**Ασφάλεια Δικτύων και Πληροφοριακών Συστημάτων**

**1η σειρά εργασιών**

**Ονόματα :**

**Νικόλας Κοξένογλου 1711** [**koxenogl@uth.gr**](mailto:koxenogl@uth.gr)

**Γεώργιος Χαλβατζής 1635 gechalva@uth.gr**

**Πασχάλης Ζαπάρας 1657** [**zaparas@uth.gr**](mailto:zaparas@uth.gr)

1. Κατασκευή του badfile:
   1. Γράφουμε την assembly που μας δίνει το root-shell σε ένα shellcode.asm αρχείο.
   2. Κάνουμε compile με το: sudo nasm –f elf64 –o shellcode.o shellcode.asm και μετά με το sudo ld –o shellcode shellcode.o για να πάρουμε το εκτελέσιμο για δοκιμή αλλά και για την εξαγωγή των assembly op codes.
   3. Βλέπουμε το objdump –d shellcode και αντιγράφουμε τα assembly op codes που τυπώθηκαν σε ένα perl script (badfile\_loader.pl) για την τύπωση των δεκαεξαδικών op codes στο badfile.
   4. Εκτελούμε gdb στο εκτελέσιμο και βρίσκουμε την διεύθυνση του buffer. Ξέρουμε ότι αμέσως μετά το buffer βρίσκεται το return address. Προσθέτουμε στο badfile script τόσα no-ops, ώστε να γίνει overwritten το return address με την διέυθυνση της αρχής του malicious payload.
2. Compilation of vulnerable file:
   1. sudo gcc –fno-stack-protector –z execstack –o stack stack.c
      1. –fno-stack-protector για απενεργοποίηση των stack canaries
      2. –z execstack για τρέξιμο κώδικα από την stack
      3. Κάνουμε compile με sudo για να μπορέσουμε να εκτελέσουμε το shell με root privileges
3. Ενέργειες στο εκτελέσιμο:
   1. Αλλαγή δικαιωμάτων στο εκτελέσιμο με sudo chmod 4755 stack
      1. Η αλλαγή γίνεται για να μπορέσουμε να εκτελέσουμε το shell με root privileges
4. Περιγραφή τρεξίματος του vulnerable program:
   1. Το πρόγραμμα εκτελείται κανονικά και διαβάζεται το badfile μέσα στο str.
   2. Η main στέλνει τα περιεχόμενα της str σε μορφή pointer.
   3. Τα περιεχόμενα μεταφέρονται στο πιο μικρό buffer της bof συνάρτησης μέσω της strcpy η οποία υπερχειλίζει τον buffer. Η return address αντικαθίσταται από την διεύθυνση της αρχής του buffer, η οποία έχει προστεθεί στο badfile.
   4. Μόλις η bof επιστρέψει, ο instruction pointer μεταφέρεται στην αρχή του buffer και ο κώδικάς μας αρχίζει να εκτελείται.
5. Παρατηρήσεις:
   1. Έχοντας απενεργοποιημένο το Address Randomization, παρατηρούμε αλλαγές στις διευθύνσεις του προγράμματος στις εξείς περιπτώσεις: a) αν εκτελείται σε gdb ή όχι   
       b) αν έχει τεθεί το set-uid root.
   2. Task 1: Ο κώδικάς μας εκτελείται κανονικά και μπορούμε να πάρουμε root-shell.
   3. Task 2: Ενεργοποιώντας το Address Randomization βλέπουμε διαφορές από τρέξιμο σε τρέξιμο. Βρήκαμε σταθερά bits στις τυχαίες διευθύνσεις τα οποία μικραίνουν το διάστημα μαντεψιάς. Οι διευθύνσεις όμως εξακολουθούν να είναι αρκετά τυχαίες για να μπορέσει κάποιος να μαντέψει την διεύθυνση και να επιτεθεί με brute-force προσέγγιση.
   4. Task 3: Αφαιρώντας το –fno-stack-protector ενεργοποιούμε τα stack canaries, bits μέσα στη stack τα οποία αν τροποποιηθούν το λειτουργικό διακόπτει την εκτέλεση του προγράμματος με το ακόλουθο error: stack smashing detected Aborted (core dumped). Τρέχοντας το εκτελέσιμο με gdb παρατηρούμε ότι μόλις ο instruction pointer ξεκινήσει να διαβάζει εντολές από το payload, επιστρέφει η συνάρτηση <\_\_stack\_chk\_fail@plt> και η εκτέλεση διακόπτεται.
   5. Task 4: Αφαιρώντας το –z execstack και αντικαθιστώντας το με –z nonexecstack αφαιρούμε την δυνατότητα εκτέλεσης της κώδικα από τη stack. Αυτό οδηγεί στο μήνυμα segmentation fault καθώς το πρόγραμμα προσπαθεί να εκτελέσει κώδικα από σελίδα η οποία έχει δικαίωμα διαβάσματος και γραψίματος. Μια τέτοια σελίδα δεν έχει δικαίωμα εκτέλεσης.