התקפות על כלים לעִבּוּד ולנִתוּחַ XML

מאת: שלמה יונה

הקדמה

מערכות לניתוח תעבורה שמיוצגת ב-XML הן מערכות שבנויות משכבות שונות של עיבוד. הפרוטוקולים והסטנדרטים שהם חלק מהמערכת האקולוגית של XML פותחו וממשיכים להיות מפותחים מתוך חשיבה על שכבות תוכנה בלתי תלויות, שניתן להשתמש בכולן או בחלקן באופנים שונים - מצב נוח מאוד בארכיטקטורה ושימושי ביותר. יחד עם זאת, אי התלות בין המרכיבים השונים מותירה פרצות רבות. התקפות על מערכות כאלה יכולות להתבצע בכל אחד מהרבדים וכן בשילוב של מספר רבדים. במאמר זה, ראשון בסדרת מאמרים, אפרט כמה מהשכבות הללו ואציג מספר התקפות על נַתְחֵי XML, כלומר על XML Parsers, בשכבות הנמוכות יותר. מפאת קוצר היריעה אנסה להסביר רק מספר מושגים ולתת רק מעט דוגמאות להתקפות. רשימת ההתקפות האפשריות רבה ומגוונת היא ויכולה לשמש חומר גלם לסדרה שלמה של מאמרים מסוג זה - במהלך הגליונות הבאים אפרסם חלקים נוספים בסדרה זאת.

אילו שכבות נפוצות במערכות לניתוח תעבורת XML?

שִׂכְבַת השִׁינוּעַ – למשל XML על HTTP: כתוכן של פרמטר ב-Query String או ב-Post Body, כתוכן של הודעה או של סדרת הודעות ב-HTTP (בשיטות שונות למשל, אך לא רק, SOAP ו-XML-RPC) ועוד. אפשר לחשוב על הזנות תוכן באינטרנט (RSS Feeds למשל) Web Services, אך לא רק. מה בנוגע להעברות קבצים שלמים? לא חסרים קבצים שיש להם ייצוג ב-XML: מסמכי Office למיניהם, תמונות ועוד. קבצים אנחנו מעבירים בשלל דרכים כמו למשל, דואר אלקטרוני, מערכות peer to peer, כוננים ניידים וכו'. במאמר זה לא אדון בשכבות השִׁינוּעַ, אך זה נושא מעניין שצופן בחובו אפשרויות רבות להתקפות בכלל ולהתקפות על מערכות שמשתמשות ב-XML בפרט.

שכבת קִידוּד התָּווים – המידע עובר בשכבות שִׁינוּעַ שונות ומשונות אך בסופו של דבר הוא מיוצג על ידי רצף של בתים (bytes). כל תוכנה שמקבלת קלט ב-XML אינה עובדת ישירות על הבתים, אלא על קִידוּד תָּווים מסוים. ישנן שיטות רבות לייצוג סדרות של בתים כרצפים של תָּווים בעלי משמעות. שיטת ASCII

היא דוגמה לקִידוּד התָווים שבה לכל רצף של 7 הסיביות (אות בינארית, 0 או 1) הנמוכות בכל בית (byte) יש מיפוי אחד ויחיד לתו. שיטה נפוצה ומקובלת לקִידוּד תָווים ב-XML היא UTF-8 המורכבת ומסובכת בהרבה, שכן היא ממפה בתים לתָווים באופן שחלקם מיוצג על ידי בית אחד ורובם על ידי שני בתים ויותר. משום ש-UTF-8 מאפשרת להשתמש ביוניקוד וגם מכילה בתוכה את ASCII באופן שמאפשר תמיכה לאחור, UTF-8 משמש שיטת קִידוּד התָווים שכל נַתָח XML חייב לתמוך בה גם כברירת המחדל.

שכבה לקסיקלית – השכבה הראשונה שבה נַתָחֵי XML מבצעים את העבודה ואוספים תָווים ליחידות בעלות משמעות. שכבה זו מקבלת תָווים משכבת קִידוּד התָווים ולפי ההקשר (ההקשר תלוי בכללי התחביר של XML ולכן האופן שבו שכבה זו עובד ומופעל תלוי בתנאי התחלה ובמשוב שמתקבל משכבת התחביר). שכבה זו מקבלת תָווים ומחליטה האם למשל התו '>' משמש כתו ככל התָווים, כמו למשל, 'A' , או שיש לו משמעות במבנה של תג, למשל התג, </foo>.

שכבת התחביר – מקבלת תָווים ומידע על מחלקות של תָווים (האם מדובר בתו תקין ומותר בהקשר ומה תפקידו). למעשה לא בכל המקרים באמת נחוץ להעביר לשכבה התחבירית גם את התָווים בעצמם – זה תלוי שימוש. כאשר מדברים על נַתָח XML לפעמים מתכוונים לכלים שממממשים את השכבה הזו. בדרך כלל כאשר מדברים על נַתָחֵי XML הכוונה היא לכלים שעושים שימוש בתוצרים של השכבה הזאת.

שכבת ממשק תכנות היישומים – מדובר בשכבה שמציעה API חיצוני מוכר וידוע. זאת השכבה שעולה בדעתם של רוב מי שמדמיינים מהו נַתָח XML. ממשקים ידועים ונפוצים הם בעיקר DOM ו-SAX. אלה אינם היחידים וטעות נפוצה היא להסיק שנַתָחֵי XML בפרט וכלים לעבוד XML בכלל חייבים לספק אחד משני מנשקים אלה. יש אופנים רבים נוספים ואחרים, אם כי אלה אינם נפוצים ואינם סטנדרטיים ולכן לא טובים בד"כ לארכיטקטורה בסגנון לגו.

שכבת ה-Namespeces – שמות תגים (Elements) ושמות של תכונות (Attributes) יכולים להיות מורכבים מתחילית ותחילית זאת יכולה להיות ממופה למחרוזת, בדרך כלל URL. זאת שיטה שמקובלת ב-XML כדי לאפשר שימוש באותם השמות בהקשרים שונים. יש המכנים את השמות כ"מילים במילון" ואת ה-Namespaces השונים כ"מילונים שונים". השכבה הלקסיקלית ושכבת התחביר עשויות לפעול באופן שונה כאשר באותו נַתָח XML מופעלת (או שאינה מופעלת) התמיכה בתקן Namespaces in XML ביחד או לחוד עם הפעלת (או אי הפעלת) התמיכה בשכבת ה-XML Schema Validation.

ההשפעות של קיום שכבה כזאת או של העדרה קובעות פרשנות ופעולת הנתח בעת שהוא פוגש בתו המיוחד ':' בשמות תגים ובשמות תכונות אך בעיקר הוא קובע את האופן שבו הנַתָּח מחפש שמות וטיפוסים בעת אכיפה בביצוע Schema Validation.

שכבת ה-**XML Schema Validation** - מבצעת אימות שהוא אינו תחבירי בלבד אלא גם סמנטי במובן מסוים: הנַתָּח בודק את מבני ה-XML ואת הערכים לפי טיפוסים שמוגדרים בְּסְכֶמָה, שנתונה. כך אפשר בְּסְכֶמָה לא רק לקבוע את מבנה ה-XML אלא גם אילו טיפוסים, בדומה לטיפוסים בשפות תכנות, משויכים לכל חלק במבנה ולכל ערך של תג ושל תכונה.

שכבות נוספות רבות קיימות וניתנות למיקום מעל כל שכבה שמעל שכבת התחביר או ה- XML Schema Validation.

לכאורה, עם השכבות הללו ניתן לספק הגנה לא רעה למערכות XML, אם הנַתָּח בנוי כהלכה ואם יש סְכֶמָה הדוקה וכתובה היטב. יחד עם זאת, גם במקרים אלה יש מקום להגנה נוספת במקומות שבהם התקנות וההגדרות של התקנים והפרוטוקולים של XML אינן מוגדרות היטב ובמקומות שבהן רמת האפליקציה שתשתמש במידע עשויה להשתמש בתוכן באופן עיוור תחת ההנחה שהוא שפיר. לפיכך, יש מקום לשלב הגנה של מערכות הגנה ל-XML כמו אלה שפותחו ב-F5 Netwoks עבור ה- Web Application Firewall ה-ASM. שכבות הגנה נוספות יכולות לכלול בדיקות של חתימות על ערכים במסמכי XML, על חלקי מסמכי XML ועל מסמכים מלאים, רשימת הגבלות שאיננה נכללת ב- XML Schema כמו הגבלה של עומק הרקורסיה למבנים מעגליים, הגבלת כמות הגדרות מיפויי ה-Namespaces והגבלות נוספות אחרות.

מה אנחנו כבר יודעים על התקפות ב-XML?

בגִליון מספר שלוש עשרה של Digital Whisper שראה אור באחד באוקטובר 2010 פרסם אפיק קסטיאל, cp77fk4r, מאמר בשם The Dark Side Of XML. במאמרו הוצגו שלוש התקפות: XML Tag Injection ויישומה במערכת מבוססת Web Services, התקפת XML Bomb והתקפת XXE. שלוש ההתקפות נוגעות ברבדים שונים של מערכות לניתוח מסמכי XML שהמשותף להן הוא הגדרות ברמת התבנית של מסמך ה-XML, מה שמכונה בעגה המקצועית XML Schema Layer. ההתקפה הראשונה מתאפשרת בגלל אי בדיקה או בגלל ליקוי בבדיקת מבנה התגים (ה-XML Elements), השנייה בשל

אפשור הגדרות DTD על ידי כותב המסמך (המשתמש), תחת הגבלת זכות זאת רק למערכת שמנתחת את ההודעה; והשלישית מכיוון שהיא מאפשרת לכותב המסמך לקבוע מדיניות גישה למשאבים חיצוניים, בדומה לשנייה, חלה מפני שלכותב ההודעה יש את הרשות לקבוע את מדיניות הפענוח והאכיפה של ניתוח המסמך. בעוד שאת הבעיה הראשונה ניתן לנטרל בקלות ע"י הפעלת XML Schema Validator שיאכוף את המבנה שתי הבעיות האחרות נובעות מתוך הנחה שניתן לתת הגדרות DTD בהודעה, מה שבכלל אינו תקין ואינו מותר בהודעות SOAP שמשמשות ב-Web Services, כך שבעיות אלה אינן אמיתיות במערכות שמנטרלות הגדרות DTD במסמכים.

גם XML Schema מאפשר פרצות אבטחה במנגנונים שהם בשליטת כותב מסמך ה-XML - כבר זה מצלצל לנו כדבר חשוד: אני אתן לכותב הודעות SOAP אנונימי מהאינטרנט אפשרות לשנות את האופן שבו המסמך שכתב ייאכף מול הסכימה שלי? אזכיר כמה שחושפים וקטורי התקפה מפתים באופן זה:

מנגנון xsi:type

מנגנון xsi:type נועד לאפשר שינויים ב-XML Schema של מסמכים שלא באמצעות עריכה של הסכימה עצמה אלא באמצעות המסמכים שצריכים לעבוד אימות מול הסכימה. השינוי שמאפשרים למשתמש לבצע הוא לקבוע בעצמו לפי איזה טיפוס על שכבת ה-Schema Validation לבצע אימות לחלק המסמך הרלוונטי. למעשה, המנגנון הזה מאפשר לשלוט בטיפוס הספציפי והקונקרטי שיזהה את החלק לטובת יישומי XML שהם XML Schema aware. מנגנון זה לא נועד לאפשר להחליט שהטיפוס שמולו ייאכף האימות הוא כלשהו, אלא לאפשר לקבוע טיפוס שהוא זהה או נורש מהטיפוס שנקבע בסְכֵמָה לאותו חלק במסמך.

בפועל, לא כל נַתָח מממש את האילוץ הזה, ואפשר למשל להחליט שקטע במסמך XML שמשמש לאימות זהות ולביצוע פעולה, שנאכף ע"י טיפוס שבודק בדיקה הדוקה ייאכף, אם בכלל, ע"י טיפוס שמניח שהזהות היא של משתמש-על ושמגוון הפעולות המותר הוא רחב יותר.

גם כאשר נאכף אילוץ הירושה - ישנם מקרים של תכנון סְכֵמָה שבהם יש טיפוס על של גישה למשאבים שממנו יורשים טיפוסים קונקרטיים: אחד של משתמש פשוט ואחד של משתמש על. על ידי שימוש ב-xsi:type יכול תוקף שיש לו ידיעה על הסְכֵמָה, לבקש שהפעולה שהכניס במסמך ששלח לשרת תהיה פעולה שתאכף מול טיפוס של משתמש-על (שלו יש זכויות יתר) ולא מול טיפוס המשתמש. בדוגמה

הבאה יש חלק ממסמך XML שבהודעות ברשת ממשתמשים לגיטימיים רואים את הקוד הבא:

```
<acc:execCmd xsi:type="accType:userPermissions"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <cmd:read resource="mq.1"/>
</acc:execCmd>
```

תוקף משנה בהודעות שיוצרת תוכנת הלקוח שלו כך:

```
<acc:execCmd xsi:type="accType:adminPermissions"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <cmd:read resource="mq.1"/>
</acc:execCmd>
```

כאשר דבר כזה מצליח ועולה יפה, אפשר גם להכניס שינויים בפקודות עצמן ובמשאבים שעליהן פועלות הפקודות.

מנגנון xsi:schemaLocation

מנגנון xsi:schemaLocation מאפשר לכותב מסמך XML לקשר למשאב חיצוני שהסמנטיקה שלו היא שהוא משמש כסְכֶמָה. הסְכֶמָה הזאת אולי אחראית להכיל את ההגדרות לטיפוס שקבענו כרגע על ידי xsi:type. דבר זה הינו מסוכן. אפשר גם להשתמש בגישות להשגת הסְכֶמָה שתבצע שכבת ה- XML Schema Validation לביצוע התקפות DoS וגם כדי לנסות ולגשת למשאבים אחרים. יתירה מזאת, אם אני התוקף קבעתי לי את הטיפוס שמולו תאכף פעולה שלי במסמך, ואני יכול לקבוע גם את מיקום הסְכֶמָה שבה מוגדר הטיפוס, אז מה מפריע לי לכתוב את הטיפוס בעצמי, להתיר לעצמי הרשאות ופעולות כרצוני, ללא מגבלות? זאת דרך להשתמש בכלי שנועד לאכיפה מחד ולגמישות מאידך – לרתום את מנגנוני האכיפה לטובת ההתקפה.

חפשו ב-Google code search ביטויים שבהם באותו התג יש שימוש גם ב-xsi:type וגם ב-xsi:schemaLocation. דבר זה מעורר השראה.

מהי דרך טובה למצוא חורים בנתחי XML?

אפשר לחשוב על כמה דרכים שהמשותף לכולן היא מתודולוגיות של בדיקות תוכנה. קוראים ברחל בתך הקטנה את כל הסטנדרטים והתקנים והפרוטוקולים המדוברים מלמטה למעלה וכותבים לכל טענה שחייבת להתקיים בדיקות שבודקות האם כך. לכל טענה שיכולה להתקיים, בודקים מה יקרה (פה אין

ודאות על תוצאה מצופה). לכל טענה ששוללת קיום, כותבים בדיקות שמוודאים שאכן כך הדבר.

לכאורה, המון עבודה. למעשה, זאת דרך נפלאה להכיר את החומר דרך הידיים, להבין, לגלות שליטה וגם לאסוף בדיקות שיכולות עם מעט אוטומציה למצוא שלל חורים וכשלים במערכות קיימות ברמת הנַתָּח. יש כבר ל-W3C אוספי בדיקות לצורך בדיקות תאימות עם התקנים שלהם. מניסיוני, הכיסוי שלהם נמוך, הם עוסקים בעיקר ב-DTD (שאינו רלוונטי כלל למערכות שעוסקות ב-Web Services) והם אינם מנסים להביא את הנַתָּח למצבים קיצוניים של ניצול משאבים, התקפות אלגוריתמיות וכן לא ממש עוברים טענה טענה בתקנים. בעת בניית תשתיות עבוד ה-XML ב-F5 הקמתי עם הצוות שלי אוסף בדיקות גדול עם כיסוי רחב מאוד של התקנים מלמטה למעלה ובנינו מערכת אוטומציה.

איך נראית בדיקה? מסמך XML משמש כקלט לבדיקה, תוכנו אמור לבדוק עניין שחשוב לנו ויש גם צורך בציון התוצאה המצופה מנַתָּח ה-XML שינתח את המסמך. לשם פשטות נביט בתוצאות המצופות הבאות:

- המסמך הוא well formed XML, ז"א המסמך מתאים לתקנים.
- המסמך הוא malformed XML, ז"א המסמך חורג מהתקן.

אם נכתוב את מסמכי ה-XML שלנו כך שהם מדגימים דוגמה חיובית (wellformed) או דוגמה שלילית (malformed) לעניין מוגדר היטב ומבודד, אזי בדיקות חיוביות שעוברות מציינות שאותם דברים שבמפרש בדקנו מתאימים לתקנים ואילו בדיקות חיוביות שנכשלות מציינות שמקרים אלה בעייתיים בנַתָּח. באופן דומה, בדיקות שליליות שעוברות מציינות שהנתח מצטיין באיתור עבירה על התקן בנושא הספציפי שנבדק ואילו בדיקות שליליות שנכשלות מציינות שהנַתָּח מעלים עַיִן ולא אוֹכֵף כהלכה את אותו החלק של התקן.

השתמשתי בבדיקות הללו ובמערכת האוטומציה שבודקת כדי לפתח נַתָּח XML שמשמש בתשתיות עבור ה-XML של F5 ולהביאו למצב שבו יש לי שליטה טובה מאוד על החלקים בתקנים שהוא אוכף ובאיזו מידה. באופן דומה בנינו בדיקות לצורך Schema Validation שהוא נושא מסובך בהרבה ולשכבות רבות נוספות, חלקן ייעודיות לצורכי אבטחה וחלקן אפליקטיביות לחלוטין (למשל, שאילתות על מסמכי XML בעזרת XPath). שימוש נוסף באותם מסמכים מתוייגים הוא בבדיקת מידת התאימות של נַתָּח אחר לתקנים וליקוייו. בהינתן תמונה ברורה על הליקויים ועל מידת (אִי)ההתאמה לתקנים אפשר לצאת לדרך ולהסיק כיצד להתקיף נַתָּח זה, יישום או מערכת שעושים שימוש בנַתָּח זה כאשר הממצאים שלנו

משמשים כווקטור התקפה.

נראה עתה כיצד אפשר לתקוף נַתָּח XML מתוך כמה בתים ראשונים בתחילת מסמך XML. ראשית, נבין כיצד מסיק נַתָּח ה-XML את שיטת קִידוּד התָּווים שיש להשתמש בה כדי לפענח נכונה את מסמך ה-XML.

ניחוש חכם של שיטת קִידוּד התָּווים במסמך XML

ננסה להבין, וכאן המקום לנסות ולפרוט לפרוטות את ההצעה בתקן (שאינה נורמטיבית, כלומר שאינה מחייבת אך רומזת בלי לשאת באחריות על כיוון לפתרון) בכל הקשור לניחוש חכם ונכון של קידוד התוים של המסמך.

לפני שנַתָּח XML יכול להסיק תָּווים ומתָּווים להסיק תגיות לצורך ניתוח תחבירי, עליו להכריע באיזה קִידוּד תָּווים עליו להשתמש כדי להמיר את שטף הבתים לתָּווים כראוי. אבל כדי שההצהרה בפרולוג על שיטת קִידוּד התָּווים תקרא ותובן, על הנַתָּח לדעת באיזה קִידוּד תָּווים להתשמש. מצב זה באופן כללי הוא מצב חסר תקנה (בֵּיצָה ותרנגולת, אם תרצו). יחד עם זאת ב-XML המצב אינו חסר תקנה באמת, שכן XML מגביל את המקרה הכללי בשני אופנים: משוער שכל מימוש של נַתָּח XML יתמוך רק במספר סופי של קִידוּדֵי תָּווים ושהכרזת שיטת הקידוד בפרולוג מוגבלת במיקומה במסמך, בתוכנה ובסדר מרכיביה. כתוצאה מכך, מתאפשרת הסקה אוטומטית של שיטת קִידוּד התָּווים שבה מקודד מסמך ה-XML. ההכרזה על שיטת הקידוד בפרולוג תשמש אם כן לשתי מטרות: האחת לאִישוּש והשנייה להפגת עמימות. למשל, במקרים שבהם תָּווי העיטור, ה-markup של XML: התָּווים שמאפיינים את התגים למשל: '<' ו-'>' נקראים בקלות כתָּווי ASCII אך טרם ברור האם מדובר בקידוד UTF-8 או אחר מתוך רשימת הקידודים מסוג ISO-8859 או אולי קידודי תָּווים אחרים שה ASCII-מוכל בהם.

ננסה להבחין בין 2 מקרים שבהם הנַתָּח יאלץ לְנַחֵש את שיטת קִידוּד התָּווים:

1. ניחוש ללא מידע חיצוני על שיטת קִידוּד התָּווים
2. ניחוש לפי סדרי עדיפויות במקרה שקיים מידע חיצוני על שיטת קִידוּד התָּווים

בהינתן מידע חיצוני על שיטת הקידוד (למשל כ-Mime Type) אפשר להסתמך עליו כל עוד אינו עומד בסתירה ל-BOM או למידע המוסק מהמסמך עצמו.

כשאין מידע חיצוני על שיטת קִידוּד התָווים או כשאין בטוחים במהימנות ניתן לנסות ולנחש באופן הבא:

התקן דורש שכל מסמך XML שאינו מיוצג ב-UTF-8 או ב-UTF-16 יכלול הצהרה של שיטת הקידוד. לפיכך, במקרים הללו אפשר להסיק שאם יש BOM ננסה לפיו לזהות את שיטת הקידוד ואחרת ננסה להסיק את שיטת הקידוד מהאופן שבו נוכל לקרוא את הרצף:

```
<?xml
```

כל מקרה אחר (פרט אולי לתווי רווח ולמשפחת ה-whitespace שאפשר להחליט שמתירים אותם, אף על פי שהתקן שולל אותם בתחילת קובץ) מרמז שהמסמך אינו well formed XML ולכן אינו תקין.

אז בעצם יש מספר סופי וקטן של צירופי בתים שמייצגים שיטות קִידוּד תָווים ב-BOM, ולכן ניתן לפתח נַתָח שמנסה בקוראו בתחילת כל קובץ להסיק את שיטת הקידוד שיש להשתמש בה לפי רצף הבתים הקצר שהוא קורא.

בטבלה הבאה אפשר לראות איך יש לפרש כל רצף בתים כרמז לשיטת קידוד הבתים כשיש BOM:

UCS-4 ב-Big Endian בסדר בתים טבעי: 1234	00 00 FE FF
UCS-4 ב-Little Endian בסדר הפוך: 4321	FF FE 00 00
UCS-4 בסדר בתים שאינו שגרתי: 2143	00 00 FF FE
UCS-4 בסדר בתים שאינו שגרתי: 3412	FE FF 00 00
UTF-16 ב-Big Endian	FE FF ## ##
UTF-16 ב-Little Endian	FF FE ## ##
UTF-8	EF BB BF

הסימון ## משמש לשני בתים כלשהם ברצף ובלבד שאין שניהם גם יחד 00.

כאשר אין לנו BOM, נוכל לנחש לפי האופן שבו הבתים יוצרים את הרצף הצפוי, ומתוך זה שנקרא כהלכה את הכרזת שיטת הקידוד בפרולוג נפיג את העמימות:

UCS-4 או שיטת קִידוּד תָּווים אחרת שעושה שימוש ב-32 סיביות כיחידה לייצוג תו ושבה תוי ASCII מיוצגים בהתאמה לפי ה-Endianess. ניתן להכריע לפי סדר הבתים על ה-Endianess.	00 00 00 3C 3C 00 00 00 00 00 3C 00 00 3C 00 00
UTF-16BE או ISO-10646-UCS-2 big-endian או שיטת קִידוּד תָּווים אחרת שעושה שימוש ב-16 סיביות כיחידה לייצוג תו ושבה תווי ASCII מיוצגים בהתאמה.	00 3C 00 3F
UTF-16LE או ISO-10646-UCS-2 little-endian או שיטת קִידוּד תָּווים אחרת שעושה שימוש ב-16 סיביות כיחידה לייצוג תו ושבה תווי ASCII מיוצגים בהתאמה.	3C 00 3F 00
UTF-8 או ISO646 או ASCII או סוג מסוּם של ISO 8859 או של Shift-JIS או של EUC או של כל סוג אחר של קידוד ב-7 או ב-8 סיביות.	3C 3F 78 6D
EBCDIC או סוג כלשהו שלו	4C 6F A7 94

שינוי שיטת קידוד התוים במסמך אסור.

התקפה באמצעות BOM או העדרו

ראינו שהשכבה הראשונה שבה יש לנַתֵּח הזדמנות לעבוד ולהשפיע היא שכבת קִידוּד התָּווים. לפי התקן כל נַתֵּח XML חייב לתמוך בקִידוּדֵי התָּווים UTF-8 ו-UTF-16. מסמכים שמיוצגים בקידודים אלה (וגם בקידוד UTF-32) יכולים (ובמקרים מסוימים אף חייבים) לפתוח ב-BOM. מדובר במספר בתים שמרמזים על שיטת קִידוּד התָּווים שעתידה להגיע, ולא זו בלבד, בשיטות לקידוד תָּווים מרובות בתים יש חשיבות גם לאֶנְדִיאַנִיוּת: ל-Big Endian לעומת Little Endian. אז מה לזה ולנו? נכתוב כמה בדיקות

לצורך בדיקת היכולת של הנתח להתמודדות עם ה-BOM. נכין מסמך XML קטן שאין בו דבר מלבד תג יחיד למשל:

```
<a/>
```

ונשמור מסמך זה בכל אחד מהקידודים שהנַתָּח שלנו אמור לתמוך בו כאשר עבור הקידודים של יוניקוד (UTF-8, UTF-16LE, UTF-16BE) ו-UTF-32 שיכול להופיע במגוון סדרים של הבתים ברביעיות) בשתי גרסאות שונות לפחות: נשמור את המסמך בקידוד המתאים פעם אחת עם ה-BOM המתאים ופעם אחת ללא ה-BOM המתאים (אפשר גם לנסות ולבדוק מה קורה כאשר נותנים BOM שאינו מתאים לקידוד). כמובן, שלכל מסמך בדיקה כזה אנחנו נסמן במערכת האוטומציה שלנו את התוצאה הצפויה: נצפה שמסמך שנשמר ב-UTF-8 ייקרא נכון כאשר יש לו BOM של UTF-8 וגם כאשר אין כלל BOM (זאת ברירת המחדל על פי התקן). בשאר המקרים נתייג לפי דרישות התקן - אם התקן איננו דורש נחשוב לרגע ונבין בעצמנו מה יִתֵּן קלט בעל משמעות ומה יִתֵּן זבל, ולפי המסקנות נכריע כיצד לצפות את התוצאות. אחרי ההכנות, מריצים את האוטומציה ומתעדים את התוצאות.

נַתָּחִים טובים יעמדו במשימה ב-100% הצלחה. נַתָּחִים משביעי רצון ידעו לעבוד נכון ב-UTF-8 לכל הפחות ובשאר המקרים יצאו יפה בהודעת שגיאה שהקידוד אינו נתמך. כל השאר יעופו או יבצעו פעולות שאינן צפויות בחלק מהמקרים. כבר יש בידינו חומר גלם כדי להזיק למערכות, עם קלט חוקי בחלק מהמקרים ועם חלק לא תקין בחלק מהמקרים – אך שימו לב לחוסר הסִימֶטרִיָה שמאפיין בעיות אבטחה רבות: קל וזול מאוד לייצר התקפה מזיקה ולעומת זאת לצד שמגן לא תמיד זול וקל באותה המידה (לפעמים יקר יותר בסדרי גודל). כל מה שנדרש לנו לייצר התקפה מטוּוַחַת היטב לנַתָּחִים שבדקנו הוא לייצר הודעות עם מספר בתים, קל וחומר כאשר שולחים חומר כזה באופן עִיוור למערכת שאמורה לעבד מסמכי XML – שולחים ומחכים לראות מה יקרה.

נמשיך באופן דומה לבדיקה מה קורה עם מסמך שמתחיל עם פרולוג.

התקפות באמצעות הפרולוג והרהורים על מימוש נכון של נַתָּח להגנה

הפרולוג הוא חלק שאינו הכרחי במסמכי XML, אך רצוי שיכיל (התקן כותב Should ולא Must). הפרולוג במסמכי XML עשוי להכיל מידע על גרסת ה-XML שהמסמך מיוצג בה, על שיטת הקִידוּד שהמסמך מקוּדד בה ושאר ירקות (ראו הגדרה במהדורה החמישית של גִרְסָה 1.0 של התֶקֶן או במהדורה השנייה של גרסה 1.1 של התֶקֶן). זה זמן טוב להזכיר שבעוד שיש גרסה 1.1 לתֶקֶן של XML, רובן ככולן של

מערכות ה-XML עדיין עובדות בגרסה 1.0, דבר היוצר בלבול. בפועל, נַתָּחֵי XML שאמורים לעבוד כמו לבני לֶגוֹ במערכות תוכנה חייבים להיות מתאימים גם לגִּרְסָה 1.0 וגם לגִּרְסָה 1.1 ולזהות, לאכוף או לתקן את הליקויים לפני שמפעילים שכבות אחרות או שמעבירים את פלט הניתוח הלאה.

כבר בשלב זה מעניין לבדוק כיצד מגיבים נַתָּחֵי XML כאשר מזינים להם פרולוג שבו גִּרְסת ה-XML אינה 1.0 או 1.1, מה למשל יקרה אם נכניס לשם כערך מספר גדול מאוד? מה יקרה אם נכניס מספר קטן מאוד בֶּדיוק גדול מאוד? מה יקרה אם נכניס ערך שאינו מספרי? – נַתָּחִים טובים יזהו את הבעיה, ידווחו ולפי קונפיגורציה עשויים להתעלם, לתקן או לפלוט את המסמך ולסרב לעבוד עליו. נַתָּחִים פחות מושקעים יתעלמו או יעופו. אם התעלמות נראית כמו פתרון טוב, אז חשבו מה יקרה לשכבות אחרות במערכת התוכנה שמצפות לקבל קלט תקין וקיבלו זבל.

מעניין לשלב את ההצהרה על שיטת הקידוד של המסמך עם מה שהנַּתָּח שלנו הסיק מעבוד הבתים הראשונים עד כה במסמך:
אם היה BOM ואחריו זוהה פרולוג - סימן שקִּידוּד התָּווים המוצהר והנקרא נכון עד לאותו המקום במסמך והנַּתָּח יכול לעבור משלב של הסקת קִּידוּד התָּווים לשלב שבו קִּידוּד התָּווים נקבע להמשך עיבוד המסמך. אבל לא בכל מקרה! מה אם ה-BOM (או העדרו) עולה בקנה אחד עם האפשרות לקרוא נכון את הפרולוג אך הקידוד שמוצהר בפרולוג שונה ממנו? למשל, לא היה BOM, הנתח ניסה לקרוא את הבתים הבאים ב-UTF-8 והצליח והגיע למצב שבו בפרולוג הקידוד המוצהר הוא בכלל אחר וסותר, נאמר UTF-16BE. בעיה? לנַּתָּח בוודאי. ומה עם התוקף? כמה מהנַּתָּחִים יעופו כאשר לא יְנַחֲשו נכון את שיטת קִּידוּד התָּווים? כמה מהם יעופו כאשר לא יקבלו את הקלט בשיטת קִּידוּד התָּווים שמצופה?

התקפה באמצעות הוראות עבוד

הוראות עיבוד, Processing Instructions, מהוות כלי לא כל כך מוכר בתחביר של XML, אך תשתיות XML בהחלט מממשות אותן. בגלל שמלבד התחביר אין מגבלות אחרות על הודעות עיבוד והן תלויות לחלוטין ברמת האפליקציה שעושה שימוש במסמך ה-XML, מסתמן וקטור התקפה יעיל. חפשו למשל ב-Google Code Search את הביטוי הרגוּלָרי הבא:

```
file:\.xml$ <\?[^xe]
```

שלילת ה-x נועדה לסנן החוצה פרולוג ושלילת ה-e נועדה לסנן החוצה באופן גס את הוראות העיבוד ל-eclipse. תוצאות החיפוש יספקו לכם רעיונות מה אפשר לעשות.

חשבו למשל על יישומים שמוכנים לקבל ולעבוד על הוראות עיבוד מסוג:

```
<?enforceNonRfc toc="yes"?>
```

נניח שנחליף את הערך ב-no? – האם נצליח לגרום לתשתיות שמתייחסות להוראות העבוד הללו להיות סלחניות יותר לתוכן מסמכי ה-XML או למידע אחר שה-XML משמש לו ככלי תעבורה? אולי זאת דרך לצמצם בדיקות וכך להחדיר התקפה?

מפתחים משאירים להם אמצעים לדבג מערכות. זה מבורך אך יכול להיות בעייתי כאשר תוקף משתמש בכלים אלה כדי לצרוך משאבים ממערכת או כדי להשיג מידע או זכויות יתר. הנה דוגמה של הוראת עיבוד שנראית לפעמים כאשר מסניפים תעבורה:

```
<?log level=”debug” ?>
```

מעניין מה יקרה אם נכניס הוראת עיבוד כזאת אחרי כל תג ותג.

דלת אחורית למערכת הקבצים בשרתים דרך מסמי XML:

```
<?include from=”adserver” file=”/path/to/ad.dhA_23B.xmlFrag”?>
```

אז מדוע שלא לנסות:

```
 <?include from=”adserver” file=”/etc/passwd”?>
```

ואחרי קצת רחרוח ובדיקת חבילות ועוד ניסוי וטעייה אפשר היה גם לבצע:

```
 <?include from=”authserver” file=”/path/to/privateKeys”?>
```

מה עושים?

תמיד אפשר לממש כראוי שכבות לניתוח ולעבוד XML - לכמה ארגונים יש את הידע ואת המשאבים לכך? משתמשים בכלים לעבוד XML שהם בדוקים כהלכה.

משתמשים בכלים ייעודיים שבנויים לאיתור ולהתמודדות עם XML Parser Attacks ושמציעים מנשק נוח לניהול ה-Security Policy – פעמים רבות משתלם להשתמש בכלים אלה גם ככלי offload לביצוע ה-Schema Validation מבחינת ביצועים, כך מקבלים בהמשך השרשרת מסמכי XML נקיים ובדוקים ושאר העיבוד יכול להתבצע תחת ההנחה שהקלט נכון.

לסיכום

הודעות ומסמכים שמבוססים על XML מעובדים באמצעות שכבות רבות שממומשות על פי תקנים רבים ומגוונים. בכל שכבה יש שלל פרצות קטנות יותר או משמעותיות יותר. כלים שונים לעבוד XML חשופים לחלק מהפרצות. במאמר זה למדנו על ייצוג שכבות מקובל וראינו מספר חולשות שמאפשרות התקפה על מערכות שמעבדות XML. ישנן עוד שכבות עיבוד רבות ופרצות נוספות. על כך עוד ייכתב במאמרים הבאים.