Time side-channel attack

By: Avi Feder && Daniel Yochanan

אז מה זה Time side-channel attack?

side-channel attack היא מתקפה המנצלת מידע שמושג מאופן היישום הפיזי או השימוש של מערכת, שניתן לנצל כדי לתקוף מערכות, או כדי להשיג מידע מסווג .

Time side-channel attack היא מתקפה מסוג זה המתבסס על מדידות זמנים של חישובים ופעולות כלשהם בצד הנתקף. כך שבאמצעות אנליזה על זמנים אלו, התוקף יכול להשיג מידע סודי כמו מפתח ההצפנה המערכת הנתקפת.

הקדמה

בחלק הראשון של הפרויקט מימשנו Timing attack על מערכת של קוד לבניין.

לפני שיכלנו להתחיל לממש את ההתקפה היינו צריכים ליצור מערכת כזאת.

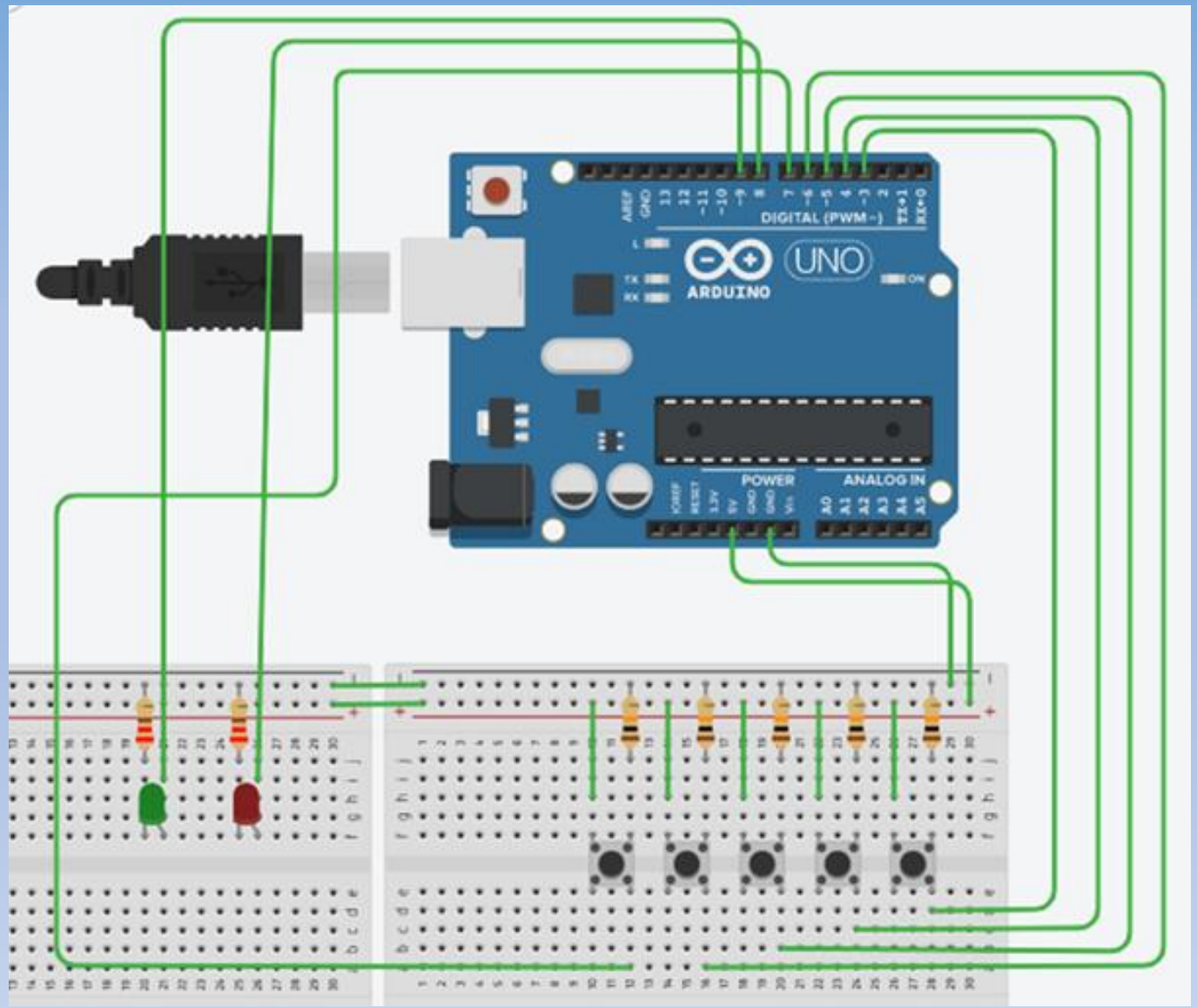
לקחנו לוח Arduino וחיברנו אליו חמישה כפתורים ושני לדים, לד ירוק הנדלק כאשר מזינים את הקוד הנכון, ולד את אדום הנדלק כאשר מזינים קוד שגוי.

כמובן שמעבר לחיבור הפיזי של הרכיבים ללוח,

עלינו לתכנת את הלוח שיפעל ככה.

לוח Arduino:

מערכת הקוד לבניין שלנו נראית כך:



החולשה בצד הנתקף

מערכת הקוד לבניין שלנו עובדת כך:

כאשר מבצעים חמש לחיצות ובעצם מזינים קוד כלשהו, המערכת משווה את הקוד שהוזן לקוד השמור במערכת באמצעות השוואת מחרוזות פשוטה.

היא עוברת בו זמנית על שתי המחרוזות (זאת שהוקשה והמחרוזת שבזכרון) ומשווה תו תו. ברגע שהיא מזהה שאחד מהתווים לא שווה היא מדליקה את הנורה האדומה.

רק כאשר המערכת תסיים לעבור על כל התווים של הקוד, כלומר אם כל התווים זהים, המערכת תדליק נורה ירוקה.

מה שקיבלנו בעצם, זאת מערכת שככל שהקוד המוזן בה יותר נכון, ככה זמן הריצה של השוואת המחרוזות יקח יותר זמן. וככה ייקח לנורה האדומה יותר זמן להידלק.

וזאת החולשה שאנחנו ננצל על מנת לפרוץ את המערכת :)

כלומר, אנו נמדוד את זמן התגובה של המערכת ללחיצות שלנו

ונסיק לפי זמן התגובה , במה תווים בקוד צדקנו.

מימוש המתקפה

את המתקפה שלנו ביצענו באמצעות לוח Tiva אשר חיברנו למגעים של הכפתורים והנורות.

כך שלוח הTiva שלנו יכול "ללחוץ" על הכפתורים ולמדוד כמה זמן לקח למערכת הקוד לבניין להדליק את הנורה האדומה או הירוקה במקרה של הצלחה.

תחילה נזין קוד שמתחיל בספרה 1 ונמדוד כמה זמן לקח לנורה האדומה להידלק, לאחר מכן נזין קוד שמתחיל בספרה 2 ונמדוד זמנים, וכך הלאה..

כעת כשיש לנו את זמני הריצה של כל הספרות עבור התו הראשון של הקוד נוכל לקבוע שהתו הראשון של הקוד מתחיל בספרה שעבורה הזמנים שמדדנו היו הכי גדולים.

וכך נעבור תו תו עד שנשיג את הסיסמא כולה ונפרוץ למערכת!

שיפור של המערכת הנתקפת?

לכאורה, על מנת לנסות למנוע מתקפה שכזו, ניתן להוסיף בכל בדיקה של תו delay של זמן רנדומלי. כך נגרום שזמן הריצה של השוואת המחרוזות לא יהיה קבוע, והתוקף לא יוכל להניח שקוד שלקח לו יותר זמן לרוץ הוא יותר נכון.


```
function timingSafeCompare($a,$b) {
    sleep(rand(0,100));
    if ($a === $b) {
        return true;
    } else {
        return false;
    }
}
```

עדיין פריץ!

הוספת delay רנדומלי אומנם מוסיפה קושי מסוים בפריצה אבל גם את זה ניתן לפרוץ.

במתקפה זו נשתמש באותה הטכניקה של המתקפה הקודמת, רק שבמקום להזין רק פעם אחת כל קוד שניסינו, נזין מספר פעמים את אותו הקוד ונחשב את ממוצע הזמנים עבור כל קוד.

כך נתקדם תו תו עד שלבסוף גם הפעם נמצא את הקוד הנכון ונפרוץ למערכת.

הרעיון מאחורי המתקפה

כפי שהראנו, כעת יש למערכת תוספת זמן רנדומלית בין [a, b].

לכן, אם נזין כל קוד מספר פעמים, ונחשב עבורו את ממוצע זמני הריצה, אנחנו נקבל בממוצע את זמן הריצה האמיתי שלו בתוספת של (a+b)/2.

כך, מכיוון שעבור כל קוד אנחנו מחשבים את ממוצע זמן הריצה שלו, אנחנו מקבלים עבור על הקודים את זמן הריצה האמיתי שלהם בתוספת קבועה לכולם.

ולכן עדיין נוכל לזהות לאיזה קוד לקח הכי הרבה זמן לרוץ ולפרוץ את המערכת!

RSA Timing attack

החלק השלישי והאחרון של הפרויקט שלנו עוסק במתקפה על מערכת RSA.

בחלק זה נממש מערכת RSA על לוח Arduino שיהיה עבורנו קופסא שחורה, שחותמת על הודעות שאנחנו שולחים אליה באמצעות המפתח הפרטי שלה.

לאחר מכן באמצעות אך ורק מדידת זמני החתימה של הודעות שונות שנשלח, נצליח למצוא ולדעת מה המפתח הפרטי של המערכת.

כיצד עובדת מערכת הצפנה RSA

במערכת RSA יש שני מפתחות א-סימטריים, אחד פרטי - d ואחד ציבורי - e. והכל בעולם מודולו n.

בשביל לחתום על הודעה m, לוח הArduino מבצע את הפעולה $m^d \ mod(n)$

באמצעות Repeated Squaring.

Repeated Squaring היא שיטה לביצוע modular exponent (פעולת החזקה והמודולו) בצורה יעילה.

בשיטה זו אנו ממירים את d (המפתח הפרטי) למספר בינארי ואז עוברים ביט ביט משמאל לימין.

בכל ביט אנו נעלה את m בריבוע. ובנוסף, אם הביט הינו 1, נכפיל את הערך שקיבלנו עם ההודעה המקורית.

כמובן שבכל איטרציה (כאשר סורקים את הביטים משמאל לימין) מבצעים הפחתה (%n) במידת הצורך כדי להשאר בעולם המודולו n.

ההפחתה הזאת היא נקודת החולשה של המערכת שננצל במתקפה.

Repeated Squaring

נראה דוגמה כיצד Repeated Squaring הופך לנו את הפעולה $12^{45}\ mod(40)$ לפעולה פשוטה:

$$12^{45} = 36572619880088371967140823026550308340274372280 32$$
$$36572619880088371967140823026550308340274372280 32\ mod(40) = 32$$

נפעל בצורה כזאת:

$12^{45}\ mod(40)$

$(45)_{10} = (101101)_2$

$12^2 = 144 = 24\ [mod(40)]$

$24^2 * 12 = 6912 = 32\ [mod(40)]$

$32^2 * 12 = 12288 = 8\ [mod(40)]$

$8^2 = 64 = 24\ [mod(40)]$

$24^2 * 12 = 6912 = 32\ [mod(40)]$

הרעיון מאחורי ההתקפה

נניח שנמצא Y, Z כך שמתקיים - $Y^3 < \mathrm{N}$ וגם - $Z^2 < \mathrm{N} < Z^3$

כלומר, Y לא יצטרך הפחתה גם אם נעלה אותו בריבוע ונכפיל בערך המקורי שלו (כלומר נעלה בשלישית).

ולעומת זאת, Z לא יצטרך הפחתה רק אם נעלה אותו בריבוע. אבל ברגע שנכפיל אותו אח"כ בערך המקורי שלו (כלומר נעלה בשלישית) הוא יצטרך הפחתה.

את שתי ההודעות האלו נשלח לArduino על מנת שיחתום אותם עם המפתח הפרטי שלו.

הרעיון מאחורי ההתקפה

תזכורת:
$Y^3 < \mathrm{N}$ ו- $Z^2 < \mathrm{N} < Z^3$

כעת לאחר ששלחנו את ההודעות לחתימה, אנו מתחלקים לשני מקרים:

<u>במקרה הראשון</u> – הביט הוא 1, לכן נצטרך להכפיל את ההודעה הנחתמת בהודעה המקורית ולא רק להעלות אותה בריבוע.

במקרה כזה, הודעה Z תרוץ יותר זמן. כי כאשר מעלים את Z בשלישית נצטרך לבצע גם הפחתה לעומת Y שלא.

<u>במקרה שני</u> – הביט הוא 0, לכן נצטרך רק להעלות את ההודעה הנחתמת בריבוע, בלי להכפיל את ההודעה הנחתמת בהודעה המקורית .

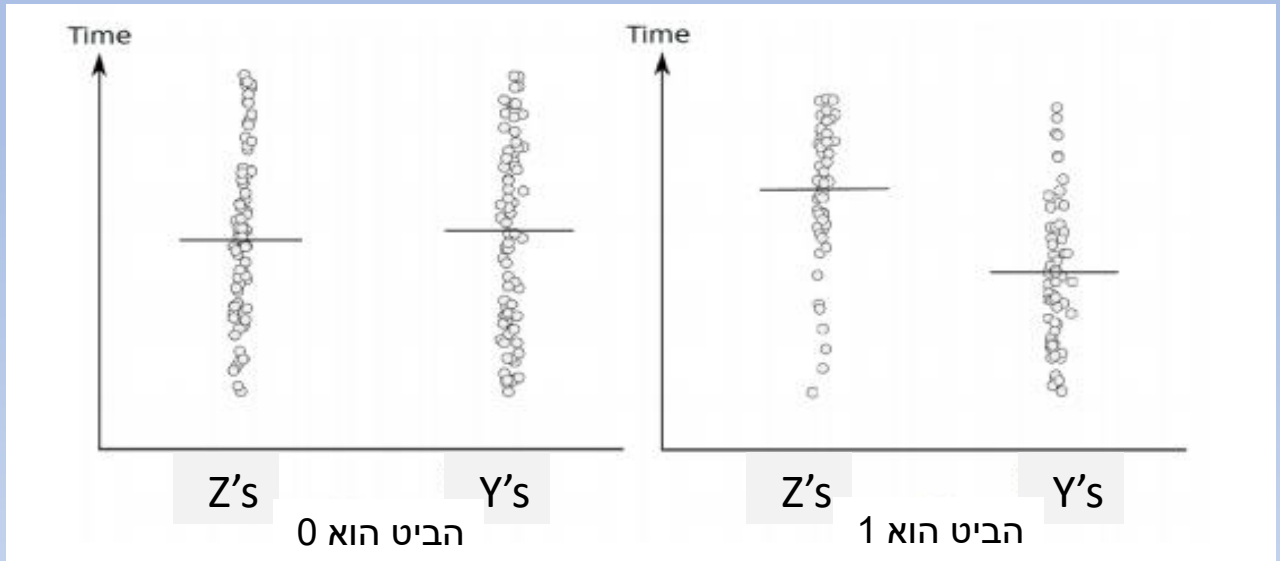
במקרה כזה שתי ההודעות ירוצו באותו זמן מכיוון ששתיהן לא יצטרכו הפחתה.

הרעיון מאחורי ההתקפה

כעת, לאחר שנשלח את ההודעות ונדע מהם זמני הריצה שלוקח לחתום אותן, נוכל להשוות בין זמני הריצה של Y וZ, ואם הם פחות או יותר שווים נדע שהביט הבא הוא 0 (כי לא הייתה הפחתה להודעה Z), אחרת נדע שהביט הוא 1 (כי להודעה Z הייתה הפחתה).

כך נעשה שוב ושוב עד שנפרוץ את כל המפתח!

כמובן שלא באמת נסתפק בלשלוח הודעה אחת מכל סוג ונשלח הרבה יותר ונעשה ממוצע.



תזכורת:
$Y^3 < \mathrm{N}$ ו- $Z^2 < \mathrm{N} < Z^3$

אם הביט הוא 1:
זמן הריצה של Z יהיה גדול בממוצע מזמן הריצה של Y מכיוון שתהיה הפחתה עבור Z.
אם הביט הוא 0:
זמני הריצה יהיו שווים בממוצע.

ההתקפה עצמה

בכדי שיהיה לנו חומר לניתוח הזמנים אנחנו שולחים אלפי הודעות רנדומליות לArduino על מנת שיחתום אותן באמצעות המפתח הפרטי שלו ומודדים עבור כל הודעה כמה זמן לקח לו לחתום אותה.

בשלב הראשוני ידוע לנו שהביט הראשון של המפתח הפרטי הוא 1 ואנחנו רוצים לגלות כל פעם מה הביט הבא במפתח.

כעת ניקח את רשימת ההודעות ששלחנו ונפעיל על כל אחת מההודעות את האלגוריתם Repeated squaring עד הביט שידוע לנו. עבור כל אחת מההודעות נבדוק האם בביט הבא בהנחה שהוא 1 יש הפחתה או לא (כלומר, האם היה צורך ב% או לא).

כך שנחלק בעצם את הרשימה שלנו לשתי קבוצות:

קבוצה א – כל ההודעות שבהן היה צורך בהפחתה אם הביט הבא הוא 1.

קבוצה ב – שאר ההודעות. כלומר ההודעות שבהן לא היה צורך בהפחתה אם הביט הבא הוא 1.

כעת נחשב את ממוצע זמן הריצה עבור כל קבוצה. ונבדוק האם הפרש הזמנים הינו זניח ולכן הביט הוא 0 או שהפרש הזמנים לא זניח ולכן הביט הוא 1.

ההתקפה עצמה

הרצנו מבעוד מועד את הסקריפט ששולח הודעות רנדומליות לArduino ומחשב את זמני הריצה עבור כל הודעה (לוקח לסקריפט לרוץ כמה שעות טובות ולכן הכנו מראש).

נריץ כעת את הקוד שמבצע ניתוח על זמני הריצה האלו ובעצם פורץ את המפתח הפרטי.

לסיכום

למדנו מהו Timing attack, הראנו כיצד לפרוץ מערכת קוד לבניין ובנוסף הראנו כיצד להתגבר על הגנות כמו הוספת delay רנדומלי ולהמשיך להצליח לפרוץ את המערכת.

כמו כן למדנו איך מערכת ה RSA עובדת ומה עלינו לעשות על מנת לפרוץ גם אותה.

כל זאת עשינו באמצעות ניתוח של זמני ריצה!

מקווים שנהניתם :)