אבטחת מערכות ויישומים ברשת - שיעור 12#

Failure to restrict URL access:

מנגנון ה-Authentication מתבסס על מנגנון access control – אכיפה על סמך זהות המשתמש. נדבר כעת על פגיעות בה המערכת נכשלת בהגנה על access. כיוון שהמערכת לא מבצעת access control בצורה נכונה או בכלל, משתמשים לא מורשים מגיעים ל-URLs שלא אמורה להיות להם גישה אליהם.

לרוב משתמש מגיע מדף לדף ע"י הקלקה על לינקים מדף אחד למשנהו, כאשר צומת ההתחלה היא לרוב דף הבית. המפתח עלול לחשוב מתוך התנהגות זו, שאם למשתמש אין לינק לדף שלא אמורה להיות לו גישה אליו אזי הוא לא יגיע אליו – **הנחת אבטחה שגויה**. הטעות נובעת מההנחה שהמשתמש יעבוד רק על בסיס הלינקים שמוצגים מולו בדף, ומה שלא מוצג מולו לא יהיה נגיש למשתמש.

כל פורץ מתחיל שרואה תבנית מסויימת על צורת הלינק ישחק עם וריאנטים שונים של צורת הלינק, למשל מספר סדרתי, ואולי יגיע לדפים שאין אליהם לינק ישיר. אם על ה-host יושבים קבצים רגישים וחשובים כמו קבצי קונפיגורציה, סיסמאות וכו', יכול התוקף לנסות לגשת לקבצי המערכת, תוך ידיעת מערכת ההפעלה על אותו host. אם המפתח לא דאג ל-access control מתאים מתוך מחשבה כמתואר לעיל, ה-host חשוף להתקפה. זו דוגמא לאבטחה באמצעות הסתרה, ושיטה זו כמובן אינה יעילה אפילו מול תוקף מתחיל.

Forceful browsing attack:

כאשר עובדים ב-web services הם חושפים לא מעט APIs, שחלקם מיועדים רק ל-services אחרים פנימיים שיפנו אליו לקבלת שירות. מבחינת המערכת אין הבדל בין אלו שמייעדים לבקשות פנימיות ובין אולו שאמורים להיות חשופים למשתמש רגיל – ואז גם הוא יכול לנצל פניות מבחוץ ל- web services שלא אמורים להיות חשופים אליו, ובפרט מתודות שלא אמורות להיות חשופות אליו.

דוגמא:

לפעמים כל שהתוקף צריך לעשות הוא לקרוא תיעוד פונקציות JS ולהבין את השימוש בהן, וכך לגשת לכל מיני מקומות רגישים שלא אמור לדעת כיצד לגשת אליהם. נושא התיעוד הוא נושא אמביוולנטי בעולם ה-web. מצד אחד קוד צריך תיעוד לצורכי תחזוקה. מצד שני תיעוד דפי HTML או קוד JS שנשלח ל-browser חושף לתוקף, שהקוד הזה פתוח אצלו, אינפורמציה על המערכת.

באפליקציוצ AJAX בעיה זו מחמירה כיוון שחלק מהאפליקציה עוברת לרוץ בדפדפן, וככל שיש יותר קוד אפליקטיבי בדפדפן, הדורש תיעוד, כך יותר נחשף בקרב תוקפים פוטנציאליים.

כיום ישנם כלים שיודעים לנקות את הקוד מתיעוד וכך הקוד האפליקטיבי שנמצא בסביבת ה-production ונשלח למשתמש נטול תיעוד. פעילות זו מקשה על התוקף להבין את הלוגיקה האפליקטיבית.

Path Traversal Attack:

ההתקפה הזו כוללת אינפורמציה על relative path של קבצים, והתוקף ירצה לבצע מניפולציה על פרמטר כזה כך שיגיעו ל-path שהאפליקציה לא התכוונה אליו, והתוקף רוצה להגיע אליו. פעמים רבות התקפות אלו מתאפיינות ע"י /.. וכד' – פקודות טיול בתיקיות מערכת הקבצים. לצורכי מניעה יש צורך לבצע הגדרת regular expression מדוייק מאוד לתבניות שמוכנה האפליקציה לקבל – וכך למנוע התקפות אלו.

כל עוד מדובר על URLs שמטרתם להביא את המשתמש לקבצי מערכת ההפעלה, אז צריך:

- לא לתת למשתמש גישה לקבצי מערכת ההפעלה הלא מורשים.
- לא לתת לאפליקציה הרשאות גישה ל-Services של מערכת ההפעלה שאינה זקוקה להם.

צריך להשתמש במנגנון access control של מערכת ההפעלה או של ה-web server, ואפילו של שניהם. שילוב שני המנגנונים – ברמת ה-web server ומערכת ההפעלה שעליה רץ יכול לתת מענה ל-forcefull browsing וגם ברמה מסויימת ל-path traversal.

Failure to restrict URL access (OWASP 2010):

תוקפים משתמשים הרבה ב-crawlers – כלי לסריקת אפליקציות web למציאת לינקים. כל עוד סורקים קצבים סטנדרטים של מערכת ההפעלה, web server, app server, הכלים הללו נותנים תוצאות טובות.

How to restrict URL access:

לא להחזיק על שרתי web מידע שלא צריך להיות שם. קבצים שצריכים להיות על שרת ה-web, צריכים אכיפה לפי file-types, למשל כן לאפשר גישה לקבצי html, htm, jpeg... ולא לאפשר גישה לקבצי bak, xml, exe וכו' – מאפיינים את ה-filetypes לפי הסיומות שלהם. החלת positive security logic תאפשר חסימת גישה לקבצי מערכת כך שהגישה תחסם כבר ברמה זו.

ישנם קבצים שמאפשרים אליהם גישה בתנאים מסויימים, וצריך לבנות מנגנון כך שהגישה תתאפשר רק תחת אותם תנאים.

שימוש ב-OS file permissions לחסימת forceful browsing:

הגבלות גישות RWX לקבצים.

Forced Browsing bypassing access control checks:

- כאשר שולחים משהו לדפדפן, הוא נמצא בידי המשתמש. כאשר נשלח לינק, ההנחה היא שמחזיק הלינק אכן מורשה לגשת. כעת יכול אותו משתמש להעביר לינק שבידו, אחרי שאותו משתמש עבר אותנטיקציה, ולהעבירו למישהו אחר לא מורשה. אם מניחים שמחזיק הלינק מורשה לגשת, הנחה זו קורסת.
- ...

Broken Access Control:

עד כה דיברנו על גישה ל-URLs שם עובדים עם מנגנוני תשתית, ועכשיו מגיעים לקטע האפליקטיבי. כאן כל הפונקציונאליות של האפליקציה דורשת אכיפה באפליקציה עצמה.

Access control – the need:

ה-Access control, authorization, הוא האופן בו שרת האפליקציה נותן גישה לתוכן או פונקציונאליות למשתמשים מסויימים ולחלק לא.

- המפתח לא תמיד מודע לכך שהוא צריך ליישם access control. את האותנטיקציה עושים פעם אחת בתחילת סשן אפליקטיבי, אך access control צריך לעשות בכל גישה של המשתמש – האכיפה הרבה יותר דורשנית.
- פעמים רבות ה-access control מורכב ולכן לא מבצע גם את משימתו כהלכה, כיוון שמורכבות גוררת חוסר אבטחה (סיכוי גבוה לפירצות).

Complex and broken access control:

כאשר ישנו מנגנון AC מורכב המפוזר בכל מיני נקודות בקוד, כיוון שהוכנס בשלבים מאוחרים לקוד (כי המוקד העיקרי היה פונקציונאליות), ישנו סיכוי גבוה לשגיאות ובעיות.

...

Broken access control consenquences:

- המשתמש יהיה חשוף לתכנים אסורים לו – כמו מידע על משתמשים אחרים.
- לבצע פעולות שאסור לו.
- שינוי תכנים באתר.

Administrative actions are prime target:

משתמשים ישאפו להשיג גישה ברמה האפליקטיבית אל הפונקציונאליות האפליקטיבית כיוון שהיא בעלת יכולות רבות. לפיכך נושא זה צריך לקבל את הפוקוס הראשוני והמשמעותי של ה-AC.

Access control policy documentation:

- יש לפיכך להגדיר כבר מהשלב הראשון של הפרוייקט את ה-AC: קבוצות משתמשים, רולים שונים, פעולות מורשות לכל רול וכו'.
- לתעד את הנ"ל כדי ליהיה ברור ומסודר מה רוצים לאכוף – הוראות מסודרות למפתחים.

Centralized access control decision point:

כתפיסה רוצים מנגנון מרכזי של קבלת החלטות. את האכיפה חייבים לבצע בכל אחד מהאלמנטים האפליקטיביים, אך רוצים שאת קבלת ההחלטות האם לאפשר גישה יעשה מנגנון מרכזי, ואז הקוד האפליקטיבי יהיה מאוד פשוט – יגדיר למפתח גישה למנגנון המרכזי עם שאילתא מסויימת ופרמטרים מסויימים, והוא יענה האם מורשה או לא מורשה, וכל שנותר לקוד האפליקטיבי הוא אכיפת ההחלטה.

ה-policy enforcement point באפליקציה מחוברת ל-policy decision point. כך כאשר רוצים לבדוק מה ה-access control המיישם במערכת, צריך לבדוק רק את ה-policy decision point, ולא לבדוק במקומות רבים בקוד.

Access control policy implementation:

- בכל תשתית או סביבת פיתוח יש את ה-access control שלה, ומומלץ להשתמש בה. לרוב מנגנוני AC מאפשרים שני סוגים של AC: programmatic או ע"י קוד אפליקטיבי, או AC דקלרטיבי – הגדרה באיזשהו configuration file.
- ...

Access control enforcement:

- פעמים רבות ישנו הטרזאקציה כלשהי המורכבת מכמה שלבים, והנטייה היא לבצע את בדיקת ה-AC בצעד הראשון, ולהניח שאם עבר את הצעד הראשון אין צורך לבדוק באחרים שיש לו הרשאה. כמובן שהנחה זו מותנית בכך שיש session management מוגדר היטב ומנהלים states, וללא תנאים אלו, וכאשר רוצים לבנות מעל זה עוד שכבה, יש לבדוק בכל צעד את ההרשאה.
- חשוב לשמור באופן מאובטח על ה-roles, ושמידע זה יהיה unpredictable, ושיהיו ראנדומיים. אם התוקף יכול לבצע מניפולציה עם זהות המשתמש – מי הוא, מה ה-roles שלו וכו' – אזי מנגנון ה-AC קורס כי הוא בודק הרשאות לא של התוסף אלא של מי שמתחזה אליו.

Client side caching:

משיקולי יעילות בעיקר דפדפנים עושים caching לאינפורמציה. כאשר משתמש עוזב דפדפן, ומגיע משתמש אחר, בבקשת דף כלשהו יתכן ויעלה cache מהגלישה של המשתמש הקודם – ויתכן שכך נחשפה אינפורמציה. לכן לכל דף אפליקטיבי מומלץ להנחות את ה-browser דרך תגים שמכניס ב- HTTP response האם לעשות caching או לא (ברירת המחדל שכן יעשה cache).

SQL injection:

Injection attacks – מצבים בהם התוקף מנצל קלט שהאפליקציה שולחת למערכות אחרות כדי לתקוף את אותן מערכות.

כמעט כל אפליקצית web משתמש ב-DB, כחלק מה-data layer, שלרוב הוא SQL והגישה אליו בשאילתות כאלה בהתאם. האפליקציה משלבת את קלט המשתמש בצורה זו או אחרת מתוך ה-HTTP request שקיבלה מהמשתמש בשאילתא לבסיס הנתונים. פעולה זו נפוצה מאוד ולפיכך התקפה זו נפוצה מאוד גם כן.

אם תוקף מצליח להכניס קלט עם תכנים עויינים אז אל מנוע ה-SQL תגיע statement חוקיים מבחינה סינטקטית אך לא מבחינת תוכן.

SQL injections causes:

- בדיקת קלט לא תקינה ב-form.
- שימוש ב-plain text ו-stored queries.
- DB access control issues.

How to protect against SQL injection:

- Input validation: השלב האפקטיבי ביותר. ניתן לא לאפשר למשל גרש או כל מיני קרקטרים מיוחדים בשם המשתמש או סיסמא – לאפשר רק תווים אלפא-נומריים. אפשרות נוספת: הגבלת גודל הסיסמא – אבל כאן יש tradeoff בין מרחב הסיסמאות לבין מה מאפשרים לתוקף לעשות. על פניו עדיף מנגנון שימנע brute-force על סיסמאות ולצרף זאת להגבלת סיסמאות ל-8 תוים למשל. תפיסה זו היא positive security logic.
- Stored procedures \ strongly typed parameterized query APIs: העברת קלט לפרוצדורות שיבצעו input validation, יעטפו את זה כמחרוזת (בצורה חכמה ונכונה) – וכך למנוע סיכון נוסף.
- הגדרת הרשאות נכונות ב-DB: מניעת חריגה גם מתוקף שמצליח לחדור לאפליקציה מהרשאות האפליקציה ב-DB. כך אם תוקף מנסה לגשת לטבלאות שהאפליקציה לא מורשית לגשת אליהן, התוקף יחסם.

ישנן הרבה דוגמאות לתנאים לוגיים ב-SQL שיכולים להתווסף כ-".. OR" ולהפוך תנאי ל-TRUE בהכרח. לכן צריך לבנות מנוע שיודע לזהות בתוך קלט ביטוי לוגי שמחזיר TRUE. אם נרצה להחיל negative security logic נתקל באפשרויות רבות מאוד והאכיפה תהיה קשה יותר ובעייתית.

Injection attack steps:

...