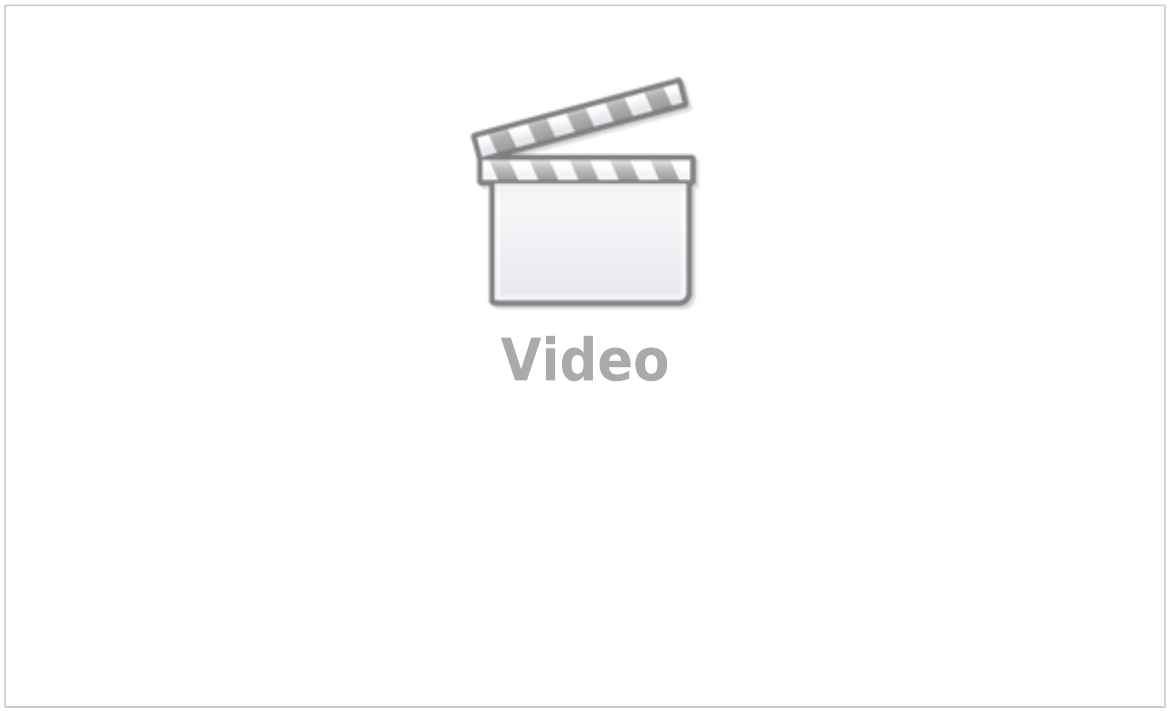
Logiciel

Outils et techniques simples pour la rétro-ingénierie



Dans cette vidéo, nous allons revisiter le vérificateur de licence de la dernière vidéo, vous pouvez avoir le même binaire compilé 64 bits depuis GitHub et vous pouvez aussi regarder la dernière vidéo où je suis allé plus en détails sur comment casser ce simple programme.$

Je vais maintenant vous montrer des outils simples et des techniques qui existent pour analyser un programme comme notre vérificateur de licence.

Cela devrait vous montrer qu'il y a un grand nombre de solutions pour résoudre ce problème.

La commande **file** est très utile pour connaître quels types de fichiers vous avez.

Donc **file**, sur notre binaire dit que c'est un ELF 64-bits exécutable pour Linux. Vous pouvez aussi faire **file** * pour avoir les informations sur tous les fichiers de votre dossier. Et nous avons aussi trouvé le code source C.

Donc cela (**file**) est très utile.

Ouvrons le programme dans un éditeur de texte comme Vim. Comme vous pouvez le voir, c'est très bizarre.

J'ai introduit la notion

Sous-titres

4 00:00:21,020 –> 00:00:25,490 Je vais maintenant vous montrez des outils simples et des techniques qui existent pour analyser un programme

5 00:00:25,490 –> 00:00:30,910 comme notre vérificateur de licence. Cela devrait vous montrez qu'il y a un grand nombre de

6 00:00:30,910 –> 00:00:33,629 solutions pour résoudre ce problème.

7 00:00:33,629 –> 00:00:39,079 La commande 'file' est très utile pour connaître quels types de fichiers vous avez. Donc 'file', sur notre

8 00:00:39,079 –> 00:00:45,020 binaire dit que c'est un ELF 64-bit exécutable pour Linux. Vous pouvez aussi faire un 'file *' pour

9 00:00:45,020 –> 00:00:51,260 avoir les informations sur tout les fichiers de votre dossier. Et nous avons aussi trouvez le code source C.

10 00:00:51,260 –> 00:00:53,450 Donc cela ('file') est très utile

11 00:00:53,450 –> 00:00:59,680 Ouvrons le programme dans un éditeur de texte comme Vim. Comme vous pouvez le voir, c'est très bizarre.

12 00:00:59,680 –> 00:01:05,259 J'ai introduit la notion d'ASCII auparavant, donc vous savez que chaque caractère à un nombre assigné

13 00:01:05,259 –> 00:01:09,290 Mais il y a des nombres qu'y n'ont pas de caractère affichable. Si vous regardez

14 00:01:09,290 –> 00:01:16,380 le 'man' de l’ASCII ('man ascii'), vous pouvez voir que par exemple, les valeurs de 0 à 1F en hexa

15 00:01:16,380 –> 00:01:23,540 ne sont pas des caractères normaux. Et l'ascii est aussi défini entre 0 et 127 (7F en héxa). Mais nos ordinateurs

16 00:01:23,540 –> 00:01:31,200 fonctionnent avec des octets, donc 8 bits, ce nombre peut donc aller de 0 à 255, et l'ASCII n'en utilise que la moitié

17 00:01:31,200 –> 00:01:36,939 Donc toutes ces valeurs bleues bizarres font parti de ces nombres qui non pas de

18 00:01:36,939 –> 00:01:43,470 caractères affichable assigné. Vous pouvez aussi faire un hexdump du fichier, pour avoir les valeurs exactes.

19 00:01:43,470 –> 00:01:49,830 'hexdump -C license_1' et vous pouvez voir qu'il y a beaucoup de 0 dans le fichier. hexdump les affiche avec des .

20 00:01:49,830 –> 00:01:55,380 mais Vim les afficher avec des trucs bleus. Mais quand vous regardez bien, il y a

21 00:01:55,380 –> 00:02:00,380 quelques chaînes intéressante là-dedans . Par exemple, au tout début “ELF”,

22 00:02:00,380 –> 00:02:06,170 qui est une 'valeur magique', qui indique que ce fichier est un exécutable.

23 00:02:06,170 –> 00:02:11,038 Vous pouvez aussi voir des chaînes pour des librairies comme libc,

qui défini des fonctions comme

24 00:02:11,038 –> 00:02:16,540 printf ou strcmp. Et on peut aussi voir des chaînes que l'on connait, les messages :

25 00:02:16,540 –> 00:02:21,250 “Checking License”, “Access Granted”, “WRONG!” et “Usage”

26 00:02:21,250 –> 00:02:28,099 et aussi la chaîne “AAAA-ZION-42-OK”. *hmmhmh… in english*

27 00:02:28,099 –> 00:02:32,170 Vous vous souvenez de la dernière vidéo où il y avait une comparaison de chaîne dedans ?

28 00:02:32,170 –> 00:02:40,040 Peut-être que la clé que l'on rentre est comparée a celle-la ? Essayons la! Accès Accordé,

29 00:02:40,040 –> 00:02:42,910 la clé de ce programme était la dedans depuis le début.

30 00:02:42,910 –> 00:02:48,370 Il y a un outil très bien appelé 'strings' qui fait ce que l'on vient juste de faire, mais mieux

31 00:02:48,370 –> 00:02:53,849 Il va scanner un fichier et afficher toutes les séquences de caractères affichables avec une certaine longueur.

32 00:02:53,849 –> 00:03:01,770 Essayons donc avec 'strings license_1'. Et voilà nos chaînes.

33 00:03:01,770 –> 00:03:07,540 La dernière fois, nous avons utilisé gdb pour lire et déboguer le fichier. Cette fois

34 00:03:07,540 –> 00:03:13,790 nous allons juste utiliser 'objdump' pour le désassemblage. ‘objdump -d license_1’.

35 00:03:13,790 –> 00:03:18,640 Vous remarquerez que ce fichier a beaucoup plus de code que juste la focntion main. C'est à cause du compilateur (gcc) qui

36 00:03:18,640 –> 00:03:23,599 met un peut plus de trucs dans le fichier binaire. En effet, les ordinateurs sont un peut plus complexe que ce qu'ils

37 00:03:23,599 –> 00:03:27,709 semblent être au premier abord. Mais tout ça est juste des choses basiques que vous trouverez

38 00:03:27,709 –> 00:03:34,050 dans n'importe quel fichier compilé par gcc. Et la plupart du temps, seul les fonctions comme le main,

39 00:03:34,050 –> 00:03:37,810 crée par l'utilisateur sont utiles. objdump peut être utilisé pour avoir beaucoup plus

40 00:03:37,810 –> 00:03:42,810 d'informations sur ce programme. Affichons tout avec ‘objdump -c license_1’ et mettons un pipe

41 00:03:42,810 –> 00:03:48,950 vers less pour y voir plus clairement. Donc la première chose qu'il dit est que ce fichier est un ELF

42 00:03:48,950 –> 00:03:55,560 pour l'architecture x86-64. La pile (stack) n'est pas exécutable, ce qui est indiqué par l’absence du 'x'

43 00:03:55,560 –> 00:03:59,629 ce qui sera intéressant quand nous regarderons les buffer overflows classiques.

44 00:03:59,629 –> 00:04:05,330 Et les dernières informations intéressante (dans cette situation) sont dans les sections. Ici nous pouvons voir que

45 00:04:05,330 –> 00:04:11,120 des données vont finir dans la mémoire. Les sections intéressantes pour nous sont .text qui contient

46 00:04:11,120 –> 00:04:19,810 notre code et commence à l'adresse 4004d0 et à une taille de 1e2(hex), si vous regardez où était la fonction

47 00:04:19,810 –> 00:04:23,830 main, vous remarquerez que c'était la.

48 00:04:23,830 –> 00:04:29,569 L'autre section est .rodata, où sont stocké nos données seulement lisibles

49 00:04:29,569 –> 00:04:35,839 Donc nos chaînes peuvent être trouvées ici. Si vous ouvrez gdb et mettez un break sur le strcmp,

50 00:04:35,839 –> 00:04:43,710 vous pouvez vérifier les registres. Et l'un deux aura une adresse venant de .rodata

51 00:04:43,710 –> 00:04:48,699 Vous pouvez afficher cette adresse avec 'x/s' et voila notre clé est encore la.

52 00:04:48,699 –> 00:04:54,520 Passons à un autre outil, il est appelé strace et il peut tracer les appels système et les signaux.

53 00:04:54,520 –> 00:05:01,479 Quand je vous ai introduit la programmation en C, nous avons utilisé printf pour afficher du texte.

54 00:05:01,479 –> 00:05:07,020 C'était une fonction que l'on avait rajouter à notre programme depuis la librairie libc. Mais printf est juste

55 00:05:07,020 –> 00:05:12,770 un 'wrapper',qui appelle une fonction que Linux nous procure.

56 00:05:12,770 –> 00:05:18,339 Linux nous offre différentes fonctions appelés syscalls. Vous pouvez en apprendre plus dans le man.

57 00:05:18,339 –> 00:05:24,349 Une de ces fonctions est 'write'. Et write peut être utilisé pour écrire du texte sur la sortie standard ,

58 00:05:24,349 –> 00:05:30,419 que nous pouvons lire dans la console. Exécutons donc notre programme avec strace.

59 00:05:30,419 –> 00:05:37,479 La première ligne est execve, qui est une fonction qui dit au kernel Linux (le coeur) d'exécuter le programme

60 00:05:37,479 –> 00:05:43,809 license_1. Après ça, beaucoup de magie opère que l'on allons ignorer pour l'instant.

61 00:05:43,809 –> 00:05:49,550 Et quelque part dans la descente, le code que j'ai écrit commence.

62 00:05:49,550 –> 00:05:57,869 Et là, vous pouvez voir l'appel de write qui est exécuté avec le texte que l'on connaît.

63 00:05:57,869 –> 00:05:59,039 Intéressant non ?

64 00:05:59,039 –> 00:06:06,069 Il y a un autre outil cool appelé 'ltrace'. Similaire à strace, il trace (dans le sens de suivre) des fonctions.

65 00:06:06,069 –> 00:06:11,349 Mais cette fois, il suit les fonctions des librairies, comme printf ou strcmp, qui viennent de libc.

66 00:06:11,349 –> 00:06:17,089 Donc ltrace nous montre leurs occurences. D'abord vous pouvez voir le printf, et ensuite le strcmp

67 00:06:17,089 –> 00:06:22,580 Et cela montre les chaînes que cela compare, donc ça nous montre sympathiquement

68 00:06:22,580 –> 00:06:25,969 comment le vérificateur de licence marche .

69 00:06:25,969 –> 00:06:31,699 Utilisons ce fichier avec une interface graphique utilisateur. Je vais utilisé hopper sur mac. Comme vous le savez sûrement

70 00:06:31,699 –> 00:06:38,589 IDAPro est très cher. Mais hopper est une bonne alternative qui est abordable

71 00:06:38,589 –> 00:06:43,860 Hopper reconnaît que c'est un exécutable ELF et peut l'analyser automatiquement pour nous.

72 00:06:43,860 –> 00:06:50,119 Cela place notre curseur sur la fonction appelée start et pas le main. Comme nous l'avons vu avec

73 00:06:50,119 –> 00:06:56,179 objdump auparavant, il y a quelques fonctions crée par le compilateur et c'est la que commence réellement le programme

74 00:06:56,179 –> 00:07:01,029 Mais ce que font ces fonctions n'est pas vraiment important pour l'instant.

75 00:07:01,029 –> 00:07:05,919 Nous sommes seulement intéressé par le main. Donc nous allons le chercher

76 00:07:05,919 –> 00:07:11,339 dans la liste des étiquettes. Et voila notre fonction main, comme avec gdb

77 00:07:11,339 –> 00:07:17,159 C'est juste un peu plus coloré et Hopper peut aussi nous montrez où les branches vont.

78 00:07:17,159 –> 00:07:22,669 A la fin de la dernière vidéo, je vous avait déjà montrez le graphe de contrôle du flux auquel

79 00:07:22,669 –> 00:07:28,969 vous pouvez accéder en haut à droite. Et une fonctionnalité cool de Hopper est son dé-compilateur.

80 00:07:28,969 –> 00:07:36,279 Décompiler pourrait paraître comme le compilateur inversé pour retrouver le code original mais non.

81 00:07:36,279 –> 00:07:42,240 En effet, le compilateur change et optimise des choses et on ne peut pas simplement l'inversé. Mais Hopper

82 00:07:42,240 –> 00:07:47,689 peut deviner comment cela avait l'air. Parfois il fait des erreurs, mais la plupart du temps

83 00:07:47,689 –> 00:07:51,849 c'est plutôt bien. Donc cela nous montre ici comment on vérifie si nous avons passer une clé en argument

84 00:07:51,849 –> 00:07:56,999 puis fait la comparaison et afficher soit “Access Granted” ou “WRONG!”

85 00:07:56,999 –> 00:08:02,319 Donc c'est plutôt sympa. A droite vous trouverez aussi un bouton pour afficher toutes les chaînes

86 00:08:02,319 –> 00:08:08,219 Et comme vous pouvez le voir, cela a aussi trouvé la clé. Quand vous cliquez dessus,

87 00:08:08,219 –> 00:08:13,419 cela va sauter à l'adresse a laquelle la clé est stockée. XREF indique un référencement croisé,

88 00:08:13,419 –> 00:08:18,309 ce qui signifie que cette adresse est référencée quelque part. Nous pouvons suivre ce XREF, et nous pouvons voir

89 00:08:18,309 –> 00:08:24,339 que c'est le code assembleur, où la clé est déplacé dans le dossier esi.

90 00:08:24,339 –> 00:08:28,689 Cela prépare les paramètres de la fonction strcmp.

91 00:08:28,689 –> 00:08:34,360 Quelques enfants ont peut être l'idée que les utilisateurs Mac sont nuls. Donc pour leur faire plaisir, nous allons installer radare2

92 00:08:34,360 –> 00:08:39,809 en clonant le dépôt depuis git. Vous devrez peut-être installer git avec 'sudo apt-get install git'.

93 00:08:39,809 –> 00:08:50,269 Pour installer radare, run 'sys/install.sh' et attendez.

94 00:08:50,269 –> 00:08:57,009 Quand c'est installé, vous pouvez allez sur le fichier license_1 et l'ouvrir avec 'r2 license_1'

95 00:08:57,009 –> 00:09:02,120 Vous vous souvenez peut être de l'adresse du objdump auparavant, c'est le début de la partie texte

96 00:09:02,120 –> 00:09:10,019 qui contient notre code. Lancez d'abord 'aaa' pour analyser automatiquement une fonction autoname.

97 00:09:10,019 –> 00:09:14,620 Puis utilisez 'afl' pour afficher toutes les fonctions que radare a trouvé

98 00:09:14,620 –> 00:09:19,209 Chaque caractères dans radare veut dire quelque chose. Et avec ? vous avez toujours l'information sur

99 00:09:19,209 –> 00:09:26,329 les caractères que vous pouvez utilisés. Donc 'a' fait des analyses

de code. 'a?' nous montre que nous pouvons

100 00:09:26,329 –> 00:09:32,889 lui coller 'f' pour analyser des fonctions. Et 'afl' liste ces fonctions. C'est logique non ?

101 00:09:32,889 –> 00:09:37,069 Ok donc afl nous montre qu'il a trouvé une fonction main.

102 00:09:37,069 –> 00:09:43,810 Changeons notre emplacement pour chercher celui de la fonction main.

103 00:09:43,810 –> 00:09:50,319 Vous pouvez aussi utiliser tab pour compléter ici. Maintenant que la localisation a changer, avec 'pdf',

104 00:09:50,319 –> 00:09:55,110 on peut afficher le code désassemblé de cette fonction

105 00:09:55,110 –> 00:10:00,250 Comme Hopper, cela nous montre le code avec de belles flèches. Et cela a crée des commentaires

106 00:10:00,250 –> 00:10:03,889 pour les chaînes qui sont référencées

107 00:10:03,889 –> 00:10:09,480 Vous pouvez aussi utiliser 'VV' pour entrer dans le mode visuel. cela montre un Graphe de contrôle de la fonction

108 00:10:09,480 –> 00:10:15,050 Vous pouvez vous déplacez avec les flèches de votre clavier. Les bords bleus indiquent que nous sélectionnons actuellement cette boite.

109 00:10:15,050 –> 00:10:20,930 Avec Tab et Shift+Tab, vous pouvez sélectionner d'autres blocs. Quand vous avez sélectionné un bloc,

110 00:10:20,930 –> 00:10:27,470 vous pouvez les déplacer avec Shift et h,j,k ou l.

111 00:10:27,470 –> 00:10:31,689 Avec 'p' vous pouvez changer de représentations. Par exemple avec ou sans les adresses au début.

112 00:10:31,689 –> 00:10:37,420 Ou alors cette vue minimaliste, qui est utile si vous avez de très grosses fonctions.

113 00:10:37,420 –> 00:10:43,209 Et avec ? vous pouvez afficher une aide. Cette aide vous indique que 'R' est le raccourci

114 00:10:43,209 –> 00:10:49,990 le plus important que vous devez connaître avec radare.Donc appuyez sur Shift+R et appréciez le moment.

115 00:10:49,990 –> 00:10:55,759 Vous pouvez aussi utiliser radare2 comme gdb pour déboguer ce programme. Pour ce faire, lancez

116 00:10:55,759 –> 00:10:59,610 radare avec -d. Cherchez la fonction main, analysez le tout

117 00:10:59,610 –> 00:11:05,180 avec 'aaa' et afficher le code désassemblé avec 'pdf'. Maintenant placez un breakpoint au début

118 00:11:05,180 –> 00:11:12,379 avec 'db' et allez dans Visual View avec 'VV'. Comme avec Vim, vous pouvez entrez en mode commande avec ':'

119 00:11:12,379 –> 00:11:19,079 où vous pouvez taper ':dc' pour lancer le programme.

120 00:11:19,079 –> 00:11:25,649 Nous arrivons donc sur le breakpoint 1 et si vous regardez bien, vous pouvez voir rip dans la première boite. Cela nous montre

121 00:11:25,649 –> 00:11:31,149 où l'instruction pointe. Avec 's' vous pouvez passer des instructions. Mais nous

122 00:11:31,149 –> 00:11:37,209 devrions utiliser 'S', autrement nous suivront des instructions que nous ne voulons pas. Donc Shift+S pour aller plus loin

123 00:11:37,209 –> 00:11:40,610 Oups, nous n'avons pas mis de clé pour la licence.

124 00:11:40,610 –> 00:11:42,600 Mais vous avez compris le principe

125 00:11:42,600 –> 00:11:47,360 J'espère que cela vous à aider à apprendre de nouveaux outils et techniques. Et rappelez vous

126 00:11:47,360 –> 00:11:52,879 qu'il n'y a pas d'outils meilleur qu'un autre. Ils ont tous des fonctionnalités et des représentations des données différentes

127 00:11:52,879 –> 00:11:58,199 Il serait logique de tous les maîtriser. Sauf radare. Certains disent que c'est le meilleur

128 00:11:58,199 –> 00:11:59,950 mais personne ne maîtrise vraiment radare.

Introduction

Pré-requis

Installation

Configuration

Utilisation

Désinstallation

Voir aussi

- **(fr)** https://www.youtube.com/watch?v=3NTXFUxcKPc

Basé sur « Article » par Auteur.