צור אליהו, 300647799

סדנה באבטחת מידע – תרגיל 3

הערה: כל ההנחות שלי ורפרנסים שהשתמשתי בהם מהרשת(stackoverflow וכו') נמצאים בסוף מסמך זה, כולל הסבר על שימוש של אותם רפרנסים בפונקציות הרלוונטיות שהשתמשתי בהן בקוד. כמו כן הוספתי את הלינקים עצמם גם בהערות בקוד.
(סימנתי הנחות כלליות בטקסט בצבע צהוב)

המבנה כללי של התוכנית:
קבצי הModule:
fw.c, fw.h, hookfuncs.c, hookfuncs.h, log.c, log.h, stateless_funcs.c stateless_funcs.h, Makefile
קבצי הInterface:
main.c, Makefile

התנהלות המערכת הכללית:

שלב ראשון:
לאחר הכנסת firewall.ko לקרנל, מוגדרים כל הdevices.
רשימת ההתקנים(Device list):
2 התקנים ראשיים – fw_rules, fw_log.

ההתקן - fw_rules

גישה להתקן הזה נעשית דרך מחלקת הsysfs בכתובת הנ"ל:
/sys/class/fw/fw_rules/
להתקן יש את **הAttributes** הבאים:
active, clear_rules, rules_size, rules_table
כאשר active, rules_size הוגדרו בדף בתרגיל.
יצרתי את **clear_rules** על מנת שיהיה אפשר למחוק את החוקים. כתיבת "0" גורמת למחיקת החוקים (כל תו אחר זוכה להתעלמות מההתקן, קריאה מההתקן לא עושה כלום).
יצרתי את **rules_table** על מנת שנוכל לקרוא ולכתוב את טבלאות החוקים של הfirewall. (הטבלאות נכתבות מקודדות לתוך הקרנל ורק קידוד חוקי\ספציפי יעבור את בדיקות התקינות של הקרנל, לכן הדרך היחידה לתקשר עם הקרנל היא דרך ממשק המשתמש שיצרתי. כל קריאת נתונים בפורמט לא נכון בפונקציית טעינת החוקים תסתיים בשגיאה ומחיקת החוק הלא חוקי)

ההתקן - fw_log

גישה להתקן הזה נעשית ב2 דרכים:
גישה להתקן עצמו, ולפעולת קריאה שלו נעשית על ידי פתיחת ההתקן /dev/log וקריאה ממנו.
בצורה הזאת מקבלים את הלוג מקודד (אם למשל מבצעים cat), לכן הדרך היחידה באמת להבין את הלוג היא להשתמש בממשק המשתמש שיצרתי שהופך את הנתונים מדטא raw למידע ברור.
גישה בעזרת מחלקת sysfs שנמצאת בכתובת /sys/class/fw/fw_log/
להתקן יש את ה **Attributes** הבאים:
log_clear, log_size
שעליהם כמובן מפורט בדף התרגיל.

שלב שני: בשלב זה, הFirewall מופעל, אך הוא ללא חוקים. לכן כל פקטה נחסמת ומצורפת ללוג (הוספתי הדפסות לתוך הקרנל כדי להבהיר שפקטה נחסמה בגלל שלא נמצא חוק שמתאים לה, שכן זהו מקרה מיוחד כי בדרך כלל יהיה חוק default שאמור להפיל כל פקטה נכנסת)
בשלב זה, ניתן להכניס חוקים לתוך הfirewall על ידי ממשק המשתמש (ולאחר מכן להשתמש בshow rules כדי לוודא שאכן הם הוכנסו)

לאחר שלב זה המערכת פועלת וחוסמת פקטות או מעבירה אותן לפי טבלת החוקים. ניתן לראות הכל בעזרת ממשק המשתמש show_log main.out/.

פירוט על כל הקבצים:

fw.c:

החלק הראשי שמגדיר את כל המודול. נמצאות בו פונקציות הinit, exit כמובן.
קובץ זה מגדיר את הdevices וכל הAttributes שפירטתי עליהם מקודם.
החלק העליון של הקובץ מכיל את ההגדרות של כל ההתקנים, וכל הAttributes שקשורים להתקנים הללו. כמו כן הוא מכיל את הFops עבור fw_log והגדרות נוספות.
קובץ זה מכיל אך ורק הגדרות ואינו מכיל את המימוש של פונקציות הhook או של פונקציות הstatelessוהlog. הוא משמש בעיקר להגדרה כללית של כל המודול.
רשימת הפונקציות:

```
ssize_t get_rules(struct device *dev, struct device_attribute *attr, char *buf);
ssize_t clear_rule_list_func(const char *buf, size_t count);
ssize_t set_rules(struct device *dev, struct device_attribute *attr, const char *buf, size_t count);
```

הפונקציה set_rules היא הפונקציה שקוראת מהמשתמש רשימת חוקים.
כפי שאמרתי, החוקים הללו מקודדים ולכן **רק חוקים בפורמט הנכון יעברו את בדיקת הקלט של הקרנל**. במידה ונשלח מידע לא נכון לקרנל, אני נוקט בצעד דיפנסיבי ומרוקן את רשימת החוקים עד עכשיו(שכן אולי הצליחו להכניס עוד חוקים לא נכונים אל תוך הקרנל)

```
ssize_t get_fw_status(struct device *dev, struct device_attribute *attr, char *buf);
ssize_t activate_fw(struct device *dev, struct device_attribute *attr, const char *buf, size_t count);
```

הפונקציה get_fw_status מחזירה כמובן את מצב הfirewall על ידי הדפסה לתוך והפונקציה activate_fw מפעילה או מכבה את הfirewall בהינתן המצב הנוכחי(היא לא תדליק אם זה כבר כבוי, אלא תתן הודעה שגיאה שהfirewall כבר למעלה)

```
ssize_t get_rules_size(struct device *dev, struct device_attribute *attr, char *buf);
ssize_t rules_size_demi(struct device *dev, struct device_attribute *attr, const char *buf, size_t count);
ssize_t clear_demi(struct device *dev, struct device_attribute *attr, char *buf);
ssize_t clear_rule_list(struct device *dev, struct device_attribute *attr, const char *buf, size_t count);
ssize_t get_log(struct file *filp, char *buff, size_t length, loff_t *off);
ssize_t open_log(struct inode *inode, struct file *file);
```

לוגיקת קריאה מהLog:
כאשר פותחים את ההתקן fw_log, אני לוקח "תמונה" של מצב הלוג ושומר את זה במשתנה בkernel space. לאחר מכן הפונקציה get_log קוראת את המשתנה הזה ושולחת אותו למשתמש באותה הצורה שראינו בכיתה.

```
ssize_t demi_set_log(struct file *filp, const char *buff, size_t len, loff_t * off);
ssize_t get_log_size(struct device *dev, struct device_attribute *attr, char *buf);
ssize_t set_log_size(struct device *dev, struct device_attribute *attr, const char *buf, size_t count);
ssize_t clear_log(struct device *dev, struct device_attribute *attr, const char *buf, size_t count);
ssize_t demi_clear_log(struct device *dev, struct device_attribute *attr, char *buf)
```

כל פונקציה שהמילה demi נמצאת בשם שלה, היא פונקציה ריקה שלא מבצעת כלום אלא רק עוזרת לי להגדיר את הAttribute הרלוונטי.

Hookfuncs.c:

כאן מוגדרים כמה דברים:

פונקציה:

int parse_packet(struct sk_buff *skb, const struct net_device *net_d, unsigned int hooknum, int dir)

זוהי הפונקציה שתופסת פקטות.

היא מקבלת פקטות מ2 מקורות אפשריים – PRE hook, POST hook בעזרת הפונקציות input_hook_func, output_hook_func

היא יוצרת פקטה מהסוג "rule" וממלאת אותה במידע הרלוונטי. לאחר מכן היא שולחת אותה לפונקציה שבודקת את חוקיות הפקטה.

כפי שראינו בשיעור, אם קיבלתי את הפקטה מהכיוון input, אנו מוסיפים לפוינטר שמחלץ את המידע מהskb עוד 20, כדי שנוכל לשלוף את המידע הנכון.

כמו כן קיימות בקובץ עוד 2 פונקציות:

int start_hooks(void);

int close_hooks(void);

שתפקידן לרשום את הפונקציות על הhooks הנכונים כמו שעשיתי בתרגיל הקודם.

Stateless_funcs.c

הקובץ מכיל את הפונקציה

int check_rule_exists(rule_t packet, int hooknum)

היא בודקת האם קיים חוק שמתאים לפקטה שנשלחה.

היא עוברת על כל התנאים ובודקת האם יש התאמה. היא עוברת על פרוטוקלים ספציפיים ובודקת גם את כל החוקיות שקשורה אליהם. במקרה והprotocol הוא other, לא בדקתי את הפורטים(שכן הם לא רלוונטים במקרה שונה מ UDPאו TCP)

log.c

לצערי גיליתי את list.h רק אחרי שכבר מימשתי את הלוג (מימשתי אותו יחסית בהתחלה)

לכן מימשתי אותו בעזרת struct פשוט של רשימה מקושרת. הגדרתי אובייקט log_node ונתתי לו שדה בשם log_entry שהטייפ שלו הוא כמובן log_row_t.

חלק מהאובייקט מכיל מצביע לnext כמובן.

בהינתן node חדש, אני עובר על רשימת הלוגים ומחפש התאמה. מצאתי – מעדכן את המונה ואת הtimstamp. אחרת – מוסיף איבר חדש בסוף הרשימה.

פונקציית המחיקה (Clear_main_log) פשוט עוברת על הרשימה ומשחררת איבר איבר. לאחר מכן היא מדפיסה שהלוג אכן שוחרר לחופשי.

הערה: הדפסת הלוג נעשית לפי הפורמט שנשלח בדוגמה, אבל החוקים לא מיושרים לגמרי (כלומר, אין ממש טבלאות מסודרות אלא רק סדר של אובייקטים. הסדר כתוב בשורה הראשונה שהלוג מדפיס. שזה תווית זמן קודם, לאחר מכן IP, לאחר מכן PORT וכו')

Main.c

קובץ זה מכיל את כל אפשרויות הממשק כפי שמפורטות בתרגיל.

הוא מכיל כמה פונקציות.

int encode_line(char * line, char* buff)

פונקציה זו בודקת חוקיות הקלט ואחראית על הפרסור שלו.

המבנה של הקלט הוא כמו שהוחלט בתרגיל. ובמידה והקלט לא עונה על המבנה – היא מחזירה

הודעת שגיאה ולא כותבת כלום לתוך ההתקן. אם הקלט כן תקין, הפונקציה מפרסרת אותו וכותבת את החוק המקודד אל תוך ההתקן.

int decode_line(char* rule_line, char* rule)
מבצעת את הפעולה ההפוכה כמובן.
היא מקבלת חוק מקודד מהקרנל, והיא אחראית לפרסור נכון שלו והדפסה לתוך היוזר ספייס.

int decode_log_line(char* log_line, char* buff)ף
היא מקבלת מחרוזת מקודדת של לוג. היא מפרסרת אותו ומדפיסה אותו למשתמש בצורה תקינה וקריאה.

הנחות נוספות והערות כלליות:

כיוונים:
כל מה שמקורו בhost2 או שמיועד לhost1, הכיוון שלו הוא **in**
כל מה שמקורו בhost1 או שמיועד לhost2, הכיוון שלו הוא **out**
בנוסף, אני מנטר את eth0(וכל התקן אחר) ומסמן כל פקטה שנכנסת או יוצאת משם או לשם כאחת שעונה על החוק "any" (כי הכיוון שלה הוא כללי ולא מיועד להוסטים)

השתמשתי בGFP_ATOMIC ולא בGFP_KERNEL משום שקיבלתי תמיד שגיאות כאשר ניסיתי לאלקץ זיכרון בתוך הhooks(הוספתי לינק ברפרנסים)

הדפסת הלוג נעשית לפי הפורמט שנשלח בדוגמה, אבל החוקים לא מיושרים לגמרי (כלומר, אין ממש טבלאות מסודרות אלא רק סדר של אובייקטים. הסדר כתוב בשורה הראשונה שהלוג מדפיס)

רפרנסים שמצאתי והשתמשתי בהם במהלך כתיבת הקוד:
http://www.tutorialspoint.com/c_standard_library/c_function_ctime.htm
שימוש בCTIME שמתרגמת ul למחרוזת שמייצגת זמן (כולל הורדה של האיבר האחרון במערך המחרוזת הזה שמייצג את סימן הירידת שורה)

https://www.kernel.org/doc/htmldocs/kernel-api/API-strsep.html
http://www.gnu.org/software/libc/manual/html_node/Finding-Tokens-in-a-String.html
שימוש בstrsep כדי לשלוף טוקנים מתוך מחרוזת (כולל שימוש נכון ובדיקת קלט)
השתמשתי בעיקר כאשר קיבלתי נתונים מהמשתמש ולא רציתי לחשוף את הקרנל ובכלליות את הזיכרון(גם ביוזר ספייס) לכיוונים חשופים מדי

http://stackoverflow.com/questions/1038002/how-to-convert-cidr-to-network-and-ip-address-range-in-c
המרה של CIDR, שזה בעצם הPrefix-size, אל Mask ממש מהצורה של 255.255.0.0 (לדוגמה)

http://beej.us/guide/bgnet/output/html/multipage/htonsman.html
שימוש ב, ntohs, htons וכל משפחת הפונקציות שממירות מhost לnetwork וכדומה.
כך הבנתי איך להמיר מחרוזות לערכים שממש מייצגים כתובות.

http://www.makelinux.net/books/lkd2/ch11lev1sec4
בתחילה הלוגים יצאו לי לא נכונים. ואז הבנתי שלא אילקצתי זיכרון עם flag הנכון. הייתי צריך להשתמש בflag - GFP_ATOMIC, שכן בעזרת זה לא קרה מקרה שבו 2 חוטים החזיקו במקומות זהים בזיכרון (ידוע שהקרנל עובד עם חוטים) כאשר פתחו לוגים שונים וגרמו לקריסה או קרנל פאניק.

הערה נוספת: ביצירת מבנה הנתונים שמייצג את הlog, יש לי הערת אזהרה קטנה. הוספתי לMakefile את ה flag: w-, כדי שיסתיר את הערה הזאת. למעט הערה זאת אין לי שום הערה אחרת, הכל מתקמפל כמו שצריך והכל רץ תקין (הנחתי שעדיף להתריע על אזהרה שאני לא בטוח איך להוריד. כנראה כי השתמשתי במבנה נתונים משלי ולא בlist.h).