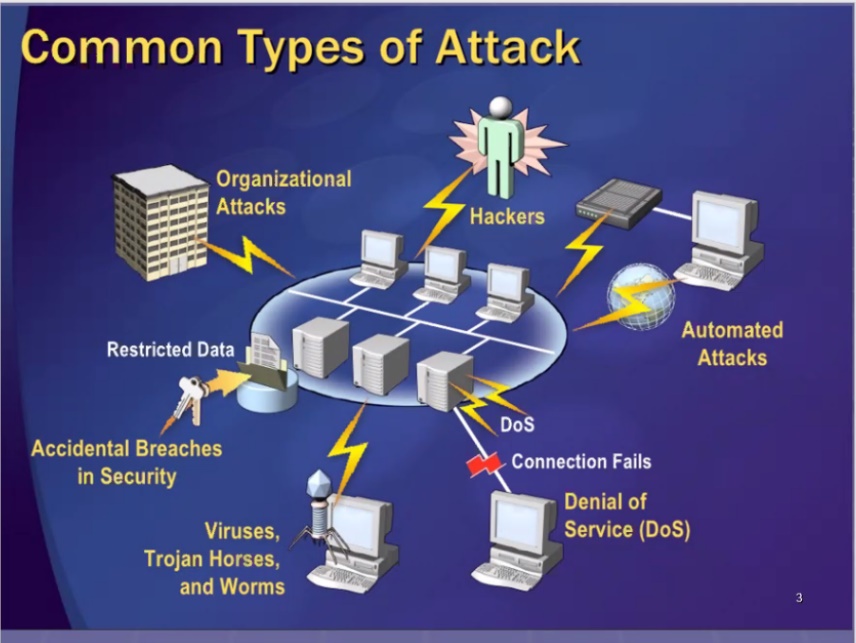
**תכנות מאובטח**

מבוא

compiler vs interpreter

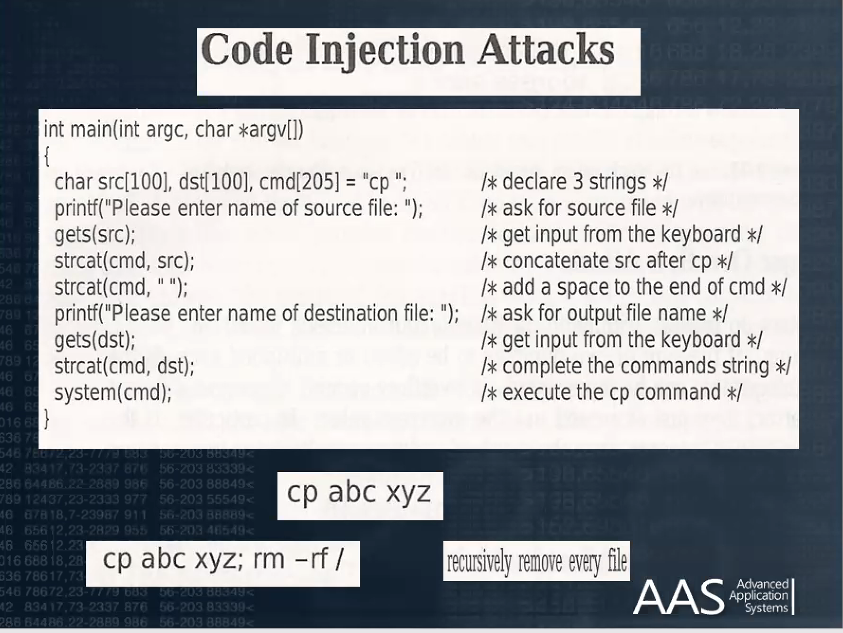
ה compiler יוצר executable מהקוד – שפת מכונה ואז מריץ אותו. כך קורה ב CPP.  
java לעומת זאת, מקומפלת על JVM, נוצר קובץ הרצה שמתאים רק לסביבת הjvm – איזשהו byte-code.   
ב jvm יש JIT- just in time compiler שאחראי להמיר את מה שאנו מריצים בJVM לשפת מכונה ואז להריץ את זה על המעבד. השלב הזה ב JIT נקרא HOT SPOT.  
  
ה interpreter מהדר שורה שורה ואז מריץ אותה – BASH, js, python



חולשות  
מציאת חולשות היא בעצם להשתמש במקומות פתוחים בקוד ולנצלם לשימוש אחר ממה שכותב הקוד התכוון.

**Snort** – כלי למניעת תקיפות DOS.  
**tunnelbear** – vpn חינמי

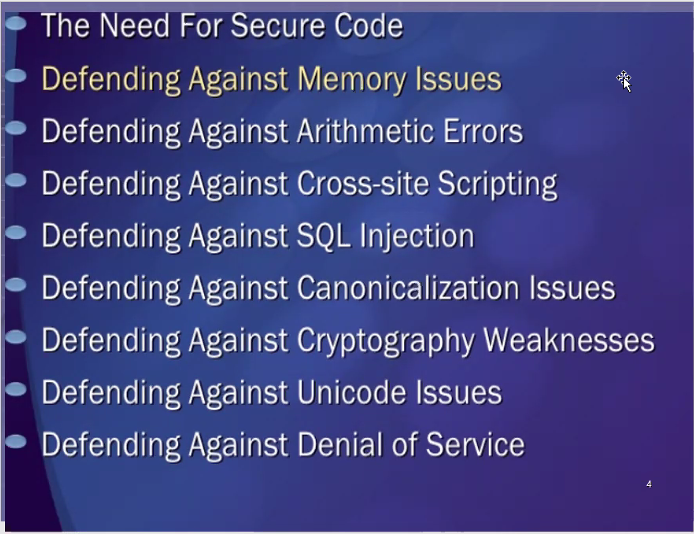
Code injection



ניתן להשתיל למשתנה dst ערך זדוני.   
בדוגמה למעלה אנו רואים כי הושתלה לו פקודה שמוחקת את כל הקבצים למערכת.  
יד לבצע ואלידציה לערכים המוכנסים במשתנים – למנוע injection, buffer overflow.

איך למנוע privilege escalation לקיחה של הרשאות של תוכנה מסוימת ולהשמיש אותם לצורכיך.  
לצורך העניין – לתוכנה שלפנינו ניתנו הרשאות מערכת גבוהה – דרכה תוקף יכול להריץ איזה פקודות שהוא רוצה.

אילו מתקפות אנו רוצים למנוע בתור מפתחים?



Arithmetic errors – חילוק באפס, החלת ערך גדול מדי ל int

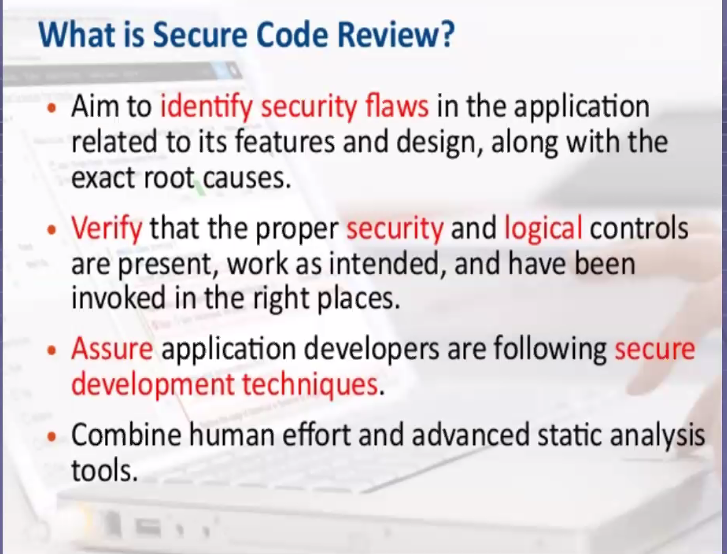
Sdlc overview – System Development Life Cycle

SRS – software requirements specification  
כל פיתוח מערכת מורכב מכמה שלבים.  
שלב האפיון הוא החשוב ביותר, מה שכתוב בו זה איך שהמערכת תיראה בסופו של דבר.  
הרובד האבטחתי באפליקציה – בעיות/פתרונות – גם הוא יקבל מענה באפיון.



**מודל מפל המים** – עובדים בשלבים לפי הסדר, עד שלא הושלם שלב אחד לא מתקדמים לשלב הבא. יש סדר בשלבי הפיתוח.  
  
**DevOps –** SDLC שמממש עקרונות agile .. עקרונות של עבודה אינטגרלית בין כל הצוותים במערכת, תוך הבנה של ה life cycle הכללי.

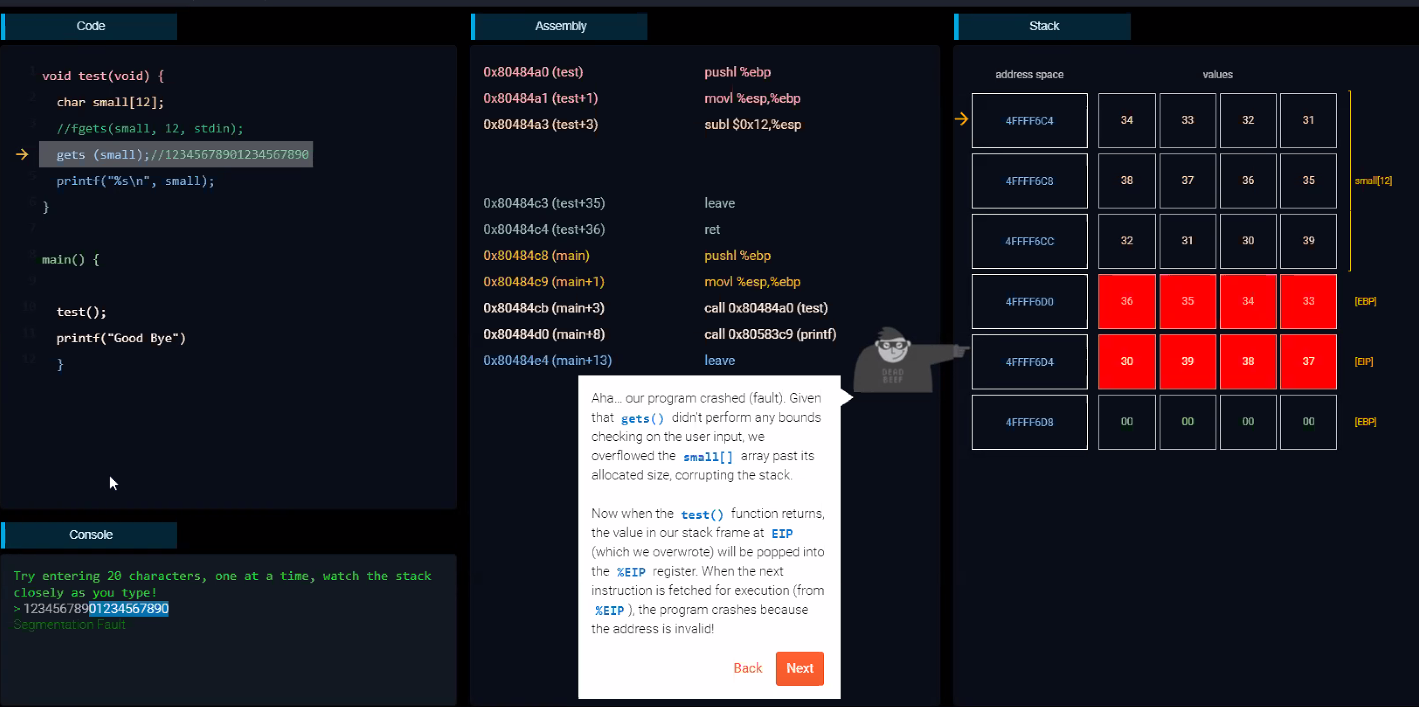
Secure code review



תוכנות לבדיקה של הקוד

**Sonarcube, whitesource, checkmark, codebashing(c++/c)**

**Codebashing**

Buffer overflow

בדוגמה שלפנינו, בפונקציה test, יש קבלת input בגודל 12.   
ניתן לראות כי המיקום המוקצה לinput, גובל במיקום בזיכרון שהוקצה לפונקציה printf שתורץ אחרי test . אם נכניס לזיכרון ערך גדול יותר מ12, הוא ייכתב למיקום של printf. זוהי חולשה בקוד, שיכולה להקריס את התוכנה.  
את החולשה הזו ניתן להשמיש כך:  
נשתיל באותו הזיכרון, כחלק מהinput, איזשהו exe שקימפלנו – פונקציה שאתה באמת רוצה להריץ, אותו קטע נוזקה שקימפלנו לשפת המכונה.   
כך בעצם, אתה כותב את הנוזקה לזיכרון של המחשב, למיקום של הפונקציה printf, והמחשב יריץ אותה כשהוא יריץ את אותה הפונקציה. כך אתה יכול להריץ מה שאתה רוצה בעצם, כנגד אותו מחשב.

- יש compilerים היום שניתן להגדיר עליהם, לא לקמפל ערכים בזיכרון סמוך לפונקציות, כדי למנוע מצבים כאלה.

**- ida**כלי ל reverese – עם הכלי ל reverse אנו מקבלים את הערכים שנמצאים בתוך המלבן הירוק בתמונה. ממנו ניתן להבין

**Gdb -**

Segmentation fault



בדוגמה שלפנינו, אנו מעבירים את המשתנה argv[1] by reference. זאת אומרת, שהמחשב מעביר לפונקציה copy את המיקום של ה byte הראשון בזיכרון, ללא ציון אורך המחרוזת. איך המחשב יודע להגיד מה הbyte האחרון שהוא צריך לקרוא ממנו בקוד אם כך?  
הפתרון הוא שהמחשב משרשר לכל מחרוזת שכזו \0 בסופה. המשמעות של זה היא שבעצם המחשב מצרף עוד byte אחד לכל string כך שבעצם, ניתן להכניס פרמטר עד 7 תווים – אחרת יהי לנו overflow – /0 ייכתב לנו byte אחד קדימה, כך שמחרוזת בת 8 בתים היא בעצם 9 בתים בזיכרון, זהו segmentation fault .

Heap

לעיתים אנו לא יודעים כמה מקום צריך להקצות לכל משתנה – אנו לא יודעים מה יהיה הגודל שלו. אם משתנה יכול לקבל ערך של מיליון או ערך של 1, אנו נרצה להקצות לו זיכרון בצורה דינאמית – כך אנו ננצל את הזיכרון שלנו גם בצורה יעילה – נקצה זיכרון לפי מה שאנחנו צריכים באותו רגע.

משתנים מסוג זה ישמרו לנו לא זיכרון אלא ב heap.

יש כמה מעברים בקומפיילר:

1. קומפילציה לשפת מכונה – קובץ obj. מכיל הפניות לכל הפונקציות
2. כאן נכנס לתמונה הlinker – הוא לוקח את כל ההפניות ומייצר לנו את קובץ ה exe עם כל ההפניות והרצף ביניהם.

כיום, הקוד שלנו מכיל המון ספריות חיצוניות .. לא נרצה שה compiler שלנו יקמפל לנו את כולם לתוך ה exe שלנו .. אלא נרצה שהם יהיו קיימים ו on the fly הקריאות בקוד ילכו וישתמשו בהם. הספריות החיצוניות הללו, מקומפלות לקבצי dll – אליהם תינתן קריאה on the fly.   
dll- dynamic library link.  
למשל, הפונקציה printf .. פונקציה שכולנו משתמשים בה .. לא נרצה לקמפל את כל הקוד שמאחוריה, אז בעצם נייבא אותה מספרייה דינאמית אחרת.   
אז בשלב הראשון הקומפיילר יחליף את printf שלנו להפנייה לdll . לאחר סיום הקומפיילר, תוך כדי ריצה יוקצה מיקום לprintf – לפונקציה החיצונית.   
המיקום המוקצה הזה תוך כדי ריצה לdll כלשהו, לפונקציונליות שלו, נשמר לנו גם על ה heap, כמו גם המשתנים הדינאמים.   
אנו צריכים להציב הגנה על הheap שלנו.

על מנת להקצות זיכרון למשתנים דינאמיים תוך כדי ריצה , ב heap שלנו, נשתמש בפונקציה – malloc(int bytes in memory) – מקבלכמה byte להקצות בזיכרון – memory allocation  
memset(pointer\*, value to write to the pointer, bytes in memory) – מי שכותב בפועל לתוך הזיכרון, מקבל את הפוינטר(הbyte הראשון של אותה כתובת), מקבל ערך לכתוב לזיכרון, ומקבל את מספר הבייטים שלרשותו.  
free(pointer) – משחרר לנו את הזיכרון מה heap

אם אנחנו לא משחררים מקום מהמערכת הפעלה – יש לנו זיכרון שהוא מוקצה אך לא נגיש.. זיכרון זה יכול להיות מושמש להתקפה .. או פשוט להביא לקריסה בסופו של דבר.

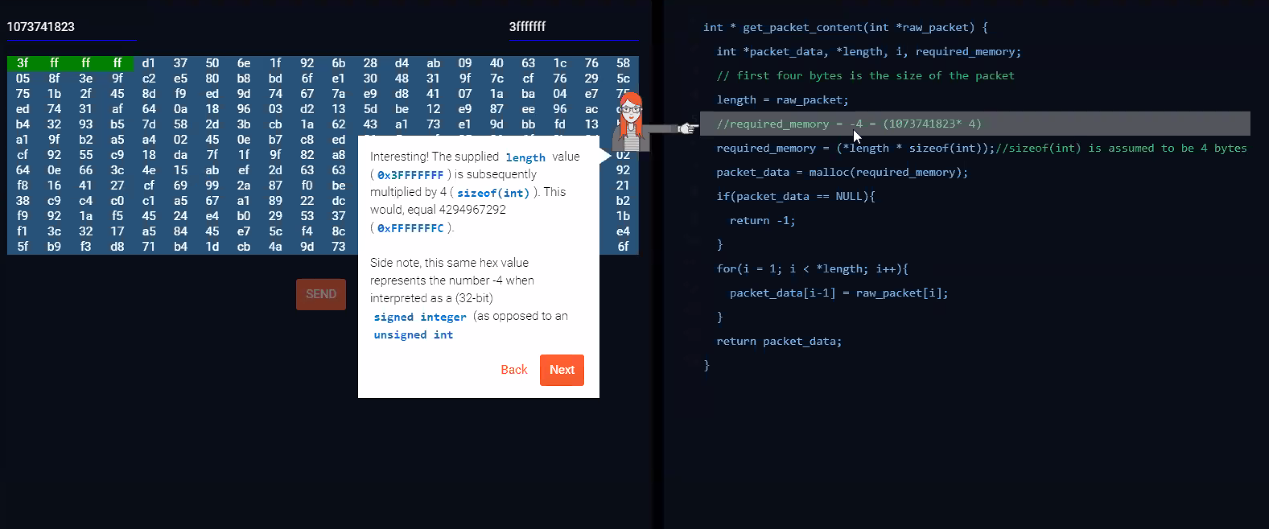
מערכת הפעלה

אחראית על ניהול המשאבים – ביניהם, ניהול זיכרון. במערכת ההפעלה שלנו ב windows לדוגמה, יש לנו dll שאחראים להקצאת זיכרון – הקוד שלנו ב c++ נבנה על הdll הנתונים מwindows.   
אנו צריכים לדאוג שאף אחד לא משנה לנו את ה dllים שאנו מסתמכים עליהם.

CVE

Database ענקי של כל חולשות הקוד הידועות. Sonarqube מודיע למה הוא התריע על משהו כחולשה ונותן reference לdescription של החולשה באותו CVE.

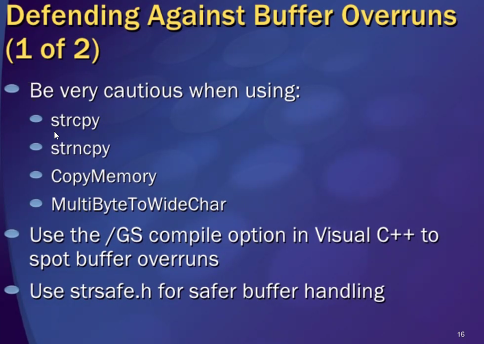
Mitre att&ck ו NIST  
תאגידים שמאגדים תחתם המון חולשות ותקנים לפיתוח מאובטח – על פיו צריך לתכנת באופן מאובטח – נחשב לסטנדרט מאובטח בארה"ב ובעולם

Integer overflows  


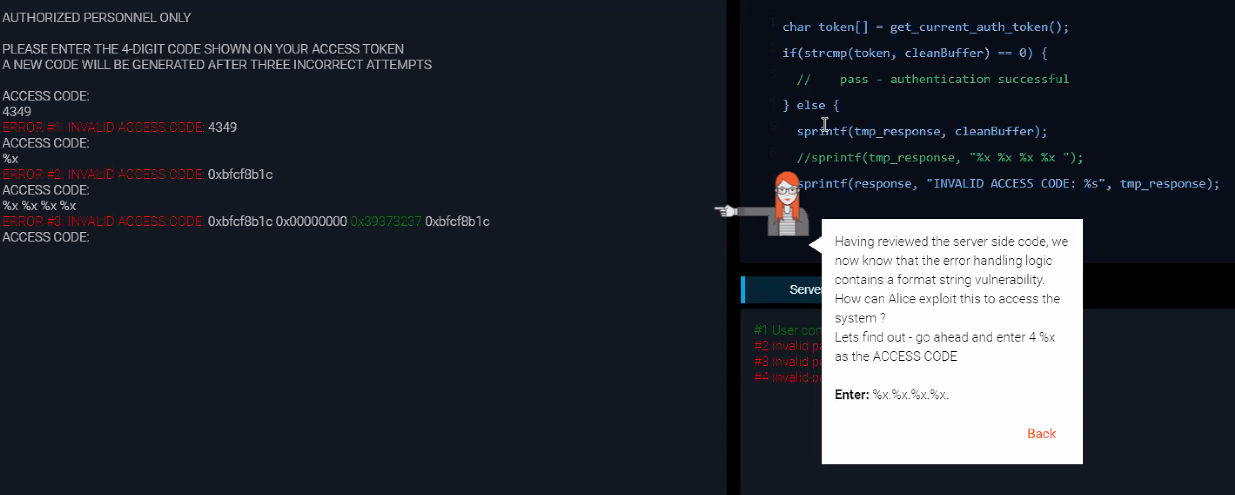
בקטע הקוד שלפנינו, יש קבלה של פקטות מהרשת והקצאת זיכרון עבורם..

בשורה המסומנת, לאחר שהוכנס ערך בעל מספר ביטים גדול ממספר הביטים הנתון ב integer, אנו מקבלים חריגה של 4 ביטים ..

Defending against buffer overruns



String format – server attack



sprintf(response, “”IB+NVALID ACCESS CODE: %s”, tmp response);

בשורה שלפנינו, יש לנו את הפקודה sprintf שתפקידה לקחת את מה שניתן בפרמטר השני שלנו  
יחד עם ה הפרמטר שהתקבל בפורמט ולהכניס לפרמטר הראשון. בעצם יש לנו כאן השמה של מידע לתוך משתנה ללא בדיקה – מבוא ל segmentation fault, buffer overflow .. יתרה מכך, הפרמטר שמוכנס לשרת מתקבל על ידי הלקוח – תקיפת remote buffer overflow .

רוב המערכות עובדות עם sprint

קימפול ללא ה Stack protector – מאפשר segmentation-fault

gcc -fno-stack-protector server.cpp -o server

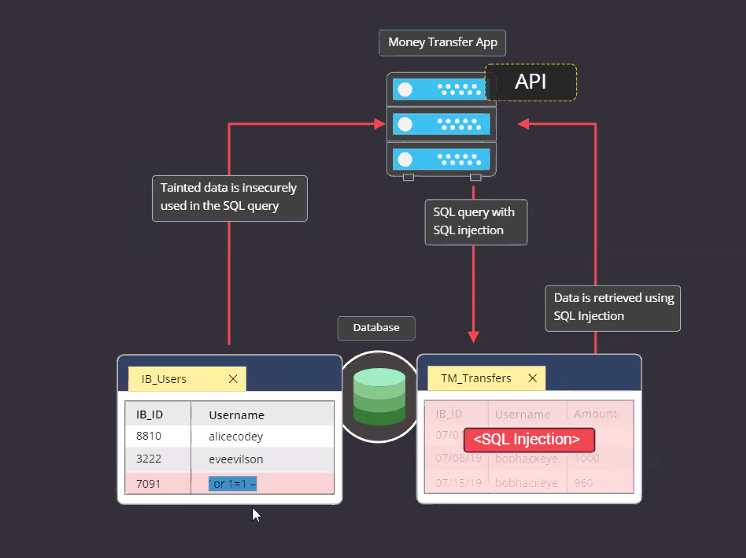
sql injection



לדוגמה, בשרת הזה נוכל להכניס בemail או בpassword –  
‘--’ – משמש להערה ב sql

’ OR 1=1 –

Second order sql injection



Sql injection ממדרגה שנייה היא כאשר תוקף מצליח להזריק לטבלה אחת ב db מספר אחד את ה sql injection .. בהמשך , אם וכאשר תיהיה שאילתה מtable אחד ל table אחר , עם הusername השמור בטבלה .. יהיה לנו injection ממדרגה שנייה.

Cmd line injection  
לעיתים, בתוכנות יש קריאה לפקודות מערכת עם פרמטרים שמתקבלים מהמשתמש.  
יש להשתמש בפקודות מובנות במערכת ולא בקריאה לפקודת מערכת .. כמה שניתן.

name = input(“enter name”)  
os.system(“mkdir ” + name)

בדוגמה הזאת למשל

ניתן לעשות לname injection של פקודה כלשהי על ידי ; בלינוקס ולהריץ מה שאנחנו רוצים ..

snort

כלי תקיפה נגד SSH

burp

כלי לבדיקות אבטחה של יישומי אינטרנט – יושב כאיזשהו proxy בין ה client ל server - ומנסה להכניס שם דברים.

Kali

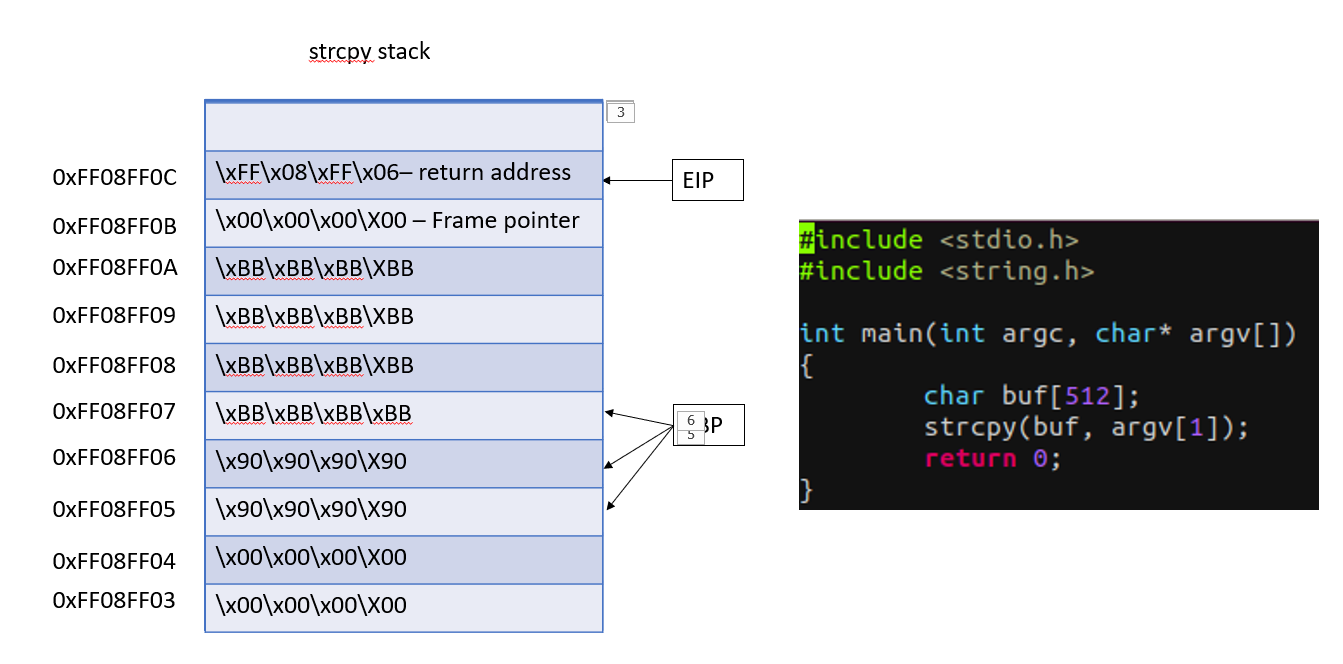
מערכת הפעלה לבדיקות ברשת

runsomewhere

כניסה למחשב, הצפנה של קבצים עם מפתח – תוכנת כופר

Modern auto

Buffer overflow exploit



דשגדש