עבודה סמינריונית

תכנות מערכות דפנסיבי

סמינר: תכנות מערכות דפנסיבי 20928

מגיש: עמית סידס

בהנחיית: פרופ' לאוניד ברנבוים

תעודת זהות: 206768780

תאריך: 12 באפריל 2021

תוכן עניינים

תוכן עניינים 1
מבוא 3
שפת C++ 4
מבנה ותחביר השפה 4
מחלקות, אובייקטים וירושה 4
פולימורפיזם 5
דוגמאת שימוש 5
מודל הזיכרון 6
איומי אבטחה 7
חולשת Stack Buffer Overflow 7
מתקפת Return-oriented Programming 7
דריסת משתנים וטבלאות וירטואליות 8
מנגנוני אבטחה 9
שפת Python 10
מבנה ותחביר השפה 10
דינמיות השפה 10
מחלקות, אובייקטים וירושה 11
דוגמאת שימוש 11
איומי אבטחה 12
חולשות ב-Deserialization 12
טעינת קובץ זדוני 12
חילוץ קבצים זדוניים 13
שימוש בפונקציות מסוכנות 13
מנגנוני אבטחה 13
תקשורת 14
אבטחה בתקשורת 14
פרוטוקולי הצפנה ואימות 14
פרוטוקול Diffie–Hellman 14
הצפנת RSA 15
תעודה וחתימה דיגיטלית 15
פרוטוקולי SSL ו-TLS 16
מתקפות שונות על פרוטוקולי תקשורת 17
מתקפת POODLE 17
מתקפת BEAST 17

מתקפת CRIME 18
מתקפת Heartbleed 18
הגנות והקלות מפני מתקפות ברשת 19
אבטחת מערכת 20
מתקפות MiTM 20
מתקפת ARP Poisoning 20
מתקפת DNS Poisoning 21
מתקפות מניעת שירות 22
מתקפת DoS 22
מתקפת DDoS 22
מתקפת DRDoS 23
הגנה מפני מתקפות מערכת 24
סיכום 25
ביבליוגרפיה 26

מבוא

עולם המחשבים והתקשורת הוא גדול ומגוון. התקדמות הטכנולוגיה עם השנים הובילה לכך שהשימוש במחשבים וברשת האינטרנט הוא חלק אינטגרלי בחיי היומיום של מרבית האנשים בעולם, והוא משפיע באופן ישיר או עקיף על כולם. יחד עם היתרונות הרבים שהביאה איתה ההתקדמות הטכנולוגית הזו, הגיעו גם אנשים אשר מנסים לנצל אותה לרעה לצורך טובתם האישית. אותם אנשים גרמו להתפתחות האבטחה בעולם המחשבים כפי שאנו מכירים אותה כיום.

אבטחת המידע והרשת הוא תחום שמטרתו להגן על אנשים, חברות, ארגונים ומדינות מפני התקפות וניצול לרעה של גורמים עוינים המנסים לפגוע בהם. תחום אבטחת המידע הוא תחום נרחב ביותר שנועד לספק הגנה במגוון צירי התקפה והוא יכול להתפרס על מספר רב של ענפים.

בסמינר זה אציג ארבעה פרקים, שכל אחד מהם יסקור חלק מסוים, מעולם אבטחה, שאותו ניתן לנצל לרעה ולהתקיף באמצעותו את המערכת המשתמשת בו. בנוסף, עבור כל סוג מתקפה כזו אציע פתרונות ושיטות הגנה שמטרתן למנוע את המתקפה או לכל הפחות להקל עליה.

הפרק הראשון יעסוק ב-שפת C++ ויציג מספר מתקפות וחולשות שניתן לנצל אם לא עושים שימוש נכון במרכיבי השפה. בנוסף, נסקור מספר מנגנוני אבטחה שהתפתחו עם השנים שמטרתם למנוע את המתקפות הללו ולהקשות מאוד על תוקפים שינסו לנצל אותם.

הפרק השני יעסוק ב-שפת Python וגם בו יוצגו דרכים שונות שבהן ניתן לנצל שימוש שגוי בשפה בשביל להתקיף את הגורמים שמריצים תוכנות הכתובות בה. לעומת C++, שפת Python נחשבת ליותר מאובטחת ועל כן ברוב המתקפות המודעות אליהן יכולה לשפר משמעותית את האבטחה מפני המתקפות השונות.

בפרק השלישי אציג מספר פרוטוקולי תקשורת וכיצד ניתן לנצל אותם בשביל לתקוף רכיבים בעולם התקשורת ולחשוף מידע פרטי העובר ברשת. בנוסף, בפרק נציג את השיפורים של אותם פרוטוקולים ויראה כיצד אותם שיפורים מונעים מתקפות וחולשות אשר היו אפשריים בגרסאות ישנות שלהם.

הפרק הרביעי והאחרון יעסוק ב-אבטחת מערכת. בפרק זה יוצגו מספר דרכים ושיטות שבאמצעותן תוקפים יכולים לנסות לפגוע במערכת שלנו ולהזיק למשתמשים שלה. בהתאם, יוצגו מספר פתרונות אשר מאפשרים למנוע או להקל על אותן תקיפות ובנוסף כיצד ניתן להעזר בחברות מסחריות אחרות בשביל למנוע אותן.

לבסוף, אציג את הסיכום של כל הפרקים הללו ואת התובנות והמסקנות שהגעתי אליהן במהלך עשיית העבודה.

שפת C++

שפת C++ נוצרה לראשונה בשלהי שנות ה-70, ע"י מדען המחשבים ביארן סטרוסטרופ כשעבד על עבודת הדוקטורט שלו [1]. בעבודתו, הוא נהג להעזר בשפת התכנות Simula, שהייתה אחת מבין שפות התכנות הראשונות שתמכו במודל התכנות-מונחה עצמים. לטעמו, השפה הזו הייתה איטית מידי לשימושים מעשיים ועל כן החליט לעבוד על פיתוח שפה חדשה המובססת על שפת C אשר תאשר שימוש במחלקות. בזמנו, הוא כינה אותה בשם "C with Classes", ועם השנים שמה שונה ל- C++. כפי שהשם מרמז, מדובר על שפת C עם עוד תוספות ושדרוגים, כפי שעושה האופרטור ++ אשר מגדיל את ערכו של משתנה ב-1.

במהלך השנים, שפת C++ התקדמה והשתדרגה באמצעות עוד תוספות ושדרוגים לשפה כמו: מחלקות, ירושה, מנגנון טיפול בחריגות (Exceptions), פונקציות וירטואליות, העמסת פונקציות, תבניות, ניהול זיכרון ועוד... [2] תוך כדי פורסמו מספר סטנדרטים המגדירים את מבנה, תחביר ומאפייני השפה, כשבכל אחד מהם הוספו, עודכנו ושונו הרבה מאוד תכונות בשפה שעזרו לה להתפתח והביאו אותה עד למצבה היום. כל 3 שנים, מאז שנת 2011 שבה יצא הסטנדרט C++11, שכלל שינויים גדולים ומשמעותיים, פורסם סטנדרט חדש שהוספו אליו כל התוספות והשינויים שסיימו לפתח עד פרסום הסטנדרט [3]. הסטנדרט האחרון שפורסם ממש לאחרונה (בעת כתיבת המאמר) הוא C++20 [4].

C++ מבוססת על השפה שקדמה לה, C. על כן, רוב התכנים של השפות זהה ואפילו יש לומר ש-C++ שומרת על תאימות לאחור עם C, וכך כל קוד C בסטנדרט הנפוץ יוכל להתקמפל (לעבור הידור – compilation) בעזרת compiler מתאים הבנוי עבור C++. לפיכך, לא אציין את כל מאפייני השפה אלא רק את הדברים שנוספו עבור C++ והופכים אותה לייחודית.

מבנה ותחביר השפה

כמו C, גם C++ היא שפה פרוצדורלית. כלומר, ישנה חלוקה בין פרוצדורות (או פונקציות) אשר כל אחת אחראית על פעולה מסוימת וכל פרוצדורה יכולה לקרוא לאחרת על מנת שתבצע את הפעולה עליה היא אחראית. בכל פרוצדורה, ניתן להגדיר ולהשתמש במשתנים אשר מיוצגים ע"י אחד מהטיפוסים הבסיסיים של השפה, או להגדיר טיפוסים חדשים שיכילו את אחד או יותר מהטיפוסים הבסיסיים האחרים [5].

בנוסף, ישנן כמה דרכים שונות לשלוט בזרימת התוכנית. הדרך הבסיסית ביותר היא באמצעות פקודת ה-if המאפשרת לבדוק האם תנאי מסוים מתקיים או לא. בנוסף, קיימות 3 פקודות המאפשרות לבצע לולאה (קטע קוד החוזר על עצמו מספר פעמים) עד שתנאי כלשהוא יופר: for, do, while. יתר על כן, קיימת הפקודה switch המאפשרת להשוות ערך של משתנה למספר רב של ערכים קבועים בצורה יעילה (בעזרת jump table לרוב – תלוי compiler), וגם שימוש בפקודות כמו continue, break ו-goto אשר משנות את זרימת התוכנית בהתאם לשימוש שלהן בקוד.

מחלקות, אובייקטים וירושה

מחלקה (Class) היא קבוצה של משתנים. בשונה מ-struct אשר קיים כבר ב-C, מחלקה כוללת בנוסף למשתני המידע גם פונקציות המוגדרות עבור המחלקה. אובייקט (Object) הוא מופע (Instance) של מחלקה. בדומה לכך שניתן להגדיר משתנים מטיפוסים בסיסיים, כך ניתן להגדיר משתנים שהם אובייקטים מטיפוס של המחלקה [6]. כל אובייקט שומר מצביע לפונקציות המחלקה המוגדרות עבורו.

ניתן ליצור מחלקה אשר יורשת ממחלקה אחרת. כך, כל אובייקט של המחלקה היורשת יכיל גם את המשתנים של המחלקה ממנה הוא יורש, ובנוסף גם את הפונקציות המוגדרות עבורה [7].

לכל משתנה במחלקה ניתן להגדיר מאפיין גישה (Access Specifier) אשר קובע מי יכול לגשת לאותו משתנה ומהיכן ניתן לגשת לאותו משתנה. מאפיין הגישה הדיפולטיבי הוא private המציין שניתן לגשת למשתנים המוגדרים באמצעותו אך ורק מפונקציות המוגדרות במחלקה של המשתנה. שאר המאפיינים הם:

1. protected אשר מגדיר שניתן לגשת למשתנה במחלקה הנוכחית וכל מחלקה היורשת ממנה.
2. public אשר מאפשר לגשת למשתנה מכל פרוצדורה הנגישה לאובייקט של המשתנה. [6]

פולימורפיזם

פולימורפיזם היא תכונה של שפות תכנות המאפשרת להשתמש באותם סימבולים, כמו שמות פונקציות או אופרטורים, למימוש שונה בהתאם לצורה שבה משתמשים בהם. C++ מממשת פולימורפיזם במספר רחב מאוד של צורות, ובכך מאפשרת תכנות גנרי אשר מונע כפילות קוד וחוסך עבודה למתכנת [8]. דוגמאות נפוצות לשימוש בפולימורפיזם בשפת C++ הן:

- העמסת פונקציות – הגדרת מספר פונקציות בעלות אותו שם אך עם חתימה שונה. הקומפיילר יודע בעזרת החתימה איזו פונקציה צריך להריץ. ניתן בצורה זו גם ליצור העמסה של אופרטורים ע"י הגדרת הפונקציות אשר ירוצו עבור כל אופרטור בשפה.
- דריסת פונקציות – החלפת מימוש של פונקציה במחלקה היורשת ממחלקה אחרת שבה הפונקציה מוגדרת כפונקציה וירטואלית. במצב כזה המצביע לפונקציה השמור באובייקט יתעדכן בהתאם לפונקציה של המחלקה אשר דרסה אותה.
- תבניות (Templates) – הגדרת פונקציה או מחלקה עבור טיפוסי מידע גנריים. בצורה זו ניתן להגדיר מימוש יחיד עבור פונקציה או מחלקה והקומפיילר ידע ליצור מימוש מתאים בכל פעם שמשתמשים בתבנית בהתאם לטיפוסים שאיתם עושים שימוש בתבנית. [9]

דוגמאת שימוש

להלן דוגמא קצרה לשימוש במחלקות, ירושה ופולימורפיזם בשפת C++:

```cpp
class Animal {
protected:
    std::string name;
    int         age;
public:
    Animal(std::string name, int age)
    {
        this->name = name;
        this->age = age;
        std::cout << "My name is " << this->name << " and I'm ";
        std::cout << this->age << " years old." << std::endl;
    }

    virtual void make_sound() = 0;
};

class Dog: public Animal {
    using Animal::Animal; // Use constructor of Animal
public:
    void make_sound() override
    {
        std::cout << this->name << " Barks!" << std::endl;
    }
};

class Duck: public Animal {
    using Animal::Animal; // Use constructor of Animal
public:
    void make_sound() override
    {
        std::cout << this->name << " Quacks!" << std::endl;
    }
};

int main() {
    Dog dog = Dog( name: "Rexy",  age: 5);
    Duck duck = Duck( name: "Nini",  age: 3);
    dog.make_sound();
    duck.make_sound();
    return 0;
}
```

בדוגמא מוגדרות 3 מחלקות. מחלקת אב ושתי מחלקות אשר יורשות ממנה. מחלקת האב נקראית Animal. במחלקה זו מגדירים משתנים המשותפים לכל בעלי החיים כמו שמם והגיל שלהם. בנוסף, במחלקה Animal מוגדרת פונקציה וירטואלית בשם make_sound. לפונקציה זו אין מימוש והיא נחשבת לפונקציה אבסטרקטית משום שלכל חיה יש קול שונה. המחלקות היורשות מ-Animal נקראות Dog ו-Duck. כאשר בשתי המחלקות דורסים את הפונקציה הוירטואלית make_sound ונותנים לה מימוש מתאים בהתאם לקול שעושה אותה חיה.

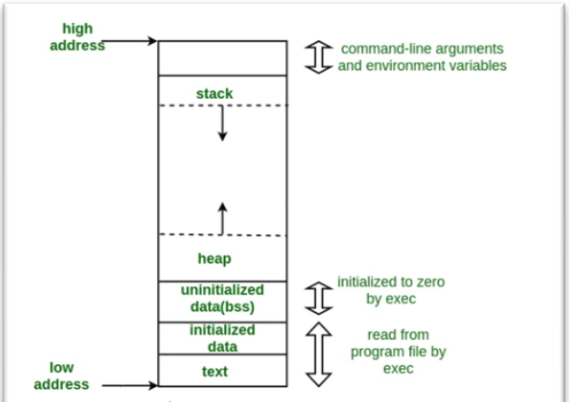
בנוסף, ניתן לראות שמשום ששם הפונקציה המוגדר במחלקה Animal מוגדר עם ה-access modifier המתאים של protected, אז ניתן לגשת לשם הזה במחלקה היורשת במימוש של הפונקציה make_sound. יש לשים לב שלא ניתן לגשת למשתנים של שם וגיל בפונקציית ה-main בעקבות זאת.

כפי שמצופה, הפלט המתקבל מהרצת הקוד הזה הוא:

```
My name is Rexi and I'm 5 years old.
My name is Nini and I'm 3 years old.
Rexi Barks!
Nini Quacks!

Process finished with exit code 0
```

מודל הזיכרון



מבנה הזיכרון במערכות הפעלה נפוצות.
נלקח מהאתר geeksforgeeks.org [10]

כמו בכל תהליך מחשב במערכת הפעלה סטנדרטית, גם תוכניות הכתובות ב- C++ מחולקות במבנה הזיכרון למקטעים (Segments) מסויימים, חלקים הכרחיים וחלקם פחות. מקטעי הזיכרון ההכרחיים ביותר הם ה-text, שבו שומר הקוד של התהליך, וה-stack, שבו נשמרת מחסנית הריצה של התוכנית הכוללת משתנים סטטיים, פרמטרים וכתובות חזרה מקריאות של פונקציות. בנוסף להם, ישנם גם את המקטעים של ה-heap (ערימה), שבו נשמרים האובייקטים וקטעי הזיכרון אשר מוקצים בצורה דינאמית, ה-bss, שבו נשמרים המשתנים אשר אינם מאותחלים (ויאותחלו ל-0 בתחילת הריצה), וה-data, שבו נשמרים כל המשתנים הגלובליים שאינם שייכים לאף scope ריצה מסוים ותמיד צריכים להשאר בזיכרון (למעשה, המשתנים הגלובליים ואלו המוגדרים כסטטיים בחלקי הקוד). [10]

בשפת C++, כמו ב-C, משתנים אשר מוגדרים ב-scope של פונקציה ישמרו ב-stack, אשר הולכת וגודלת לכיוון המקטע של ה-heap. בנוסף להם, לעיתים גם נשמרים ב-stack הפרמטרים המועברים לפונקציות או אוגרים (registers) אשר רוצים לשמור את ערכם לצורך שימוש בהם. ויתר על כן, לרוב גם נשמרת שם כתובת החזרה מהקריאה לפונקציה או כתובת המחסנית של הפונקציה הקודמת, ולעיתים גם ה-stack canary (או stack cookie) אשר נשמרת על מנת להגן על תקינות המחסנית.

ב- C++, בשביל להקצות אובייקט בזיכרון בצורה דינאמית יש להשתמש באופרטור new ובאופרטור המקביל delete בשביל לשחרר את ההקצאה, זאת להבדיל מ-C שבה יש להשתמש בפונקציה malloc ומשפחתה ו-free לצורך השחרור (אולם ניתן להשתמש בהם גם ב- C++ כמובן). האופרטורים new ו-delete קוראים לפונקציית ה-constructor וה-destructor (בהתאמה) ומקצים או משחררים בלוקי זיכרון על ה-heap [11]. בנוסף, כדי למנוע מצבים של דליפות זיכרון עקב הקצאת אובייקטים ללא שיחרורם (אשר עלולים להגרם הרבה במצבים בהם נזרקת שגיאת ריצה), הוספו לשפה מחלקות הממשות מצביעים חכמים (Smart Pointers) ע"י מחלקות כמו unique_ptr ו-shared_ptr, אשר דואגים לשחרר את האובייקט ברגע שהמצביע אליו משוחרר מהזיכרון או כאשר המצביע האחרון לאותו אובייקט משתחרר או נדרס.

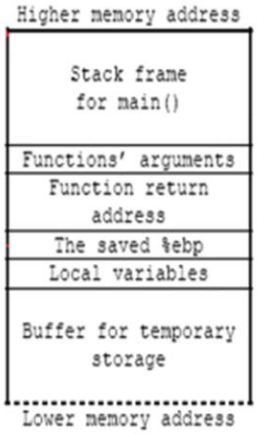
ניתן לשים לב כי קטעי הזיכרון של ה-stack וה-heap מתקרבים אחד לשני ככל שמשתמשים בהם יותר, לכן כמות המקום של כל אחד מהם מוגבלת ולפעמים הקצאות דינאמיות יכשלו או, במקרה הגרוע יותר, התוכנית תקרוס עקב דריסת מידע של קטע זיכרון כלשהו ע"י השני.

איומי אבטחה

חולשת Stack Buffer Overflow

Stack Buffer Overflow היא חולשה מסוג Buffer Overflow, שהיא קבוצה של חולשות אשר בהן נעשה שימוש זדוני בתוכנה בשביל לדרוס מידע בזיכרון מעבר למקום שהוקצה עבור ה-buffer שבו נשמר המידע הזדוני שנכתב.

כידוע, buffer (חוצץ) הוא חלק רציף בזיכרון הנדיף של התוכנה, שבו נשמר מידע הנעשה בו שימוש זמני בתוכנה. כאשר מפתח התוכנה אינו בודק גבולות גישה עבור הגישות ל-buffer הזה, עלול להיווצר מצב שבו התוכנה ניגשת לחלקים בזיכרון שאינם נמצאים באיזור המוקצה ל-buffer, לרוב לחלקים שבאים אחריו (כלומר, בכתובת גבוהה יותר). לעומת Heap Overflow, שבה ה-buffer נשמר באיזור ה-heap ע"י הקצאה דינאמית, בחולשת ה-Stack Buffer Overflow ה-buffer מוקצה בצורה סטאטית על מחסנית הריצה של התוכנית. כפי שכבר הוזכר קודם, בהרבה מאוד ארכיטקטורות, באותו חלק בדיוק נשמרים נתונים חשובים כמו משתנים מקומיים, מצביע לתחילת המסגרת (frame) של הפונקציה, והחשוב מכל: כתובת החזרה שממנה הפונקציה תחזור לפונקציה שקראה לה. [12]



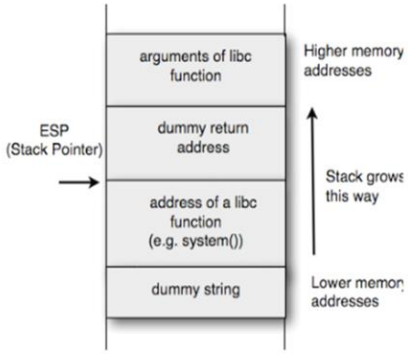
דוגמא לפורמט המחסנית בארכיטקטורת x86
נלקח מאתר tenouk.com - [53]

בהנתן שקיימת חולשה מסוג Stack Buffer Overflow, משתמש זדוני, אשר יש לו גישה למידע הנכתב לאותו ה-buffer, יכול לדרוס את המידע שנמצא אחריו ובכך להשפיע על הריצה של התוכנית ותקינות הקוד. בפרט, מכיוון שבהרבה ארכיטקטורות כתובת החזרה נשמרת על המחסנית אחרי ה-buffer, טכניקה נפוצה היא לדרוס את כתובת החזרה למיקום שנמצא בתוך ה-buffer עצמו, שלמשתמש הזדוני יש שליטה בתוכנו. בטכניקה זו המשתמש הזדוני יכתוב ל-buffer קטע קוד שאינו תלוי בחלקים אחרים בקוד או במיקום שלו, אשר נקרא shellcode, ובכך יוכל לדרוס את כתובת החזרה לכתובת של הקוד שלו ולהריץ קוד לפי בקשתו.

קיימות מספר רב מאוד של שיטות שונות ומגוונות אשר מציעות פתרונות על מנת לפתור בעיות הנוצרות מחולשות כאלו, שעליהן ארחיב בהמשך בתת-פרק על מנגנוני אבטחה.

מתקפת Return-oriented Programming

Return Oriented Programming היא טכניקה להרצת קוד אשר מבוססת על חולשת Stack Buffer Overflow. בחלק מהמקרים, בניסיון להשמיש חולשת Stack Buffer Overflow, לא נוכל לשלוח את ה-shellcode שלנו עקב מגבלות מסוימות כמו כמות תווים (למשל, הגבלה של גודל הודעה) או מניעה של תווים מסוימים במידע (למשל, רק תווים טקסטואלים). בנוסף, ישנן כמה הגנות, המפורטות ב-מנגנוני אבטחה, אשר אינן מאפשרות להריץ קוד אשר נמצא במקטע הזיכרון של ה-stack, ולכן גם במקרים אלו שליחת shellcode על גבי ה-buffer אינה מועילה לנו.

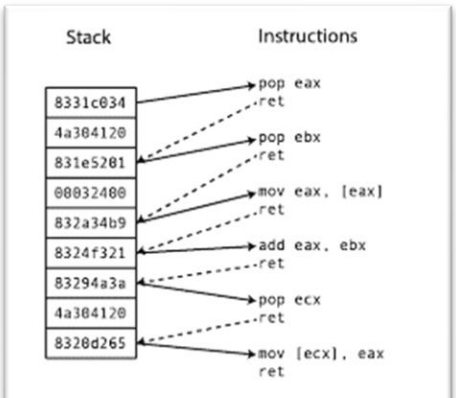


דוגמא להשמשת return-into-libc
נלקח מהמאמר [59]

עבור מקרים אלו הומצאה טכניקת Return Oriented Programming, או בקיצור, ROP. בטכניקה זו, התוקף מסתמך על העובדה כי יש לו שליטה מלאה על התוכן אשר נמצא במחסנית הריצה, ע"י דריסה שלו עם תוכן ה-buffer. בהרבה ארכיטקטורות, ובפרט בארכיטקטורת x86, ה-calling convention הנפוץ מגדיר כי יש להעביר את הפרמטרים לפונקציות על גבי מחסנית הריצה. לפיכך, מכיוון שהתוקף יכול לשלוט במחסנית, הוא יכול, באופן תיאורטי, לקבוע אילו פרמטרים יכתבו למחסנית ובנוסף איזו פונקציה תקרא עם הפרמטרים הללו ע"י דריסה של כתובת החזרה לכתובת הפונקציה. לרוב, קריאה לפונקציה אחת אינה מספיקה. על כן, התוקף יבצע את אותו התהליך שוב ושוב ע"י בניית "מחסנית ריצה" הכוללת כתובות של פונקציות (בתור כתובת חזרה) ופרמטרים עבורן, על פי ה-calling convention המתאים. כמובן שהתהליך הזה יכול להתבצע מספר בלתי מוגבל של

פעמים (בהיקפים סבירים בהתאם לחומרה), בהינתן שאין הגבלות המונעות זאת (כמו הגבלה של גודל ה-buffer). השימוש הזה ב-ROP נקרא return-into-library. זאת משום שהתוקף דורס את ה-buffer ברצף של קריאות לפונקציות אשר נמצאות בספריית פונקציות סטנדרטית כלשהיא, לדוגמא libc או STL. ספריות כאלו נחשבות למאוד נפוצות במערכות ההפעלה שלהן ולכן התוקף יכול להניח בנוחות כי המימושים לפונקציות נפוצות בהן גם נמצאים במרחב הזיכרון של התוכנית.

במקרים אחרים, בהם ה-calling convention אינו מאפשר לנו לשלוח את הפרמטרים לפונקציות על גבי המחסנית, אנו נאלץ למצוא דרך שבה נוכל לשנות את ערכי האוגרים (registers) של המעבד, שבהם מועברים הפרמטרים לפונקציות. בנוסף, במקרים שבהם קשה למצוא את מיקום הפונקציות (בגלל ASLR לדוגמא) נשתמש באותן דרכים בשביל להשפיע על ריצת התוכנית ואולי אפילו לקרוא לקריאות מערכת (syscalls) בעצמנו ולא בעזרת פונקציית ספריה.



דוגמא ל-ROP עם gadget-ים.
נלקח מהמאמר [60]

בשביל שנוכל לשנות ערכים של אוגרים בצורה נוחה ופשוטה שתאפשר לנו לבצע מספר רחב של פעולות, אנו נרצה לחפש gadget-ים רבים ככל שנוכל. Gadget הוא רצף של פקודות מעבד (instructions) אשר מבצע מספר מצומצם של פעולות כשבסופו מופיע הפקודה ret, אשר מבצעת חזרה מקריאה לפונקציה, ובפועל שולפת את כתובת החזרה שלנו מהמחסנית וקופצת אליה. למשל, בארכיטקטורת x86, gadget טוב היכול לשמש להרבה דברים הוא ה-gadget הכולל את הפקודות: pop eax; ret. ב-gadget הזה מבצעים שליפה מהמחסנית (שבתוכן שלה אנו שולטים) לאוגר ה-eax ולאחר מכן מבצעים ret וקופצים לכתובת החזרה (שגם עליה ניתן לשלוט). תוקף, אשר מעוניין להריץ קוד מבלי לשלוח אותו על גבי ה-buffer, יכול לבנות מחסנית ריצה שלמה המבוססת על gadget-ים כאלו, אשר רצים אחד אחרי השני כשבכל אחד מהם מבצעים מספר קטן של פקודות, המקדמות את לוגיקת הקוד שהתוקף רוצה להריץ, ואחריהן מבצעים ret וקופצים ל-gadget הבא...

באופן תיאורטי, אם תוקף נגיש ל-gadget מהסוג הזה עבור כל אוגר אשר קיים במעבד, הוא למעשה יכול לשנות כל אוגר לכל ערך שיבחר ובכך להריץ כל סוג קוד שרק יבחר, בהתאם למגבלות על תוכן ה-buffer.

ישנם סוגים רבים של gadget-ים אפשריים המאפשרים דברים שונים. ישנו gadget הנקרא gadget קריאה, אשר מבצעים בו טעינה של מידע מכתובת השמורה באוגר מסוים לתוך אוגר אחר. בהנחה שניתן לשלוט בתוכן האוגר שממנו קוראים, ניתן לקרוא מכל כתובת שהתוכנית נגישה אליה. בהתאם, ישנו gadget הנקרא gadget כתיבה אשר מבצע את הדבר ההפוך ומאפשר לכתוב ערך של אוגר לכתובת השמורה באוגר אחר, ובכך התוקף יכול לכתוב כל מידע שיבחר לכל כתובת הנגישה לתוכנית. שילוב של gadget-ים כאלו מאפשר לתוקף להזליג מידע חשוב לגבי התוכנה, ובהנחה שמדובר בשרת רשתי, דבר כזה עלול להיות מאוד מסוכן לנתקף.

דריסת משתנים וטבלאות וירטואליות

```
// Compiled with -fno-stack-protector

bool authenticate()
{
    char password[20] = { 0 };
    int authenticated = 0;

    std::cout << "Insert secret password: ";
    std::cin >> password;
    if (strncmp(password, "MY_SECRET_PASSWORD", 20) == 0)
    {
        authenticated = 1;
    }

    return authenticated;
}
```

בחלק מהמקרים שבהם ננסה להשמיש חולשת Stack Buffer Overflow, אנו עלולים להתקל בהגנות מסוימות (כמו stack canary) אשר מונעות מאיתנו לדרוס את כתובת החזרה על מחסנית הריצה. אבל, הגנות אלו עדיין אינן פותרות באופן מלא השמשות אפשריות של החולשה. למשל, בדוגמאת הקוד המופיע משמאל ניתן לראות פונקציה בשם authenticate האחראית על בדיקה אם המשתמש מאומת, ולפי זה מאפשרת לו לבצע פעולות שמשתמש רגיל לא יכול. בדוגמא זו, ניתן להשפיע על בקרת הזרימה (flow control) של התוכנית מבלי לדרוס בכלל את כתובת החזרה או כתובת מסגרת המחסנית השמורה. הכנסת הסיסמא 0123456789012345678901234567 8, אשר ארוכה יותר מגודל ה-buffer ששמו password, תגרום לכך שהמשתנה השני המוגדר על המחסנית, authenticated, ידרס ע"י התווים של הסיסמא שהוכנסה ומכיוון שהתווים הטקסטואליים שונים מהתו '\0' הערך הבוליאני שיוחזר מהפונקציה יהיה true.

המקרים הללו של דריסה משתנים אומנם אינה נפוצה, בעיקר בגלל שרוב ה-compiler-ים כיום דואגים לשמור מערכים בסוף רשימת המשתנים כך שלא יהיה ניתן לדרוס בעזרתם משתנים אחרים (ולשם כך הוסף הדגל של fno-stack-protector- בדוגמא), אך הם עדיין קיימים: אם המשתנה השמור על המחסנית הוא struct אשר בו שמור מערך, ה-compiler לא יוכל לבצע את אותה אופטימיזציה מכיוון שסדר משתני המידע ב-struct מובטח שישאר זהה, או, כאשר ישנם 2 מערכים, האחד שנמצא בכתובת גבוהה יותר עלול להדרס ע"י השני מבלי שניתן לזהות זאת בקלות ללא פגיעה משמעותית בביצועים.

בנוסף, עוד דריסה אפשרית אשר עלולה להגרם בעקבות Stack Buffer Overflow ומאפשרת להתחמק מהגנות דומות היא דריסה של פונקציות וירטואליות, שהוא קונספט ייחודי של ++C אשר אינו קיים ב-C. כזכור, כל אובייקט של מחלקה שבה מוגדר פונקציות וירטואליות מכיל, כחלק ממשתני המידע שלו, עוד מצביע נוסף: מצביע לטבלה וירטואלית המכילה כתובות של פונקציות וירטואליות של האובייקט [13]. הטבלה הזו, אומנם, לרוב שמורה במקטע זיכרון מיוחד בשם rodata, אשר מאפשר קריאה בלבד, ולכן לא ניתן לדרוס את הכתובות של הפונקציות. אבל, מה שכן ניתן לעשות עם חולשת Stack Buffer Overflow שבה מוגדר אובייקט על המחסנית, הוא לדרוס את המצביע לטבלה הוירטואלית שלו, ולגרום לכך שיצביע, באופן תיאורטי, לכל מקום שנבחר. בפרט, כנראה שנרצה שהוא יצביע למקום כלשהו בתוך ה-buffer שלנו. במקום זה נוכל לקבוע בקלות את כתובת הקוד שירוץ ברגע שיקראו לפונקציה של אותו אובייקט. מתקפות כאלו ואחרות מוסברות בפירוט במאמר על[14] VTint , שגם מסביר כיצד ניתן להגן מפניהן.

מנגנוני אבטחה

ישנם אינספור מנגנוני אבטחה אשר מגנים מפני שלל בעיות וחולשות נפוצות בתוכנה, חלקם ממומשים בעזרת ה-compiler וחלקם בנויים כחלק ממערכת ההפעלה והמערכת המריצה את התוכנה. בחלק זה אציג מספר שיטות שונות שבעזרתן ניתן למנוע אחד או יותר מאיומי האבטחה שהצגתי למעלה:

1. NX bit – מנגנון ה-NX bit (No-eXecute bit) הוא מנגנון הממומש במעבדים אשר מאפשר למערכת הפעלה לקבוע באילו חלקים בזיכרון יהיה ניתן להריץ קוד. רוב מערכות ההפעלה משתמשות במנגנון זה בשביל לממש את הקונספט של $W \oplus X$ (W xor X), אשר קובע כי כל קטע בזכרון יהיה ניתן או לכתיבה או להרצה, ואף פעם לא יאופשרו שניהם יחדיו. כך, מערכת הפעלה אשר מממשת תמיכה במנגנון תקבע כי מקטע ה-text לא יאפשר כתיבה, ואילו מקטע ה-stack לא יאפשר הרצה [15]. בגרסאות החדשות של מע' ההפעלה Windows, לדוגמא, המימוש הזה נקרא DEP – Data Execution Prevention, וניתן (אך לא מומלץ) לכבות אותו בהגדרות. בכך, ניתן למנוע את השיטה הקלאסית של כתיבת shellcode לתוך buffer והרצה שלו. אך עדיין, הרבה איומי אבטחה אחרים שהוזכרו מעלה מצליחים להתגבר על הגנה זו.
2. ASLR – Address Space Layout Randomization, או בקיצור ASLR, היא שיטה הממומשת במערכת ההפעלה אשר גורמת לכך שבכל ריצה של תוכנית, החלקים השונים בזיכרון ימופו לכתובות אקראיות. כך, הרצה של שיטות מסוג ROP, ובפרט return-into-library, יהיו הרבה יותר קשות לביצוע, שכן התוקף יאלץ לדעת לאיזו כתובת נטענה הספרייה בזיכרון ורק לאחר מכן יוכל לקפוץ לפונקציות הספרייה או ל-gadget-ים הרלוונטיים שמצא.
3. Stack Canary – מנגנון זה, אשר נקרא גם Stack Cookie לפעמים, הוא מנגנון הממומש ע"י ה-compiler ומאפשר הגנה בזמן ריצה על כתובת החזרה השמורה על המחסנית. מימוש המנגנון לרוב מתבצע ע"י כתיבת 4 בתים או יותר של מידע בין כתובת החזרה לשאר משתני הפונקציה, ל-4 הבתים האלה קוראים canary או cookie. המיקום של ה-canary הוא אינו מקרי, בהנחה שהדריסה של המחסנית היא רציפה, בשביל לדרוס את כתובת החזרה, יאלץ התוקף גם לדרוס את ה-canary. כשכל פונקציה מסיימת את ריצתה, היא קודם כל בודקת שה-canary לא נדרסה, ורק לאחר מכן מבצעת את הקפיצה לכתובת החזרה. אם בזמן ריצה זיהינו שה-canary נדרסה, התוכנית תזרוק שגיאה ותפסיק את הריצה.
4. Reversed Stack – מנגנון נוסף, הפחות מוכר מהשאר, הוא מנגנון הקובע כי כיוון המחסנית יהיה הפוך מהמקובל כיום. משמעות הדבר היא שכעת, במקום שה-buffer ידרוס את תוכן המחסנית, הוא ידרוס מידע אקראי המופיע אחריה, ושברוב המקרים לא יפריע לריצת התוכנית. בארכיטקטורת ARM המנגנון הזה מובנה במעבד (למרות שאינו בשימוש נפוץ) ואילו ב-x86 פירסמו מאמרים המציגים כיצד ניתן לבצע זאת עם פגיעה מינימאלית ביעילות התוכנית, לדוגמא, המאמר הבא: [16].

שפת Python

שפת Python היא שפת תכנות עילית (high-level), דינאמית ומפורשת (interpreted). היא שפה יחסית מודרנית הנחשבת לאחת משפות התכנות הקלות והפשוטות ביותר להתחיל לפתח בהן, עבור אנשים שאינם מגיעים מתחום הפיתוח. היא הופצה לראשונה בתחילת שנות ה-90 ע"י חידו ואן רוסום אשר המשיך לעבוד עליה ולפתח אותה עד לאחרונה בשנת 2018 שבה הפסיק לפקח על התפתחותה, כשבדרך הוא נעזר בקהילת המתכנתים הרחבה שהתפתחה לשפה ותרמה רבות לפיתוח שלה. השפה פותחה במטרה לאפשר למפתחים לכתוב קוד ברור, קריא ופשוט תוך זמן קצר ביחס לשאר השפות הקיימות. פשטות השפה, שבה ניתן לכתוב קוד במבנה הדומה מאוד לשפה האנגלית, בשילוב עם אינספור ספריות הקוד-הפתוח הקיימות עבורה ברשת הופך את תהליך הפיתוח לקל ופשוט בהרבה גם עבור פרויקטים גדולים ורחבים. [17]

ל-Python קיים מנהל חבילות בשם Pip המאפשר להוריד ולעדכן בקלות כל חבילה, או מאגר (repository), אשר כוללת פיתוח של אדם או חברה אחרת המתחזקים אותה, ובכך להשתמש באותה חבילה בפיתוח תחת הרישיון שבאמצעותו החבילה מופצת. מרבית החבילות מופצות באמצעות רשיונות עם הגבלות מינימליות, כדוגמאת MIT, BSD ו-GPL [18], ועל כן כל מפתח או חברה יכולים להשתמש בהם בחופשיות ובקלות.

במהלך השנים שוחררו מספר רב של גרסאות ועדכונים עבור השפה, שהעיקריים מביניהם היו בשנת 2000 שבה שוחררה גרסאת Python 2.0 ובשנת 2008 שבה שוחררה גרסאת Python 3.0, שניהם מכילים עדכונים ראשיים (major release) והביאו לשינויים משמעותיים בשפה, הן במימוש והן בפונקציונאליות שלה.

מבנה ותחביר השפה

כאמור, שפת Python תוכננה להיות פשוטה וברורה, ועל כן תחביר השפה כולל הרבה מילות מפתח אשר מובנות לנו כבני אדם גם מבלי להכיר אותן קודם, מילים כמו: is, in, and, or, not, None וכו'... קוד בשפת Python מחולק לבלוקי קוד, אשר ניתן להבחין ביניהם ע"י הזחה (indentation). כאשר שורה מסוימת מוזחת יותר מהשורה הקודמת לה סימן שהתחיל בלוק קוד חדש והוא נגמר כאשר ההזחה חוזרת למצב הקודם. ניתן להתחיל בלוק חדש של קוד רק לאחר פקודת קוד המאפשרת זאת, לדוגמא: if, for, while, def ועוד פקודות דומות אשר משפיעות על בקרת הזרימה או מגדירות משתנה בשפה. כל הפקודות הללו שאחריהן מתחילים בלוק חדש של קוד מסתיימות בנקודותיים ":". אם קוד בלוק מסוים מתחיל לרוץ (בהתאם לזרימת התוכנית), הוא יריץ בו את כל השורות אחת אחרי השניה, אלא אם פקודה מסוימת גרמה לכך שיקרה משהו אחר, כמו לדוגמא התחלה של בלוק חדש או אחת ממילות המפתח break, continue, return ועוד, אשר גורמות לשינוי בבקרת הזרימה של התוכנית.

דינמיות השפה

קוד פייתון רץ ע"י interpreter (מפרש), אשר קורא את קובץ הקוד בזמן ריצה ומריץ אותו פקודה אחר פקודה. המשמעות של כך היא שהקוד בשפה מורץ ע"י interpreter בצורה כמעט זהה לאיך שהוא מופיע בקוד עצמו, ללא תהליך של compilation ואחריו linking כפי שקורה בשפות אחרות כדוגמאת C++. לדבר זה ישנן יתרונות וחסרונות.

החסרון המרכזי לכך הוא היעילות: מכיוון שה-interpreter צריך לפרסר בזמן ריצה את שורות הקוד ולהריץ אותן בעצמו, לוקח יותר זמן להריץ אותן לעומת תוכנית אשר כתובה ב- C++ שבה קובץ ההרצה כבר נמצא בפורמט שהמעבד יודע לזהות ולהריץ. עוד חיסרון הוא שה-interpreter לא יודע מבעוד מועד אילו ספריות הקוד יצטרך ובאילו חלקים, ולכן הוא יאלץ לטעון אותן בזמן ריצה כאשר נעשה בהם שימוש, ובמידה והן לא קיימות הקוד עלול לקרוס לאחר שכבר בוצעו פעולות חלקיות של התוכנה והדבר עלול להשאיר את התוכנה במצב אינו מוגדר, בהנחה שהמפתח לא ציפה שיריצו את התוכנה ללא הספריה הזו.

למרות זאת, ישנם הרבה מאוד יתרונות אחרים אשר גוברים על החסרונות ברוב המקרים והופכים את השפה לפופולרית מאוד, כפי שהיא היום. היתרון המרכזי הוא שהקוד הנכתב יכול לרוץ על כל פלטפורמה קיימת, כל עוד קיים עבורה interpreter. ואכן קיימים מספר רב של סוגים לשלל מערכות הפעלה אשר ממומשים בטכנולוגיות שונות, כמו: CPython, IronPython, Jython ועוד... בנוסף, יתרון נוסף אשר קיים ב-Python בעקבות הדינמיות הגדולה שלה הוא שניתן לייצג בה כל ביטוי שנרצה, ללא הגבלות של גודל (עד כדי הגבלות

חומרתיות), וזאת משום שכל משתנה בשפה מנוהל ע"י מנהל זיכרון פנימי המרחיב את גודל המשתנים בהתאם לשימוש בהם ויכול אפילו לשנות את סוגם תוך כדי ריצה. כתוצאה מכך, האפשרות של דליפות זיכרון קטנה משמעותית משום שאותו מנהל זיכרון יודע לזהות מתי מסתיים שימוש במשתנה כלשהו וה-garbage collector מפנה אותו. יתרון נוסף אשר מספקת הדינמיות הוא האפשרות להגדיר אובייקטים תוך כדי ריצה. למשל, ניתן להגדיר מחלקה שלא מכילה משתנים (attributes), אבל להגדיר עבורה פונקציה אשר תקרא כאשר מבקשים משתנה שלא קיים עבורה, הפונקציה תחזיר ערך כלשהו בהתאם ללוגיקה שלה (לפונקציה זו קוראים getattr).

מחלקות, אובייקטים וירושה

כמו בהרבה שפות אחרות, גם בשפת Python ניתן להגדיר מחלקות אשר עוזרות למפתח לייצג טיפוסים מורכבים המכילים מספר משתנים. משתנה מטיפוס של מחלקה מסוימת נקרא אובייקט של המחלקה, והמשתנים של האובייקט נקראים attributes. בשונה מ- C++, ה-attributes המוגדרים לכל אובייקט של המחלקה אינם חייבים להיות זהים לכולם, ולמעשה ניתן ליצור attributes חדשים גם לאחר יצירת האובייקט ואפילו לעשות זאת מחוץ לבלוק הקוד המגדיר את המחלקה. הדבר כמובן מתאפשר בעקבות הדינמיות של השפה והוא מהווה יתרון וחסרון כאחד: השפה אינה מגבילה אותך בתור מפתח, אבל בעקבות כך ניתן לממש דברים מסוימים בצורה מסובכת ומסורבלת. מעבר ל-attributes, ניתן גם להגדיר פונקציות עבור אובייקטים של המחלקה, כשההבדל ביניהן לפונקציות רגילות הוא לא יותר מכך שפונקציות אלו מצפות לקבל את האובייקט של המחלקה כפרמטר הראשון. וכמובן, גם ב-Python, כמו הרבה שפות אחרות, ישנה תמיכה בירושה ותכנות מונחה-עצמים ולמעשה כל מחלקה, גם אם מוגדר כך באופן מפורש וגם אם לא, יורשת מהמחלקה Object שמייצגת את האובייקט הבסיסי ביותר. כשמחלקה יורשת ממחלקה אחרת, היא למעשה מכילה את כל הפונקציות המוגדרות עבורה, והיא יכולה לדרוס אותן בהתאם. ה-attributes עצמם אינם מועברים אוטומטית, משום שהם נוצרים בזמן ריצה כאשר הפונקציות של מחלקת האב נקראות, אבל גם המחלקה היורשת יכולה לגשת אליהם. בנוסף, Python תומכת בירושה מרובה, כלומר, מחלקה אחת אשר יורשת מכמה מחלקות שונות, למרות שהמאפיין הזה של השפה מאוד שנוי במחלוקת, ולכן פחות משתמשים בו [19].

דוגמאת שימוש

להלן מימוש ב-Python של הקוד שהדגמנו בפרק על C++:

```python
from abc import ABC, abstractmethod

class Animal(ABC):  # Inheritance from ABC - the built-in abstract class
    def __init__(self, name, age):
        self.name = name
        self.age = age
        print(f"My name is {name} and I'm {age} years old.")

    @abstractmethod
    def make_sound(self):
        pass

class Dog(Animal):
    def make_sound(self):
        print(f"{self.name} Barks!")

class Duck(Animal):
    def make_sound(self):
        print(f"{self.name} Quacks!")

def main():
    dog = Dog("Rexi", 5)
    duck = Duck("Nini", 3)
    dog.make_sound()
    duck.make_sound()


if __name__ == '__main__':
    main()
```

בדומה לדוגמא ב- C++, גם כאן מוגדרות 3 מחלקות. מחלקת אב בשם Animal ושתי מחלקות יורשת בשם Dog ו-Duck.

מחלקת ה-Animal מוגדרת כמחלקה אבסרקטית משום שמלבד ה-constructor היא אינה מממשת פונקציות נוספות, אלא רק מגדירה אותן, כדוגמאת הפונקציה האבסטרקטית בשם make_sound.

המחלקות Dog ו-Duck יורשות מ-Animal וכל שהן עושות הוא לדרוס את המימוש של הפונקציה האבסטרקטית make_sound בהתאם לקול של אותה החיה.

ניתן לשים לב שבשונה מ- C++, כאן משתני המידע של המחלקה (name, age) אינם מוגדרים כמשתנים אלה פשוט נוצרים דינמית ב-constructor של Animal, וכך המחלקות היורשות נגשות אליהם.

הפלט של הדוגמא זהה לפלט שמופיע בדוגמא של C++.

איומי אבטחה

ראשית, אומר כי מרבית מאיומי האבטחה הקיימים ב-Python נגרמים עקב שימוש לא נכון בשפה ובספריות הנלוות לה. מפתח אשר מודע לאבטחה יכול לבדוק לגבי כל ספרייה מובנית, ע"י צפייה בתיעוד שלה, האם היא בטוחה לשימוש ומאילו דברים צריך להמנע על מנת לשמור על הקוד מאובטח ובטוח לשימוש.

חולשות ב-Deserialization

כפי שכבר הוזכר, הדינמיות של שפת Python מספקת לה הרבה יתרונות, אחד מהם הוא האפשרות להגדיר וליצור אובייקטים חדשים תוך כדי ריצה. בהרבה תוכנות רשת אשר צריכות להעביר אובייקטים ממחשב אחד למחשב אחר דרך הרשת, נעשה תהליך של serialization, מעין קידוד אשר מאפשר להמיר את נתוני האובייקטים לרצף בתים אשר ניתן לשלוח בהודעת TCP או UDP. כשההודעה מגיעה ליעדה, הוא מבצע בה תהליך הפוך, הנקרא deserialization, שבו הוא ממיר את רצף הבתים בחזרה לאובייקט המקורי. ב-Python, ישנו מנגון serialization מובנה הנקרא pickle. בעזרתו, המחלקה של האובייקט המועבר ברשת לא חייבת להיות מוגדרת בתוכנה הרצה בצד של מקבל ההודעה, ו-pickle ידע לזהות את סוג האובייקט ולהמיר אותו כפי שצריך. על פניו, נראה ש-pickle מאפשר לבצע משימה מסובכת כמו המרה של אובייקט בצורה פשוטה ונוחה, כפי ש-Python מנסה לספק למפתח. אבל, השימוש בו עלול להוות איום אבטחה חמור אם משתמשים בו עם מידע אשר הגיע ממקור אינו אמין. ידוע כבר מזה שנים שהשימוש ב-pickle הוא מסוכן, וקיימים מספר מאמרים אשר מראים כיצד ניתן לנצל זאת להרצת קוד בצורה מרוחקת, לדוגמא [20], [21] ועוד רבים אחרים... בעמוד התיעוד הרשמי של הספרייה ניתן למצוא אזהרה אשר אומרת שאין להשתמש בה עם מידע שהגיע ממקור לא אמין או לא מאומת [22]. בנוסף להרצת קוד, פורסמו מס' מאמרים המציגים כיצד ניתן לנצל את המימוש הלקוי של הספרייה על מנת ליצור מתקפת DOS (Denial Of Service) ע"י שליחת הודעה בפורמט מסוים אשר גורמת לצריכת זיכרון גדולה מאוד ביחס לגודל ההודעה הנשלחת. במאמר [23] לדוגמא, מציגים כיצד ניתן לשלוח הודעה שגודלה 5MB שגורמת לספריית pickle ליצור מספר רב מאוד של מילונים (dict) בעת ביצוע ה-deserialization. כל מילון שנוצר צורך בזיכרון 224 בתים, וע"פ החישוב שבוצע ההודעה כולה גורמת לצריכת זיכרון של כ-1GB בסה"כ. שליחת מספר הודעות מהסוג הזה בקצב גבוהה עלולה בהחלט לפגוע בפעילות תקינה של שרת.

טעינת קובץ זדוני

בשפת Python קיים מנגנון פנימי אשר אחראי על טעינת ספריות חיצוניות בזמן ריצה. כשהקוד מנסה לטעון סיפרייה (package) כלשהיא בעזרת הפקודה import, התוכנית בודקת אם המודול כבר טעון (ע"י בדיקה של sys.modules), ואם לא, היא מריצה סדרה של אובייקטים הנקראים finders ו-loaders אשר מטרתם לחפש את המודול של הסיפרייה כפי שהוגדר להם לעשות [24]. פרוטוקול הטעינה הזה עובד בדומה ל-loader-ים הקיימים במערכות ההפעלה השונות, בדומה לצורה שטוענים DLL ב-Windows או SO (Shared Object) ב-Linux. ההבדל המרכזי והחשוב הוא שברוב המקרים, בשביל להוסיף DLL או SO במיקומים הרלוונטיים שיגרמו לטעינה שלו, צריך משתמש בעל הרשאות גבוהות. לעומת זאת, בהתקנת Python אין שום הגדרת אבטחה המונעת ממשתמש כלשהו במערכת להוסיף או לשנות קבצי Python השייכים למודלים בסיסיים. המשמעות היא שכל משתמש מאומת יכול לשנות את התוכן או להחליף את הקבצים הללו וקוד הטוען אותם יריץ את הקוד הזה בעזרת ההרשאות שלו, דבר היכול להוות חולשת PE (Privilege Escalation). ישנן עוד דרכים שונות לבצע את הדבר הזה, והן ברובן תלויות בקינפוג שגוי של הרשאות בתיקייה שבה מחפשים את המודולים [25].

בנוסף, ב-Python קיים מנהל חבילות מובנה אשר מאפשר להוריד חבילות וסיפריות שלמות, שנוצרו ע"י אנשים בקהילה, דרך הרשת. אומנם כל חבילה מופצת ביחד עם כל הקוד שלה פתוח לרווחה, אך תמיד קיים החשש שאיפשהו בקוד ישנה פירצה או אפילו backdoor אשר עלולים להוות בעיית אבטחה רצינית [26].

חילוץ קבצים זדוניים

בתת-פרק של חולשות ב-Deserialization הצגנו כיצד קבלת מידע אשר עובר deserialization עשויה להוות פירצת אבטחה אם הגיע ממקום לא אמין. באופן דומה, קבלת קבצים דחוסים (שעברו קיבוץ, compression) עשויה בחלק מהמקרים להוות בעיית אבטחה רצינית אם לא מוודאים את מקורם, תקינותם או שימושם כמו שצריך. מעבר לעובדה שישנן פונקציות ספציפיות אשר אינן בטוחות והתיעוד הרשמי של Python מזהיר מהשימוש בהן [27] (ואפילו פירסמו מספר מאמרים על השמשה של החולשות שלהן [28]), ישנן מספר רב של חולשות ופירצות הנמצאו עבור סיפריות הקיבוץ המובנות של Python, והמספר הזה רק הולך וגדל עם הזמן. זה נכון שכמעט תמיד מוציאים תיקונים (או לכל הפחות אזהרות) לגבי החולשות הנמצאות, אך הדבר מרמז לכך שלא בטוח שמצאו עד היום את כל החולשות האפשריות ואבטחתן של אותן ספריות מוטלת בספק. החולשות הבאות [29], [30], [31] הן רק חלק מהחולשות שמצאו לאחרונה בסיפריות הקיבוץ (או איגוד קבצים) עבור קבצים מסוג zip, tar, ועוד...

שימוש בפונקציות מסוכנות

כמו בכל שפה, גם ב-Python קיימות מספר פונקציות מסוכנות אשר שימוש לא זהיר בהן עלול לאפשר לתוקף זדוני לנצל פירצות כמו הדלפת מידע, מניעת שירות (DOS) או הרצת קוד מרוחקת (RCE). הפונקציות הראשונות, ושככל הנראה המסוכנות ביותר, הן exec, eval, os.system, רוב הפונקציות במודול subprocess [32] ואפילו input בגרסאות של Python 2 [33]. ברוב הפונקציות הללו ישנן אזהרות המתריעות מפני הסכנות שהן עלולות להביא, אך ישנן הרבה פונקציות נפוצות אחרות המאפשרות לפגוע בתקינותן של תוכניות כמו לדוגמא שימוש בפונקציה open כאשר נתיב הקובץ מושפע מקלט של המשתמש עלול לגרום לדליפת מידע או אפילו לפריצה של המערכת כולה ללא בדיקות מתאימות.

מנגנוני אבטחה

בשונה מ- C++ ושפות תכנות נמוכות אחרות (low-level languages) הדומות לה, Python היא שפה די מוגנת: עבור החולשה בתת-פרק חולשות ב-Deserialization, קיימת חבילה הנקראית Hickle אשר פותחה על ידי מספר חוקרים מאוניברסיטאות שונות ומטרתה לממש חלופה טובה ובטוחה ל- pickle [34].

באשר לשאר, אומנם הצגנו מספר דרכים אשר עלולות להוות פירצות אבטחה, שחלקן אפילו חמורות, אבל אם נבחן כל אחת מקרוב נשים לב שברוב המקרים מפתח אשר מודע לאבטחה יכול למנוע אותן בקלות מאוד. בחלקן, בדיקה מקיפה של התיעוד מראה לנו ממה צריך להמנע וכיצד להשתמש במנגנוני השפה הרלוונטיים כראוי בצורה מאובטחת. יתר על כן, ברובם המוחלט של האיומים, ווידוא ואימות המידע המתקבל מהמשתמש ימנעו לחלוטין את האפשרות לפירצה כלשהיא. עם זאת, תמיד קיימת האפשרות של פירצת אבטחה הנובעת מטעות לוגית בקוד המפתח (באג), ועל כן כל תוכנית, בכל שפה אשר תהיה, תמיד עלולה להיות חשופה לבעיות אבטחה כאלו ואחרות, ובפרט תוכנית הכתובה ב-Python. ובכל זאת, הדבר הבטוח ביותר שימנע את מרבית הבעיות הוא ווידוא ואימות קלט מהמשתמש, ויש לבצע אותו תמיד!

תקשורת

החשיבות של תקשורת בין מחשבים כבר הוכיחה את עצמה מאוד עם השנים. חיבור של אנשים במקומות שונים בעולם והפצת אינפורמציה בצורה גלובלית במהירות הם רק חלק קטן מהיתרונות הרבים שמעניקה לנו רשת האינטרנט המקשרת בין המחשבים ורכיבים אחרים כיום. אך, יחד עם היתרונות הללו, צצו גם הבעיות הבלתי נמנעות שבאות איתה.

לפני תקופת האינטרנט, בשביל שמישהו יוכל לקבל גישה למידע של אדם אחר הוא היה צריך להשיג נגישות פיזית לרכיב שעליו מאוחסן המידע. כיום, בעולם המודרני, המחשבים של רוב האנשים מחוברים לאינטרנט והחיבור הזה מספק לאדם זדוני נגישות מרוחקת למחשב, מבלי שיצטרך לדעת ולגשת למיקומו הפיזי. כמובן, הדבר הזה אינו מובן מאליו והוא דורש לרוב שימוש בחולשה כלשהי או ניצול אדם, אשר אינו מודע להשלכות האפשריות, בשביל שיספק את הגישה הזאת.

בנוסף, השימוש של מחשבים באינטרנט משמעותו החלפת מידע בין המחשבים. בגלל שהמידע הזה עובר בהרבה תחנות מעבר עד שמגיע ליעדו, שום דבר לא מונע מאחת התחנות הללו לקרוא או לשנות את המידע הזה, ובפרט לא להעביר אותו. על כן, יש צורך במנגנונים מסוימים אשר ימצאו פתרון לבעיות הללו.

אבטחה בתקשורת

תקשורת האינטרנט, כפי שאנו מכירים אותה כיום, מוגדרת ע"י ארגון ה-OSI (International Organization for Standardization), ארגון התקינה העולמי. הסטנדרט שאותו קבע ארגון ה-OSI נקרא "מודל 7 השכבות של OSI" (The 7 Layers of the OSI model), לפיו, תקשורת האינטרנט בין מחשבים מחולקת ל-7 שכבות שכל אחת מהן אחראית על תפקיד אחר. בכל שכבה, מוגדרים מספר פרוטוקולים אשר מגדירים כיצד צריך להבין ולהגיב למידע המתקבל בשכבה הזו. למרות ששכבת הייצוג (Presentation) אחראית על הצפנת התקשורת, הפרוטוקולים בהם נעסוק בחלק זה שייכים לשכבת האפליקציה. יש שיגידו כי מודל ה-TCP/IP מדמה יותר את רשת האינטרנט כיום: במודל זה שכבות השיחה (Session), הייצוג והאפליקצייה שולבו יחדיו לשכבת אפליקצייה יחידה.

רשת האינטרנט מורכבת ממספר רב של רכיבי תקשורת היודעים ומכירים מספר רב של פרוטוקולים שבעזרתם ניתן להעביר הודעות מישות אחת ברשת אל ישות אחרת. כל הודעה הנשלחת מישות מסוימת יכולה לעבור במספר תחנות ביניים בדרכה אל היעד, כאשר כל תחנה כזאת נגישה באופן מלא למידע המועבר בהודעה. על כן, כדי למנוע חשיפת מידע של הודעה העוברת ברשת, ישנם מספר פרוטוקולים שמטרתם להצפין את המידע בצורה בטוחה כך שיהיה חסין מפני שבירה של הצופן. בנוסף, מכיוון שההודעות אינן מועברות ישירות מתחנת המוצא אל תחנת היעד, אלא דרך כמה תחנות ביניים, אף אחד מהצדדים לא יכול לדעת בוודאות ששולח ההודעה הוא באמת המקור שלה, ושלא מדובר בישות אחרת המתחזה אל הישות המוצהרת כשולח ההודעה. על כן, קיימים פרוטוקולים לאימות בין ישויות ברשת כדי שצד אחד יוכל לאמת שמי שעונה להודעות שלו הוא אכן הישות אליה פנה. הפרוטוקולים הללו מבוססים בעיקר על סוגים שונים של הצפנות, שלהן תכונות מיוחדות המבטיחות שלא יהיה ניתן לרמות אותן או לנצל אותם ללא כח חישוב גדול במיוחד (שרבים סבורים שרק לממשלות יש כח כזה [35]).

פרוטוקולי הצפנה ואימות

פרוטוקול Diffie–Hellman

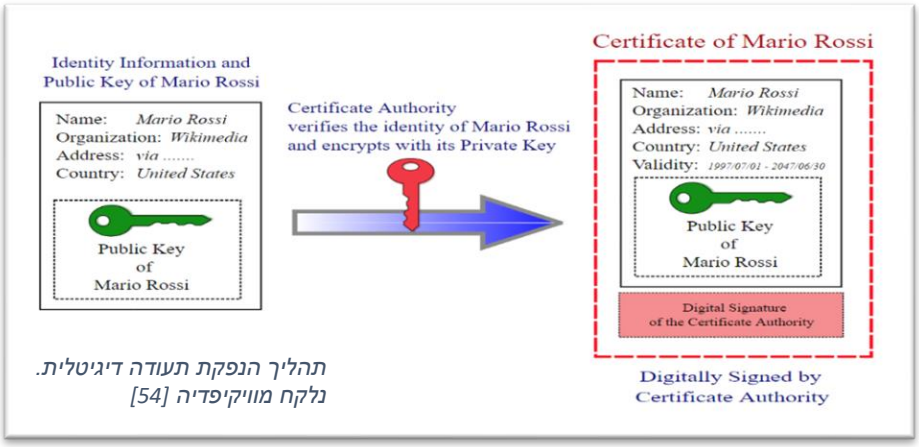
פרוטוקול Diffie-Hellman הוא פרוטוקול, שפותח על ידי ויטפילד דיפי ומרטין הלמן, שמטרתו היא שיתוף מפתח הצפנה (לעיתים קרוי "סוד") בין שני גורמים ללא חשיפת המפתח בזמן שליחתו. בעזרתו, שתי ישויות ברשת יכולות לשתף ביניהם מפתח סודי שבאמצעותו ישתמשו להצפנה סימטרית מבלי שלאף אחד מהצדדים יהיה מידע כלשהו לגבי הצד השני, וכל זה תוך הגנה מפני תוקף זדוני אשר מאזין על התקשורת ביניהם לצורך חשיפת המידע הנשלח בין השניים. יש לשים לב כי שיטה זו אינה מספקת הגנה מפני מתקפות מסוג MiTM (Man in The Middle – האיש שבאמצע) שבהן התוקף מתחזה לכל אחד מהנמענים וממסר ביניהם את המידע הנשלח, תוך כדי החלפת פרטים שמטרתם להצפין את התקשורת, כמו הפרוטוקול המדובר. הפרוטוקול מבוסס על עיקרון מתמטי הנקרא "בעיית הלוגריתם הבדיד" שעיקרה הוא הקושי בגילוי המעריך של חזקה בהינתן הבסיס ותוצאת החישוב שלה תחת המספרים השלמים. בכך הפרוטוקול מממש פונקציה חד-כיוונית שהיא המרכיב העיקרי וההכרחי של כל הצפנה אסימטרית. [36]

הצפנת RSA

עוד סוג של הצפנה אסימטרית היא הצפנת RSA, הקרויה על שם הממציאים שלה: Rivest, Shamir, Adleman. הצפנה זו מאפשרת לאחד מהצדדים להצפין מידע שברצונו לשלוח לצד השני, כך שרק האיש אליו מיועדת ההודעה יוכל לפענח אותה. התהליך הזה ממומש ע"י יצירת זוג מפתחות, מפתח פרטי ומפתח ציבורי, בצד המעוניין לקבל את המידע המוצפן. ברוב המקרים, הצד המקבל יהיה שרת המעוניין לקבל מידע חסוי מלקוח כלשהו, מבלי ששאר הישויות ברשת יוכלו לגלות את המידע. לשם כך, השרת ייצור את זוג המפתחות, וישלח ללקוח את המפתח הציבורי, מפתח גלוי הנשלח לכולם, שבעזרתו ניתן להצפין מידע. הלקוח יקבל את המפתח הציבורי, יצפין בעזרתו את המידע הרלוונט וישלח זאת לשרת. בקבלת ההודעה, השרת ישתמש במפתח הפרטי שלו, אשר נשמר עליו מקומית ולא נשלח לאף אחד, ויפענח את ההודעה המוצפנת שהצפין הלקוח בעזרת המפתח הציבורי. בצורה זו, הלוקח הצליח להעביר מידע לשרת בצורה מאובטחת מבלי שיסכמו מראש על מפתח משותף או כל מידע אחר. גם כאן, מתקפות מסוג MiTM יכולות להכריע את ההצפנה הזו אם התוקף יודע לזהות את הפרוטוקול ולהחליף את המפתח הציבורי הנשלח ללקוח למפתח ציבורי שהוא יצר בעצמו, ובקבלת המידע לפענח ולהצפין אותו בחזרה עם המפתח הציבורי של השרת. בדומה לפרוטוקול Diffie-Hellman, גם הצפנת RSA מבוססת על עקרון מתמטי המסתמך על כך שלא ניתן לפרק מספר מספיק גדול לגורמים הראשוניים שלו בזמן סביר, ובכך מממש הפרוטוקול פונקציה חד-כיוונית. [37]

תעודה וחתימה דיגיטלית

תעודה דיגיטלית (Digital Certificate) היא הדרך שבה ישויות ברשת מאמתות את המוען של הודעה מסוימת. תעודה דיגיטלית מכילה פרטים של גוף או ארגון ובנוסף את המפתח הציבורי המשוייך לאותו גוף או ארגון. לדוגמא, כאשר אדם גולש באינטרנט לאתר כלשהו, אותו אתר שולח לו את התעודה הדיגיטלית שלו והלקוח מאמת, בעזרת נתונים השמורים אצלו ע"י הדפדפן, שאכן השרת שענה לו הוא השרת שאליו ניסה לפנות ושאכן המפתח הציבורי שאיתו השרת רוצה להשתמש לצורך ההצפנה הוא אותו מפתח המצויין בתעודה. בצורה זו, תוקף זדוני לא יוכל לרמות לקוחות של שרת מסוים להשתמש במפתח שהתוקף יצר משום שאותו מפתח ציבורי לא יתאים לזה המשוייך לתעודה הדיגיטלית של השרת. בכך, תעודה דיגיטלית מספקת אימות של הישות שאיתה אנו מתקשרים. ישנם גופים וארגונים הנקראים רשויות אמון (Certificate Authorities) שתפקידם להנפיק תעודות דיגיטליות אלו ולאמת שגוף שאליו משוייכת התעודה הוא גוף לגיטימי שאינו ינצל את התעודה הדיגיטלית שלו למטרות ניבזיות, ובמידה וכן יוכלו לקשר את הגוף הזה לאנשים הרלוונטיים האחראיים עליו. אותם גופים אשר מנפיקים תעודות דיגיטליות דואגים לחתום עליהם באמצעות המפתח הפרטי שלהם, וכך, כל לקוח יכול להשתמש במפתח הציבורי בשביל לאמת את החתימה הדיגיטלית ולוודא שהתעודה הדיגיטלית מהימנה. לצורך תהליך החתימה והאימות, נדרשת הצפנה אסימטרית הכוללת מפתח פרטי ומפתח ציבורי, כדוגמאת RSA, המבטיחה קיום של פונקציית מלכוד חד-כיוונית כדי שרק רשויות האמון יוכלו להנפיק תעודות דיגיטליות שחתומות על ידן.



תהליך הנפקת תעודה דיגיטלית.
נלקח מוויקיפדיה [54]

בנוסף לאימות של מוען ההודעה, ניתן להשתמש בחתימה דיגיטלית בשביל לאמת את התוכן של ההודעה, ובכך להבטיח כי אף גורם חיצוני לא שינה אותה בדרכה אל הנמען. הדבר נעשה לרוב ע"י המרת המידע לקוד קצר, באמצעות פונקציית גיבוב (Hash function), המתאר את המידע וחתימה על אותו קוד בעזרת מפתח פרטי. כך, הנמען יכול להפעיל את אותה פונקציית גיבוב על התוכן ולאמת אותו בעזרת המפתח הציבורי והחתימה.

פרוטוקולי SSL ו-TLS

פרוטוקול SSL (Secure Sockets Layer) הוא פרוטוקול המאפשר ליצור חיבור מוצפן מאומת ובטוח בין שרתים ולקוחות ברשת האינטרנט. הוא משתמש במגוון רחב של פרוטוקולי הצפנה ואימות, כמו Diffie-Hellman, RSA, AES ועוד רבים אחרים בשביל להבטיח את כל תכונות האבטחה שמטרתו לספק. עם השנים, יצאו מספר גרסאות של הפרוטוקול וביניהן SSL 1.0, SSL 2.0, SSL 3.0 ואחריהן פותח פרוטוקול אחר שבה להחליפו הנקרא פרוטוקול TLS (Transport Layer Security), וממנו יצאו הגרסאות הבאות: TLS 1.0, TLS 1.1, TLS 1.2 ו-TLS 1.3, שהיא הגרסא העדכנית ביותר לפרוטוקול נכון לזמן כתיבת הפרק. בכל גרסא של כל אחד מן הפרוטוקולים, שופרו ועודכנו הפרוטוקולים בשביל למנוע חולשות ובעיות שנמצאו בהם. בין היתר, ככל שהתקדם הזמן, מחשבים נהפכו להרבה יותר יעילים בביצוע פעולות חישוביות ועל כן נדרשה התאמה של אלגוריתמים מסוימים כמו הצפנות סימטריות ופונקציות גיבוב (Hash). לדוגמא, בעבר השתמשו בהצפנה סימטרית מסוג DES שהיום ידועה מאוד כלא בטוחה. אותה הצפנה שודרגה לאחר מכן ל-3DES שגם היא כבר אינה בטוחה כיום והוחלפה ע"י AES שבה משתמשים כיום. פרוטוקול SSL עדיין תומך בשימוש ב-DES בגרסאות 2.0 ו-3.0 והדבר נחשב לבעיית אבטחה. בנוסף, גם גרסאות SSL 2.0, SSL 3.0, TLS 1.0, TLS 1.1, TLS 1.2 עדיין תומכות ב-3DES וגם זה עלול להוות בעיית אבטחה. בגרסת TLS 1.3 השימוש ב-3DES כבר לא נתמך ומשתמשים רק בהצפנות מסוג AES הנחשבות, לפחות היום, כבטוחות ומאובטחות [38]. בנוסף, גם פונקציות הגיבוב (Hash) יתפתחו עם הגרסאות ופונקציות כמו MD5 או SHA (SHA1, SHA128, SHA224) כבר לא נחשבות לבטוחות ועל כן הופסקה התמיכה בהן בגרסת TLS 1.3 [39].

כפי שנאמר, פרוטוקול TLS מספק חיבור תקשורת אמין מוצפן ובטוח. הוא עושה זאת ע"י אימות זהות השרת בעזרת התעודה הדיגיטלית שלו, החלפת מפתחות בטוחה ע"י פרוטוקולים מאובטחים כמו DiffieHellman או RSA, ושימוש במפתחות אלו להצפנת התקשורת השוטפת בין השרת ללקוח, בעזרת הצפנות מאובטחות כדוגמאת AES וסוגיה. הצורה שבה TLS יוצר חיבור כזה נקראית TLS Handshake. כשמתחיל חיבור בין לקוח לשרת, לדוגמא חיבור TCP, הדבר הראשון שמתבצע לאחר יצירת החיבור הוא תהליך ה-Handshake של TLS. בתהליך הזה קורים מספר דברים הנחוצים ליצירת חיבור TLS [40]:

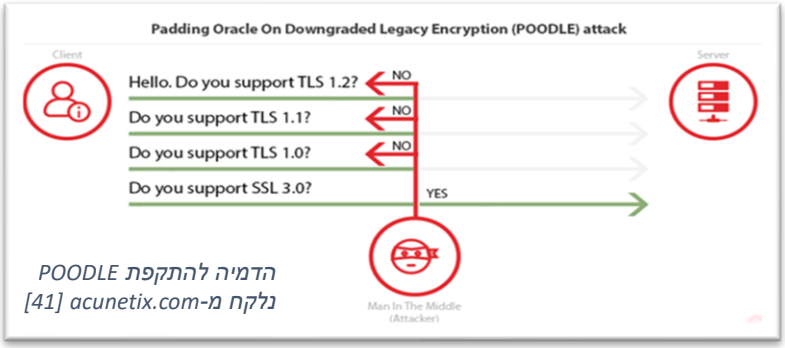
1. הלקוח והשרת מחליטים באיזו גרסא של TLS הם יעבדו, שכן שני הצדדים צריכים לתמוך באותה גרסא כדי להבין כיצד לפרש את ולהגיב אל המידע המתקבל. אם אחד הצדדים אינו תומך ב-TLS בחלק מהמכרים הם יכולים להתפשר ולהשתמש ב-SSL, אך הדבר עלול להוות בעיית אבטחה.
2. הלקוח והשרת מחליטים באיזו שיטות הצפנה, פרוטוקולי החלפת מפתחות ובאילו פונקציות גיבוב הם ישתמשו. כמו כאן, שני הצדדים צריכים לתמוך באותם פרוטוקולים ושיטות כדי שיוכלו לתקשר אחד עם השני.
3. השרת מציג את התעודה הדיגיטלית שלו, והלקוח מאמת אותה מול גורם שלישי אמין.
4. הלקוח יוצר מחרוזת אקראית הנקראית Premaster secret, מצפין אותה בעזרת המפתח הציבורי של השרת ושולח אותה אליו.
5. השרת מקבל את אותה מחרוזת מוצפנת ומפענח אותה.
6. הלקוח והשרת יוצרים מפתח הצפנה משותף בעזרת מחרוזת ה-Premaster secret שיש ברשותם, וכעת הם יכולים להשתמש בפרוטוקולי הצפנה סימטריים לצורך התקשורת הרצופה.

בעזרת שלבים אלו, נוצר חיבור TLS מאובטח ומוצפן בין לקוח ושרת, מבלי ששיתפו אחד את השני במידע כלשהו לפני כן ומבלי שסיכמו מראש על מידע משותף שישתמשו בו.

מתקפות שונות על פרוטוקולי תקשורת

מתקפת POODLE

מתקפת POODLE, או Padding Oracle On Downgraded Legacy Encryption, היא מתקפה מוכרת על במנגנון ה-Handshake של TLS. המתקפה מנצלת חולשה בפרוטוקול שבו תוקף יכול לבצע מתקפת MiTM ובה הוא משפיע על תהליך ה-Handshake בכך שהוא גורם ללקוח להשתמש בפרוטוקול אבטחה ישן, SSL 3.0. מכיוון שהשלב שבו השרת והלקוח מחליטים על גרסאת הפרוטוקול שבה ישתמשו מתרחש לפני אימות זהותו של השרת, אז תוקף יכול להתחזות אל השרת בשלב הזה. לאחר שהתוקף מתחזה לשרת ואומר שהוא אינו תומך בגרסאות שאינן מתקדמות יותר מ-SSL 3.0, הלקוח מתפשר ומשתמש בגרסא זו. [41]

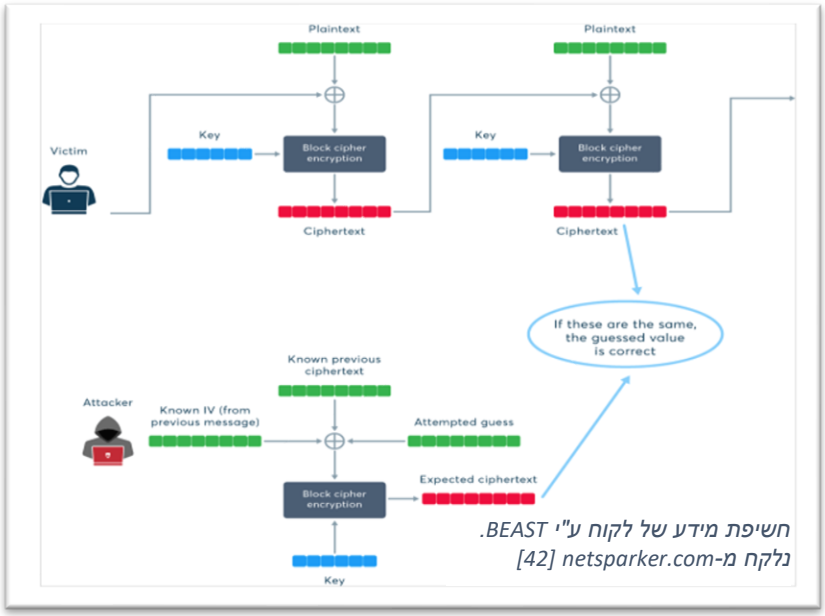


הדמיה להתקפת POODLE
נלקח מ-acunetix.com [41]

לאחר מכן, אם גם השרת תומך בגרסא זו, אז הם יתחילו לנהל את התקשורת שלהם בעזרתה והתוקף יוכל לנצל עוד חולשה במנגנון ההצפנה הסימטרית של הגרסא הזו בשביל לפענח חלק מהמידע הנשלח מהלקוח לשרת. המידע הזה עלול להכיל מידע רגיש ולהוות בעיית אבטחה לשרת, ללקוח או שניהם ביחד.

מתקפת BEAST

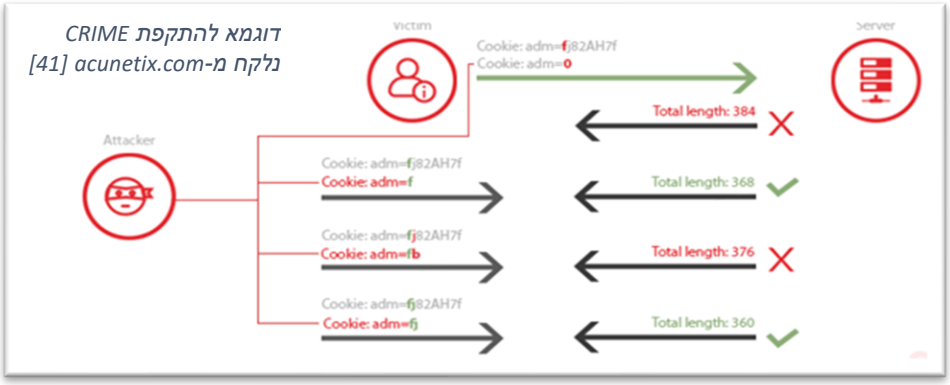
מתקפת BEAST, או Browser Exploit Against SSL and TLS, גם היא מתקפה המנצלת חולשה בגרסאות ישנות של TLS ו-SSL, ספציפית בגרסאות TLS 1.0 ו-SSL 3.0. המתקפה הזו מנצלת, גם היא, חולשה במנגנון ההצפנה הסימטרי: אם תוקף הצליח להשיג שליטה בדפדפן של הלקוח, הוא יכול להזריק הודעות בתקשורת וכך לגלות משתנים ששימשו את הלקוח בהצפנת המידע. בעזרת ניחושים מושכלים והזרקת הודעות, התוקף יכול לנחש את ה-IV (Initialization Vector) ולאחר מכן לבצע הסנפה פאסיבית על התקשורת בשביל לנחש את המידע העובר בין הצדדים. עם זאת, המתקפה הזו דורשת מספר רב של נסיונות ניחוש בשביל להצליח לגלות מידע חיוני וגם דורשת שליטה של התוקף בדפדפן מבעוד מועד, ועל כן אינה נחשבת למתקפה פרקטית. [42]



חשיפת מידע של לקוח ע"י BEAST.
נלקח מ-netsparker.com [42]

מתקפת CRIME

מתקפת CRIME, או Compression Ratio Info-Leak Made Easy, היא מתקפה על מנגנון הדחיסה של פרוטוקולי SSL ו-TLS המאפשרת לתוקף לחשוף מידע הנשלח מהלקוח לשרת למרות שהוא מוצפן. במתקפה זו, תוקף מנצל את הצורה שבה עובדים רוב אלגוריתמי הדחיסה בכך שהוא מזריק הודעות עם מידע מסוים שהוא מצפה שיופיעה בהודעה שהלקוח שולח לשרת. אם אכן המידע הזה הופיע, אז מכיוון שהוא חוזר על עצמו, גם בהודעה של הלקוח וגם בהודעה שהוזרקה ע"י התוקף, אז גודל ההודעה הדחוסה יהיה קטן יותר משום שמידע שחוזר על עצמו ניתן לדחיסה בקלות. המידע הזה כמובן יהיה מוצפן, אך בכל זאת הגודל הכולל שיתפוס יהיה קטן יותר. כך, התוקף יכול לבצע מספר ניחושים בשביל לזהות איזה מידע מופיע בהודעה המקורית ולגלות לאט לאט יותר מידע ממנה, בהתאם לתגובה שמוחזרת מהשרת. המידע הזה יכול להיות מידע רגיש של פרטי התחברות או קוקיות (cookie) השומרת מידע על הלקוח המאפשר הזדהות מול השרת. גם כאן, התוקף יצטרך לשלוט בצורה כזאת או אחרת על הלקוח דרך הדפדפן או צורה אחרת כלשהי, כדי שיוכל להזריק הודעות בשמו. [41]



מתקפת Heartbleed

מתקפת Heartbleed, בשונה משאר המתקפות שתוארו לעיל, היא מתקפה המכוונת כלפי שרתים יותר מאשר לקוחות. המתקפה מנצלת חולשה בספרייה המוכרת OpenSSL, הממשת את הפרוטוקולים של SSL ו-TLS, בשביל לגנוב מידע רגיש השמור בשרת. אותו מידע עלול להיות סיסמאות, מסמכים, פרטי חשבונות, מפתחות הצפנה, עוגיות (Internet Cookies) ועוד...

ניצול החולשה הוא פשוט למדי. בפרוטוקול TLS מוגדר מנגנון המאפשר ללקוח לבדוק האם השרת עדיין קיים וזמין לתקשורת מולו, בדומה למנגנון ה-KeepAlive בפרוטוקול TCP. הדרך שבה הלקוח עושה זאת היא ע"י שליחה בקשה מסוג hearbeat עם מחרוזת כלשהי והגודל שלה, לדוגמא "Hello" ו-5. על השרת לטפל בבקשה זו ולהשיב בחזרה הודעת תגובה מתאימה עם אותה מחרוזת, בדוגמא "Hello". החולשה הייתה במימוש של ספריית OpenSSL, שבה לא נבדק כי הגודל שהתקבל זהה לגודל המחרוזת. לכן, תוקף יכל לשלוח את אותה בקשה עם גודל גדול כרצונו, והשרת היה שולח מידע לפי הגודל שביקש. הדבר מוביל לכך שכל המידע ששמור בזיכרון של השרת לאחר אותה מחרוזת, ישלח בחזרה לתוקף לפי גודל המידע שביקש. [43]

המחשה של מתקפת Heartbleed. נלקח מ-acunetix.com [41]

מספר דברים הפכו את המתקפה הזו להרבה יותר מסוכנת משאר המתקפות שתוארו. אחד הדברים הללו הוא העובדה שהשמשת החולשה לא דורשת גישה כלשהי ללקוח או לשרת. התוקף מתחבר בצורה לגיטימית לשרת ומשתמש בשירות שהוא מספק, ואז שולח לו את אותה בקשה זדונית שגורמת לזליגת מידע. כל זאת מבלי שהלקוח או השרת יוכלו לזהות שהפרטים שלהם נגנבו. דבר נוסף שהפך את המתקפה הזו למסוכנת היא ההיקף שלה: ספריית OpenSSL שבמימוש שלה קיימת החולשה היא בין הסיפריות הכי נפוצות למימוש הפרוטוקול של TLS ומעריכים כי בעת פרסום החולשה מעל ל-300 אלף שרתים ציבוריים היו חשופים לחולשה [44]. בנוסף, העובדה כי החולשה הייתה קיימת לאורך תקופה של כשנתיים הגדילה עוד יותר את ההיקף של המתקפה, שכן הרבה מהשרתים הספיקו לעדכן את המוצרים שלהם לגרסא פגיעה של ספריית OpenSSL במשך התקופה הזו, עד היום קיימים שרתים המריצים גרסא פגיעה כלשהי של הספריה.

הגנות והקלות מפני מתקפות ברשת

כפי שראינו, קיימות מספר רב של מתקפות אפשריות בתווך רשת האינטרנט, הן ללקוחות והן לשרתים. על כן, חשוב מאוד להיות מודעים לסוגים שונים של מתקפות אלו ולוודא שעושים הכל כדי למנוע מתקפות דומות בעתיד. במרבית המתקפות, הדרך למנוע אותן היא פשוטה ביותר: לוודא שמעדכנים לעיתים תכופות את כל הגורמים האחראיים על התקשורת ברשת, בין אם מדובר בספריית אבטחה, כדוגמאת OpenSSL, או מימוש לשרת פרוקסי, כדוגמאת nginx, עבור שרתים ובין אם מדובר באנטי-וירוס או דפדפן עבור לקוחות. חולשות חדשות צצות באופן יומיומי, ועל כן עדכון תדיר של התוכנות הללו יוודא שרמת החשיפה תשאר מינימלית.

בנוסף, מעבר לעדכוני גרסאות הכוללים תיקוני אבטחה, חשוב לא פחות לוודא כי אתם לא תומכים בגרסאות ישנות של פרוטוקולים כדוגמאת TLS ו-SSL. שכן בגרסאות ישנות אלו ישנן מספר חולשות שהן חלק מהסיבה שהוציאו עבורם עדכוני גרסא, ותמיכה בהם משמעותה להיות חשוף לאותן חולשות.

מעבר לכך, מצופה מחברות וארגונים ממשלתיים, שמחזיקים במידע רגיש של לקוחות או רוצים לוודא שאף מידע חשוב לא יעבור ליידיים לא נכונות, להשתמש בשירותים ומוצרים של חברות אשר מנטרות ובודקות את המידע שעובר ברשת שלהן וכך יגדלו הסיכויים לזהות, למצוא ולחסום מתקפות כאלו ואחרות. זאת משום שלעולם לא ניתן לדעת אם הארגון שלך חשוף למתקפה או חולשה, ועל כן חשוב תמיד להיות מודע לאבטחה כיוון ששום דבר לא יכול להבטיח הגנה אבסולוטית מפני מתקפות ברשת.

אבטחת מערכת

בפרקים הקודמים דיברנו על בעיות אבטחה הקשורות למרכיבים ספציפיים במערכת שלנו, שבהם עלולים לנצל חולשות ובעיות אבטחה בשביל להשיג שליטה על המערכת או להזליג מידע רגיש ממנה. בפרק זה נעסוק בבעיות אבטחה אשר מתמקדות במימוש של המערכת עצמה, ולא בתהליכים מסויימים הרצים עליה.

ברוב המקרים, מערכות הפעלה הנפוצות שבהם נשתמש לא יהיו חשופות לחולשות ובעיות אבטחה שיאפשרו לתוקף זדוני להשתלט על המערכת או לחשוף מידע, ובמידה וכן ככל הנראה שישחררו עדכון שבו הבעיות הללו יטופלו. ובכל זאת, ישנן כמה סוגים של מתקפות אשר מנצלות את העובדה כי השרת שלנו צריך לספק שירות כלשהו ללקוחות, ועל כן עליו להיות בקשר איתם ולתמוך בפרוטוקולים מסוימים כדי שיוכלו לתקשר איתו. מכיוון שהפרוטוקולים מתוקננים ע"י תקן, מערכת ההפעלה מחוייבת לפעול על פיו על מנת לתמוך בפרוטוקול, ולכן היא גם חשופה לבעיות המובנות בו. את חלק מהבעיות הללו נציג בהמשך.

מתקפות MiTM

מתקפת MiTM (Man in The Middle) היא מתקפה שבה אדם זדוני משיג יכולת לשלוט בניתוב התעבורה הרשתית של מחשב כלשהו, לשנות אותה או לקרוא (להסניף) את התקשורת הזו באופן פאסיבי. מתקפות מהסוג הזה הן כמעט בלתי נמנעות משום שבכל תקשורת רשתית העוברת באינטרנט ההודעות שלנו מועברות דרך מספר רכיבים בדרך, וכל אחד מהם למעשה יכול לגשת למידע הנשלח בהודעה. לכן, ברוב המקרים נרצה להצפין את ההודעות שלנו במידה והמידע הנשלח יכול להחשב כמידע רגיש.

כדאי לנסות ולהתגונן מהמתקפות הללו כמה שיותר, שכן הן עלולות להשפיע על זמן התגובה שלנו בתקשורת (אם מדובר בהתקפה אקטיבית) ובנוסף שילוב שלהן עם חולשות אחרות עלול להוות בעיית אבטחה משמעותית ביותר.

מתקפת ARP Poisoning

מתקפת ARP Poisoning היא מתקפה אשר מכוונת כנגד מנגנון הנקרא ARP, או Address Resolution Protocol. מנגנון ARP הוא למעשה פרוטוקול תקשורת המגדיר כיצד יבוצע התרגום מכתובת אינטרנט (IP) בשכבת הרשת לכתובת פיזית (MAC) בשכבה הקו [45]. הפרוטוקול מגדיר מספר סוגים של הודעות שמחשב יכול לשלוח ברשת המקומית שלו (LAN) בשביל לתשאל מחשבים על הכתובות הפיזיות שלהן. מבין ההודעות הללו, ישנן שתי הודעות עיקריות המאפשרות לבצע את המתקפה: [46]

1. הודעת ARP Request – מחשב מתשאל את הרשת המקומית מה הכתובת הפיזית של המחשב בעל כתובת אינטרנט נתונה.
2. הודעת ARP Reply – מחשב בעל כתובת האינטרנט שאותה תשאלו משיב ומדווח על הכתובת הפיזית שלו.

בנוסף לפרוטוקול, ברוב מערכות ההפעלה קיים זיכרון מטמון (Cache) אשר שומר טבלה קצרה הממפה בין כתובות אינטרנט לכתובות פיזיות שהמחשב ניגש אליהן לאחרונה או לעיתים תכופות. הזיכרון הזה חוסך הרבה תעבורת רשת מיותרת.

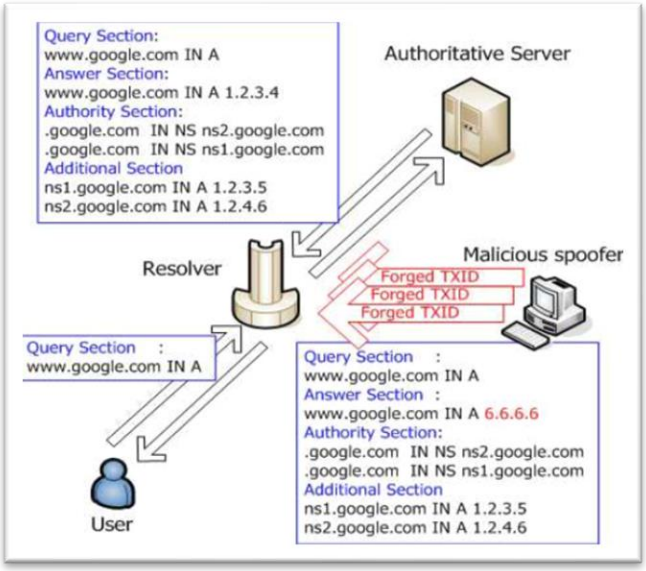
במתקפה ARP Poisoning, מטרתו של תוקף זדוני היא לגרום למחשב כלשהו לחשוב שהכתובת הפיזית של מחשב אחר שאליו הוא מנסה לגשת היא הכתובת הפיזית של אותו תוקף. כך, מחשב התוקף יקבל את ההודעות המיועדות אל מחשב אחר ובכך יוכל לקרוא אותן, לשנות אותן ולהחליט לאן ישלחו הלאה (אם בכלל). ניתן לעשות זאת בכמה דרכים, אך הדרך הנפוצה היא לשלוח באותן תדיר מאוד הודעות ARP Reply המדווחות כי הכתובת הפיזית עבור כתובת האינטרנט של המחשב, שאליו מיועדת ההודעה, היא הכתובת הפיזית של מחשב התוקף. הרעיון מאחורי השיטה הזו הוא שברגע שהמחשב של הקורבן (אותו מחשב שעליו תתבצע ההתקפה) ישלח הודעת ARP Request עבור הנמען, הוא ישר יקבל את אחת מהודעות ה-ARP Reply של התוקף מבלי שהמחשב האמיתי יספיק להגיב לבקשה. ברגע שזה קורה, הכתובת הפיזית של מחשב התוקף נרשמת בזיכרון המטמון של מחשב הקורבן והחל מרגע זה כל תעבורה שתשלח לכתובת האינטרנט הזו תגיע לתוקף. [46]

ברוב המקרים, תוקף לא יהיה נגיש בקלות לרשת המקומית ויוכל לבצע שליחה של הודעות מסוג ARP בשכבת הקו. אבל, ע"פ המאמר [47], הציגו מספר שיטות שבהן מתקפה כזאת על רשת אלחוטית של ארגון עלולה להוות בעיית אבטחה רצינית גם על הרשת הקווית שלה וגם על רכיבים אחרים ברשת האלחוטית.

מתקפת DNS Poisoning

בדומה למתקפת ARP Poisoning, מתקפת DNS Poisoning היא מתקפה אשר מטרתה להשפיע ולשנות את המידע השמור במנגנון ה-DNS. DNS, או Domain Naming System, הוא מנגנון שמטרתו לתרגם בין כתובת טקסטואלית של שרת (Domain) ובין כתובת האינטרנט שלו (IP). בדומה ל-ARP, גם עבור ה-DNS מוגדר פרוטוקול אשר מגדיר כיצד ישויות באינטרנט יתשאלו ישויות אחרות, לרוב שרתי DNS יעודיים, לגבי כתובות האינטרנט של שרתים מסוימים, ועוד פעולות שמגדיר הפרוטוקול. כמו כן, גם ל-DNS יש זיכרון מטמון (Cache) שמטרתו לשמור את הכתובות האחרונות שתורגמו על מנת לחסוך בתעבורת רשת ולאפשר ביצועי רשת מהירים יותר, וגם אותו ניתן לנצל לרעה ו"להרעיל" אותו עם כתובת זדונית כך שהקורבן ישלח את ההודעות לתוקף במקום למחשב אליו ניסה לפנות.

פרוטוקול ה-DNS הוגדר לראשונה אי שם בשנות ה-80, כשהמודעות לאבטחה ברשת אינה הייתה גבוהה כפי שהיא היום. על כן, הפרוטוקול אינו פותח עם מנגנוני אבטחה מיוחדים במיוחד: בשביל לרמות מחשב, ששלח בקשה לתרגום כתובת DNS, ולגרום לו לחשוב שהשרת הייעודי ענה לו, כל שנדרש לעשות הוא להתאים את כתובות ה-IP בהודעה, להתאים את סוג ההודעה לסוג המתאים ולנחש מספר באורך 16 סיביות (ששמו TXID) המוגרל בשליחת הבקשה. על פניו נראה כי ההגרלה של מספר כזה מונעת ברובה את האפשרות לזייף תגובה מתאימה שתגיע לפני תגובת השרת, אך במאמר [48] מתוארת שיטה ובה מסבירים כיצד ניתן לבצע זאת ומראים כי בעזרת העקרון של "פרדוקס יום ההולדת", בשביל לנחש את המספר שגודל $N = 16$ סיביות, ידרשו סדר גודל של $O\left(2^{\frac{N}{2}}\right) = O(2^8)$ נסיונות בשביל להצליח לזייף הודעה מתאימה עם כל הפרמטרים הרלוונטיים שתגרום להרעלת הזיכרון של ה-DNS. המספר הזה אינו גדול בכלל ביחס לכמות המידע הנשלח בימינו ברשת האינטרנט ומתקפה מהסוג הזה בהחלט יכולה להוות איום אבטחה לשרת ולמערכת שרצה עליו.



הדגמה של מתקפת DNS Poisoning.
נלקח מהמאמר [48]

במאמר גם מציגים כיצד ניתן לשפר את החולשה הזו ולהגדיל משמעותית את הסיכויים לבצע אותה בכך שכופים על השרת לשלוח בקשות DNS, ובכך מקטינים את הזמן שלוקח לביצוע ההתקפה, מגדילים את רוחב הפס כדי לשלוח כמה ניסיונות של זיוף הודעה עבור בקשה אחת, ועוד...

מתקפות מניעת שירות

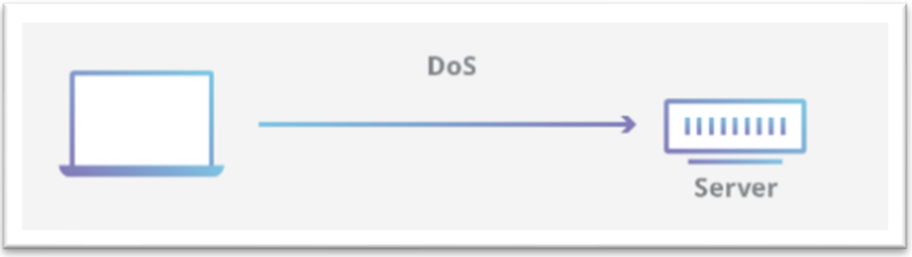
מתקפת DoS

מתקפת DoS, או Denial of Service (בתרגום חופשי: מתקפת מניעת שירות), היא מתקפה שמטרתה למנוע מישות רשתית, בדרך כלל שרת ברשת, להעניק שירות לישויות אחרות, כדוגמאת לקוחות הפונים לשרת בשביל השירות שאותו מספק. לכל שרת ברשת ישנה כמות מוגבלת של משאבים, אם מדובר ברוחב הפס שלו, המגדיר את כמות המידע שהוא יכול לשלוח ולקבל בזמן נתון, או במשאבי החומרה שלו המגדירים כמה פעולות חישוב הוא יכול לבצע או כמה נתונים הוא יכול לשמור בזיכרון שלו. ברגע שהשימוש באחד או יותר מהמשאבים הנ"ל חורג מהכמות שאותה השרת יכול לספק, המשאב הזה יכול להוות צוואר בקבוק גדול לכל הפעולות במערכת:

1. אם כמות המידע הנשלח ומתקבל חורגת מרוחב הפס, אז בקשות יתחילו "להזרק" ולא לקבל מענה.
2. אם כמות הפעולות לחישוב לא יתבצעו מספיק מהר, הפעולות ימשיכו להצטבר והזמן לביצוע כל פעולה יגדל לאט לאט עד שחלקן לא יתבצעו כלל או שכל המערכת תתחיל להתקע כי תהליכים ממתינים לפעולות שלהם יותר מידי זמן.
3. אם כמות הנתונים הנשמרים על כמות הזכרון (הנדיף) תגדל משמעותית מגודלו של הזיכרון אז מידע יכתב לזכרון לטווח ארוך (לרוב בכונני SSD, לפחות בימינו) לעיתים תכופות והדבר יעכב את ביצועי המערכת משמעותית.

כל המקרים הללו, ועוד הרבה אחרים, יגרמו לכך שהשרת שעליו רצים התהליכים ושאליו פונים הלקוחות יתפקד בצורה פחות טובה ומהירה ובאופן כללי העבודה מולו תהפך למעיקה או לעיתים בלתי אפשרית. כאשר מדובר בשרת של חברה או ארגון, מתקפה מהסוג הזה יכולה לפגוע במוניטין ובעיקר לפגוע כלכלית בצורה מאוד קשה. במאמר [49] מסופר על כמה מקרים מפורסמים של מתקפות מהסוג דומה ונאמר כי שעה, שבה לא ניתן שירות ע"י שרתים של חברה מסוימת (downtime), יכולה לעלות בין 300,000$ ל-1,000,000$, מה שמדגיש עוד יותר את החשיבות של מניעת מתקפות כאלו ומתקפות דומות.

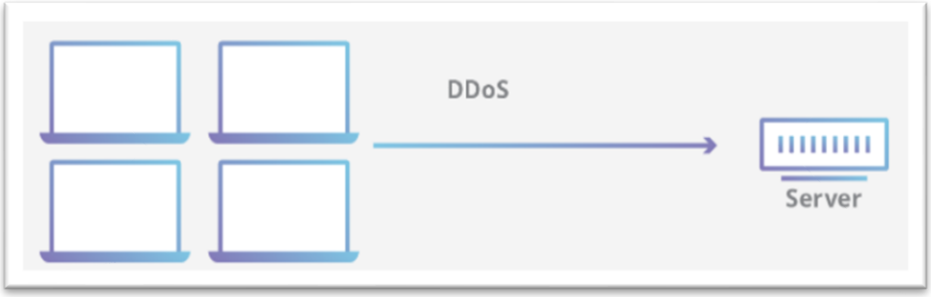
מתקפת DoS היא המתקפה הפשוטה ביותר מבין מתקפות מניעת השירות. במתקפה זו מחשב יחיד מנסה לגרום לעומס על שרת מסוים כדי שלא יוכל לשרת את הלקוחות שלו. בשביל שזה יקרה, לרוב אותו מחשב יצטרך כמות משאבים גדולה יותר, או לכל הפחות זהה בגודלה, לזו של השרת. מכיוון שכמות משאבים כזו יכולה לעלות לא מעט כסף, לרוב היא אינה פרקטית למימוש, והמתקפות הנפוצות מהסוג הזה משדרגות את המתקפה הזו בהיקף שלה ובתחכום שלה. חלק מסוגי המתקפות הללו נציג בפרקים הקרובים.



המחשה של מתקפת DoS. נלקח מ-cloudflare.com [55]

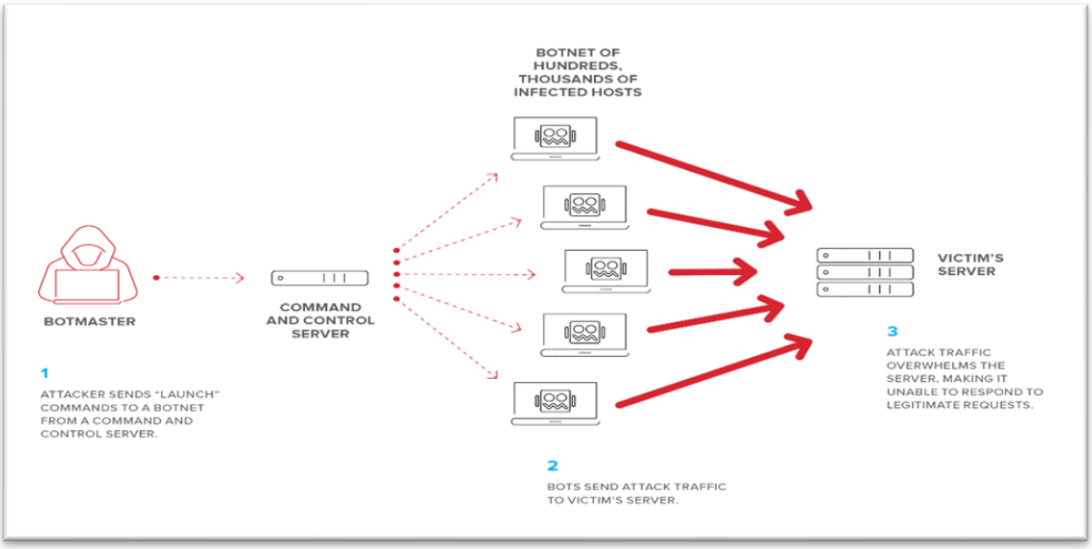
מתקפת DDoS

מתקפת DDoS, או Distributed Denial of Service, היא מתקפה הזהה במימושה למתקפת DoS הרגילה, רק שבמקום מחשב אחד אשר מנסה להשבית את השרת, כמה מחשבים עושים זאת בו זמנית, ועל כן הוספת המילה Distributed, אשר מציין כי המתקפה מבוצעת בצורה מבוזרת בין מספר מחשבים בו-זמנית. כפי שנאמר, בביצוע מתקפת DoS יש צורך בכמות משאבים גדולה מאוד (בהתאם למשאבי השרת) ועל כן ניתן לעקוף מגבלה זו ע"י ביצוע מתקפת DDoS שבה משתמשים בכמות רבה של מחשבים בעלי משאבים קטנים של חישוב ורוחב פס (שוב, ביחס לשרת).



המחשה של מתקפת DDoS. נלקח מ-cloudflare.com [55]

ברוב המקרים של מתקפת DDoS, המתקפה מבצעת באמצעות רשת מחשבים הנקראת Botnet, קבוצה של מחשבים המחוברים יחדיו באמצעות הרשת ומסנכרנים בינם ובין עצמם לאיזה שרת לשלוח הודעות ואיזה פעולות לבצע מולו כדי שיחדיו הם יגרמו לו להשבתה של השירות שלו. בדרך כלל יהיה מחשב מרכזי המופעל ע"י התוקף אשר ישלוט דרכו על כל שאר המחשבים ברשת ה-Botnet וכך תנוהל המתקפה באופן ידני. [50]

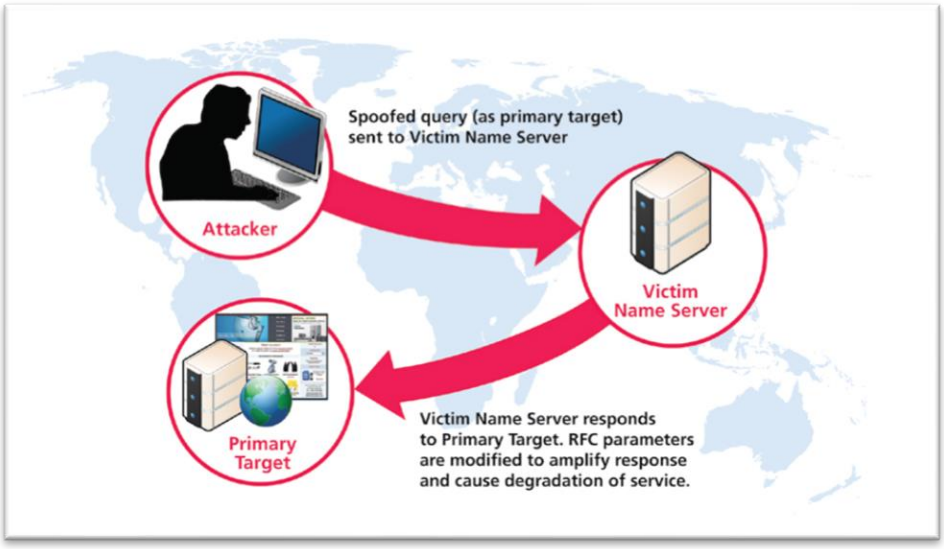


דוגמא להתקפת DDoS באמצעות Botnet. נלקח מ-f5.com [56]

מתקפת DRDoS

מתקפת DRDoS היא עוד מתקפה מסוג DOS שבה התוקף מנצל שרתי צד שלישי, כדוגמאת DNS או NTP, בשביל להגדיל את רוחב הפס של המחשבים התוקפים (השייכים ל-Botnet) ובכך להגדיל את העומס על רוחב הפס של השרת הקורבן וליצור צוואר בקבוק בתווך קבלת הבקשות. במתקפה DRDoS, או בשמה המלא Distributed Reflected Denial of Service, התוקף, השולט ברשת מחשבי ה-Botnet, אינו שולח את הבקשות אל השרת הקורבן כדי להציף אותו. במקום זאת, התוקף מפקד על מחשבי ה-Botnet לשלוח בקשות, שנראות לגיטימיות, לשרתים אמיתיים המספקים שירותים לשאר המחשבים ברשת. אותן בקשות יראו כבקשות אמיתיות לחלוטין, אך מה שהתוקף עושה הוא לשנות את כתובת המקור שלהן (Source IP Address) לכתובת של השרת הקורבן. כך למעשה, השרת הקורבן יקבל את התגובות לאותן בקשות שנשלחו ע"י מחשבי ה-Botnet. הסיבה לשינוי הזה היא שככל ששולחים יותר מידע לשרת הקורבן, כך ניתן להשבית אותו יותר בקלות. אבל, מכיוון שמחשבי ה-Botnet מוגבלים בכמות המידע שהם יכולים לשלוח בזמן נתון (רוחב פס קטן יחסית) אז בעזרת השיטה הזו הם יכולים לגרום לשליחת יותר מידע לשרת הקורבן עם פחות מידע. הסיבה לכך היא שהבקשות המזויפות, שנשלחות לשרתים הלגיטימיים, קטנות מאוד בגודלן ביחס להודעות התגובה להם, כך אם לדוגמא גודל הבקשה הוא 20 בתים והתשובה לבקשה הוא 2000 בתים, ניתן להכפיל את כמות המידע פי 100 ע"י שימוש במתקפה זו. ע"פ המידע בוויקיפדיה [51] הצליחו להגיע בעבר למקדם הכפלה של 50,000!

במאמר [52] מציגים כמה דרכים שבה ניתן לנצל את פרוטוקול DNS בשביל לבצע מתקפה של DRDoS. כידוע, DNS הוא פרוטוקול המאפשר לתרגם בין שמות domain לכתובת אינטרנט (IP) של שרתים. במאמרים מציגים כיצד שליחת בקשות מסוג ANY, TXT או A (סוגי בקשות של פרוטוקול DNS) יכולים ליצור תגובות מתאימות שארוכות בהרבה מאורך הבקשות, ולהגדיל את מקדם ההכפלה שישפר את המתקפה. בחלק מהמקרים ישנן דרישות מסוימות לביצוע המתקפה כמו הרשאות מתאימות של שרת הקורבן במקרה של בקשות מסוג ANY, אך בכולם ניתן לנצל את פרוטוקול ה-DNS ולהגדיל את מקדם ההכפלה, לפעמים פי 100 או יותר [51].



המחשב של התקפת DRDoS. נלקח מהמאמר [52]

הגנה מפני מתקפות מערכת

בכל המתקפות המתוארות היה מרכיב אחד משותף: ניצול של פרוטוקולים נפוצים לצורך התקפה של שרת או מחשב אשר משתמש בפרוטוקולים אלו. על כן, הגנה אחת, אשר אינה תמיד אפשרית, היא ביטול השימוש בפרוטוקולים אלו. אם הפרוטוקולים הללו אינם נמצאים בשימוש במחשב / שרת שעליו מנסים להגן, ניתן לבטל אותם ע"י הגדרות שונות במערכות ובכך למנוע מתקפות באמצעותם. הבעיה היא שבמרבית המקרים אותם פרוטוקולים הם פרוטוקולים חיוניים לעבודת המערכת, כמו ARP או DNS שהצגנו בפרק זה, ולכן לא ניתן לבטל אותם. במקרים כאלו, נדרשת עבודה יותר טובה ומעמיקה בנושא. ישנן מספר חברות אבטחה המשווקות מוצרים שכל מטרתם הם אבחון וניטור המידע העובר ברשת. מוצרים כדוגמאת Firewall וכדומה אשר יכולים לבדוק אילו הודעות נשלחות, מהיכן, באיזו תדירות, מה התוכן שלהן ועוד הרבה דברים מורכבים אשר מאפשרים לאותם רכיבים לזהות מתקפות פוטנציאליות ולהתריע ולעיתים גם למנוע אותן. רכיבים כאלו יכולים לשמש גם כהפרדה בין הרשת המקומית הסגורה לבין האינטרנט וגם בתוך רשת מקומית שעלולה להיות חשופה.

בנוסף, מתקפות מסוג DoS הן מתקפות מוכרות מאוד וישנם מספר חברות אבטחה שכל מטרתן הוא למנוע מתקפות מסוג זה. לאותן חברות יש כמות אסטרונומית של משאבי רשת והם מאפשרים לשרתים מסחריים להעביר את כל התעבורה שלהם דרכם, וכך הם יכולים לחסום איומי מערכת עוד לפני שהגיעו לשרתים עצמם. כיוון שלחברות הללו יש הרבה מאוד משאבי רשת, הם באופן עקרוני חסינים מפני תוקפים ברשת משום שלרוב ארגוני התקיפה אין מספיק משאבים בשביל להשפיע על החברות הגדולות הללו.

סיכום

עולם אבטחת המידע הוא עולם גדול ומורכב. ישנן הרבה מאוד סוגים של מתקפות וחולשות שיכולים לנצל ולהשמיש על מנת לפגוע בפרטיות ובמידע שלנו. אם מדובר בהתקפות המנצלות חולשות בקוד התוכנות שנכתבו על ידי המפתחים, כמו שהוצג בפרקים שפת C++ ו-שפת Python, או במתקפות המנצלות חולשות ומימושים של פרוטוקולי תקשורת, כפי שהוצגו בפרק תקשורת, וגם מתקפות שמטרתן לפגוע בכלל המערכת שמנהלת את המידע שלנו ומספקת שירותים לשאר המחשבים ברשת, כפי שהראתי בפרק אבטחת מערכת.

את כל המתקפות הללו ניתן למנוע או להקל בעזרת מספר שיטות וטכניקות אשר הוצגו בפרקים הקשורים אליהן, אך בכל אחת מהן מרכיב חשוב בהגנה מפניהן הוא המודעות. המודעות לאבטחה בעולם המחשבים והתקשורת לא הייתה תמיד גבוהה כפי שהיא היום, ואילו גם היום אנו שומעים לעיתים קרובות על מתקפות שונות אשר מכוונות כלפי ארגונים או חברות. הגברת המודעות לאבטחה היא בין הדברים החשובים ביותר שניתן לעשות בשביל למנוע מתקפות, גם כלפי חברות וארגונים וגם כלפי אנשים פרטיים. אף פעם לא נוכל להגן מפני כל החולשות והמתקפות הקיימות, אך הגברת המודעות לאבטחה תוביל בסופו של דבר לכך שלכל מתקפה או חולשה חדשה ימצאו פתרון טכנולוגי כזה או אחר שימנע או יקל עליהם, ויפיצו את העדכונים הרלוונטיים לגורמים הרלוונטיים כדי למנוע אותם. בהתאם, יותר אנשים יקבלו את אותם עדכוניים ויתקינו אותם כי הם מודעים לכך שהם מספקים הגנה מפני אותם מתקפות או חלשות אשר צצות מידי יום.

לפיכך, אני מאמין שכל אדם אשר עושה שימוש במחשבים או רכיבי תקשורת כאלו ואחרים צריך להיות מודע לסכנות הקיימות בשימושים הללו, ולפעול בהתאם כדי למנוע אותם. בנוסף, ארגונים וחברות המספקות שירותים דרך אותם אמצעים צריכים להגן על עצמם, ולא פחות חשוב מכך על הלקוחות שלהם, אשר סומכים עליהם במתן שירות בטוח ואמין.

ביבליוגרפיה

[1] Albatross, "History of C++," [Online]. Available: http://www.cplusplus.com/info/history/.

[2] Albatross, "C++ - Brief Description," [Online]. Available: http://www.cplusplus.com/info/description/.

[3] "C++11," [Online]. Available: https://en.wikipedia.org/wiki/C++11.

[4] "C++20," [Online]. Available: https://en.wikipedia.org/wiki/C++20.

[5] "Wikipedia - C++," [Online]. Available: https://en.wikipedia.org/wiki/C++.

[6] "Classes," [Online]. Available: http://www.cplusplus.com/doc/tutorial/classes/.

[7] "Friendship and Inheritance," [Online]. Available: http://www.cplusplus.com/doc/tutorial/inheritance/.

[8] "Wikipedia - Polymorphism," [Online]. Available: https://he.wikipedia.org/wiki/פולימורפיזם_(מדעי_המחשב).

[9] "Templates," [Online]. Available: https://www.cplusplus.com/doc/oldtutorial/templates/.

[10] "Memory Layout," [Online]. Available: https://www.geeksforgeeks.org/memory-layout-of-c-program/.

[11] "Dymanic Memory," [Online]. Available: http://www.cplusplus.com/doc/tutorial/dynamic/.

[12] P. LACROIX and J. DESHARNAIS, "Buffer Overflow Vulnerabilities in C and C++," Department of Computer Science and Software Engineering, Université Laval, Quebec, 2008.

[13] "Exploiting C++ VTABLES: Instance Replacement," 11 May 2013. [Online]. Available: https://defuse.ca/exploiting-cpp-vtables.htm.

[14] C. Zhang, C. Song, K. Z. Chen and Z. Chen, "VTint: Protecting Virtual Function Tables' Integrity," in *Network and Distributed System Security Symposium*, San Diego, 2015.

[15] "NX bit," [Online]. Available: https://en.wikipedia.org/wiki/NX_bit.

[16] A. G. M. F. Babak Salamat, "Reverse Stack Execution in a Multi-Variant Execution Environment," School of Information and Computer Sciences - University of California, Irvine, 2012.

[17] "Python - Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Python_(programming_language).

[18] T. Einat, "Most Common Licenses for Python Packages," [Online]. Available: https://snyk.io/blog/over-10-of-python-packages-on-pypi-are-distributed-without-any-license/#:~:text=Most%20Common%20Licenses%20for%20Python,by%20at%20least%20100%20packages)..

[19] A. M. Fawzi Albalooshi, "A Comparative Study on the Effect of Multiple Inheritance Mechanism in Java, C++, and Python on Complexity and Reusability of Code," *International Journal of Advanced Computer Science and Applications,* vol. 8, no. 6, pp. 109-116, January 2017.

[20] D. Hamann, "Exploiting Python pickles," [Online]. Available: https://davidhamann.de/2020/04/05/exploiting-python-pickle/.

[21] A. Kumar, "SOUR PICKLE : Insecure Deserialization with Python Pickle module," [Online]. Available: https://medium.com/@abhishek.dev.kumar.94/sour-pickle-insecure-deserialization-with-python-pickle-module-efa812c0d565.

[22] "Pickle — Python object serialization," [Online]. Available: https://docs.python.org/3/library/pickle.html.

[23] K. Tanaka and T. Saito, "Python Deserialization Denial of Services," in *Computational Science/Intelligence & Applied Informatics*, Springer, 2018, pp. 15-25.

[24] "The import system," [Online]. Available: https://docs.python.org/3/reference/import.html.

[25] S. Whang, "Privilege Escalation: Hijacking Python Library," [Online]. Available: https://medium.com/@klockw3rk/privilege-escalation-hijacking-python-library-2a0e92a45ca7.

[26] A. Shaw, "10 common security gotchas in Python and how to avoid them," 16 June 2018. [Online]. Available: https://hackernoon.com/10-common-security-gotchas-in-python-and-how-to-avoid-them-e19fbe265e03.

[27] "ZipFile - extractall," [Online]. Available: https://docs.python.org/3/library/zipfile.html#zipfile.ZipFile.extractall.

[28] A. Abraham, "Exploiting insecure file extraction in Python for code execution," 28 September 2017. [Online]. Available: https://ajinabraham.com/blog/exploiting-insecure-file-extraction-in-python-for-code-execution.

[29] MITRE, "CVE-2019-20907 - TAR archive leading to an infinite loop," 13 July 2020. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2019-20907.

[30] MITRE, "CVE-2019-9674 - denial of service via a ZIP bomb," 4 February 2020. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2019-9674.

[31] Nandiya, "CVE-2013-7338 - ZipFile Denial Of Service," [Online]. Available: https://www.cvedetails.com/cve/CVE-2013-7338/.

[32] M. Baggett, "Pen Test Poster: "White Board" - Python - Python Reverse Shell," 31 January 2017. [Online]. Available: https://www.sans.org/blog/pen-test-poster-white-board-python-python-reverse-shell/.

[33] S. GeeksforGeeks, "Vulnerability in input() function – Python 2.x," [Online]. Available: https://www.geeksforgeeks.org/vulnerability-input-function-python-2-x/.

[34] S. C. P. E. M. M. E. O. C. R. B. Y. Danny Price, "Hickle: A HDF5-based python pickle replacement," *The Journal of Open Source Software,* vol. 3, no. 32, p. 1115, 2018.

[35] J. Kloc, "Did the NSA just crack RSA encryption?," 2 March 2020. [Online]. Available: https://www.dailydot.com/debug/nsa-rsa-encryption-crack-prime-numbers/.

[36] Wikipedia, "Diffie–Hellman key exchange," [Online]. Available: https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange.

[37] "RSA," [Online]. Available: https://en.wikipedia.org/wiki/RSA_(cryptosystem).

[38] "TLS - Cipher," [Online]. Available: https://en.wikipedia.org/wiki/Transport_Layer_Security#Cipher.

[39] "TLS 1.3," [Online]. Available: https://en.wikipedia.org/wiki/Transport_Layer_Security#TLS_1.3.

[40] cloudflare.com, "What is a TLS handshake?," [Online]. Available: https://www.cloudflare.com/learning/ssl/what-happens-in-a-tls-handshake/.

[41] A. Prodromou, "TLS Security 6: Examples of TLS Vulnerabilities and Attacks," 31 March 2019. [Online]. Available: https://www.acunetix.com/blog/articles/tls-vulnerabilities-attacks-final-part/.

[42] Z. Banach, "How The BEAST Attack Works," [Online]. Available: https://www.netsparker.com/blog/web-security/how-the-beast-attack-works/.

[43] M. Carvalho, J. DeMott, R. Ford and D. A. Wheeler, "Heartbleed 101," *IEEE Security and Privacy Magazine,* vol. 12, no. 4, pp. 63-67, 2014.

[44] "Heartbleed," [Online]. Available: https://en.wikipedia.org/wiki/Heartbleed.

[45] "Address Resolution Protocol," [Online]. Available: https://en.wikipedia.org/wiki/Address_Resolution_Protocol#:~:text=The%20Address%20Resolution%20Protocol%20(ARP,in%20the%20Internet%20protocol%20suite..

[46] C. Nachreiner, "Anatomy of an ARP Poisoning Attack," 18 November 2012. [Online]. Available: http://csci6433.org/Papers/Anatomy%20of%20an%20ARP%20Poisoning%20Attack%20_%20WatchGuard.pdf.

[47] B. Fleck and J. Dimov, "Wireless Access Points and ARP Poisoning," December 2013. [Online]. Available: https://digilander.libero.it/SNHYPER/files/arppoison.pdf.

[48] S. Son and V. Shmatikov, "The Hitchhiker's Guide to DNS Cache Poisoning," in *Security and Privacy in Communication Networks*, Springer, 2010, pp. 466-483.

[49] P. Nicholson, "Five Most Famous DDoS Attacks and Then Some," A10networks, 27 July 2020. [Online]. Available: https://www.a10networks.com/blog/5-most-famous-ddos-attacks/.

[50] "Botnet - Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Botnet#:~:text=A%20botnet%20is%20a%20number,the%20device%20and%20its%20connection..

[51] "Denial-of-service attack," [Online]. Available: https://en.wikipedia.org/wiki/Denial-of-service_attack#Amplification.

[52] Prolexic, An Analysis Of DrDoS DNS Reflection Attacks, Prolexic, 2013.

[53] "Buffer Overflow," [Online]. Available: https://www.tenouk.com/Bufferoverflowc/Bufferoverflow4.html.

[54] "Public key certificate," [Online]. Available: https://en.wikipedia.org/wiki/Public_key_certificate.

[55] "What is a Denial-of-Service (DoS) Attack?," Cloudflare, [Online]. Available: https://www.cloudflare.com/learning/ddos/glossary/denial-of-service/.

[56] D. Walkowski, "What Is a Distributed Denial-of-Service Attack?," F5 Labs, 5 June 2019. [Online]. Available: https://www.f5.com/labs/articles/education/what-is-a-distributed-denial-of-service-attack-.

[57] D. Wagner and B. Schneier, "Analysis of the SSL 3.0 Protocol," *Proceedings of the Second USENIX Workshop on Electronic Commerce,* pp. 29-40, 18-21 November 1996.

[58] N. Mavrogiannopoulos, F. Vercauteren, V. Velichkov and B. Preneel, "A cross-protocol attack on the TLS protocol," in *ACM conference on Computer and communications security*, Raleigh, 2012.

[59] S. K. F. A. E.-S. O. Tamar Fatayer, "A key-agreement protocol based on the stack-overflow software vulnerability," in *IEEE Symposium on Computers and Communications*, Riccione, 2010.

[60] D. W. Nicholas Carlini, "ROP is Still Dangerous: Breaking Modern Defenses," in *Proceedings of the 23rd USENIX Security Symposium*, San Diego, USENIX, 2014, pp. 385-399.