digital whisper
Technologic Papers

HTML5 - מנקודת מבט אחרת

מאת: לירן בנודיס ואלעד גבאי

הקדמה

בזמן האחרון אנו שומעים עוד ועוד על HTML5 - יש הגורסים כי זוהי הטכנולוגיה שתשנה את פני הרשת (ואף תהווה תחליף לטכנולוגיות פיתוח רבות), אחרים מתעסקים בגישות העיצוב החדשות שטכנולוגיה זו מאפשרת ואחרים מתעסקים בשאלה האם HTML5 תחליף את Flash של Adobe או Silverlight של מיקרוסופט.

על אף השפע העיצובי והטכנולוגי ש-HTML5 מביאה איתה, ישנם לא מעט אנשים שלא מדברים על אפשרויות העיצוב החדשות שטומנת בתוכה HTML5, לא מתעסקים בדיונים על אפשרויות לשדרג אתרים קיימים בעזרתה ואפילו לא על אפשרויות השיווק החדשות. אנשים אלו, שהתעמקו ב-HTML5 והטכנולוגיות הנלוות, רואים סיבות לדאגה - HTML5 טומנת בתוכה, בין שאר הירקות, הרבה בעיות פרטיות ואבטחת מידע.

בסדרת מאמרים הבאים, נסקור את הנקודות הבעייתיות ב HTML5 ובטכנולוגית הנלוות. מאמר זה יסקור את התגים והתכונות החדשות ב-HTML5, את מנגנון ה-XDR/CORS וכמובן ידון בהשלכות של אלו על מידת האבטחה של התקן החדש.

יש לזכור כי כרגע HTML5 הינו תקן בפיתוח ולכן חלק מדוגמאות הקוד המובאות במאמר זה לא יעבדו בכל הדפדפנים.

HTML (HyperText Markup Language)

HTML היא שפת תגיות המיועדת להצגת תוכן בעמודי אינטרנט. זו שפה קלה יחסית ללמידה וניתן להציג להציג בעזרתה אינסוף תכנים באינסוף צורות שונות. את תקן השפה קובע ארגון ה-W3C (ארגון תקינת הרשת העולמי).

גרסת ה-HTML המוכרת והיציבה, אשר נקבעה ב-1999 ולא שונתה מאז, הינה HTML4.01, אליה מתלווים CSS, JavaScript ו-HTML DOM, ויוצרים את השפה הדינמית DHTML. מאז, עולם המחשוב כמו גם האינטרנט השתנו רבות, ויש צורך לענות על צרכי מפתחי האתרים שתמיד דורשים יותר ויותר, וכמו כן גם על צרכי הקהל אשר צורך את המידע ותמיד דורש אפשרויות נוספות, ובכל פעם צורך את המידע בצורה חדשה ובפלטפורמות שונות (פלאפונים, מכשירי Tab, טלוויזיות, windows8, טוסטר משולשים, ועוד...). לפיכך, גבר הצורך בעדכון שפת HTML.

הדור הבא של התקן הינו HTML5, שגרסת התקן הראשונה שלו יצאה ב-2008 ועדיין נמצא בפיתוח.

כיום, כאשר דרישת הקהל, ובעקבותיו דרישת המפתחים, לפיצ'רים חדשים ופונקציות נוספות היא מיידית, לא תמיד ניתן לספק דרישה זו בזמן. דוגמה חייה היא HTML5. לאחרונה ישנם הרבה אתרי אינטרנט המשתמשים בתקן ה-HTML5. תקן זה עוד נמצא בפיתוח ועתיד לצאת רשמית רק ב-2014!

העניין שלנו ב-HTML נובע בדיוק מסיבה זו. כיום נוצר מצב שהתקן של HTML5 לא קבוע ומשתנה לעתים תכופות.

רוב הדפדפנים הפופולריים כבר תומכים בחלק מן האלמנטים והפיצ'רים החדשים שמציע התקן, כל דפדפן על פי בחירתו ובצורת מימוש שונה. בנוסף על כל זה, יש את עניין הפיצ'רים הנוספים של HTML5, שעצם הוספתם של חלק גדול מהם יוצר בעיות אבטחה.

התגיות החדשות

ב-HTML5 ישנן-28 תגיות חדשות, ניתן לסווגן לסוגים:

תגיות חלוקת הדף וטקסט:

התגיות הללו לא משנות או מבצעות שום דבר בנוגע לעיצוב, אלא רק נותנות סמנטיקה עבור התוכן אותו הן מייצגות. סמנטיקה זו עוזרת מאוד למנועי חיפוש ורובוטים.

חלוקה לדוגמה של הפוסט האחרון ב-DigitalWhisper בעזרת התגים החדשים:

כמו שרואים בתמונה, תג ה-header מסמן את ראש העמוד, בעוד שתג ה-footer מסמן את סופו (לא עוד תג div עם id המתאר את משמעות התוכן). בנוסף, ניתן לראות כי קיים תג article, המכיל את תוכן המאמר, תג nav המכיל את תפריט הניווט של האתר, ותג aside אשר מכיל את התפריט הצדדי, כאשר בתוכו תגי section המגדירים את החלקים השונים בתפריט.

לא נפרט יותר מידי על תגים אלו במאמר זה, אך אלו מכם המתעניינים יכולים למצוא יותר מידע בקישורים בסוף הפרק.

תגיות מולטימדיה:

HTML5 מאפשרת מספר תגי מולטימדיה המשמשים להטמעת קול, סרטונים ופלאגינים בעמוד.

תג הווידאו והאודיו מאפשרים לנו להציג תוכן ממקור (origin) אחר, וזה יאפשר לנו להציג סרטונים אהובים מאתרים אחרים ללא כל צורך לאחסן את אותם הסרטים בשרתים שלנו, או להשמיע רשימות מוזיקה שלמות ומתעדכנות מאתרי מוזיקה מובילים. אפשרות זאת יוצרת בעיית אבטחה משני כיוונים:

כאשר אתר תמים מציג תוכן זדוני, וכאשר אתר זדוני מציג תוכן תמים.

מצד אחד, כאשר עמוד תמים מציג תוכן זדוני, החשש העיקרי הוא שמקור המדיה הזדוני עלול להכיל סקריפט כלשהו אשר ינסה לבצע אינטראקציה עם תוכן העמוד ולשנות אותו. הדפדפנים יכולים לבצע הפרדה בין ה-context של התוכן המוצג לבין זה של העמוד המציג אותו, לדוגמה, אם בעל העמוד חפץ להציג אנימציית svg בתוך תג video, על הדפדפן להפריד בין השתיים כך שהתג יראה את עצמו כעמוד נפרד ללא אב ולא יוכל לשלוט על ה-DOM בעמוד שהטמיע את הוידיאו.

מצד שני, כאשר עמוד זדוני מציג תוכן תמים, החשש הוא שהעמוד יוכל לשלוף מידע על אותו התוכן שבצורה אחרת לא יכל להשיג. דוגמאות למידע שהעמוד הזדוני יכול להציג הן: משך הסרטון\אודיו, משקל הקובץ, עצם קיומו של הקובץ, מידע נוסף על השרת המאחסן את הקובץ ועוד.

חשיפה של מידע מסוג זה לאתר זדוני היא כבר בעייתית, אך קיימת בעיה חמורה עוד יותר: אם הדפדפן חושף מידע נוסף על התוכן (כמו שהרבה גופים אוהבים לעשות), כמו כתוביות, רשימת כותרים או פרקים או כל מידע אחר. מידע שכזה אולי לא נראה חשוב כל כך, אבל תחשבו על התרחיש הבא:

נניח כי קיימת חברת סטארט-אפ אשר עובדת על פיתוח של רעיון אשר עתיד להניב לחברה המון כסף. במחשבי החברה קיימים סרטונים אשר מציגים את שלבי הפיתוח עד כה. בכדי לעדכן משקיעים ולהלהיב משקיעים פוטנציאלים, לסרטונים יש כתוביות על מנת שגם משקיעים אשר לא דוברים את שפת הקודש

יוכלו להבין שהחברה בדרך לכסף הגדול. עובד תמים מן החברה גולש לעמוד זדוני מתוך אחד ממחשבי החברה. בעת הגלישה לעמוד, מורץ ברקע על הדפדפן סקריפט זדוני אשר מציג (בגלוי או בסתר) לעובד את הסרטונים הרגישים הללו. כמובן שכעת הסקריפט יוכל לראות מה אורך הסרטונים. ובאם דפדפן הגולש החליט לספק קצת יותר מידע, יוכל העמוד הזדוני גם לראות את הכתוביות עבור הסרטים, ואלו כבר יספיקו לו בכדי לדעת מהו מוצר הקסם עליו החברה עובדת. בנוסף לבעיות אלו יש לקחת בחשבון כי מימוש המנגנון הנ"ל הינו מטלת הדפדפן, בניגוד למצב כיום שמטלה זו מבוצעת על ידי פלאגינים חיצוניים, בעיקר על ידי הנפוצים כמו Adobe Flash ו-Microsoft Silverlight אשר ידועים כמקורות טובים לבעיות אבטחה. כעת הדפדפנים יצטרכו לממש זאת בעצמם, וכנראה שנראה שוב חורי אבטחה. הדפדפן יצטרך להתמודד עם מספר גדול יותר של פורמטים מאשר היה צריך בעבר (למשל avi, mp4, mp3, ...) ובין השאר יצטרך להתמודד מול התקפות עתידיות אשר ישתמשו בחולשות בפורמטים של קבצי ווידאו ואודיו, ובנגנים אשר יממש על מנת להציגם.

יש לשים לב כי למרות שרוב מה שכיום תוספים כמו Adobe Flash ו-Microsoft Silverlight עושים בעמודי אינטרנט, כעת יכול להתבצע בעזרת HTML5 ו-JS, עדיין HTML5 מאפשר לנו להטמיע תוספים (plugins) בעמוד בעזרת תג ה-embed, בתוספים אלו יכולים להיות פרצות אבטחה בעצמם ובמקרה שכזה אלו יכולים להוות סיכון עבורנו ועבור המשתמשים.

תג ה-canvas:

תג ה-canvas מאפשר רינדור דינמי של תמונות בעזרת סקריפט, או במילים אחרות, ניתן לצייר על תג ה-canvas בעזרת javascript.

בכדי ליצור canvas נכתוב את קוד ה-HTML הבא:

```
<canvas id="example" width="200" height="200">
This text is displayed if your browser does not support HTML5 Canvas.
</canvas>
```

וכעת נצייר עליו בעזרת JS מלבן אדום:

```
var example = document.getElementById('example');
var context = example.getContext('2d');
context.fillStyle = "rgb(255,0,0)";
context.fillRect(30, 30, 50, 50);
```

עד כאן הכל טוב ויפה. מה הבעיה?

אם קיים מצב בו סקריפט ממקור (origin) אחד יכול לגשת למידע אשר נמצא במקור שונה, הסקריפט יכול לקרוא את הפיקסלים מן ה-canvas ולדעת מהי התמונה המוצגת.

בכדי למגר את הבעיה, החליטו ב-W3C להוסיף דגל (flag) בשם origin-clean. הדגל מקבל ערך בוליאני, true אם כל האובייקטים המצוירים ב-canvas (או המשמשים לציור ה-canvas) נמצאים באותו origin כמו העמוד המכיל את אלמנט ה-canvas ו-false אחרת.

התנאים היותר מדויקים נמצאים כאן:

http://dev.w3.org/html5/spec/Overview.html#security-with-canvas-elements

גם עם הפתרון הזה, הבעיה עדיין קיימת ובכל פעם שה-canvas משתמש באובייקט ממקור אחר, סקריפט כלשהו שרץ ב-context של העמוד יוכל לדעת בדיוק מה מוצג על ה-canvas. להרחבה:

http://html5example.net/tag/WebGL%20Browser%20Exploitation

תגיות טפסים

ב-HTML5 נוספו כמה תגי טפסים נוספים כמו, תג ה-output אשר אמור לתחום פלט מסוים הנוצר על ידי סקריפט, ותג ה-datalist אשר מחזיקה אפשרויות לבחירה והשלמה אוטומטית בתיבת טקסט. אך התג הכי מעניין שנוסף הוא תג ה-keygen אשר מייצר זוג מפתחות רנדומליים.

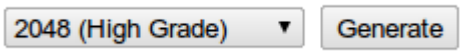
תג ה-keygen

כמו שנאמר, תג זה מייצר זוג מפתחות רנדומליים המתאימים להצפנה א-סימטרית המשמשים לאותנטיקציה. כאשר הטופס (שבו נמצא התג) ישלח, המפתח הפרטי ישמר ב-Local keystore והמפתח הפומבי נארז ונשלח אל השרת.

לדוגמה, נוכל ליצור את הטופס הבא:

```
 <form method="POST" action="http://www.server.com">
<keygen name="pubkey">
<input type="submit" name="createcert" value="Generate">
</form>
```

אשר הדפדפן (כרום) ירנדר בצורה הבאה:



וב-Drop Down List תהיה רשימה של ה-keylength הנתמכים על ידי הדפדפן.

כאשר נלחץ על Generate, הדפדפן ייצר שני מפתחות (פרטי וציבורי) ואת המפתח הפומבי ישלח לשרת http://www.server.com.

לתג זה יש מספר תכונות:

- תכונת ה-challenge אשר מקבלת מחרוזת (אם לא מוגדר, מקבל ערך של מחרוזת ריקה).
- תכונת ה-keytype אשר מגדירה את האלגוריתם ששימש ליצירת את המפתח (כרגע אין רשימה של סוגי הצפנות שעל דפדפנים לממש, אך שדה זה יכול להכיל לדוגמה את הערך "RSA")

המפתח הפומבי ומחרוזת ה-challenge מקודדים בעזרת DER ונקראים PublicKeyAndChallenge. לאחר מכן, ערך זה נחתם בעזרת המפתח הפרטי בכדי ליצור ערך הנקרא SignedPublicKeyAndChallenge. ערך זה נשלח לשרת ביחד עם ערך מפתח ההצפנה הפומבי.

משהו נוסף שיהיה מעניין לראות הוא כמה רנדומלי יהיה רצף הבתים הנוצר. גם בעבר נראו אלגוריתמים אשר כביכול נתנו רצף בתים רנדומלי, אך לבסוף התגלה כי הרנדומליות לא כל כך רנדומלית.

בנוסף לתגים, ישנם כמה ערכים נוספים אשר ניתן לשים בתכונת ה-type של תג ה-input, למשל: search, url, datetime, email, color ועוד. אלו משפיעים בעיקר על תצוגת שדה הקלט ועל אימות תאימות הקלט לסוג השדה על ידי הדפדפן.

כל התוספות הללו לא מהוות בעיית אבטחה מבחינה עקרונית, אך ברגע שכולן ימומשו בדפדפנים יכול מאוד להיות שצורת המימוש תהייה פגומה. בדיקות תאימות קלט הן דבר מסובך, ויעבור זמן עד שנוכל לתת לדפדפן לבצע זאת עבורנו בעיניים עצומות. גם אז נצטרך לחשוב על כל הדפדפנים ולא רק על אחד שהמימוש שלו נכון. כמובן שזה לא חוסך לנו את הבדיקות בצד השרת. משתמש זדוני יכול בקלות לעקוף בדיקות בצד הקליינט, וגם משתמש שאינו זדוני ומשתמש בדפדפן ישן אשר אינו מבצע בדיקות אלו בצורה טובה או בכלל לא, ואף כלים אוטומטים למילוי טפסים יוכלו לשלוח לשרת מידע אשר לא אומת.

אם כן, אסור להסתמך על בדיקות אלו לא בצד הקליינט (אם כי זה פחות קריטי), ובטח שלא בצד שרת! **מנגנוני הבדיקה הללו נועדו אך ורק בכדי להוסיף לחוויות המשתמש ולא כמנגנון אבטחה.**

את רשימת התגים החדשים של HTML5 והסבר עליהם ניתן לראות בלינקים הבאים:

- http://www.w3schools.com/html5/html5_new_elements.asp
- http://www.htm.co.il/2009/10/26/html-5-%D7%9E%D7%94-%D7%A0%D7%9B%D7%A0%D7%A1-%D7%95%D7%9E%D7%94-%D7%99%D7%95%D7%A6%D7%90-%D7%97%D7%9C%D7%A7-%D7%90/

התכונות החדשות:

ב-HTML5 ישנן תכונות גלובליות אשר חלות על כל התגים, כמו תכונת ה-contenteditable אשר קובעת כי המשתמש יכול לערוך את תוכן התג.

כמו כן ישנן גם תכונות חדשות עבור תגים מוכרים כמו תכונת ה-autofocus אשר בעת מילוי טופס תעשה פוקוס אוטומטי על השדה הרצוי, תכונת ה-placeholder אשר בעת מילוי טופס תראה לנו דוגמה למה אמור להיות באותו השדה, תכונת ה-required אשר מגדירה כי שדה מסוים בטופס הינו שדה חובה, תכונת ה-pattern תגרום לדפדפן לבדוק שהערך בשדה ה-input שעבורו יישמנו את התכונה יתאים לביטוי הרגולרי שקיבלה התכונה כערך.

בנוסף, ב-HTML5 לא חייבים לתת לכל תכונה ערך, וכאשר כותבים ערך שהוא ללא רווחים, לא חייבים לשים יותר גרשיים (למרות שזה נראה טוב יותר עם גרשיים). ניתן למשל לכתוב את התג הבא:

```
<input type=email placeholder="hello@mail.com" autofocus required / >
```

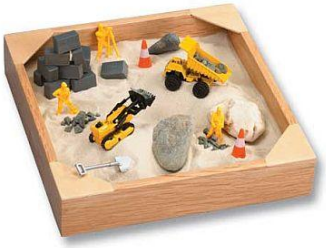
דבר נוסף שחלקכם ימצאו שימושי הוא תכונות מותאמות אישית (Custom Attributes), זאת אומרת שאם יש תכונה שהייתם רוצים להוסיף והיא לא קיימת ב HTML5, אתם יכולים לכתוב אותה ולא תיגרם כל שגיאה. ניתן לאחר מכן להשתמש בה ב-JS וב-CSS. זוהי דרך מצוינת להכניס עוד מידע הרלוונטי לעמוד אך לא נמצא לו מקום מתאים. דוגמה טובה לשימוש בזה היא ליצור תכונת explanation המכילה הסבר על המושג שהתג שהיא נמצאת בו תוחם, ובעזרת JS להציג את אותו הסבר כאשר עוברים על המילה. כך ההסבר יכלל ב-HTML אשר אמור להכיל את התוכן.

כמובן שגם את התכונות המותאמות ניתן לנצל אם קיים מצב כמו בדוגמה. המנגנון אשר מדפיס את ההגדרות של המושגים משתמש ב-document.write, ומעביר לו כפרמטר את ערך ההגדרה. כעת נניח אפילו שיש IDS, IPS או כל מנגנון אחר אשר מסנן כל תכונה קיימת ב-HTML וגם ב-HTML5 כדי שלא נצליח להחדיר שום תכונה או מאורע, אשר ישמשו להרצת סקריפט, לתוך התג הנ"ל. אם המנגנון לא עובר התאמה לתכונות החדשות שבעל האתר הוסיף (וזה משהו שקל יחסית לפספס), נוכל להזריק את אותה התכונה למקומות החשופים ל-XSS (כמובן שהם נבדקים על ידי המנגנון ולכן בעל האתר חושב שהם בטוחים), ולגרום להרצת סקריפט.

תג ה-iframe:

תג ה-Iframe הינו תג המאפשר להטמיע עמוד אינטרנט אחר ישירות בתוך האתר. קיימות הרבה בעיות אבטחה בדבר זה. כחלק מן הניסיון לפתור בעיות אלו, וגם בכדי להוסיף לפונקציונלית של ה-iframe, הוסיפו לתג ה-iframe כמה תכונות מעניינות.

תכונת ה-sandbox



תכונת ה-sandbox היא תכונה שימושית ביותר. כאשר נוסיף תכונה זו לתג Iframe, הדפדפן בעצם יוריד את ההרשאות של הדף הזה וימנע ממנו לבצע פעולות מסוימות שעשויות לסכן את המשתמש ואת האתר עצמו. תכונה זו יכולה לקבל ארבעה ערכים:

- allow-forms - מאפשר שליחת טפסים מן הדף המוטמע.
- allow-same-origin - מתייחס לתוכן הדף כאילו הגיע מן הדומיין של האתר המטמיע (בין השאר מאפשר לסקריפטים בדף המוטמע לגשת לאלמנטי DOM, לעוגיות ו-localstorage הנמצאים בדומיין של הדף המטמיע).
- allow-scripts - מאפשר הרצת סקריפטים בדף המוטמע (למרות שעדיין הסקריפטים אינם יכולים להקפיץ הודעות pop-ups).
- allow-top-navigation - מאפשר ללינקים ולסקריפטים לנווט את העמוד המטמיע (למשל, בזמן טעינת הדף לקשר את העמוד העליון לעמוד פישינג).

בנוסף, במקרה שלא מכניסים אף אחד מן הערכים הללו, אלא נותנים ערך ריק, הדפדפן יחסום את כל הפעולות הנ"ל. זאת אומרת שכאשר הערך הוא ריק, הדפדפן יראה את העמוד המוטמע בתור עמוד מדומיין אחר, שליחת טפסים והרצת סקריפטים מן הדף המוטמע לא תתאפשר, לינקים לא יוכלו לנווט את העמוד המטמיע ותוספים אחרים (plugins) לא יכולו לרוץ.

חדי העין (וחדי הקרן) בוודאי שמו לב שבמקרה ובאתר המוטמע מוגדרים גם allow-same-origin, גם allow-scripts וגם העמוד המוטמע ב-iframe הוא מאותו מקור, האתר המוטמע יכול בקלות לגשת לדף המטמיע ולבטל בעזרת סקריפט את תכונת ה-sandbox ולבטל כל הגנה שזו סיפקה, קוד לדוגמה:

inner.html:

```
<script>
top.document.getElementsByTagName("iframe")[0].attributes.removeNamedIte
m("sandbox");
</script>
```

outer.html:

```
<iframe src="http://127.0.0.1:80/inner.html" sandbox="allow-same-origin
allow-scripts">
</iframe>
```

תכונת ה-srcdoc

תכונת ה-srcdoc מאפשרת למפתח להכניס לתוך ה-Iframe קוד HTML של האתר המוטמע. זאת אומרת, במקום לתת את הלינק לדף שברצוננו להטמיע ולגרום לדפדפן לבצע בקשה עבור התוכן של דף זה, ניתן לבצע את הבקשה בעצמנו ולהכניס ישירות את קוד ה-HTML.

במחשבה ראשונה הרעיון נשמע קצת מוזר, כי הרי באותו אופן יכולנו להעתיק את קוד המקור של העמוד לתוך העמוד שלנו. אך יש לזכור כי חלק מהתפקיד של תג ה-Iframe הוא למנוע גישת תוכן לא מורשה לתוכן העמוד שלנו (בין השאר בעזרת תכונת ה-sandbox).

במידה ולתג Iframe קיימות גם תכונת ה-src וגם תכונת ה-srcdoc תמיד תהיה עדיפות לתכונת ה-srcdoc. רק במצב של דפדפן מיושן אשר לא מכיר את תכונה זו יהיה שימוש בתכונת ה-src.

תכונת ה-seamless

מפתחים רבים משתמשים בתגי HTML עם סמנטיקה מאוד ספציפית עבור תוכן לא מתאים בטענה ש"זה נראה יפה". תכונת ה-seamless היא תכונה בוליאנית שמעלימה את הגבולות של תג ה-Iframe וגורמת לו להיראות כחלק מן העמוד.

ניתן להשתמש בה בצורה הבאה:

```
<iframe seamless>
```

אך זה לא כל מה שתכונה זו עושה. התכונה לא רק גורמת לדף המוטמע להיראות כחלק מהעמוד, היא גם גורמת לדפדפן לרנדר את הדף המוטמע כחלק מהעמוד!

כמה דוגמאות להתנהגות הדפדפן במקרה שהתכונה מופיעה בתג Iframe מסוים:

- לינקים בתוך ה-Iframe ינווטו מחדש את הדף כולו ולא רק את ה-Iframe (אלא אם ב- target צוין _self).
- ה-css של הדף המוטמע ישפיע לא רק על ה-Iframe אלא יצורף גם אל ה-css של העמוד המטמיע וישפיע על העמוד כולו.
- רוחבו של ה-Iframe יהיה זהה לזה של כל אלמנט אחר באותה רמה ובעל רוחב אוטומטי.
- גובהו של ה-Iframe יהיה גובה של מסגרת התוכן המוטמע בתוכו (Bounding Box) בהינתן הרוחב שהוגדר בנקודה הקודמת.
- כאשר תוכנת נגישות כמו קורא קולי יקרא את העמוד, הוא יקרא גם את תוכן ה-Iframe בלי להכריז על כך שמדובר בדף נפרד.

- הדפדפן מתייחס אל הדף המוטמע כאילו הוא חלק מן הדף המטמיע, גם כאשר נריץ סקריפט. לדוגמה, אם בדף המטמיע יהיה סקריפט אשר סופר את מספר הלינקים בעמוד, גם הלינקים אשר נמצאים בתוכן של הדף המוטמע יספרו.

בקשות בין מקורות שונים (Cross-Origin Requests - XDR ו- Cross Origin Resource Sharing - CORS)

בימים שלפני HTML5, בקשות AJAX הוגבלו על ידי "Same Origin Policy", אשר מאפשרת לבקשות עבור משאב מסוים להתבצע רק בתוך אותו דומיין, זאת אומרת שהעמוד http://www.site.com/index.html יכל לבצע בקשות רק עבור משאבים הנמצאים תחת http://www.site.com. ב-HTML5 ביטלו הגבלה זו, ומאפשרים לבקשות AJAX להתבצע בין מקורות שונים.

למה לבטל את ההגבלה? זהו לא משהו חדש, כבר היום מפתחי אתרים יוצרים מנגנונים משלהם על מנת לאפשר בקשות בין מקורות, על מנת לספק למשל APIs ולנצל יותר מן הפוטנציאל של הרשת (תחשבו על מערכות התגובות של פייסבוק המאפשרת הטמעה באתרים שונים).

מה חדש?

בקשות למשאבים ממקורות אחרים יכלו להתבצע גם עוד לפני HTML5, הרי עוד הרבה לפני ה-AJAX וה-HTML5 דפדפנים היו מבצעים בקשות עבור משאב הנמצא בדומיין אחר, אם על ידי תג img, script או iframe.

אך בבקשות מסוג זה לעמוד לא הייתה גישה לתוכן המשאב. הדפדפן רק הריץ את המשאב או רינדר אותו והציג אותו בעמוד.

ההבדל הוא ש-XDR מאפשר ל-JavaScript לקרוא את תוכן המשאב על ידי תכונת ה-' responseText'.

איך זה פועל?

XDR הוא מעין מפרט על שמגדיר איך לאפשר בקשות Cross-Origin אל עמוד מסוים. מפרט זה מגדיר מספר כותרים (headers) שניתן להוסיף לבקשות ותגובות של HTTP.

הכותרים שמוגדרות ב-XDR

XDR מגדיר מספר כותרים, חלקם יוכלו בבקשות מן השרת וחלקם בתוך תגובות השרת לבקשות אלו. אנו נסביר כאן רק על הכותרים העיקריים:

Access-Control-Allow-Origin

Access-Control-Allow-Origin נמצאת בכותר של תגובה מן השרת, ומגדירה לאילו דומיינים מותר לבקש בבקשת Cross-Origin את המשאב.

התחביר הוא:

```
Access-Control-Allow-Origin = רשימת דומיינים | *
```

(כאשר התו '*' יגרום לכך שכל דומיין יוכל לבצע את הבקשה.)

זה כנראה יהיה ה-header הבעייתי ביותר. הרבה מפתחים ישימו כערך '*' ובכך בעצם יצרו דלת פתוחה לכל מי שרוצה לבצע XDR.

Access-Control-Max-Age

Access-Control-Max-Age נמצאת בכותר של תגובה מן השרת, ומגדירה לכמה זמן התגובה לבקשת preflight (יוסבר בהמשך) תישמר ב-cache.

התחביר הוא:

```
Access-Control-Max-Age = מספר שניות
```

Access-Control-Allow-Methods

Access-Control-Allow-Methods נמצאת בכותר של תגובה מן השרת, ומגדירה באילו שיטות מותר לבקש בבקשת Cross-Origin את המשאב.

התחביר הוא:

```
Access-Control-Allow-Methods: GET/POST/OPTIONS/DELETE/...
```

ניתן להגדיר גם כמה שיטות על ידי רשימת שמן עם רווח ביניהן.

Access-Control-Allow-Headers

Access-Control-Allow-Headers נמצאת בכותר של תגובה מן השרת, ומגדירה עם אילו custom headers מותר לבקש בבקשת- Cross-Origin את המשאב.

התחביר הוא:

```
Access-Control-Allow-Headers: custom headers names
```

Access-Control-Allow-Credentials

Access-Control-Allow-Credentials נמצאת בכותר של תגובה מן השרת, ומגדירה האם התשובה לבקשה יכולה להיעשות כאשר דגל ה-Credentials (יוסבר בהמשך) דלוק.

כאשר כותר זה חוזר עם תשובה לבקשת preflight, הוא מגדיר האם הבקשה האמיתית יכולה להיעשות כאשר דגל ה-Credentials דלוק.

התחביר הוא:

```
Access-Control-Allow-Credentials: true | true: %x74.72.75.65
```

Origin

Origin נמצאת בכותר של בקשה לשרת, ומגדירה מאיזה דומיין נעשית הבקשה.

Access-Control-Request-Method

Access-Control-Request-Method נמצאת בכותר של בקשה לשרת, וכאשר הדפדפן מבצע בקשת preflight, הוא משתמש בכותר זה על מנת להגדיר באיזו שיטה הוא ירצה לבצע את הבקשה האמתית.

Access-Control-Request-Headers

Access-Control-Request-Headers נמצאת בכותר של בקשה לשרת, וכאשר הדפדפן מבצע בקשת preflight הוא משתמש בכותר זה על מנת להגדיר עם אילו custom headers הוא ירצה לבצע את הבקשה האמתית.

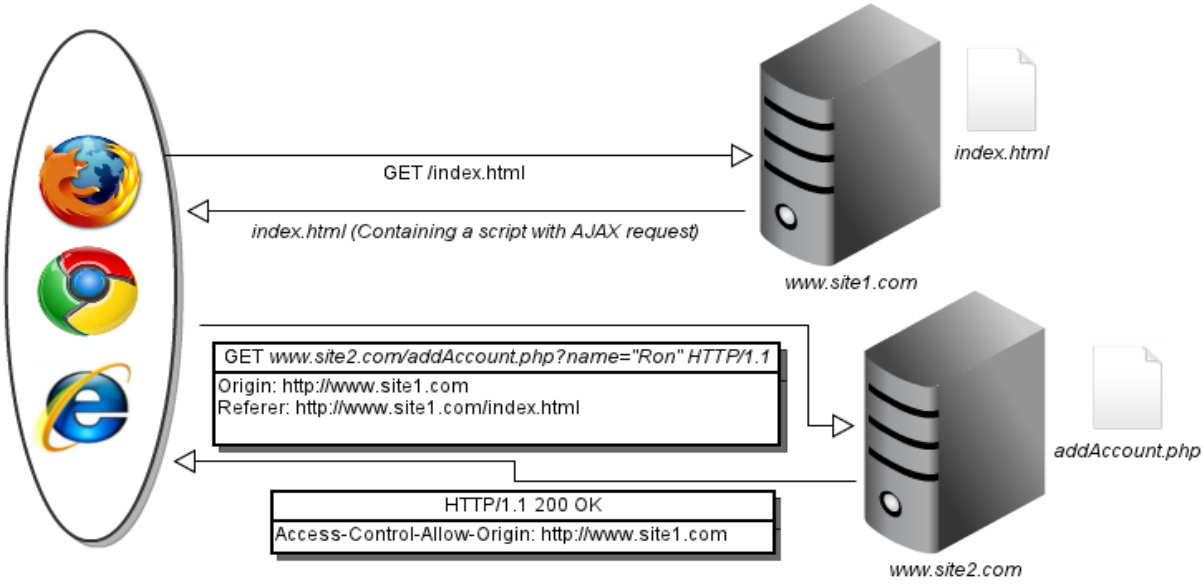
שימוש ב-XDR

ניתן להשתמש במפרט ה-XDR בעמוד http://site2.com/addAccount.php בכדי לאפשר לעמוד http://site1.com/index.html ולעמודים נוספים לבצע בקשת Cross-Origin אליו בצורה הבאה:

בכדי לעשות זאת נוסיף את ה-header:

```
Access-Control-Allow-Origin: http://site1.com
```

לתשובה שהשרת של http://site2.com מחזיר בעת בקשת הדף addAccount.php.



בעת ביצוע בקשה מן העמוד: http://site1.com/index.html לעמוד: http://site2.com/addAccount.php הדפדפן יבצע את הצעדים הבאים:

1. יבצע בקשה לעמוד http://site2.com/addAccount.php
2. יבדוק את ה-headers של התשובה שהוא קיבל חזרה, ויחפש אישור לבקשת Cross-Origin מן הדומיין http://site1.com
   a. אם הדפדפן מצא header אשר מגדיר שלדומיין הזר מותר לבצע את הבקשה הזו, העמוד יוכל לקרוא את תוכן התגובה.
   b. אחרת, העמוד לא יוכל לקרוא את תוכן התגובה.

ניתן לחשוב שסדר זה בעייתי, שהרי הדפדפן קודם מבצע את הבקשה ורק אז לפי תגובת השרת מוודא אם מותר לו לבצע את הבקשה. אבל לפעמים עצם בקשת עמוד מסוים עלולה לגרום לפעולות לא מאושרות בצד השרת.

אך זהו לא באמת סיכון מכוון שגם בצד שרת ניתן לבדוק מיהו הדומיין אשר מבצע את הבקשה על ידי קריאה של ה-origin header שם, בכל בקשה שנעשית, הדפדפן מכניס כערך את הדומיין אשר ממנו התבצעה הבקשה.

כך, במקרה שהשרת זיהה שדומיין לא מורשה ביצע את הבקשה, הוא יכול להחזיר עמוד ריק במקום התוכן המקורי ולא לבצע שום פעולות.

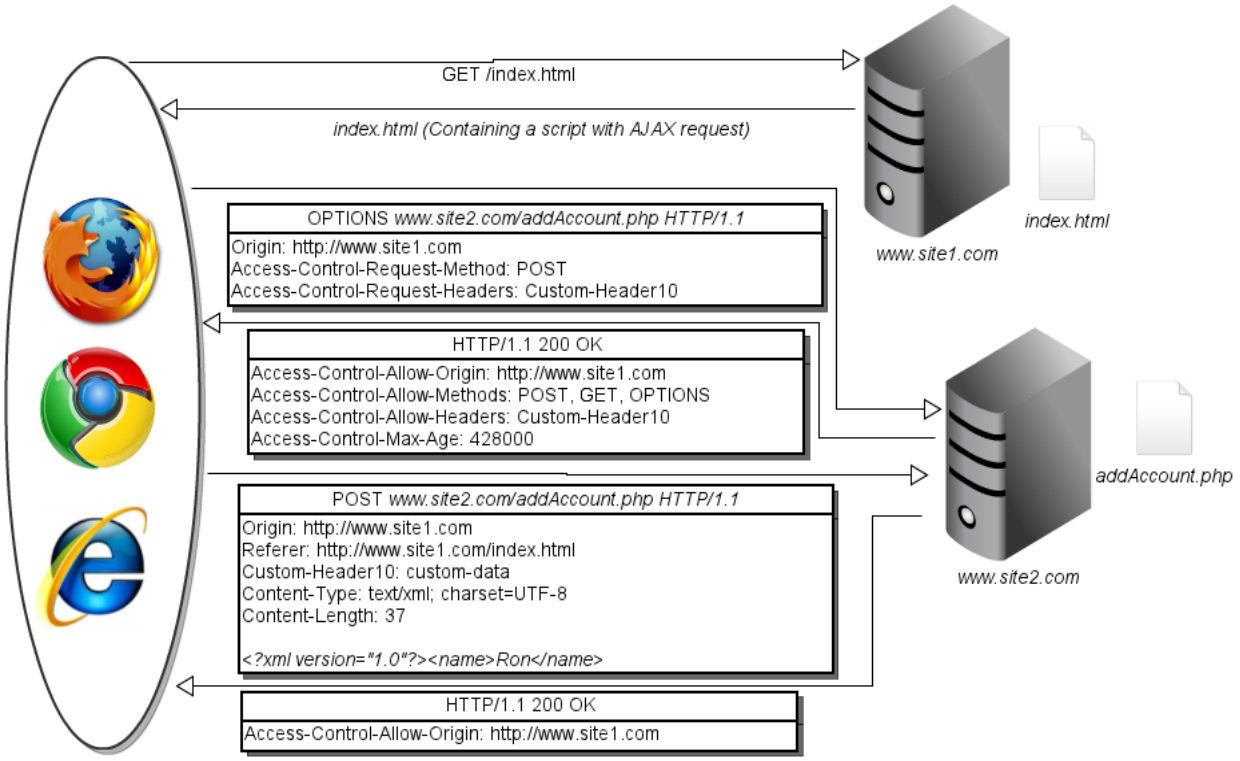
בקשת preflight

בנוסף לבקשות הרגילות, קיימת גם preflight request, אשר קודם שולחת בקשת OPTIONS בהתחלה בכדי לבדוק האם לדומיין מותר לבצע בקשת Cross-Origin, ורק אחר כך תבוצע הבקשה (בתנאי שלדומיין מותר לבצע אותה).

מנגנון זה קיים מכיוון שלפעמים עצם שליחת הבקשה אינה בטוחה.

הדפדפן מבצע את בקשת ה-preflight אוטומטית, ללא צורך בקוד נוסף אם מתקיים:

- הבקשה נעשית בשיטה אשר שונה מ-GET או POST, וגם אם הבקשה נעשית בשיטה אשר שונה מ-POST כאשר ה-Content-Type הנשלח יהיה שונה מ: application/x-www-form-urlencoded, multipart/form-data, או text/plain (או בעברית: אם הבקשה שולחת תוכן XML לשרת עם Content-Type של application/xml או text/xml אז הדפדפן יבצע את הבקשה בעזרת preflight).
- ישנם custom headers בבקשה (לדוגמה, ב-header יש שדה בשם "10custom-header"):



בעת ביצוע בקשה מן העמוד http://site1.com/index.html לעמוד http://site2.com/addAccount.php הדפדפן יבצע את הצעדים הבאים:

1. הדפדפן ישלח בקשת Option לעמוד http://site2.com/addAccount.php כאשר בנוסף הוא מצרף את הכותרות Access-Control-Request-Method ו-Access-Control-Request-Headers.
2. כעת, אם השרת מאשר לדומיין הזר לבצע את הבקשה עם הפרמטרים הללו הוא יענה בתגובה OK.
3. אם השרת אישר לדפדפן לבצע את הפעולה, הוא ישלח כעת את הבקשה המקורית.
4. השרת יקבל אותה וישלח בחזרה את תוכן העמוד.

דגל ה-Credentials

דגל זה מסמן האם ה-Credentials של המשתמש יצורפו לבקשה.

Credentials הם: עוגיות, אותנטיקציות HTTP שנעשו, ו-Certificates של הקליינט עבור SSL.

בברירת המחדל, דגל ה-Credentials כבוי. כלומר, הדפדפן בברירת המחדל אינו מצרף עוגיות לבקשות שנעשות ב-XDR אלא אם הוגדר אחרת, ובמקרה שכזה הוא שולח את הכותרות המתאימות אשר שואלות את השרת האם זה בסדר מבחינתו. במקרה שכן, ה-Credentials נשלחים עם הבקשה. אחרת הבקשה לא מתבצעת.

כותרת ה-Access-Control-Allow-Origin לא יכולה להיות '*' כאשר דגל ה-Credentials דלוק.

בעיות אבטחה של XDR

אפשור כללי

אפשור כללי זוהי הטעות הכי ברורה והכי קלה שמפתחים יכולים לעשות. כמו שכבר אמרנו, הכותרת 'Access-Control-Allow-Origin' אמורה להכיל רשימה של הדומיינים אשר יכולים לבצע בקשת Cross-Origin לעמוד, אך היא יכולה גם לקבל ערך wildcard ('*') אשר יאפשר לכל אתר לבצע בקשות Cross-Origin. ניתן לנצל זאת במספר דרכים:

- אם קיים איזשהו דף השייך לרשת פנימית (intranet) של ארגון כלשהו ואין אליו גישה מבחוץ. עובד בארגון אשר גולש בטעות לאתר זדוני מאחד ממחשבי החברה המחוברים לרשת הפנימית (וגם לחיצונית) עלול לחשוף את המידע ברשת הפנימית לאתר הזדוני, כאשר האתר הזדוני יבצע

בקשת XDR לעמוד ברשת הפנימית, וכך יוכל לקרוא את המידע בו ולשלוח אותו לבעל האתר הזדוני.

- מקרה דומה הוא למשל אם גוגל מאפשרת אופציה נוספת כאשר מחפשים בשירות החיפוש שלה (למשל מחיקת תוצאות מן החיפוש). אך אופציה זו קיימת רק כאשר מחפשים מתוך הרשת הפנימית. נניח בנוסף שגוגל מאפשרת לכל אתר לעשות לה XDR. כאשר עובד גוגל יגלוש לאתר זדוני, האתר הזדוני יוכל לבצע בקשת XDR ולמחוק רשומות מן תוצאות החיפוש של גוגל.

- נניח שמצאנו באתר מסוים SQL Injection בעמוד שניתן לבצע אליו XDR מכל דומיין. אם נשלוף את כל המידע בעמוד בעצמנו, הלוגים של השרת יצביעו עלינו באופן ברור וניתפס. אך נוכל ליצור קוד JS אשר מבצע את השליפה ושולח את התוצאות אלינו בעזרת XDR. את הקוד JS נשים באתר שלנו או שנעביר אותו דרך אתר עם XSS אל משתמש כלשהו. כאשר קורבן יבקר באתר בו שמנו את ה-JS, הדפדפן שלו ישלוף את הנתונים מן האתר וישלח אותם אלינו. בדיקות בלוגים של האתר יראו כי המשתמש הוא זה שביצע את השליפות, מכיוון שכותרות של בקשות בדרך כלל לא נשמרות בלוגים. הקורבן לא יכול לטעון שהמחשב שלו נפרץ, מכיוון שבדיקה על מחשב הקורבן לא תראה עקבות של malware או כל צורת השתלטות אחרת.

שגוי Include

נניח כי קיים מצב שבו לשרת יש קובץ common.php לדוגמה, אליו הוא מאפשר בקשות XDR, והמון קבצים בשרת עושים לקובץ זה Include. עלול להיווצר מצב שעמוד אשר לא התכוונו לתת הרשאות לבצע אליו בקשות XDR עושה Include לקובץ זה, אשר בתורו מוסיף את הכותרים שמאפשרים בקשות XDR, וכעת יהיה ניתן לבצע בקשות XDR אל העמוד וגם אל כל עמוד אחר אשר יבצע Include לקובץ זה.

פתרון אפשרי לבעיה זו הוא לבצע בדף common.php בדיקה מול "מילון" המכיל מידע על דפים אליהם אנו מאפשרים בקשות XDR, ועל פי "מילון" זה לשלוח את הבקשות המתאימות, ולא להסתמך על כך שעשו Include לקובץ זה.

הסתמכות יתר על כותר ה-Origin

יש לזכור שכותר ה-Origin נוסף על ידי הדפדפן, וכל אחד יכול להנדס בקשות-HTTP משלו עם כותר-Origin שיכיל כל ערך שירצה.

לדוגמה, הקוד הבא מכיל פרצה:

```
<?php
      if($_SERVER['HTTP_ORIGIN'] == "http://intranet.andlabs.org”){
      header('Access-Control-Allow-Origin: http://intranet.andlabs.org');
      //perform some important action
```

```
      print >>> sensitive internal information <<< ;
}
else{
    print >>> normal page <<< ;
}
?>
```

תוקף יכול בקלות לזייף את כותר ה-Origin שלו, ובעזרת החולשה לבצע פעולות חשובות ולקבל גישה למידע רגיש.

שמירת preflight cache לתקופות ארוכות

לביצוע בקשת preflight יש overhead, ולכן ביצוע הבקשה לוקח יותר זמן מאשר בקשת XDR רגילה. לכן יש אפשרות לשמור את תוצאת ה-preflight ב-cache, כך שבפעם הבאה שנרצה לבצע את אותה בקשה לא נצטרך לבצע את בקשת ה-preflight שוב ונוכל לפעול על פי הנתונים שיש ב-cache. שמירת נתוני ה-preflight ב-cache נעשית על ידי הוספת הכותרת 'Access-Control-Max-Age' לתשובת השרת לבקשת ה-preflight.

שמירה של cache לזמן ארוך יכולה לגרום לבעיית אבטחה. נניח שהשרת עדכן את מדיניות הגישה אליו. דפדפנים ישתמשו ב-cache שלהם המכיל את המדיניות הישנה ולא במדיניות החדשה. הזמן המומלץ הוא 30 דקות.

מתן אמון יתר בשותף

ישנם שני צדדים ב-XDR. הצד שמבקש את המידע והצד שמוסר את המידע. צריכה להיות כמות מסוימת של אמון בין השניים. האתר המבקש סומך על כך שהמידע שיקבל חזרה הוא אמין ובטוח, בעוד האתר שמוסר את המידע סומך על כך שהצד השני מוסמך לבצע את הבקשה ולקבל את המידע.

גם אם שני הצדדים הם לגיטימיים (וגם אם אותו אדם כתב את שניהם), יש להפקיד את כמות האמון המינימלית במידע שמספק הצד שני. זאת אומרת שעדיין עלינו לבדוק שהמידע אינו מכיל סקריפטים זדוניים, מידע מטעה (כאשר אפשר) וכו'. זאת מכוון שגם אם הצד השני לגיטימי, יכול להיות שתוקף הצליח לקבל אליו גישה וכעת משתמש בו בכדי לתקוף אותנו.

כעת נביא דוגמה דמיונית למקרים שכאלו: נניח ש-DigitalWhisper היה מציג את עדכוני הטוויטר של תוכנית החדשות האהובה עליכם בעמוד הראשי.

אתר DigitalWhisper יבצע בקשות XDR אל טוויטר בכדי לדעת מה היו העדכונים האחרונים, וכמובן שטוויטר מאפשר ל-DigitalWhisper לבצע בקשות אלו, שהרי אלו בקשות לגיטימיות. טוויטר בחזרה ישלח את העדכונים האחרונים שתוכנית החדשות פרסמה בקידוד HTML. בין השניים יש אמון, ולכן

DigitalWhisper לא מבצע בדיקות קלט ופשוט מדפיס את העדכונים לעמוד וטוויטר מאפשר ל-DigitalWhisper לבצע מגוון רחב של פעולות.

תחריש מספר 1 - לתוקף יש שליטה על טוויטר

כמו שאמרנו, מכיוון ש-DigitalWhisper סומך על טוויטר שישלח מידע מקודד HTML הוא מכניס את המידע לעמוד ללא בדיקות קלט. כעת, מכיוון שטוויטר בשליטת התוקף, הוא יכול לשלוח מידע זדוני המכיל תגיות HTML, וסקריפטים אשר DigitalWhisper יציג למשתמשים ובכך יסכן אותם.

תחריש מספר 2 - לתוקף יש שליטה על DigitalWhisper

נניח שטוויטר סומך על DigitalWhisper ולכן נותן לו גישה למגוון רחב של אופציות. DigitalWhisper יכולים לקרוא ציוצים, לשנות ציוצים, להוסיף משתמשים, למחוק משתמשים וכו'. כעת, מכוון ש-DigitalWhisper בשליטת התוקף, הוא יכול לבצע בקשות זדוניות על שרתי טוויטר בשם DigitalWhisper.

חשוב מאוד שהצד המבקש יוודא את תקינות התגובה שהוא קיבל, ושהצד המוסר יחשוף רק את האפשרויות הכרחיות עבוד הצד השני. גם אם תוקף קיבל שליטה על צד אחד, אין זה אומר שהוא צריך לקבל שליטה על הצד השני באופן אוטומטי.

DDoS בעזרת XDR

יש לזכור כי קיימים שני סוגים של בקשות XDR: בקשה פשוטה ובקשה עם preflight. ישנם רק כמה מקרים בהם משתמשים ב-preflight, ולרוב הבקשות משתמשים במנגנון הפשוט.

נזכיר, במנגנון הפשוט אנו שולחים בקשה לשרת, מקבלים את המידע ואיתו את הכותרים, ואנו בודקים על פי הכותרים האם לדומיין אשר יצר את הבקשה מותר לראות את המידע. אם כן, ניתן לו גישה, אחרת לא.

לעומת זאת, במנגנון ה-preflight הדפדפן ישלח בקשת Options עם הכותרים המתאימים, כך שהשרת יידע באיזו שיטה ועם אילו כותרים הבקשה האמיתית תתבצע, מקבל תשובה מן השרת עם הכותרים המתאימים ובודק האם הבקשה המקורית יכולה להתבצע. נזכור גם שבעת בקשת XDR פשוטה (ללא preflight) השרת מבצע את כל החישובים הנחוצים בכדי ליצור את התגובה (ומתייחס לבקשה כבקשה לגיטימית). זהו רק הדפדפן אשר מחליט האם הבקשה היא באמת לגיטימית או לא.

ניתן להשתמש בזה בכדי לבצע התקפות DDOS בשכבת האפליקציה.

נניח לדוגמה שקיים עמוד המבצע פעולות חישוביות רבות ולבסוף שולח בחזרה את תוצאות חישוביו. בעת ביצוע XDR לעמוד, העמוד יבצע את החישובים וישלח את התוצאות, רק לאחר מכן הדפדפן יחליט האם לאפשר לעמוד שביצע את הבקשה לראות את תוכן התשובה או לא. אך זה כבר לא משנה, מכיוון וגרמנו לביצוע הפעולות על מחשבי השרת.

כעת נוכל לחפש עמודי אינטרנט המכילים Persistent XSS ולהזריק לתוכם קוד JS אשר מבצע את בקשת ה-XDR הזו.

בעוד אנשים ייכנסו לאתרים אלו, הדפדפנים שלהם יבצעו את בקשות ה-XDR, ובכך בעצם יעמיסו על מחשבי האתר.

היינו יכולים להשיג את אותה התוצאה בעזרת החדרת תג של תמונה בעל תכונת src מתאימה לעמודי האינטרנט המכילים Persistent XSS, אך ב-XDR נוכל להשתמש גם במקרה של בקשת POST/DELETE/PUT. היינו יכולים גם להטמיע iframe עם src מתאים או לשלוח טופס (כאשר צריכים לשלוח את הבקשה ב-POST לדוגמה) מתוך iframe, אך XDR נותן ביצועים טובים יותר, בעזרת XDR ניתן לשלוח כ-100,000 בקשות בדקה (בכרום וספארי).

פתרון חלקי לבעיה זו הוא לבצע בדיקות על כותרת ה-Origin בצד השרת, וכאשר ה-Origin אינו אתר שאנו רוצים שיבצע לנו XDR לא נבצע שום עיבוד בצד השרת. יש לזכור כי למרות שהתוקף יכול לזייף בעצמו את כותרת ה-Origin, הוא לא יכול לעשות זאת אצל משתמשים תמימים הנכנסים לאתר הזדוני, ולכן, מכוון שב-DDOS עסקנו, כל עוד אם התוקף בעצמו ישלח את הבקשות עם ה-Origin המזויף זו כבר לא תהיה מתקפת DDOS, לפחות לא מאסיבית (יכול להיות שלתוקף יש כמה מחשבים) אלא מתקפת DOS רגילה (או DDOS קטנה) שספק שתשפיע על השרתים שלנו.

בנוסף, פתרון זה רק מקשה עלינו לנצל את הבעיה, שכן אם נשלחות בקשות רבות מידי גם רכיבי הרשת בדרך אלינו יהיו עמוסים (כמו Switchs, Routers, וכו').

אפשרויות ניצול חדשות למתקפות ישנות

למרות שחלק מהתוספות החדשות ב-HTML5 אינן גוררות בעיות אבטחה חדשות, הן יוצרות וריאציות חדשות לבעיות ידועות.

XSS

באמצעות תגים חדשים

מנגנון הגנה יעיל על מנת למנוע (או לפחות לצמצם) מתקפות XSS, כש-encoding לא אפשרי (כאשר מקבלים תוכן עשיר מה-user) זהו מנגנון של פילטרים.

כמו בכל מנגנון פילטרים, יכולה להיות רשימה לבנה ורשימה שחורה. לצערנו (או שלא), רוב המנגנונים משתמשים בגישה של רשימה שחורה.

פילטרים שחוסמים למשל תגים כמו '<script' או '<img' וכו', על מנת למנוע מאיתנו להכניס סקריפט שלנו שירוץ. אך כבר דברנו על התגים החדשים של HTML5, ובעזרתם נוכל לעקוף מנגנוני פילטור אלו, ולהריץ את הסקריפט הנחשק.

למשל, תגי המולטימדיה החדשים עבור video ו-audio, להם יש מאורע של מה לעשות בזמן שגיאה (onerror).

וכמו תמיד, שגיאות אנחנו יכולים ליצור גם בעצמנו. כאשר נכניס src לא לגיטימי, מאורע ה-onerror של תג ה-video\audio יתרחש. לדוגמה:

```
<video src="0" onerror="javascript:alert(1)"></video>
```

```
<audio src="0" onerror="javascript:alert(1)"></audio>
```

כאשר שמים כמה תגי source מוקפים בתג video/audio, הדפדפן יעבור על כל רשימת ה-source-ים עד שיימצא source בו הוא תומך ואותו יפעיל. גם לתג ה-source קיים מאורע onerror אשר יקרה כאשר ה-source הספציפי לא נתמך, ולכן עוד דרך לגרום למאורע ה-onerror היא לגרום לכך שכל הפורמטים של תגי ה-sources ששמנו יהיו לא נתמכים, לדוגמה:

```
<video><source onerror="javascript:alert(1)">
```

```
<audio><source onerror="javascript:alert(1)">
```

כעת נעלה רמה אחת למעלה, פילטרים אשר חוסמים את התווים '>' '<'. לא נוכל להזריק תגיות, אך נניח (הנחה סבירה יחסית) כי הטקסט שנכנס כקלט נכנס כערך של תכונה של תג, ולכן נעבור לאפשרויות הבאות.

באמצעות תכונות חדשות

נסתכל לדוגמה על הקוד הבא:

```
<input type='text' value='someUserInput' />
```

לפני HTML5, על מנת לנצל חולשת XSS זו, היינו משתמשים למשל במאורע onmouseover, וכאשר המשתמש היה עובר עם הסמן על מקום זה, היה מורץ הסקריפט:

```
someUserInput = ' onmouseover='alert("XSS HTML4 version")
```

באמצעות HTML5, המספק לנו תכונה חדשה לתג ה-input, תכונת autofocus שמטרתה לשים את שדה הטקסט בפוקוס עם טעינת הדף אוטומטית, אנו מקבלים כוח רב יותר, ונוכל בקלות לגרום לסקריפט לרוץ ישר עם טעינת הדף ללא צורך במעבר ידני של הגולש עם הסמן, כלומר ללא אינטראקציה עם הקורבן:

```
someUserInput = ' onfocus='alert("XSS HTML5 version")'
autofocus='autofocus
```

ציינו קודם כי תכונת ה- autofocus לא חייבת לקבל ערך. כלומר, השורה הבאה שקולה:

```
someUserInput = ' onfocus='alert("XSS HTML5 version")' autofocus
custom='
```

שימו לב ששמנו תכונת custom, שהיא custom attribute, סתם בכדי להיפטר מהגרש האחרון שסוגר את הערך של value בלי שגיאות.

כמו כן:

```
someUserInput = ' onfocus=alert("XSS HTML5 ++ version") autofocus
custom='
```

(לא חייבים להקיף את ערכי התכונות בגרשיים)

מה שיקרה, זה ששמנו את השדה בפוקוס אוטומטי, וכל עוד הוא בפוקוס, ירוץ ה-alert.

משום שאנחנו לא רוצים ליצור אינסוף דיאלוגים נוספים, כלומר שהסקריפט ירוץ כל עוד השדה בפוקוס, ננצל את העובדה שרק שדה אחד יכול להיות בפוקוס בזמן נתון. כאשר שני תגי input מכילים autofocous, הדפדפן בסופו של דבר יעביר את הפוקוס מהראשון שהוגדר לשני. כלומר, הראשון בשלב הזה 'יפסיד אותו'. מזכיר לנו- event כלשהו? (אהמ-onblur)

```
someUserInput = ' onblur='alert("I love Israel")' autofocus /><input
autofocus custom='
```

ונקבל הרצה פעם אחת ללא אינטרקציית המשתמש. כמובן שאם הוא יחזור לפוקוס על השדה ויצא ממנו שוב אז הסקריפט שוב ירוץ. אך נוכל בהרצת הסקריפט בפעם הראשונה לדאוג לכך שמצב כזה לא יקרה.

מה קורה אם יש פילטור ספציפי על ארועים (events), כמו למשל onerror, onload ואירועים נפוצים אחרים, וכעת לא נוכל להשתמש בהם?

לשם כך נשתמש באירועים החדשים שבאים יחד עם HTML5 כמו למשל onforminout ו-onformchange.

לדוגמה:

```
<form id=formid onforminout=alert(1)><input></form>
```

```
<button form=formid onformchange=alert(2)>
```

ואפילו אם נעלה עוד שלב, פילטור חזק יותר באמצעות ביטוי רגולרי אשר מזהה תחילית של 'on', לאחריו מילה ואז הסימן =. כלומר, הביטוי הרגולרי:

```
on\w+=/i
```

פילטר חזק, אך גם זה לא ימנע מאיתנו להשתמש בתכונות החדשות של HTML5 אשר יעברו את הפילטר הזה בקלות, כמו למשל:

```
<form id="formid" /><button form="formid"
formaction="javascript:alert(1)">
```

עוד וקטורי תקיפה חדשים תוכלו למצוא ב- http://www.html5sec.org

ביצוע התקפות CSRF בעזרת XDR

מתקפת CSRF - Cross Site Request Forgery, מתבסס על העובדה שהמשתמש סומך על הדפדפן (וגם על האתר הנתקף). זו נגרמת מכיוון שבכל פעם שהדפדפן שלח בקשה לעמוד מסוים הוא מצרף את ה-Credentials (עוגיות, SSL Certificate, HTTP Authentication) אשר מזוהות עם אותו מקור לבקשה.

כך לדוגמה אם היינו מחוברים לאתר של הבנק שלנו ובאותו הזמן היינו נכנסים לאתר זדוני המכיל את תג התמונה הבא:

```
<img src="www.bank.com/transfer.php?amount=1000&to=EvilBankAccount" />
```

הדפדפן שלנו היה מבצע בקשת GET בכדי לקבל את התמונה, ובקשה זו הייתה גורמת להעברה כספית אל חשבוננו (שכן באותו הזמן הבנק זיהה אותנו על פי ה-Credentials שהדפדפן צירף).

פתרונות לבעיה זו הגיעו ממעבר לשיטת ה-POST (שלא פותרת את הבעיה לגמרי - בעזרת יצירת טופס מוחבא ב-iframe ושליחתו אל אתר הבנק ניתן לשלוח בקשה ב-POST, אך זה טיפה יותר מסורבל) ועד שליחת Challange (או Token רנדומלי) בחזרה עבור כל פעולה שכזו.

בעזרת XDR, ניתן לבצע כעת CSRF גם בבקשות DELETE ו-PUT!

לכן שרתים אשר לא מוגדרים טוב ומאפשרים בקשות שכאלו ללא אותנטיקציה (ואפילו עם אותנטיקציה) עלולים להיות פגיעים, תוקף יהיה מסוגל לנצל את ה-Credentials של משתמש בעל הרשאות כתיבה ומחיקה ולבצע פעולות אלו במקומו, וכך להשתיל, לשנות ולמחוק עמודים מן האתר. כדי שזה יקרה, על השרת לאפשר בקשות XDR עבור שיטות אלו באופן ידני (זה לא נעשה אוטומטית, ובברירת המחדל זה לא מתאפשר). אך נזכור ש-XDR יכול לבצע בקשות גם לרשת הפנימית, ושם יש סיכוי גדול יותר למצוא פרצות אבטחה שכאלו, וקיבלנו איום ממשי.

לסיכום

במאמר זה סקרנו והצגנו חלק קטן משלל הפיצ'רים והתוספות ש-HTML5 מאפשר לנו.

ראינו כיצד פיצ'רים חדשים עלולים ליצור בעיות אבטחה חדשות, לחדש אפשרויות ניצול לבעיות ישנות וגם כיצד פיצ'רים חדשים באים לעזרתנו בניסיון לפתור בעיות אבטחה ישנות (sandbox), על אף שגם פתרונות אלו עדיין לא הגיעו לכדי שלמות.

ישנן הרבה תוספות ושינויים שהצטברו ויצרו את מצב האינטרנט כיום (וכנראה מצב האינטרנט בעתיד), ברובם לא נגענו, ובמאמרים הבאים נמשיך ונסקור את שאר התוספות שנעשו ל-HTML5, ונתעסק בשאלות חשובות כמו: האם מנגנון ה-Drag&Drop ייצור גל מתקפות חדשות מסוג Clickjacking, כיצד ניתן להשתמש ב-History API בכדי לבצע מתקפות פישינג.

נמשיך ונסקור גם את התוספות השונות שנעשו מסביב ל-HTML5, וגם הן מהוות חלק בלתי נפרד מהשינוי המסיבי שהאינטרנט עובר כיום, כמו Websockets, Geolocation, Client Storage וכן הלאה, שהן אמנם חלק מהחידושים שבאים יחד עם HTML5, אך בניגוד למה שנהוג לחשוב, הן אינן חלק מ-HTML5. גם פה נענה על שאלות חשובות כמו: האם מנגנון ה DatabaseStorage יהיה חשוף למתקפות מסוג SQL Injection? מה ההבדל בינו ובין Session Storage, Database Storage או Global Storage? ולמה אנחנו צריכים כל כך הרבה שיטות לשמור מידע בצד הלקוח?

נראה גם כיצד ניתן לבצע Port Scanning בעזרת בקשות XDR ו-WebSockets, ונציג דרכים חדשות לנצל משתמשים תמימים לצרכינו הזדוניים, כל אלו ועוד במאמרים הבאים.

קצת קישורים

- http://www.andlabs.org
- http://code.google.com/p/html5security
- https://developer.mozilla.org/en/HTML/HTML5
- http://mashable.com/2011/04/29/html5-web-security
- http://www.softwaremag.com/focus-areas/security/featured-articles/what-does-html5-mean-for-security

על הכותבים

לירן בנודיס ואלעד גבאי, בני 19, סטודנטים בשנה ד' בהנדסת מחשבים באוניברסיטה העברית. סטודנטים במעבדת האינטרנט של האוניברסיטה העברית:
http://internetlabhuji.wordpress.com

תודות

תודה ל:

ישראל חורז'בסקי (SRO), אפיק קסטיאל (cp77fk4r) ואוהד אסולין.

ואבאנגהההה!!!!