ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΛΕΙΤΟΥΡΓΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ 2013

2η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

ΚΡΥΠΤΟΓΡΑΦΙΚΗ ΣΥΣΚΕΥΗ VirtIO ΓΙΑ QEMU-KVM

ΤΕΛΙΚΗ ΑΝΑΦΟΡΑ

ΟΜΑΔΑ B04

ΣΑΡΛΗΣ ΔΗΜΗΤΡΙΟΣ 03109078

ΤΖΑΝΝΕΤΟΣ ΔΗΜΗΤΡΙΟΣ 03109010

ΓΕΝΙΚΗ ΠΕΡΙΓΡΑΦΗ

Αντικείμενο της 2ης Εργαστηριακής Άσκησης είναι ο σχεδιασμός και ανάπτυξη εικονικής συσκευής κατά το πρότυπο VirtIO για το περιβάλλον εικονικοποίησης QEMU-KVM . Σκοπός είναι οι διεργασίες που εκτελούνται μέσα στο εικονικό μηχάνημα (VM) να έχουν πρόσβαση μέσω της εικονικής συσκευής στη πραγματική κρυπτογραφική συσκευή του host (τύπου cryptodev-linux) .

ΚΡΥΠΤΟΓΡΑΦΗΜΕΝΗ ΕΠΙΚΟΙΚΩΝΙΑ ΠΑΝΩ ΑΠΟ TCP/IP

Για την επαλήθευση της ορθής λειτουργίας του οδηγού – κρυπτογραφικής συσκευής VirtIO , αναπτύχθηκε ένα απλό εργαλείο για κρυπτογραφημένη επικοινωνία πάνω από TCP/IP sockets με χρήση του BSD Socket API. Σχεδιάστηκε έτσι ώστε να επιτρέπει την επικοινωνία πολλών ταυτόχρονων πελατών με κοινό server , σενάριο παρόμοιο με αυτό της υπηρεσίας Internet Relay Chat (IRC) .

Κατά την εκτέλεση του εργαλείου chat καθορίζονται μέσω των command line arguments ο τύπος του χρήστη (server ή client) και το κλειδί κρυπτογράφησης αποτελούμενο από αλφαριθμητικούς χαρακτήρες . Η χρήση ενός hard coded κλειδιού είναι μάλλον λιγότερο ασφαλής και ευέλικτη . Αντίθετα , η επιλογή του user-defined key επιτρέπει την αλλαγή του κλειδιού χωρίς να απαιτείται recompilation και επιτρέπει σε πελάτες να συνδέονται σε διαφορετικούς servers με το εκάστοτε κλειδί χρησιμοποιώντας το ίδιο εκτελέσιμο .

Συνεχίζοντας την ανάλυση πρώτα με την πλευρά του server , αφού εγκαταστήσει με επιτυχία το listener socket και ξεκινήσει το session με την κρυπτογραφική συσκευή , ο server περιμένει για incoming connections – εξυπηρετώντας παράλληλα την κίνηση με τους ήδη υπάρχοντες clients . Για να μπορούμε να ελέγχουμε ταυτόχρονα πολλά sockets για αποστολή δεδομένων από/προς τον server ή πιθανά log ins/outs χωρίς να «μπλοκάρουμε» , χρησιμοποιούμε τη συνάρτηση **`select`** και τα κατάλληλα file descriptor sets (`fd_set`) . Με τη βοήθεια των macros `FD_SET` και `FD_ISSET` και μίας ουράς (`server_ops.c`) που αποτελεί τους logged-in clients εξυπηρετούμε την κίνηση στο δίκτυο ενημερώνοντάς τους (με κρυπτογραφημένα πακέτα από τον server) όταν κάποιος client στείλει ένα μήνυμα ή αποσυνδεθεί . Τα μηνύματα των clients είναι ήδη κρυπτογραφημένα από τους ίδιους και γίνονται απλώς echo από τον server.Η αποκρυπτογράφηση θα πραγματοποιηθεί από κάθε client ξεχωριστά .

Αντίστοιχα , στην περίπτωση του client , ζητούνται αρχικά το hostname και η θύρα στην οποία θέλει να συνδεθεί (τα στοιχεία του server). Μόλις ολοκληρωθεί η σύνδεση με τον server , ο client ξεκινά και εκείνος δικό του session με την κρυπτογραφική συσκευή . Για την επικοινωνία με τον server χρησιμοποιείται πάλι η `select` για τους ίδιους ακριβώς λόγους. Στην περίπτωση αποστολής δεδομένων προς τον server , το μήνυμα διαβάζεται από το πληκτρολόγιο του χρήστη , κρυπτογραφείται και αποστέλλεται . Στην περίπτωση της λήψης , το μήνυμα αποκρυπτογραφείται από τον ίδιο τον client και εμφανίζεται στην οθόνη .

Όσο αφορά την διαδικασία της κρυπτογράφησης , χρησιμοποιείται ένα συγκεκριμένο αρχείο ως initialization vector , αντίγραφα του οποίου πρέπει να έχουν όλοι οι χρήστες έτσι ώστε να γίνεται σωστά η διαδικασία της (απο)κρυπτογράφησης . Επίσης, για λόγους ασφάλειας και ομαλής λειτουργίας του πρωτοκόλλου κρυπτογράφησης AES , τα αρχικά πακέτα που αποστέλλονται μέσω TCP/IP έχουν σταθερό μήκος το οποίο καθορίζεται στο `socket-common.h` .Με βάση αυτή τη σύμβαση , όταν διαβάζουμε αυτό το πληκτρολόγιο χρησιμοποιούμε zero padding έτσι ώστε να διατηρείται σταθερό το μέγεθος των πακέτων που αποστέλλονται και «επιμένουμε» (`insist_write`) για να αποσταλούν πάνω από το TCP/IP . Αντίστοιχα , «επιμένουμε» (`insist_read`) για να διαβάσουμε δεδομένα που προέρχονται από ένα TCP socket . Όλες οι παραπάνω λειτουργίες υλοποιούνται στο αρχείο `io_ops.c`

ΚΡΥΠΤΟΓΡΑΦΙΚΗ ΣΥΣΚΕΥΗ VirtIO

Όπως έχει ήδη αναφερθεί , σκοπός της Άσκησης είναι ο σχεδιασμός και υλοποίηση μίας εικονικής κρυπτογραφικής συσκευής VirtIO για το QEMU και ο αντίστοιχος οδηγός συσκευής για τον guest πυρήνα Linux μέσα στην εικονική μηχανή (VM) . Για την ανάπτυξή τους χρησιμοποιήθηκε το split-driver model : frontend (πυρήνας Linux του VM) και backend (κώδικας για το QEMU) και η μεταξύ τους επικοινωνία επιτυγχάνεται με τη βοήθεια του VirtIO.

FRONT-END DRIVER

Ύστερα από την εγκατάσταση του virtio_crypto module στον πυρήνα της εικονικής μηχανής, ο πυρήνας του guest θα πρέπει να εξυπηρετεί τις κλήσεις συστήματος ενός user-space προγράμματος προς την συσκευή crypto_devs για την έναρξη ενός session (open) και τις λειτουργίες (απο)κρυπτογράφησης (ioctl) και να τις μεταβιβάζει μέσω του προτύπου VirtIO στον host (QEMU διεργασία) όπου θα πραγματοποιούνται οι αντίστοιχες κλήσεις στην πραγματική κρυπτογραφική συσκευή (cryptodev-linux) . Για την υποστήριξη των παραπάνω λειτουργιών από την πλευρά του quest , τροποποιήσαμε κατάλληλα τα αρχεία `crypto-chrdev.h` , `crypto-ioctl.c` και `crypto-vq.c`. Ακολουθεί η ανάλυση των επιμέρους λειτουργιών όπως πραγματοποιούνται για να εξυπηρετήσουν το user-space πρόγραμμα .

- **Άνοιγμα Συσκευής :** όταν το userspace πρόγραμμα εκτελέσει το system call `open` για να εκκινήσει την κρυπτογραφική συσκευή crypto_devs , ο πυρήνας του guest εκτελεί την open του συγκεκριμένου module (**`crypto-chrdev.c`**) οπότε συμβαίνουν τα εξής : αντιστοιχίζει το αρχείο προς-άνοιγμα με μία υπάρχουσα εικονική κρυπτογραφική συσκευή (`get_crypto_dev_by_minor`) και στη συνέχεια στέλνει αίτημα ελέγχου (`send_control_msg`) στον host να εκκινήσει την πραγματική συσκευή . Οποτεδήποτε ο πυρήνας του guest θέλει να επικοινωνήσει με την συσκευή του host , θα χρησιμοποιεί τον file descriptor που επέστρεψε η κλήση της open του host και αποθηκεύτηκε στο `crdev→fd` .

- **Επικοινωνία/Χρήση Συσκευής** : αφότου το user-space πρόγραμμα «ανοίξει» την κρυπτογραφική συσκευή , μπορεί να χρησιμοποιεί τη συσκευή μέσω της ioctl , των κατάλληλων command flags (`CIOCGSESSION` , `CIOCCRYPT`, `CIOCFSESSION`) και arguments (`struct session_op` ή `crypt_op`) και τότε ο πυρήνας εκτελεί τον κώδικα της ioctl (`crypto-chrdev.c` → **`crypto-ioctl.c`**). Για να εξασφαλιστεί ότι το user-space πρόγραμμα δεν έχει πρόσβαση σε διευθύνσεις μνήμης του πυρήνα τις οποίες μπορεί να αλλοιώσει , τα arguments της ioctl αντιγράφονται σε kernel space με τη χρήση της **`copy_from_user`** . Παράλληλα , επειδή μερικά από τα πεδία των δομών session_op και crypt_op είναι δείκτες προς διευθύνσεις μνήμης του guest , αντιγράφονται το περιεχόμενο μνήμης όπου αυτοί δείχνουν (πραγματικά δεδομένα) με τη χρήση της **`memcpy`** καθώς στη συνέχεια θα αποσταλούν στον host (διαφορετικό μηχάνημα, διαφορετικό address space mapping) . Επειδή κατά την επεξεργασία από το host, κάποια από τα πεδία των kernel-space δομών αλλοιώνονται (`key,dst,src,ivp`) από την ioctl κλήση του host στην πραγματική κρυπτογραφική συσκευή , θα πρέπει να τα πρώτα να τα αντιγράψουμε σε προσωρινές μεταβλητές στον guest και μετά να αποστείλουμε το ioctl πακέτο στον host . Έπειτα , αφού λάβουμε την απάντηση του host (αποτέλεσμα της κλήσης ioctl) , πρέπει να επαναφέρουμε τα πεδία χρησιμοποιώντας τις αντίστοιχες προσωρινές μεταβλητές και να επιστρέψουμε το αποτέλεσμα πίσω στον χρήστη – userspace μέσω της συμμετρικής **`copy_to_user`** . Τέλος , υπάρχει η περίπτωση μία κλήση της ioctl στον host να αποτύχει οπότε θα πρέπει να ενημερώνεται και το userspace πρόγραμμα . Για να υλοποιήσουμε αυτή την επέκταση , προσθέσαμε ένα επιπλέον πεδίο success στη δομή `crypto_data` που περιέχεται στο αρχείο `crypto.h` Αρχικοποιείται από την ioctl του guest με την τιμή 1 και σε περίπτωση σφάλματος της κλήσης ioctl στην πραγματική κρυπτογραφική συσκευή , ενημερώνεται με την τιμή 0 από τον host .

- **Αποστολή Πακέτων Quest's Kernel σε Host(QEMU) :** για την αποστολή δεδομένων προς το host (διεργασία QEMU) ο πυρήνας του quest χρησιμοποιεί τη συνάρτηση `send_buf` . Συμπληρώσαμε την υλοποίηση της συνάρτησης στο αρχείο `crypto-vq.c` έτσι ώστε να επικοινωνεί με τον host με το πρότυπο VirtIO . Αρχικά , προσθέτουμε τον output scatter-gather buffer μέσω της `virtqueue_add` στην output virtqueue και ενημερώνουμε τον backend οδηγό για την προσθήκη μέσω της `virtqueue_kick` .

BACKEND DRIVER

Ύστερα από την ενσωμάτωση του backend driver στον κώδικα του QEMU, κάθε userspace πρόγραμμα (διεργασία) QEMU που εκκινεί εικονική μηχανή με εικονικό

υλικό virtio-crypto-pci θα πρέπει να επιτρέπει στο VM να χρησιμοποιεί την κρυπτογραφική συσκευή `/dev/crypto` εκτελώντας κατάλληλες κλήσεις συστήματος και επικοινωνώντας μέσω VirtQueues . Για την υλοποίηση των απαραίτητων λειτουργιών , τροποποιήσαμε κατάλληλα τα αρχεία `virtio-crypto.{c,h}` του βοηθητικού κώδικα . Ακολουθεί η ανάλυση των επιμέρους συναρτήσεων όπως εκτελούνται από τη QEMU διεργασία :

- **Αρχικοποίηση Συσκευών PCI** : αρχικά για την υλοποίηση των εικονικών συσκευών PCI στο περιβάλλον εικονικοποίησης QEMU-KVM κατά την εκκίνηση της εικονικής μηχανής εκτελείται η συνάρτηση `virtio_crypto_init` έτσι ώστε να δημιουργηθούν οι αντίστοιχες συσκευές και να αρχικοποιηθούν οι κατάλληλες VirtQueues .

- **Εξυπηρέτηση Μηνυμάτων Ελέγχου** : όταν ο guest στείλει ένα μήνυμα ελέγχου προς τον host (`virtqueue_kick`) μέσω της `c_ovq` VirtQueue εκτελείται ο handler `control_out` , λαμβάνει τα δεδομένα από την VirtQueue και αφού τα τροποποιήσει κατάλληλα τα στέλνει στην `handle_control_message` . Στην υλοποίηση της τελευταίας , ελέγχουμε το control event που έστειλε ο guest . Στην περίπτωση όπου ο guest επιθυμεί να «ανοίξει» την κρυπτογραφική συσκευή (`VIRTIO_CRYPTO_DEVICE_GUEST_OPEN`) τότε στην πλευρά του host εκτελείται η πραγματική open στο `/dev/crypto` και ο file descriptor επιστρέφεται πίσω στον πυρήνα του guest μέσω της send_control_event όπου ο τροποποιημένος buffer τοποθετείται πίσω στην `c_ivq` VirtQueue και ενημερώνεται ο frontend driver μέσω interrupt (`virtio_notify`). Στην περίπτωση όπου ο guest επιθυμεί να κλείσει τη συσκευή , εκτελείται απλά η close στον host χωρίς περαιτέρω ενημέρωση του frontend driver .

- **Εξυπηρέτηση ioctl πακέτων** : όταν ο guest στείλει δεδομένα στον host μέσω της ovq VirtQueue εκτελείται ο handler `handle_output` , λαμβάνει τα δεδομένα (buffer) από την VirtQueue (`virtqueue_pop`) και αφού τα τροποποιήσει κατάλληλα τα στέλνει στην `crypto_handle_ioctl_packet` . Η τελευταία συνάρτηση αποτελεί και τον βασικό κορμό του backend για τη διαδικασία της (απο)κρυπτογράφησης . Ανάλογα με το είδος της λειτουργίας (`cmd`) εκτελείται η ioctl με τα αντίστοιχα ορίσματα . Για την σωστή εκτέλεση της κλήσης ioctl θα πρέπει πρώτα να αντιγράψουμε τα κατάλληλα πεδία (`keyp,dstp,ivp`) στις αντίστοιχες δομές της `cr_data` και μετά να εκτελέσουμε την ioctl στην πραγματική συσκευή . Η εκτέλεση αυτή θα τροποποιήσει αυτές τις δομές και για αυτό το λόγο θα πρέπει να τις έχουμε πρώτα αντιγράψει σε τοπικές μεταβλητές στον frontend driver . Παράλληλα , σε περίπτωση αποτυχίας ενημερώνεται το πεδίο success της δομής `virtio_crypto_buffer` έτσι ώστε να ενημερωθεί και ο guest . Σε κάθε περίπτωση μετά την αντίστοιχη κλήση της ioctl , ο τροποποιημένος buffer τοποθετείται στην `ivq` VirtQueue και ενημερώνεται ο frontend driver μέσω interrupt (`virtio_notify`).