אבטחת מערכות ויישומים ברשת - שיעור #8

נושא היום: Web Authentication:

הבחנה בין identification ל-authentication:

- Identification: אמצעי דרכו מזהים את זהות המשתמש בפני המערכת. זה יכול להבחר ע"י המשתמש כל עוד זה ייחודי באותה מערכת. אלמנט זה אינו סודי ואמור להיות גלוי.
- Authentication: השלב בו המשתמש מוכיח למערכת שהוא אכן בעל הזהות שהוא טוען שהוא. שלב זה משלים את שלב ה-identification, וזאת נעשה ע"י authentication factor – אמצעי הוכחה למערכת.

מדוע שלב ה-authentication כה חשוב? כיוון שבכל מערכת שמנהלת מידע משתמשים בד"כ תשמור מידע על אותם משתמשים, ועל המערכת לשמור על המידע הפרטי של המשתמשים שלא יזלוג למשתמשים אחרים או לגורמים אחרים לא מורשים. המידע האישי יכול להיות רגיש: מידע פיננסי, רפואי וכד'. ה-authentication הוא בסיס ל-access control.

User Authentication Factors:

המשתמש מוכיח את זהותו באמצעות authentication factor. ישנם 3 משפחות עיקריות:

- Something you know: מידע שהמשתמש אמור לזכור ולדעת – כמו סיסמא.
- Something you have: רכיב כלשהו שבידי המשתמש, כמו כרטיס חכם, כרטיס אשראי וכו'.
- Something you are: זיהוי ביומטרי כלשהו של המשתמש כמו טביעת אצבע, זיהוי רשתית וכו'. אמצעים אלו דורשת חומרה ודיוקה לקוי. כמעט ולא נעשה בכך שימוש בעולם ה-web.

שילוב שני אמצעים ומעלה משתי קטגוריות נקראת strong authentication.

Authentication Procedures:

מקובל לחלק את האותנטיקציה לכמה קטגוריות:

- Two-party authentication: בתהליך ההזדהות ישנם שני צדדים – המשתמש והמערכת. One-way או two-way: זיהוי הלקוח מול השרת, או זיהוי כפול – גם שרת מול לקוח. בתהליך צריכה להיות היכרות מוקדמת בין הלקוח לשרת, נדרש תהליך registration.
- Three-party authentication: עבור מצבים בהם לא רוצים היכרות מוקדמת. דוגמא לכך היא זיהוי על בסיס certificate בפרוטוקול SSL, כאשר ה-CAs הם הצד השלישי שמזהה את השרת מול הלקוח (לכיוון השני – בד"כ לא מיושם). בעולם ה-web זיהוי לקוח מול שרת יהיה לרוב דרך 2-way.
- Single sign ON: האידאל היא הזדהות דרך גורם אחד מרכזי, וכל שירות שדורש זיהוי יקח את הזהות מאותו שרת מרכזי אליו הזדהינו.

HTTP Authentication:

כאשר ישנו משאב ב-web server שהוגדר לגביו access-control לגבי משאבים באותו שרת, ומתקבלת בקשה לאותו משאב, אז אם מתקבלת הודעה 401 – unauthorized והמשתמש מתבקש להזדהות בפני השרת. פרוטוקול HTTP מגדיר שני סוגי הזדהות:

<u>Basic HTTP authentication</u>:

כאשר המשתמש מתבקש להזדהות כלפי השרת, ה-browser מקפיץ חלון user/pass, המשתמש מכניס את הנתונים שלו, הדפדפן מקודד את האינפורמציה ב-64bit ושולח לשרת במסגרת אחד ה-headers אותה מצרף ל-HTTP request. הנתונים נשלחים **מקודדים ולא מוצפנים**. השרת מקבל את ה-header (מסוג authorization), מבצע decoding לאינפורמציה, משווה לנתונים במסד נתונים שלו, בודק את ה-role של המשתמש (אם אושר) ובודק האם הוא רשאי לגשת לאותו משאב. אם כן – נותן את המשאב (הודעה 200 – אישור; אם הוא לא מאושר – נשלחת שוב הודעה 401 – unauthorized).

אם נשלחה בקשה עבור משאב אחד, עבור משאב שני השרת יבקש שוב מהמשתמש אותנטיקציה. בכדי להקל על המשתמש, כדי שלא יצטרך להזין פרטים אלו שוב ושוב, הדפדפן שומר (כל עוד חלון הדפדפן פתוח) את הנתונים שהוכנסו וכך הדפדפן יודע ליצור את ה-header המתאים בלי לבקש מהמשתמש user/pass.

אם לא נבצע את האותנטיקציה של basic http authentication מעל ערוץ לא מאובטח, הנתונים יהיו גלויים שכן הם מקודדים ולא מוצפנים. לכן לכל הפחות יש להשתמש ב-SSL להעברת הנתונים בין הלקוח לשרת. ה-SSL עוזר במניעת reply, כלומר לקחת בקשה קיימת (הכוללת הזדהות) ולשלוח אותה שוב – בקשה זו תקינה ולכן השרת יחזיר למשתמש (הלא חוקי) תשובה.

Challenge-response:

בפרוטוקול זה המשתמש מבצע identification. הוא מקבל כתגובה challenge, והמשתמש מפעיל פונקציה, יכולה להיות חד-כיוונית, על ה-challenge ומפתח (something you know). התוצאה נשלחת לשרת שיודע את האתגר שנשלח והמפתח הסודי של המשתמש. אם החישוב שביצע המשתמש היה hash, השרת יוכל לבצע בעצמו את החישוב ואם קיבל את אותה תוצאה – המשתמש אומת.

מדוע מנגנון זה טוב יותר מהעברת ססמא מעל הרשת? כך הרבה יותר קשה לזהות את הססמא מתוך הנתונים, והחוזק עדיין תלוי בחוזק המפתח. כל בקשת הזדהות הוא challenge אחר, והשרת אמור לזכור במה השתמש. נסיונות reply לא יעבדו בשיטה זו.

<u>HTTP Digest Authentication</u>:

משתמש ב-challenge authentication. כאשר המשתמש מקבל את ה-challenge הוא יחשב את התגובה שלו באמצעות פונקציית hash קריפטוגרפית – MD-5 למשל, הכוללת בין היתר את ה-nonce. סה"כ שלוש הפעלות של MD-5.

The Autherization Request Header:

מבחינת המשתמש הוא לא יודע איזה סוג אותנטיקציה, הוא מקבל בקשת user/pass. ה-browser לוקח את הנתונים הללו ומחזיר את ה-response הכולל רשימה של פרמטרים (מפורט במצגת) ועל סמך נתונים אלו השרת יכול לחשב את כל מה שצריך כדי לבצע אימות על הזדהות המשתמש. ה-response מקודד על ידי קידוד הקסדצימלי, וע"י הצפנת 128 ביט סה"כ מתקבלים 32 תוי hex.

<u>HTTP digest authentication advanteges</u>:

השרת צריך לשמור רשימת nonces ששלח ושלא יהיה מוכן לקבל שנית (ניתן להחזיק time-stamps על ה-nonces). צריך למצוא איזון מתאים בין זכרון שישמור את ה-nonces שנשמרו כדי למנוע reply, nonces שנשלחו.

Application layer Authentication:

<u>Form based authentication</u>:

הטכניקה הפופולרית ביותר. כאשר כל הנושא ה-login מנוהל ברמה האפליקטיבי, גם ה-logout יהיה אפליקטיבי. כאן כמעט תמיד חייב להיות שבקשת ה-login תעבור ב-SSL כיוון ששם עוברים ה-user/pass בגלוי. ישנן אפליקציות מעטות שמיישמות מנגנוני digest-authentication בקוד jsp ב-browser, אך הרוב מעבירים גלוי על גבי SSL.

יתכן מצב בו המשתמש סוגר את ה-browser אך אם לא עשה app logout השרת חושב שהוא עדיין פתוח ורץ.

<u>Form based authentication and cookies</u>:

כיצד מנהליך את ה-session האפליקטיבי המאפשר לחבר את הבקשות השונות של המשתמש לסשן אפליקטיבי אחד? באמצעות cookies. ה-cookies מחזיקות ticket שהשרת יוצר. ה-ticket מכיל username, time of issue, period of validity, user's source address ובנוסף secret. היקט צריך להיות חתום ע"י דיגיטלית, סימטרית או אסימטרית. הטיקט הוא message digest hash קריפטוגרפי (למשל) על כל הנתונים הללו. השרת יוכל לבדוק את אמינות ושלמות ה-ticket. אם כן, יוכל להוציא משם את הנתונים.

<u>מדוע צריכים ניהול סשן אפליקטיבי</u>:

עובדים מעל פרוטוקול HTTP שהוא stateless, וגם באספקט האבטחה. השרת יקצה למשתמש session-id, ישלח אותו אל המשתמש כ- response header, ויבקש מהדפדפן לשלוח לו בכל בקשה חדשה את אותו session-id, וכך ידע השרת לחבר כל בקשה לסשן האפליקטיבי בשרת המנוהל עבור אותו משתמש. כלומר: session-id הוא הדרך "לשרשר" requests על אותו סשן אפליקטיבי.

הצורך ב-session management הוא גם אפליקטיבי, כמו לשמירת user preferences. הדרך למימוש session management על בסיס session id יכול להיות על בסיס 3 אפשרויות:

- שילוב ב-session id בתוך ה-url, וכך הוא נשלח כחלק מה-url.
- שילוב ה-session id כ-hidden parameter בתוך ה-form.

- שימוש במנגנון ה-cookies: כאשר השרת שולח cookie לדפדפן, וה-cookie.

URL-based solutions:

מנגנון כזה נוח אך אינו מאובטח:

- ה-url נשמר בהרבה caches בדרך (לוגים של שרתים ו-firewalls).
- ה-url יכול להיות חשוף ב-refferer.

ה-session id יהיה חשוף כך להרבה גורמים לאורך כל הדרך. הבעיה מכאן היא שניתן למצוא session id ולהשתמש בו (session hijack).

Form-based with hidden parameter:

פותר את בעיית הופעת ה-session-id בכל כך הרבה מקומות בעייתיים, ולכן אופציה זו טובה יותר. ההגבלה היא שניתן להכניס רק ב-form.

סיכום:

ה-session id הוא אלמנט של ה-session authentication, ואם גונבים אותו בזמן שסשן פתוח, זה שקול למציאת user/pass.

Attacks techniques on session id:

- Interception: אם העברת הפרמטרים לא נעשית מעל SSL, תוקף יכול לחטוף סשן אקטיבי וכך להתחזות למשתמש. באפליקציה שיש בה סשן אפליקטיבי ותהליך הזדהות – משלב הלוגין לשלב הלוגאאוט חייבים לעבוד מעל SSL, אחרת ה-session id חשוף.
- קוד malicious שיכול להגיע ל-cookies של המותקף ולשלוח אותן לתוקף (cross-site scripting attack).
- אם ה-session ids לא אקראיים או מספיק ראנדומיים, ניתן למנות עליהם.
- Session fixation (לא הסביר).

Protection on session id:

- הצפנת תקשורת כנגד interception.
- דאגה לאקראיות וגודל מספק ב-session id כדי למנוע מניה.
- לדאוג להיות לא חשופים ל-cross-site scripting.
- לדאוג ש-session id יוחלף בתדירות מספקת כך שגם אם באמצעות אנומרציה ניתן למצוא אותו, זמן האנומרציה לא יספק עד שיוחלף המזהה לאחד אחר.
- לדאוג שה-session id לא יימצא בלוגים.
- רק השרת קובע את ה-session id, אחרת יש חשיפה ל-session fixation.

Cookies:

זהו token או קובץ מידע קצר... במצגת יש תיאור מדוייק.