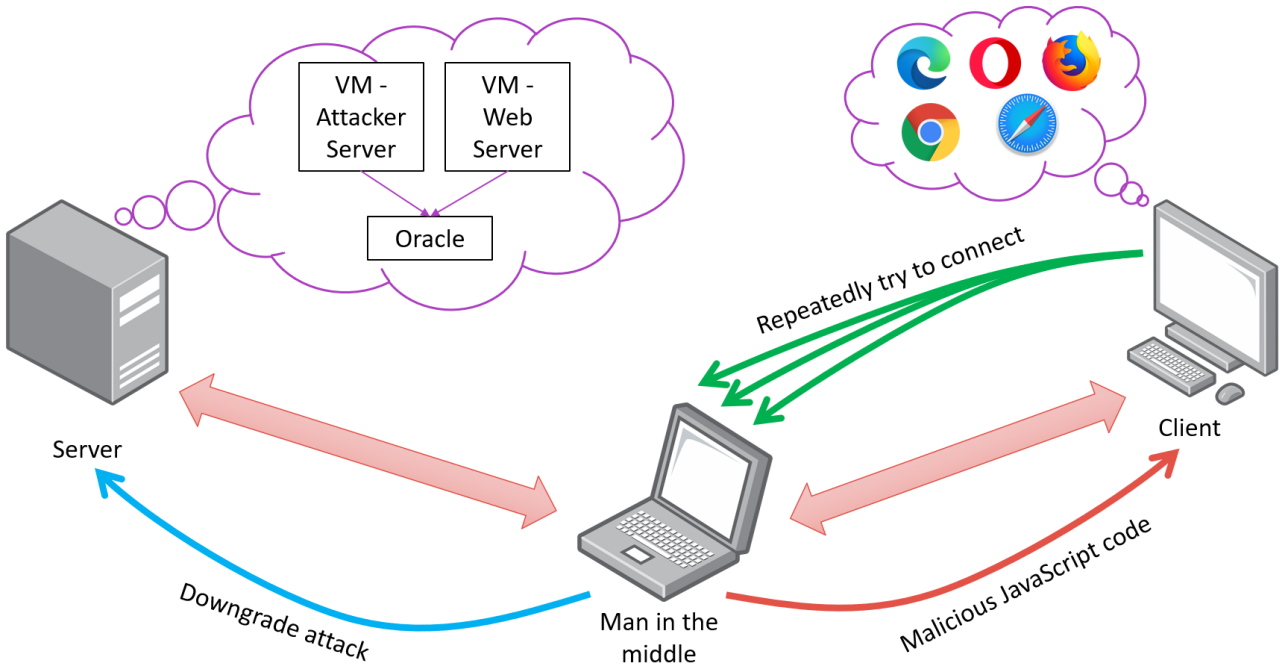
Names: Shaked Fish
Mariam Zoabi

IDs: 206317554
315661629

The 9 Lives of Bleichenbacher's CAT: New Cache Attacks on TLS Implementations

Are modern implementations of PKCS #1 v1.5 secure against padding oracle attacks?



תוכן עניינים

1. הקדמה

א. החולשה

התקפה בעזרת ערוצי צד במיקרו-ארכיטקטורה הינה התקפה מבוססת על ניצול חולשה חומרתית ותכנותית,
אשר מאפשרת לנו להדליף מידע לגבי התהליך הקורבן.
מתוך המאמר בחרנו להשתמש בהתקפת Flush & Reload (F&R).

היררכית הזיכרון של המחשב מורכבת ממספר חלקים:

- הזיכרון הראשי (RAM), הוא הזיכרון הגדול ביותר והאיטי ביותר.
- זיכרון מטמון (Cache), הוא נמצא ישירות בתוך המעבד, אשר מחולק למספר שכבות, L1, L2, L3.
  כאשר L1 היא השכבה המהירה ביותר אך הקטנה ביותר,
  עד L3 שהיא הגדולה ביותר מתוכן והאיטית יותר.
  ובנוסף, L3 היא גם משותפת לכל ליבות המעבד.
- קיימים עוד זיכרונות שונים אשר לא משפיעים על ההתקפה (כמו HDD וכד').

התקפת F&R מתבצעת לפי השלבים הבאים:

1. התוקף משתמש בפקודת clFlush בשביל למחוק את המידע של התהליך הנתקף מכל שכבות המטמון.
2. התוקף ממתין, כדי לתת זמן לתהליך הנתקף לגשת לזיכרון (או לא לגשת).
3. התוקף מנסה לגשת למידע בעצמו ומודד את המזמני הגישה למידע,
   אם הגישה היא איטית אז הוא יודע שהוא נגש לזיכרון הראשי ומחזיר "החטאה"
   (כלומר התהליך הנתקף עוד לא הביא בעצמו את המידע מהזיכרון הראשי),
   אחרת הגישה הייתה מהירה אז הוא יודע שהוא נגש לזיכרון מטמון ומחזיר "פגיעה"
   (כלומר התהליך הנתקף הביא בעצמו את המידע מהזיכרון הראשי).

מבחינה חומרתית, ההתקפה מנצלת את ה-Inclusiveness של מעבדי Intel,
כלומר שאם מידע נמצא במטמון ברמה כלשהי בליבה, אז הוא נמצא גם כן במטמון ברמות שמעליו.
לכן, ידוע לנו שכאשר אנו מבצעים את פקודת clFlush המידע נמחק מכל רמות המטמון,
גם בליבה עליה התוכנית הנתקפת רצה.
ללא תכונה זאת של המעבד, ביצוע השלב הראשון של ההתקפה יכול לא להשפיע על התהליך הנתקף (אשר המידע יכול עדיין לא היות קיים בשכבות L1, L2 אשר לא משותפות בין כל הליבות) ובכך ההתקפה לא תעבוד.

מבחינה תוכניתית, ההתקפה מנצלת קוד שלא רץ בזמנים קבועים, כלומר אם כן היה רץ בזמן קבוע, אז התקפתנו הייתה מחזירה את אותה תשובה בכל פעם (פגיעה או החטיאה) ולכן התוקף לא היה מצליח ללמוד מידע חדש על הנתקף.
יחד עם זאת, בשביל לבצע את ההתקפה נדרש שהזיכרון של התהליכים התוקף והנתקף יהיה משותף כדי שהתוקף יוכל לבצע clFlush על הזיכרון של הנתקף בזיכרון הראשי.

תוצאות ההתקפה שימושיות לנו, כיוון התוקף יכול לדעת האם התוכנית הנתקפת נגשה למידע כלשהו בזיכרון,
דבר זה מדליף לנו מידע חיוני על תהליך הריצה של התוכנית הנתקפת.

ב. רקע מערכתי

מתוך המאמר בחרנו בהתקפת על שרת OpenSSL.

התקפת מתבצעת לפי השלבים הבאים:

1. התוקף משתלת על התקשורת בין השרת ללקוח ויוצר Man-in-the-Middle (MITM).
2. התוקף מבצע התקפת Downgrade ובכך מאלץ את התקשורת בין השרת ללקוח לעבוד בעזרת פרוטוקול TLS v1.2 וגם מאלץ הצפנת RSA עם פרוטוקול ריפוד PKCS#1 v1.5.
3. התוקף משתמש בהתקפת Beast בשביל להחדיר קוד זדוני לדפדפן של הלקוח המאלץ אותו לפנות לשרת שוב ושוב ובכך מאלץ את השרת לבצעה פיענוח עבור התוקף.
4. בעזרת התקפת ערוצי צד, התוקף מגלה מידע על ההודעה המפוענחת ע״י השרת, אשר יוצר Oracle המאפשר לתוקף לפענח בעצמנו את הפרטים הסודיים של הלקוח.

בשביל ליעל את ההתקפה (ולגרום לה לעבוד תחת מגבלות הזמן), התוקף יכול בו זמנית לפנות למספר שרתי קצה של אותו חברה ולבצע את אותה התקפה, ולבסוף לאחד את תוצאות ההתקפות.

ההתקפה הנ״ל הינה מבוססת על ניצול חולשה חומרתית ותכנותית,
אשר מאפשרת לנו לזייף את התקשורת בין השרת ללקוח ובכך לבצעה את פעולות בשם הלקוח.

מבחינה חומרתית, ההתקפה מנצלת:

- **יכולת ביצוע התקפת ערוצי צד** – כלומר המחשב שעליו רץ השרת פגיע לאותן חולשות שצוינו בסעיף הקודם.

מבחינה תוכניתית, ההתקפה מנצלת:

- **יכולת הרצת קוד של התוקף באותו מחשב עם השרת** – בשביל לבצע התקפת ערוצי צד.
  באופן כללי, בעזרת התקפת Prime & Probe התוקף יכול להריץ קוד במכונה וירטואלית על אותו מחשב שבו רץ השרת (דבר פופולארי כאשר השרת נמצא בענן ציבורי).
  באופן ספציפי יותר להתקפתנו, בשביל לבצע F&R נדרש גם שהזיכרון יהיה משותף בין התהליך של השרת והתהליך של התוקף, כלומר הם צריכים לרוץ תחת אותה מערכת הפעלה.
- **יכולת האזנה/עריכת התקשורת בין השרת והלקוח** – כדי שהתוקף יוכל לבצע MITM.
- **יכולת שליחת בקשות פענוח טקסט לשרת** – בשביל שהתוקף יוכל להפעיל את ה-Oracle שלו ולגלות מידע על הפענוח.
- **שכל שרתי הקצה של אותו חברה משתמשים באותם תעודה (Certificate)** – כדי שנוכל להשתמש באותם נתונים של הלקוח לבקש פענוחים מהשרתים.

תוצאות ההתקפה שימושיות לנו, כיוון שהן יאפשרו לנו לזייף את התקשורת בין הלקוח לשרת,
כלומר להעביר בקשות לשרת על שם הלקוח.

ג. רקע על התקפות קודמות

להלן התקפות קודמות אשר עזרו בפיתוח ההתקפה הנוכחית:

1. **התקפות Padding Oracle** [5, 8, 11, 42] – התקפות המנצלות חולשה בפרוטוקול ריפוד ע"י שימוש ב-Oracle כלשהו שיכול לגלות מידע כלשהו לגבי השרת.
2. **התקפת Bleichenbacher** [11] – היא התקפה המנצלת את מבנה הריפוד בפרוטוקול PKCS#1 v1.5 ע"י שימוש בהתקפת Padding Oracle, זוהי אחת משיטות ההתקפה בהם משתמשים במאמר.
3. **התקפות עדכניות בסגנון Bleichenbacher** [12, 40, 42, 47, 48, 72] – אלו התקפות שגם הן משתמשות ב-Padding Oracle בצורה דומה ל-Bleichenbacher.
4. **התקפה מקבילית של Padding Oracle** [12, 42, 52] – בדומה להתקפה Padding Oracle, אך היא מתבצעת על מספר שרתים בו זמנית.
5. **התקפת Downgrade** [18, 41] – היא התקפה בה התוקף מאלץ את הנתקף להשתמש בפרוטוקול תקשורת ספציפי שהוא מעוניין בו.
6. **התקפת Man in the middle** [18, 41] – היא התקפה בה התוקף מתחזה לשרת אליו הלקוח מנסה להתחבר, התוקף מעביר אל השרת את המידע שהלקוח מנסה להעביר, אך יכול להיות שהתוקף קורה את המידע הזה או עורך אותו.
7. **התקפת Beast** [23] – זאת התקפה בה התוקף מנצל חולשה של פרוטוקול TLS המאפשרת לו לפצח את התקשורת בין השרת ללקוח על ידי ניסיונות חוזרים של יצירת תקשורת עם השרת, ופענוח של חלק קטן מהפתח הסודי בכל פעם.
8. **התקפות Cache** [2, 3, 6, 10, 12, 14, 16, 17, 22, 24, 25, 26, 28, 30, 31, 32, 33, 37, 38, 39, 40, 42, 43, 44, 46, 48, 50, 55, 56, 57, 60, 64, 66, 67, 69, 70, 71, 72] – אלו התקפות ערוצי צד, המאפשרות לגלות מידע לגבי התוכנית הנתקפת לפי הגישה שלה לזיכרון.

ד. תרומת המאמר

המאמר עונה על השאלה: האם יישומים מודרניים של פרוטוקול PKCS #1 v1.5 בטוחים מפני התקפות Padding Oracle? והתשובה לכך היא לא.

המאמר מראה זאת על ידי הצלחת ההתקפה על 7 מתוך 9 שרתים פופולריים לאותה תקופה ואיכן יש לתקן את הקוד הפגיע בשרתים הללו.

בנוסף המאמר מציג שיטה יעילה לביצוע ההתקפה כדי לעמוד בהגבלת הזמן (30 שניות) התחברות הלקוח לשרת.

ההתחדשות שהמאמר מביא הינה טכניקת ההקבלה, אשר מציגים קשר חדש בין התקפות Padding Oracle לבין הבעיה הווקטורים - הווקטור הקרוב ביותר (CVP), אשר מאפשר לתוקף לפענח מפתח RSA בעל 2048 סיביות תחת הגבלת הזמן של 30 שניות שנאכף על ידי כמעט כל דפדפני האינטרנט.

ה. תרומת הפרויקט

הפרויקט מציג מימוש מלא של התקפת Bleichenbacher בעזרת התקפת F&R בשביל ליצור את ה-Oracle הנדרש להתקפה.
ההתקפה שלנו היא על שרת OpenSSL.

לשם סימולציית ההתקפה, יצרנו שרת TCP המשתמש ב-OpenSSL בשביל ביצוע הצפנה ופענוח,
בנוסף מימשנו לקוח עם סיסמא מוצפנת, והתוקף מנסה לפענח את הסיסמא ללא ידיעת המפתח ה-RSA הפרטי.

ובכך הפרויקט משיג הוכחת היתכנות שהתקפה המתוארת במאמר עובדת.

פערים שקיימים מהמימוש שלנו למאמר:

- במימוש שלנו תקפנו רק שרת אחד מתוך 9 השרתים שצוינו במאמר.
- לא מימשנו התקפת Beast על הלקוח מכוון שהקוד הזדוני שלנו כבר רץ במחשב של הלקוח.
- לא מימשנו MITM בין הלקוח לשרת כוון שהשרת הוגדל לעבוד רק עם הצפנת RSA ופרוטוקול ריפוד PKCS#1 v1.5 .
- לא מימשנו התקפה על מספר שרתים בו זמנית, כפי שהמאמר מחדש.
- ההתקפה שלנו לא הוגבלה ב-30 שניות להצלחה, אלה היא הוגבלה בפענוח נכון של בית אחד מתוך סיסמת הלקוח.

2. תיאור טכני

א. הנחות נדרשות לביצוע ההתקפה

להתקפה נדרשים שני מחשבים פיזיים שונים.
אחד בשביל להריץ שרת, אותו אנו תוקפים, למחשב זה נקרה "שרת".
השני בשביל להריץ את קוד הלקוח ואת קוד התוקף, למחשב זה נקרה "לקוח".

1. חומרתי:
לשרת אנו השתמשנו במחשב עם מעבד Intel i5-3570@3.40GHz, 6MB L3 Cache, 16GB Ram,
חשוב בשביל שההתקפה תעבוד שהמעבד יהיה Inclusive.

ללקוח אנו השתמשנו במחשב עם מעבד Intel i7-8550U@1.80GHz, 8MB L3 Cache, 16GB Ram,
אין הגבלות לגבי חומרתו.

2. תוכניתית:
בשרת השתמשנו במערכת הפעלה Ubuntu 18.04, כל מערכת הפעלה מבוססת הפצת Ubuntu יכולה לעבוד.
בנוסף, נדרש יכולת שיתוף זיכרון בין תהליכים בשביל לבצע F&R,
ניתן לבצע את אותה התקפה גם בעזרת Prime & Probe אבל בחרנו ב-F&R בשביל לקבל תוצאות טובות יותר.

בלקוח השתמשנו במערכת הפעלה Windows 10, שדרכה השתמשנו ב-WSL 1.0 בשביל לקבל גישה ל-Ubuntu 18.04. כל מערכת הפעלה מבוססת הפצת Ubuntu יכולה לעבוד,
ניתן גם להריץ את קוד הלקוח מתוך Virtual Machine ללא פגיע הצלחת ההתקפה, אולם ההתקפה תמשך יותר זמן.

ב. מרכיבי ההתקפה

אנחנו בחרנו לממש את התקפתנו בדומה לניסוי המוצג במאמר ב- Section V-A,
בו אנו תוקפים את שרת OpenSSL, את פונקציית RSA_padding_check_PKCS1_type_2 שלו.

1. **בשרת**:

- השתמשנו בגרסת 1.0.2 של OpenSSL ממרץ 2018
(אותה גרסה של OpenSSL מהזמן בו המאמר נכתב) - קישור.

בשביל למצוא את המקומות בהם אנחנו מעוניינים לבצע Monitor בעזרת התקפת ה-F&R הוספנו Symbols
במקומות המתאימים בקוד של OpenSSL, כלומר בתוך הפונקציה RSA_padding_check_PKCS1_type_2
לפני הקריאה ל-RSAerr הוספנו label בשם "probe1" ובשורה הראשונה של הפונקציה RSAerr הוספנו "probe2".
הקבצים הערוכים שלנו נמצאים ב-

```
code.zip/computer_2_server/openssl
```

בשביל להתקין אותם, יש למקם אותם בתיקייה OpenSSL שהורדנו מהקישור הנ"ל, בנתיבים הבאים:

```
openssl/crypto/rsa/rsa_pk1.c
openssl/crypto/err/err.c
```

בשביל להתקין את OpenSSL לאחר עריכת הקבצים יש להריץ את הפקודות הבאות מתוך תיקיית OpenSSL:

```
./config -d
make
make install
```

נשם לב שהוספנו "-d" בשביל שנוכל לראות את ה-debugging symbols בקובץ ELF שנוצר,
ובכך נוכל לראות את ה-labels שהוספנו.

- בשביל לבצע F&R השתמשנו בספריית Mastik – קישור.

בעזרת פונקציות הספרייה כתבנו בעצמו את קוד ה-Oracle הנמצא ב-

```
code.zip/computer_2_server/oracle/oracle.c
```

יש לשם אותו ב-

```
mastik/demo/oracle.c
```

בשביל להתקין את Mastik יש להריץ את הפקודות הבאות מתוך תיקיית Mastik:

```
sudo apt-get install -y binutils-dev
sudo apt-get install -y libdwarf-dev
./configure
make
```

בנוסף, יש צורך להגדיר HugePages על ידי הפקודה:

```
sudo sysctl -w vm.nr_hugepages=1024
```

נשם לב שקוד ה-Oracle מכיל מספר חלקים:

i. **חלק ראשון, שורות 111-131**: בו אנו מחפשים את ה-Offset של ה-Labels שהוספנו בתוך מרחב הזיכרון של OpenSSL, מידע זה ידוע לפי ה-ELF של OpenSSL, ניתן לראות זאת על ידי הרצת הפקודה:

```
readelf -Ws /usr/local/ssl/bin/openssl | grep probe
```

לאחר שה-Offset ידוע לנו, ניתן לשתף את הזיכרון של התוכנית שלנו (Oracle) עם OpenSSL,
זה מתבצע ע"י הפקודה map_offset של Mastik.

ii. **חלק שני, שורות 137-145**: כאן אנו מחכים ש-OpenSSL יתחיל לרוץ.

iii. **חלק שלישי, שורות 148-164**: פה אנו מבצעים את ה-F&R בעזרת הפונקציות של Mastik: clflush, delayloop, memaccesstime.
iv. **חלק רביעי, שורות 167-177**: כאן אנו קובעים אם התקבל Hit או Miss, על פי הניסוי במאמר הם קבעו Hit רק כאשר הם קיבלו Hit בכל המקומות עליהם הם ביצעו Monitor, אחרת Miss, וכך גם אנחנו עשינו.

הערה: אם OpenSSL הותקן במקום שונה ממיקום ברירת המחדל שלו, אז יש צורך לשנות את הפרמטרים PATH_TO_OPENSSL, BINARY_OPENSSL בהתאם.

- בשביל לדמות שרת המשתמש ב-OpenSSL להצפנת ופענוח כתבנו שרת בעצמנו, המשתמש ב-TCP Protocol בשביל התקשורת בינו לבין הלקוח.

קוד השרת נמצא ב-

code.zip/computer_2_server/server

בשביל להתקין את השרת יש לוודא ש-gcc מתוקן ואז לבצע make מתוך תיקיית השרת.
בשביל להתחיל להריץ את השרת (יש להתקין את OpenSSL לפני הרצת השרת) יש להריץ את הקובץ:

server/bin/server.bin

פונקציונליות שהשרת יכול לתת ללקוח:
i. **יצירת מפתח פרטי וציבורי של RSA** בעזרת OpenSSL.
ii. **קבלת טקסט והחזרת ההצפנה שלו** (בעזרת OpenSSL) – פונקציה זאת משמשת רק בשביל התחלת הניסוי בו אנו זרוקים לקובץ מוצפן (סיסמא של משתמש) ואותו ננסה לפענח (בעזרת התקפת Bleichenbacher).
iii. **פענוח טקסט מוצפן** – נשם לב שפונקציה זאת לא מחזירה את הפענוח (או את השגיאה אם הפענוח נכשל), אלה רק מריצה את קוד הפענוח של OpenSSL בלבד.
iv. **שולח ללקוח את ה-Modulus וה-Exponent של המפתח הציבורי**.
v. **יציאה וכיבוי השרת**.

פירוט מדויק עבור שימוש פרוטוקול התקשורת בין השרת ללקוח ניתן למצוא ב-

code.zip/computer_2_server/server/Protocol_Usage.txt

נשם לב שהשרת נכתב בצורה הכי "generic" בשביל לאפשר להריץ את השרת במחשבים אחרים,
לשם כך יש ערוך את הפרמטרים הבאים בהתאם:
DEFAULT_PORT – זהו הפורט שבו השרת ישתמש בשביל תקשורת ה-TCP שלו, אם הפורט המצוין לא פנוי, אז כאשר השרת יתחיל לרוץ הוא ימצא פורט אחר פנוי (והוא יודפס למסך).
PATH_TO_OPENSSL – אם OpenSSL לא הותקן במיקום הבירורת מחדל שלו.
RSA_MOD_BIT_SIZE – הצפנת RSA מאפשרת גדלים שונים עבור המפתח שלה (256/1024/2048/...)
הקוד שלנו מאפשר בחירת גודל המפתח הרצוי.
הפרמטרים הנ"ל נמצאים ב-

code.zip/computer_2_server/server/src/parameters.h

2. **בלקוח**:

- בשביל לבצע את ההתקפה, יצרנו סימולציה של לקוח ותוקף באותה קוד.

הקוד נמצא ב-

code.zip/computer_1_client/client

תהליך ריצת הקוד:
i. בקשה מהשרת להצפין את סיסמת הלקוח.
ii. הרצת התקפת Bleichenbacher, כאשר הוא משתמש ב-Oracle שכתבנו,
כלומר הלקוח שולח את טקסט מוצפן לשרת שוב ושוב, ובאותו הזמן הלקוח פונה לקוד ה-Oracle שיבצע F&R,
על פי תוצאת ה-Oracle הוא יכול לקבוע אם ההצפנה היא PKCS#1 v1.5 conforming
ובכך ליצר Oracle בהתאם לדרישות Bleichenbacher.
נשם לב, שלפי המאמר התבצעו 6 קרואות ל-Oracle ורק אם 5 מתוכם הם conforming אז אנחנו מחזירים conforming, אחרת לא. המימוש שלנו עובד בצורה זאת גם כן.

iii. הדפסת הטקסט המפוענח.

בשביל להריץ את הקוד נדרש Python 3.

כדי להריץ את הקוד יש להריץ את הפקודה הבא מתוך תיקייה ה-client:

```
python3 src/main.py
```

נשם לב שגם קוד הלקוח נכתב בצורה "generic" ולכן יש לתאם את הפרמטרים שלו גם כן:
modulus_size – גודל המפתח של RSA, צריך להיות אותו גודל כמו בשרת.
host – כתובת ה-IP של השרת.
port – הפורט עליו השרת מאזין.
הפרמטרים הנ"ל נמצאים ב-

```
code.zip/computer_1_client/client/src/param.py
```

- בשביל שהלקוח יוכל לתקשר עם ה-Oracle שנמצא בשרת,
כתבנו Script קצת המשתמש ב-SSH בשביל לשלוח פקודה שתריץ את ה-Oracle.

ה-Script נמצא ב-

```
code.zip/computer_1_client/client/script/call_oracle.sh
```

יש לשם לב, לשנות את פרטי ה-SSH בשביל שה-Script ירוץ
(כתובת ה-IP של השרת, המשתמש שאיתו מתחברים לשרת, הנתיב שבו נמצא ה-Oracle).

ג. מהלך ההתקפה

אנו מניחים שכל ההתקנות מסעיף ב' הותקנו.

ראשית, יש להדליק את השרת, כלומר להריץ:

```
./server/bin/server.bin
```

לשם לב למספר ה-Port שיודפס למסך, לדוגמא 4430.
אם ה-Port שונה מה-Port שצוין בפרמטרים של הלקוח אז לעדכן אותו.

שנית, להריץ את הלקוח, כלומר להריץ:

```
python3 client/src/main.py
```

כעט, ההתקפה מתבצעת, במסך השרת ניתן לראות את הפלט של OpenSSL
ואת הדפסות מהשרת שלנו (שאומרות מה הוא עושה בכל רגע).
במסך הלקוח, כל מספר queries יודפס למסך כמה זמן עבר מתחילת ההתקפה ולבסוף יודפס כל פרטי ההתקפה,
כלומר הסיסמא המקורית, המפתח המוצן שהתוקף מנסה לפענח, הפענוח של התוקף, ומספר בטים בהם הפענוח הצליחה.

ברגע שהתקפה תסתיים הלקוח יכבה את השרת.

3. מדדי הצלחה

א. צורת המדידה

הקוד של הלקוח מדפיס כמה זמן לוקח לכל query לרוץ, ובכך אנחנו יכולים לדעת כמה זמן לקח לכל query בנפרד, כמה זמן לקחה כל התקפה וכמה queries בסה"כ נדרשו להתקפה.

ב. מדידת תוצאות הפרויקט

את ההתקפה הרצנו במספר מצבים שונים:

1. **שינוי מספר ה-clock cycles בין שלב ה-Flush ושלב ה-Reload ב-Oracle**.
בניסויינו בדקנו כאשר ה-delay הוא 500 או 1000 clock cycles.
לממצאנו כמות הזמן עבור כל query הייתה מהירה יותר כאשר הקטנו את ה-delay
אך לא ראיינו שינוי בסיכוי הצלחה של הניסוי.
2. **שינוי במספר הסיביות במפחת ה-RSA**.
בניסויינו בדקנו עבור 256 סיביות ו-2048 סיביות.
עבור 256 סיביות, ראיינו שנדרשים כ-230 queries לפענוח, ההתקפה נמשכה בין 5 עד 15 דקות כתלות בשאר השינויים.
עבור 2048 סיביות, ראיינו שנדרשים כ-2100 queries לפענוח, ההתקפה נמשכה בין שעה לשעתיים כתלות בשאר השינויים.
3. **כאשר מתקבל מה-Oracle שהטקסט המוצפן הוא conforming,**
יש לבדוק מספר פעמים נוספים כדי לוודא שזה נכון.
בניסויינו בדקנו עבור 6 בדיקות (כמו במאמר) ו-10 בדיקות.
מספר ה-queries לא משתנה על פי השינוי הזה,
אך הגדלת מספר הבדיקות מגדיל משמעותית את כמות הזמן עבור כל query.
על פי הבדיקות שלנו, כאשר אנו מבצעים 6 בדיקות, לכל query לוקח כ-10 עד 15 שניות.
כאשר מספר הבדיקות הוא 10 אז כל query לוקח כ-20 עד 25 שניות.
העלאת מספר הבדיקות כן השפיע על הסיכוי ההצלחה של הניסוי לטובה
(כמות הבתים שבהם הסיסמא המקורית שווה לסיסמא המפוענחת ע"פ ההתקפה).
נציין גם כן, שניסינו לבצע את הניסוי עם 20 בדיקות,
אך כמות השניות עבור כל query עלה ל-40 שניות מה שגרם לריצת הניסוי להיות ארוכה מדי.

בתור הוכחת היתכנות, צירפנו את תוצאות ריצה בא קיבלנו פענוח נכון של שני בתים של הסיסמא.
קבצים האלו נמצאים בתיקיית `proof_of_concept`.
בתיקייה זאת נמצאים המפתח הפרטי והציבורי שהשתמשנו בהם באותו ניסוי,
קובץ טקסט עם המייצג את הסיסמא (לא מוצפנת), קובץ מוצפן של הסיסמא (שאותו אנו מנסים לפענח),
קובץ של הפענוח שהתקבל מהתקפה, פלט הסופי של התוכנית שמראה שקיימים שני בתים נכונים והפלט המלא של הלקוח.

4. References

a. Code that we use

(1) Implementation of the Bleichenbacher attack – link.
(2) Basic TCP server in C – link.
(3) Snippets of code taken from StackOverflow (The place they being used can be found in the code in comments start with "Credit: <link>") – link1, link2, link3, link4, link5, link6, link7.

b. From the article

[1] "The ICSI Notary," http://notary.icsi.berkeley.edu/#connection-cipherdetails.
[2] O. Aciiç¸mez, "Yet another microarchitectural attack: Exploiting ICache," in CSAW, 2007.
[3] O. Aciiç¸mez, S. Gueron, and J. Seifert, "New branch prediction vulnerabilities in OpenSSL and necessary software countermeasures," in IMA Int. Conf., 2007.
[4] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thom´e, L. Valenta, B. VanderSloot, E. Wustrow, S. Z. B´eguelin, and P. Zimmermann, "Imperfect forward secrecy: How Diffie-Hellman fails in practice," in CCS, 2015.
[5] N. J. AlFardan and K. G. Paterson, "Lucky thirteen: Breaking the TLS and DTLS record protocols," in IEEE SP, 2013, pp. 526–540.
[6] T. Allan, B. B. Brumley, K. E. Falkner, J. van de Pol, and Y. Yarom, "Amplifying side channels through performance degradation," in ACSAC, 2016.
[7] R. Bardou, R. Focardi, Y. Kawamoto, L. Simionato, G. Steel, and J. Tsay, "Efficient padding oracle attacks on cryptographic hardware," in CRYPTO, 2012.
[8] M. Ben-Or, B. Chor, and A. Shamir, "On the cryptographic security of single RSA bits," in STOC, 1983.
[9] N. Benger, J. van de Pol, N. P. Smart, and Y. Yarom, ""Ooh aah... just a little bit" : A small amount of side channel can go a long way," in CHES, 2014.
[10] D. J. Bernstein, "Cache-timing attacks on AES," 2005.
[11] D. Bleichenbacher, "Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1," in CRYPTO, 1998.
[12] H. B¨ock, J. Somorovsky, and C. Young, "Return of Bleichenbacher's oracle threat (ROBOT)," in USENIX Sec, 2018.
[13] D. Boneh and R. Venkatesan, "Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes," in CRYPTO, 1996.
[14] F. Brasser, U. M¨uller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A. Sadeghi, "Software grand exposure: SGX cache attacks are practical," in WOOT, 2017.
[15] B. B. Brumley and N. Tuveri, "Remote timing attacks are still practical," in ESORICS, 2011.
[16] J. V. Bulck, F. Piessens, and R. Strackx, "SGX-Step: A practical attack framework for precise enclave execution control," in SysTEX@SOSP, 2017.
[17] ——, "Nemesis: Studying microarchitectural timing leaks in rudimentary CPU interrupt logic," in CCS, 2018.
[18] S. Checkoway, J. Maskiewicz, C. Garman, J. Fried, S. Cohney, M. Green, N. Heninger, R.-P. Weinmann, E. Rescorla, and H. Shacham, "A systematic analysis of the Juniper Dual EC incident," in CCS, 2016.
[19] T. Dierks and C. Allen, "The TLS Protocol Version 1.0," RFC 2246, Jan. 1999.
[20] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1," RFC 4346, Apr. 2006.
[21] ——, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246, Aug. 2008.
[22] C. Disselkoen, D. Kohlbrenner, L. Porter, and D. M. Tullsen, "Prime+Abort: A timer-free high-precision L3 cache attack using intel TSX," in USENIX Sec, 2017.
[23] T. Duong and J. Rizzo, "Here come the ⊕ ninjas," 2011.
[24] D. Evtyushkin, D. Ponomarev, and N. B. Abu-Ghazaleh, "Understanding and mitigating covert channels through branch predictors," TACO, vol. 13, no. 1, 2016.
[25] D. Evtyushkin, D. V. Ponomarev, and N. B. Abu-Ghazaleh, "Jump over ASLR: attacking branch predictors to bypass ASLR," in MICRO, 2016.
[26] D. Evtyushkin, R. Riley, N. B. Abu-Ghazaleh, and D. Ponomarev, "BranchScope: A new side-channel attack on directional branch predictor," in ASPLOS, 2018.
[27] Q. Ge, Y. Yarom, D. Cock, and G. Heiser, "A survey of microarchitectural timing attacks and countermeasures on contemporary hardware," J. Cryptographic Engineering, vol. 8, no. 1, 2018.
[28] Q. Ge, Y. Yarom, and G. Heiser, "No security without time protection: We need a new hardware-software contract," in APSys, Aug. 2018.
[29] D. Genkin, L. Pachmanov, I. Pipman, E. Tromer, and Y. Yarom, "ECDSA key extraction from mobile devices via nonintrusive physical side channels," in CCS, 2016.

[30] D. Genkin, L. Valenta, and Y. Yarom, "May the fourth be with you: A microarchitectural side channel attack on several real-world applications of Curve25519," in CCS, 2017.
[31] D. Genkin, L. Pachmanov, E. Tromer, and Y. Yarom, "Drive-by keyextraction cache attacks from portable code," in ACNS, 2018.
[32] D. Gruss, R. Spreitzer, and S. Mangard, "Cache template attacks: Automating attacks on inclusive last-level caches," in USENIX Sec, 2015.
[33] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+Flush: A fast and stealthy cache attack," in DIMVA, 2016.
[34] N. Howgrave-Graham and N. P. Smart, "Lattice attacks on digital signature schemes," Des. Codes Cryptography, vol. 23, no. 3, 2001.
[35] M. S. Inci, B. G¨ulmezoglu, T. Eisenbarth, and B. Sunar, "Co-location detection on the cloud," in COSADE, 2016.
[36] M. S. Inci, B. G¨ulmezoglu, G. Irazoqui, T. Eisenbarth, and B. Sunar, "Cache attacks enable bulk key recovery on the cloud," in CHES, 2016.
[37] Intel, "Speculative execution side channel mitigations," https://software.intel.com/sites/default/files/managed/c5/63/336996-Speculative-Execution-Side-Channel-Mitigations.pdf, May 2018.
[38] G. Irazoqui, T. Eisenbarth, and B. Sunar, "S$A: A shared cache attack that works across cores and defies VM sandboxing - and its application to AES," in IEEE SP, 2015, pp. 591–604.
[39] G. Irazoqui, M. S. Inci, T. Eisenbarth, and B. Sunar, "Lucky 13 strikes back," in ASIA CCS, 2015.
[40] T. Jager, S. Schinzel, and J. Somorovsky, "Bleichenbacher's attack strikes again: Breaking PKCS#1 v1.5 in XML encryption," in ESORICS, 2012.
[41] T. Jager, J. Schwenk, and J. Somorovsky, "On the security of TLS 1.3 and QUIC against weaknesses in PKCS#1 v1.5 encryption," in CCS, 2015.
[42] V. Kl´ıma, O. Pokorn´y, and T. Rosa, "Attacking RSA-based sessions in SSL/TLS," in CHES, 2003.
[43] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Haburg, M. Lipp, S. Mangard, T. Prescher, M. Schwartz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in IEEE SP, 2019.
[44] S. Lee, M. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado, "Inferring fine-grained control flow inside SGX enclaves with branch shadowing," in USENIX Sec, 2017.
[45] A. K. Lenstra, H. W. Lenstra, and L. Lov´asz, "Factoring polynomials with rational coefficients," Mathematische Annalen, vol. 261, no. 4, 1982.
[46] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in IEEE SP, 2015.
[47] J. Manger, "A chosen ciphertext attack on RSA optimal asymmetric encryption padding (OAEP) as standardized in PKCS #1 v2.0," in CRYPTO, 2001.
[48] C. Meyer, J. Somorovsky, E. Weiss, J. Schwenk, S. Schinzel, and E. Tews, "Revisiting SSL/TLS implementations: New Bleichenbacher side channels and attacks," in USENIX Sec, 2014.
[49] D. Migault and I. Boureanu, "LURK extension version 1 for (D)TLS 1.2 authentication," IETF, Internet-Draft draft-mglt-lurk-tls12-01, 2018.
[50] A. Moghimi, G. Irazoqui, and T. Eisenbarth, "CacheZoom: How SGX amplifies the power of cache attacks," in CHES, 2017.
[51] Mozilla, "SSL handshake key exchange algorithm for full handshake," https://mzl.la/2BQjcMO.
[52] P. Q. Nguyen, Public-Key Cryptanalysis, ser. Contemporary Mathematics. AMS–RSME, 2009, vol. 477.
[53] P. Q. Nguyen and I. E. Shparlinski, "The insecurity of the Digital Signature Algorithm with partially known nonces," J. Cryptology, vol. 15, no. 3, 2002.
[54] OpenSSL, "RSA public encrypt," https://www.openssl.org/docs/man1.0.2/crypto/RSA private decrypt.html.
[55] Y. Oren, V. P. Kemerlis, S. Sethumadhavan, and A. D. Keromytis, "The spy in the sandbox: Practical cache attacks in JavaScript and their implications," in CCS, 2015.
[56] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: The case of AES," in CT-RSA, 2006.
[57] C. Percival, "Cache missing for fun and profit," in Proceedings of BSDCan, 2005.
[58] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in CCS, 2009.
[59] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," Commun. ACM, vol. 21, no. 2, 1978.
[60] E. Ronen, K. G. Paterson, and A. Shamir, "Pseudo constant time implementations of TLS are only pseudo secure," in CCS, 2018.
[61] PKCS #1 v2.2: RSA Cryptography Standard, RSA Laboratories, 2012.
[62] S. Schmidt, "Introducing s2n, a new open source tls implementation," https://aws.amazon.com/blogs/security/introducing-s2n-a-newopen-source-tls-implementation/, 2015.
[63] The Sage Developers, SageMath, the Sage Mathematics Software System (Version 8.3), www.sagemath.org, 2018.
[64] Y. Tsunoo, T. Saito, T. Suzaki, M. Shigeri, and H. Miyauchi, "Cryptanalysis of DES implemented on computers with cache," in CHES, 2003.

[65] V. Varadarajan, Y. Zhang, T. Ristenpart, and M. M. Swift, "A placement vulnerability study in multi-tenant public clouds," in USENIX Sec, 2015.
[66] Y. Xiao, M. Li, S. Chen, and Y. Zhang, "STACCO: differentially analyzing side-channel traces for detecting SSL/TLS vulnerabilities in secure enclaves," in CCS, 2017.
[67] M. Yan, C. W. Fletcher, and J. Torrellas, "Cache telepathy: Leveraging shared resource attacks to learn DNN architectures," CoRR, vol. abs/1808.04761, 2018.
[68] Y. Yarom, "Mastik: A micro-architectural side-channel toolkit," cs.adelaide.edu.au/~yval/Mastik/Mastik.pdf, 2017.
[69] Y. Yarom and K. Falkner, "FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack," in USENIX Sec, 2014.
[70] X. Zhang, Y. Xiao, and Y. Zhang, "Return-oriented Flush-Reload side channels on ARM and their implications for Android devices," in CCS, 2016.
[71] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-tenant sidechannel attacks in PaaS clouds," in CCS, 2014.
[72] ——, "Cross-tenant side-channel attacks in PaaS clouds," in CCS, 2014.