צור אליהו, 300647799

סדנה באבטחת מידע – תרגיל 5

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 הכללי של התוכנית:

קבצי הModule:
fw.c, fw.h, hookfuncs.c, hookfuncs.h, log.c, log.h, stateless_funcs.c stateless_funcs.h, stateful_funcs.c, stateful_funcs.h, exploits.c, exploits.h, data_leak.c, data_leak.h ,Makefile
קבצי הInterface:
main.c, Makefile (ועוד קבצי חוקים וhosts שקיבלנו ושמתי בתיקייה הנ"ל, סתם לנוחות, לא חובה להשתמש.)

התנהלות המערכת הכללית:

אני כמובן מדלג פה על ההסבר של התרגיל הקודם. בתמציתו, אני ממשיך מיד מהשלב שבו יש טבלת חיבורים סטטית, והפקטות קודם כל עוברות דרכה. לאחר מכן הן עוברות בטבלה הדינמית שסוגרת ופותחת חיבורים בהתאם למכונת הTCP.
אין התקנים חדשים, רק פונקציות נוספות על החיבורים הקיימים, וכרגע מנטרים גם את פרוטוקול SMTP בנוסף לקודמים.

ההתקפות:
Coppermine Photo Gallery 1.4.14 - Remote Command Execution Exploit
phpFileManager 0.9.8 Remote Code Execution

המחקר והמסקנות מפורטים גם במצגת, יותר בהרחבה. פה אציג את המבנה הכללי של הפונקציות שיצרתי. ההגנה כלפיהן נמצאת בקבצי exploits.c, exploits.h
חשוב לציין שהפונקציונליות הרגילה שלהם נבדקה במקביל והיא תקינה, לכן כל הפונקציות יודעות למנוע את ההתקפות אבל לא פוגעות בפעילות התקינה של האפליקציה.

int check_for_php_attack(char* http_command)
הפונקציה הזאת בודקת שימוש בפיצ'ר המסוכן CMD, שנמצאת בPhpFileManager.
זהו פיצ'ר שלא כוון לשימוש "חופשי" ללא סיסמא, אבל הוא עדין פתוח לחלוטין גם אם לא הגדרת סיסמא לכלי (הצורה הדיפולטיבית של הכלי זה לעבוד ללא סיסמא.)
לכן הדרך היחידה להגן מהמצב הזה זה לחסום את הפיצ'ר הזה, שכן הוא מאפשר הרצת **כל** פקודה אפשרית (במקור נועד להקל על מפתח האפליקציה, כאשר המפתח לא חשב על העובדה שהוא מפיץ קוד שיכול להיות מאוד פגיע, הוא פשוט השתמש בו באופן "אישי" ולא חשב לעומק על מה הוא עושה)
פירוט נוסף במצגת.

int check_coppermine_attack(char* http_command)
הפונקציה הזאת בודקת סניטציה של משתנים מסוכנים באפליקציה Coppermine photo gallery.
ישנם 3 משתנים מסוכנים שלא עוברים בדיקת "ניקיון" לפני שמריצים אותם. Angle, quality, clipval.
הפונקציה פשוט מנטרת פקודות POST, ובודקת שהמשתנים שעוברים הם מהצורה החוקים של angle=NUMBER&clipval=NUMBER&...

ולא דחפו לשם כל מיני אפשרויות להרצת פקודות. הבדיקה העיקרית היא שהתווים שבאים מיד אחרי המשתנים הם עונים על הגדרות החוקיות הנכונות.

Data Leak Prevention:

הוספתי לStateful_funcs.c עוד פונקצייה של SMTP.
בנוסף, כל פקטה שהולכת לפורט 25 תייגתי כפקטת SMTP ופתחתי את החיבור המתאים בטבלת החיבורים.
הוספתי עוד מצבים למכונת המצבים שלי –

SMTP_HANDSHAKE
SMTP_START
SMTP_CONNECTED
SMTP_END

כאשר HANDSHAKE זה בשביל לחיצת היד של הTCP, START זה בשלב "לחיצת היד" של SMTP. הוא יוצא מהשלב הזה ועובר לשלב הCONNECTED כאשר הפקטה החדשה מכילה את המילה DATA. מילה זו אומרת שמהפקטה הבאה, אני אמור לקבל את ה"תוכן" של ההודעה. לינק לאיך למדתי את זה נמצא בסוף מסמך זה.

קובץ הdata_leak.c -
בכדי לדעת אם יש במחרוזת שנשלחת דרך פרוטוקול HTTP או SMTP קוד של שפת C, חקרתי קצת בגוגל מה הדרך הכי טובה לעשות את זה. הוספתי הסברים בקבצים עצמם, וגם לינקים בסוף שגרמו לי להבין מה כדאי לי לחפש. אני אוסיף פה את הדברים העיקרים שלמדתי. וכמובן שיש הסבר נוסף במצגת:

1. סוגריים ריקים מיד אחרי מילים (למשל func()), שכן זה לא משהו הגיוני שיקרה "בשפה" המדוברת.
2. כמות מסויימת של סוגריים מסולסלים (מעל מספר סף מסוים. נכון לעכשיו – 8 מכל הסוגים)
3. חץ(->) בין שתי מילים. למשל pointer->next, מצביע על קיום של מצביע בקוד.
4. אופרטורים שלא אמורים להיות בטקסט. למשל && או ==, אותם ספרתי מספר פעמים, ואם אחרי כמות מסויימת(בין 2 ל3) יש כאלה, הנחתי שזה קוד.
5. סוגריים מכוננים({{}})
6. מילים מיוחדות כמו #include, #define
7. חיפשתי קיום של פונקציית Main על ידי חיפוש המבנה **int main (*) {*}**
8. מבנה של לולאת while בצורה של **while (*) {do}**
9. מבנה של לולאת for בצורה של **for (*start*; *threshold*; *) {do}**
10. מבנה של if מהצורה של **if (*){do}**

אם אחת מהקטגוריות הזאת מתקיימת בטקסט לאחר פרסור הpayload, אני מתייחס למייל כמכיל קוד של C ולכן זורק את הפקטה.
כמובן שיש עוד דרכים לפתור, אבל הדרך הזאת הצליחה לעלות על רוב קטעי הקוד הפשוטים שכתבתי, גם כשניסיתי להעלים נקודה פסיק או דברים כאלה ולהתרכז רק ב"אלגוריתם" עצמו.
בעיני אין "שיטה מושלמת" וכנראה שלכל שיטה תהיה את הנפילות שלה, אבל הבדיקות הללו עזרו לי לתפוס את רוב הקוד שניסיתי.

קובץ **הdata_leak.c** מכיל את כל הפונקציות שמנטרות את המידע שעובר דרך פקטות הSMTP, ופקטות HTTP ומחפשות את הנקודות שפורטו פה למעלה.

הנחות נוספות והערות כלליות:

כך למדתי איך לתקן את הטעות מהתרגיל הקודם – חילוץ שדות לא זהיר.
תיקנתי את זה על ידי בדיקה שהגודל של SKB אכן תואם לפקטת IP שאני מצפה לראות.

http://portbunny.recurity.com/svn/PortBunny/trunk/PortBunny/sniffer/parse_tree.c

טעות נוספת שתוקנה – בתרגיל הקודם שרשרתי את החוקים החדשים שנטענו.
אז התיקון הוא שבטעינת חוקים חדשים, נמחקים החוקים הקודמים ולא משורשרים לחוקים הקיימים כפי שקיבלתי הערה בתרגיל הקודם.

כאן למדתי איך לעשות את החלק של הDLP
איך עובד תהליך פרוטוקול הSMTP:
http://www.tcpipguide.com/free/t_SMTPMailTransactionProcess.htm

מצאתי דיונים מעניינים ברשת על איך לגלות קוד שנשלח דרך הרשת, מה צריך לחפש כדי לזהות אותו. בהם נעזרתי כשעשיתי את החלק של הDLP

http://programmers.stackexchange.com/questions/87611/simple-method-for-reliably-detecting-code-in-text