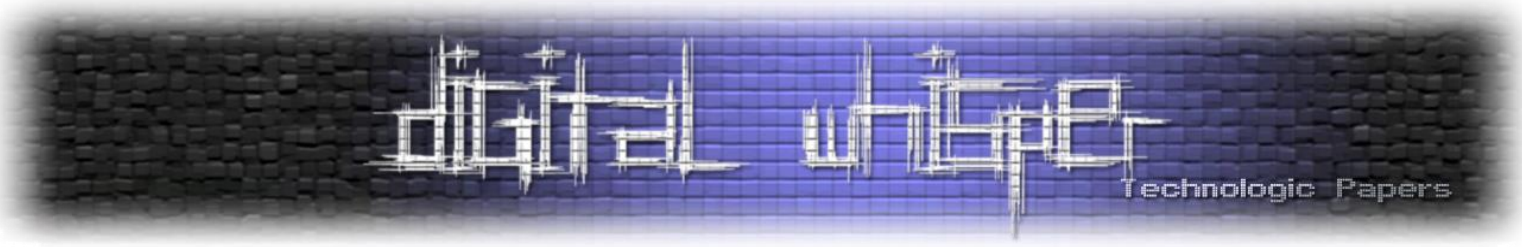

Java Script Security

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

האפליקציה שלך פגיעה - שלא באשמתך

אפתח בתרחיש המפחיד הבא: מתכנתים שעברו הכשרה סטנדרטית בכתיבת קוד מאובטח, יודעים ומבצעים Input validation ו-Output encoding ממש לפי הכללים, ועם כל זאת, האתר או האפליקציה שלהם פגיעים ל-Cross Site Scripting שהינה מתקפה חמורה.

מעבר לכך, אפליקציה שעד היום הייתה מוגנת לחלוטין, מספיק שנשייך אליה ספריית JS מסויימת, ובלי שאפילו שהשתמשנו בה עדיין, האפליקציה פגיעה. על כך ועוד, במאמר שלפניכם.

יציין שהתכנים המובאים כאן הינם רק "טעימות" קטנטנות ממספר קורסים (פיתוח מאובטח ב-HTML5, PhoneGap, AngularJS ו-NodeJS) המועברים ע"י חברת AppSec Labs[1] ובאדיבותה.

הקדמה - השתלטות Java Script בחיינו

היקף קוד ה-Java Script שנכתב גודל משנה לשנה, ועל פי tiobe[2], היא נמצאת ברשימת עשרת השפות הנפוצות ביותר. היא התחילה כשפת עזר לאתרי אינטרנט, כשבפועל כל הלוגיקה נכתבה בצד השרת (ASP, C++ וכו'), ועם השנים גם הלוגיקה עברה לקליינט - לדפדפן, או במילים אחרות - ל-JS. תקן HTML5 הוסיף יכולות נוספות לשפה, ואיפשר לבצע פעולות שעד היום כלל לא היו לדפדפן, כמו קבלת GEO Location, Offline storages ועוד.

HTML5 והתקדמות הסמארטפונים פתחו את העולם להתאמת אתרים למובייל. כאשר במקום להעסיק צוות מתכנתים לכל פלטפורמה, אפשר לתת לצוות ה-Web שפיתח עד היום, להתאים את האתר ל-Mobile ולחסוך בזמן ומשאבי פיתוח. נוסיף לכך מערכות סטייל PhoneGap שנותנות לאפליקציה שלנו עוד יכולות על המכשיר, באמצעות Java Script ונגלה שכיום המון קוד נכתב בשפה הזו.

[1] https://appsec-labs.com/
[2] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

אבל זה לא רק אתרי אינטרנט ומובייל. גם תוספים לדפדפנים נכתבים ב-Java Script, Widgets של Win7, אפליקציות Metro של Win8, SSB גם Widgets של Win7, SSB[3] (Site-Specific Browser) שבה ממירים אתר אינטרנט לאפליקציית Desktop.

אולם הכי מסוכן וחשוב, זו הזליגה של JS לשרתים, פריימוורקים כדוגמת NodeJS שמריצים Java Script כשפת צד-שרת. נרחיב על הנושא בהמשך.

Angular JS - DOM Based XSS

כיום, כל אתר אינטרנט מודרני משתמש לפחות בספריית JS אחת. הספרייה הבסיסית היא בדר"כ JQUERY. ספרייה נוספת שמתפתחת בקצב מהיר הינה AngularJS[4]. ספרייה של גוגל, שעוזרת לרנדר ולשלוט בתוכן הדפים בקלות.

מה שהשפה הזו נותנת מעבר למה שהכרנו עד כה, הינו רנדור של כל תוכן הדף בזמן הטעינה. לדוגמא, ניתן לראות קוד שנראה כך:

```
<h1>{title}</h1>
```

אנחנו רואים תגית פתיחה וסגירה מסוג h1 וביניהן Place Holder שמפנה ל-Model בשם title. ברגע שערך המודל ישתנה, תוכן התגית יתעדכן.

כעת נראה קוד פשוט של אתר אינטרנט ב-PHP שמבצע Output encoding נגד XSS (Cross Site Scripting), באמצעות הפונקציה htmlspecialchars[5], ממש "By the book":

```
<?php
      $description = htmlspecialchars($_GET[' description']);
      echo "<div>$description</div>";
?>
```

בקוד הזה, המתכנת מכניס קלט מפרמטר בשורת הכתובת בשם Description (כגון victim-site.com/x.php?description=abc) לתוך הפונקציה htmlspecialchars ומשם לתוך משתנה בשם description. לאחר מכן היא מדפיסה לדפדפן את תוכן המשתנה בין תגיות div.

[3] http://en.wikipedia.org/wiki/Site-specific_browser
[4] http://angularjs.org/
[5] http://php.net/manual/en/function.htmlspecialchars.php

תשאלו כל איש אבטחת מידע רגיל, והוא יגיד שהקוד כתוב נכון והינו מאובטח, כיוון שהפונקציה htmlspecialchars מקודדת כל תו מהתווים & " < > בקידוד מתאים והדפדפן יידע להציג אותם ולא להריץ אותם.

אך מתברר שאם לדף מקושרת הספריה AngularJS (גרסה עדכנית!), בפעם אחת הקוד נהפך לבעייתי. כמו שראינו, אנגולר יודע לרנדר חלקים מהדף באמצעות { } וללא התגיות הקלאסיות < >. זה כבר רמז לבעיה... ואכן נמצא שיש לא מעט אפשרויות לנצל זאת ולהגיע להרצת קוד.

אם להשתמש בלינק שבדוגמא הקודמת, מה שהתוקף יעשה, יהיה לשלוח למשתמש את הלינק הבא:

```
victim-
site.com/x.php?description={{''.toString.constructor('alert(1)')()}}
```

התוצאה תהיה מודעת Alert שתקפוץ למשתמש. דוגמא זו נמצאה על ידי Alex Kouzemtchenko[6].

מעט הסבר על הניצול: המתודה constructor של כל פונקציה מקבלת כפרמטר קוד ומייצרת בתשובה function() { your_code }. כאן השתמשנו בסטרינג ריק " כדי לשרשר לו את המתודה toString של-constructor שלה פנינו והעברנו לו את הקוד alert(1). ה-Response של הקוד הזה הוא מצביע לקוד:

```
function anonymous() { alert(1) }
```

כעת נוסיף לכך () שיגרום לקוד הזה לרוץ. ונעטוף ב {{ }} שיגרמו לספריית Angular להריץ אותו:

```
{{''.toString.constructor('alert(1)')()}}
```

הבעיה הזו קיימת גם בגרסה הכי עדכנית של AngularJS. פרוייקט מיוחד[7] נפתח בגוגל קוד כדי לסכם הן את הבעיות הקיימות ב-AngularJS והן בעיות שקיימות בספריות JS אחרות.

DOM Based XSS

בנוסף למקרה הנפוץ, שבו אנחנו מדפיסים קלט (כגון window.name שניתן לשליטה ע"י הכנסה של הדף ל-Iframe) ללא קידוד מתאים, אציין דוגמאות שבהן דף אינטרנט לוקח קלט ומריץ אותו כ-JS. הסיכון ברור, התוקף ינסה לשלוט בקלט ולהוסיף לשם פקודות שיקדמו את מטרתו.

[6] https://communities.coverity.com/blogs/security/2013/04/23/angular-template-injection
[7] https://code.google.com/p/mustache-security/

נתחיל מהמקרה הפשוט - הרצה באמצעות eval:

```
eval(jsCode + controlledInput)
```

דף אינטרנט יכול להריץ String כ-JS (ביודעין או בלא תשומת לב) גם באמצעות תזמון:

```
setInterval(jsCode + controlledInput, timeMs)
setTimeout(jsCode + controlledInput, timeMs)
```

והמקרה הקלאסי האחרון, באמצעות events:

```
tag.onmouseover = jsCode + controlledInput
```

כשבוחנים לעומק מגלים שיש מספר מקרים מעט נדירים, אבל עדיין אפשריים. נתחיל משינוי המקור (source) של תגית סקריפט:

```
script.src = controlledInput
```

במרבית הדפדפנים ניתן גם להריץ סקריפט באמצעות תגית סקריפט עם המתודות ,text, .textContent. innerText.

מקרה מעניין יותר הוא המקרה הבא (F קפיטליסטית):

```
Function (jsCode + controlledInput)
```

ולקינוח נסיים במשהו שעובד רק על גרסאות IE שמספרן בין 6 ל-11:

```
execScript(jsCode + controlledInput, "JScript")
```

אפשר לראות סיכום יפה של המקרים + דוגמאות נוספות, בפרוייקט ייעודי[8] שהוקם אף הוא בגוגל קוד.

תוספים והרחבות של דפדפנים

אחרי שהבנו ש-JS יכול להיות מסוכן, ננסה לכוון גבוה על מנת לנצל את מקסימום הנזק. בגישה הוותיקה שבה JS רץ בקליינט, המטרה שלנו היא לפגוע בכמה שיותר משתמשים ו/או בכמה שיותר אתרים. אם כן, הדבר המתאים ביותר הוא לנצל בעיית אבטחה בתוספים של דפדפנים. תוסף של דפדפן רץ בדר"כ על כל האתרים, ואם נצליח לזהם את אחד המקורות שלו (Local storage מאיזשהו סוג), או לפחות לגרום לו להריץ קוד במיידי, נוכל להריץ JS על כל האתרים.

[8] https://code.google.com/p/domxsswiki/wiki/ExecutionSinks

פלוס נוסף שיש לתוסף, שהוא רץ עם הרשאות גבוהות יותר משרץ JS באתר. כך לדוגמא הוא יכול לקרוא Cookies שהם HTTPONLY.

כאן המקום להפנות למחקר רחב שבוצע ב-2010 ע"י עמנואל בורנשטיין ופורסם (בלעדית!) ב-Digital[9] לגבי ניצול הרחבות של Firefox.

אציין, שבגוגל כרום:

1. ניתן באמצעות JS, ממש בקלות לדעת אם תוסף מותקן בדפדפן, וכך לבצע מתקפה ממוקדת תוסף (מקור[10]):

```
var detect = function(base, if_installed, if_not_installed) {
      var s = document.createElement('script');
      s.onerror = if_not_installed;
      s.onload = if_installed;
      document.body.appendChild(s);
      s.src = base + '/manifest.json';
}
detect('chrome-extension://' + addon_id_youre_after, function()
{alert('boom!');});
```

2. כאמצעי הגנה גוגל דורשים ומכריחים תוספים לרוץ עם CSP, כך שקוד JS לא יכול לרוץ inline, לא יכול לרוץ ישירות מ-Remote, ואחרון חביב - לא יכול לקבל לתוך eval סטרינגים (מחרוזות) (מקור[11]).

JS on server side

אחרי שהצצנו בקליינט, בואו נראה מה קורה בשרת. בשלוש שנים האחרונות החלה נהירה לכיוון NodeJS. דרך העבודה של NodeJS היא א-סינכרונית, כך שלדוגמא כשאנחנו מבצעים פניה ל-DB, אנחנו/השרת לא ממתינים לתשובה, אלא מגדירים CallBack ובינתיים השרת פנוי לטפל בבקשות אחרות. כשתגיע התשובה - היא תמופה ל-CallBack.

הנקודה החשובה עבורנו, זה שכמו שהשם מרמז NodeJS למעשה בנוי ונכתב ב-JS. כך שאותן בעיות שראינו קודם לכן, החל כמובן ב-XSS לסוגיו השונים הפשוטים, המשך בהרצה דינמית של סקריפט באמצעות eval ודומיו עד לבעיות שינבעו עקב השימוש בספריות. אנחנו צפויים לראות גם ב-NodeJS.

[9] http://www.digitalwhisper.co.il/files/Zines/0x0F/DW15-4-FFExtsploits.pdf
[10] http://blog.kotowicz.net/2012/02/intro-to-chrome-addons-hacking.html
[11] http://developer.chrome.com/extensions/contentSecurityPolicy.html

כמובן שניתן לכתוב דינמית בכל שפה אם ממש מתעקשים, אבל ללא ספק - ב-JS ניתן לראות זאת ביתר שאת. אז מה בכל זאת התחדש כאן? שהמתקפה לא רצה על הדפדפן של המשתמש אלא ישירות בשרת. ולמעשה מדובר על השגת שליטה בשרת עצמו. מה שמעלה את אפשרויות ה-Exploitation, הסיכון וכו'.

JS on iOS7

לסיום, אציין ש-iOS7 מכילה תמיכה[12] בכתיבת אפליקציות Native ב-JS. זה אכן פתח רציני לבעיות אבטחה ואני משער שעוד נשמע על כך.

מקורות

- https://appsec-labs.com/
- http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html
- http://en.wikipedia.org/wiki/Site-specific_browser
- http://angularjs.org/
- http://php.net/manual/en/function.htmlspecialchars.php
- https://communities.coverity.com/blogs/security/2013/04/23/angular-template-injection
- https://code.google.com/p/mustache-security/
- https://code.google.com/p/domxsswiki/wiki/ExecutionSinks
- http://www.digitalwhisper.co.il/files/Zines/0x0F/DW15-4-FFExtsploits.pdf
- http://blog.kotowicz.net/2012/02/intro-to-chrome-addons-hacking.html
- http://developer.chrome.com/extensions/contentSecurityPolicy.html
- http://strongloop.com/strongblog/apples-ios7-native-javascript-bridge/

[12] http://strongloop.com/strongblog/apples-ios7-native-javascript-bridge/

לסיכום

במאמר זה סקרתי בקצרה על השימושים השונים של JS. ראינו מעט סיכונים וניצולים אפשריים, כולל כאלה שבאים "באשמת" ספריות חיצוניות, המאפשרים לתוקף להשיג שליטה ולהריץ פקודות JS בהתאם לבחירה שלו.

אשמח לקבל פידבק (Israel@appsec-labs.com), תודה מראש. המאמר נתרם כדי לחזק את הקהילה הישראלית ובעברית (אפיק לא הסכים שאכתוב באנגלית!).

ישראל חורז'בסקי [Sro.co.il]

Tech Leader ב-AppSec-Labs.