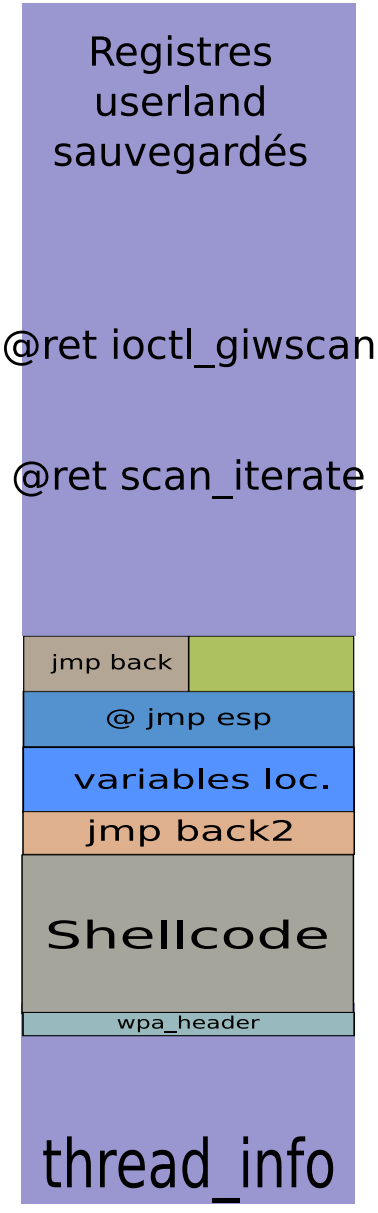
- Injection du code dans l'espace d'adressage
  - Il semble naturel d'utiliser la trame 802.11
  - Notre information element se retrouve sur la pile noyau du processus ayant réalisé l'appel système
  - Nous allons y placer notre shellcode

Stratégie d'exploitation

- Contrôle du flot d'exécution
  - Il semble naturel d'écraser la sauvegarde de EIP sur la pile
  - En utilisant un jmp esp
  - Trouvé quelque part en espace utilisateur ou noyau
  - Le VDSO en espace utilisateur contient un jmp esp bien pratique
    - dd if=/proc/self/mem bs=4096 skip=$((0xFFFFE)) count=1 of=vdso.so
    - Entre la fin de l'ELF et la fin de la page: jmp esp
    - Indépendant du processus et de la version du noyau

Pile noyau


jmp back
@ jmp esp
variables loc.
jmp back2
Shellcode
wpa_header


Registres
userland
sauvegardés
@ret ioctl_giwscan
@ret scan_iterate
jmp back
@ jmp esp
variables loc.
jmp back2
Shellcode
wpa_header
thread_info

La problématique du shellcode en mode noyau

- Revenons à une situation connue
- Retournons vite en Userland
  - On obtient la sauvegarde du pointeur de pile utilisateur en haut de la pile noyau
  - On recopie un shellcode utilisateur sur la pile utilisateur
  - On modifie la sauvegarde de EIP utilisateur
- À partir de là nous pouvons directement faire un iret
  - Cela donne un exploit indépendant de la version du noyau et du processus réalisant l'appel système
  - Mais cela tue la pile 802.11

Sauvons le Wifi

- On essaie de faire en sorte que le noyau reprenne son exécution "normalement"
  - Retour sur notre grand parent
  - Émulation de l'épilogue du parent
    - Restauration des registres (Merci Stéphane!)
    - Déverrouillage d'un spinlock (ouch!)
- Notre shellcode "userland" s'exécute au retour de l'appel système
- La pile 802.11 va bien
- On peut même aller jusqu'à reprendre l'exécution du processus normalement

Résultat

- Module pour Metasploit (utilisant Metasm) exploitant toute machine Linux avec une carte Atheros scannant ses réseaux
- Deux types de cibles
  - Une cible générique fonctionnant partout mais détruisant la pile 802.11
  - Des cibles plus spécifiques mais laissant un système en parfait état
    - Par exemple Ubuntu 6.10
  - Possibilité d'un exploit multi-cibles puisque l'on obtient l'exécution de code manière totalement générique
- Possibilité d'utiliser n'importe quel payload Metasploit

Démo

Ca a marché?

- La première faille publique remote kernel pour Linux !
  - Avec en plus un exploit fiable
  - Une autre raison d'utiliser PaX
    - KERNEXEC contre les remotes
    - UDEREF contre certains locaux
- Intégration prochaine dans Metasploit de payloads kernel

Ca n'a pas marché?

- Qui a fait un DoS?
- Qui a lancé un autre exploit? ;)
- Il n'est pas possible de se protéger, même avec WPA

Conclusion

- Le fuzzing permet de défricher le terrain
  - La plupart des erreurs de programmation découvertes par fuzzing sont classiques
  - L'espace des tests ne peut être couvert

Conclusion (2)

- L'efficacité d'un fuzzer dépend de :
  - La compréhension du protocole
  - La gestion des états du protocole
  - La possibilité de détecter les tests induisant des dysfonctionnements
  - La possibilité de rejouer les tests efficacement
- Dans tous les cas, l'investigation d'un bug nécessite des compétences pointues
  - et du temps!

Remerciements

- Yoann Guillot, Raphaël Rigo, Stéphane Duverger, Franck Veysset
- Des questions?