

Challenge Securitech 2006: scramblito

Julien Tinnès

France Telecom R&D

ESIEA - 19 Mai 2006





Présentation de scramblito

- Challenge 11
- Démon inetd pour OpenBSD Intel
 - ▶ Tournant sur un serveur OpenBSD 3.8
- Permet de 'scrambler' ses données par le réseau



Fonctionnement de scramblito

- Affiche une bannière
- Lit un entier 'taille' sur le réseau
- Vérifie $\text{taille} < \text{sizeof}(\text{buffer})$
- Lit 'taille' dans buffer sur le réseau
- Calcule le CRC de 'SECRET'+'buffer'
- Xor 'SECRET'+'buffer' avec ce CRC
- Renvoie la clé et la chaîne scramblée



Objectifs du challenge

- Trouver la faille
- Exploiter la faille
 - ▶ Injecter son payload
 - ▶ Prendre le contrôle du processus et exécuter le shellcode
 - Déjouer l'ASLR
 - Déjouer la non exécutabilité W^X
- Faire un shellcode localisant le valideur et l'exécutant

Faille



- Int overflow (signé)
 - ▶ $\text{size} + 4 < \text{sizeof}(\text{buffer})$
- $\text{size} = 0xFFFFFFFF$ ne marche pas
 - ▶ Passé en argument à `read()`
- $\text{size} = 0x7FFFFFFFF$ induit un stack overflow



Injecter son payload

- Payload xor-é par son propre CRC:

$$\text{xoredpayload} = \text{payload} \oplus \text{crc}(\text{payload})$$

- On s'impose un CRC (targetcrc)
- On génère

$$\text{payload} = \text{targetpayload} \oplus \text{targetcrc}$$

- On modifie quelques bits inutiles de payload de sorte à ce que son CRC vaille targetcrc

CRC



- Un message peut être vu comme une suite à support fini de bits
 - ▶ C'est à dire un polynôme sur $GF(2)$ (ou vecteur d'un EV sur $GF(2)$)
- CRC de taille m = division polynômiale par un polynôme de degré m
- CRC de degré 32 du message M avec le polynôme P

$$M \times X^{32} \bmod P$$

- CRC est une fonction linéaire si on considère le message comme un vecteur de $GF(2)^{\text{longueur}}$ et le CRC comme un vecteur de $GF(2)^{32}$



CRC (2)

- CRC32 IEEE induit quelques modifications:
 - ▶ Inversions de bits dans les octets du message et dans le résultat final
 - ▶ Shift register initialisé à -1, not du résultat final
- Plus linéaire mais affine
- On fait un changement de base
 - ▶ $CRCN(X) = CRC(X) \oplus CRC(0)$
- CRCN linéaire
- P étant premier, on sait qu'on peut obtenir tous les CRCs en jouant sur 32 bits consécutifs du message

CRC (3)



- Méthode algorithmique de calcul d'un antécédant:
 - ▶ ajouter des multiples de P au CRC cible
 - TweakCRC
- Possibilité de force brute 'intelligent'
 - ▶ Brute-force des antécédants de la base normale (ou d'une base de Gauss)
- Possibilité de force-brute
 - ▶ Pas de nonce dans le protocole
 - ▶ CRC de 32 bits seulement



Déjouer l'ASLR

- Brute-force
 - ▶ Ok en local mais les paquets sont fragmentés avant d'arriver au serveur
 - Pas de possibilité de gros nop-sled
 - ▶ Nécessite de ne pas bruteforcer le CRC
- Return-to-the-stager:
 - ▶ Retourner sur un read()
 - ▶ Lui faire lire le shellcode dans une zone à adresse fixe
- In-CRC-shellcode
 - ▶ s'arranger pour que CRC vaille 'jmp esp'
 - ▶ retourner sur le CRC!



Déjouer la non executabilité

- Retourner sur un dl_bind pour résoudre mprotect
 - ▶ Puis mprotect PROT_EXEC d'une zone non exécutable
- Changer de segment
 - ▶ Faille publiée dans MISC 23
 - ▶ Retourner sur un 'far ret'



Shellcode

- Pour ceux qui bruteforcent le CRC:
 - ▶ Faites un stager!
- Pas d'AS readdr sous bsd, utiliser getdirentries
- Utiliser shellforge
 - ▶ Phil quand publieras-tu le patch pour le linker script?



Pile finale

