אבטחת מערכות ויישומים ברשת - שיעור 9#

נושא היום: Web Session Management:

פעם שעברה דיברנו על כך שכל HTTP request הוא בפני עצמו, ולכן כדי ליצור קשר בין בקשות שונות של המשתמש לסשן אפליקטיבי של המשתמש, יש session id – מתבצע אחרי login של המשתמש אל המערכת, והוא אמור להיות מוחזר לשרת בכל בקשה, ויש לשמור עליו היטב. כך הוא מאפשר ל- web app. לקשר את הבקשה לסשן אפליקטיבי.

אמרנו שה-session id הוא גם Identification אך גם authentication של המשתמש אל מול השרת. ניתן לראות אותו כמעיין סיסמא למשך חיי הסשן ולכן יש להתייחס אליו בהתאם ל-credentials – לשמור על סודיותו, שלמותו אחרת תהיה חשיפה לגניבת ה-session id, והגונב יכול להתחזות למשתמש בעל אותו session – ישלב את המזהה בבקשות שלו, ויראה לשרת כמו אותו משתמש.

ישנם 3 מנגנונים בסיסיים כדי להעביר את המזהה אל השרת וחזרה:

- שילוב ה-session id ב-url: יתרון הוא שזהו מנגנון גנרי וקל למימוש, וחיסרון הוא שהמזהה כך חשוף לחלוטין – ב-cache של מערכות בדרך, בהיסטוריה של הדפדפן וכו'. הוא גנרי אך מאוד חשוף לגורמים לא מורשים.
- שילוב ב-hidden parameter ב-form: החשיפה כך הרבה יותר קטנה, לא חשוף ב-history, referrer, הדפסה, לוגים (לרוב), כן חשוף ב-source בבקשות שנעשות ב-post method (לא חשוף ב-get method).
- HTTP cookies – תחילת ההרצאה היום.

Cookies:

Cookie – פיסת אינפורמציה שנשלחת מתוכנית אחת לשניה ומטרתה היא שהתוכנית השניה תחזיר אותה לתוכנית הראשונה. תוכן ה-cookie לא אמורה להיות מובנת לתוכנית ב', כל מטרתה היא שתוכנית ב' תחזיר אותה מתישהו ל-א' בכדי לגרום לה להתייחס לה בהקשר מסויים. לא אמורה להיות מובנת למקבל, אלא רק להחזיר אותו (נראה הסתייגויות לכך).

דוגמא מהחיים האמיתיים: בשמירת חפצים, כאשר מקבלים פתק, הוא חסר משמעות למקבל הפתק. כאשר ניתן את הפתק לאחראי, הוא ידע להחזיר את החפץ השמור אלי, אך מבחינתי הוא חסר משמעות.

HTTP Cookies:

כאשר בנו את מנגנון ה-HTTP הבינו את הצורך ביכולת קישור בקשות זו לזו. על כן הוגדר מנגנון HTTP cookies שנשלח מהשרת אל הדפדפן, כאשר הדפדפן אמור לשלוח בכל בקשה לשרת גם את ה-cookie. דרך ה-cookies ניתן גם ללא ידיעת זהות המשתמש לעקוב אחר פעילות המשתמש וללמוד על אופי הפעילות שלוף ולכן ניתן לנטרל cookies, אלא שהשימוש בו כיום נרחב מאוד בעולם ה-web ולא רק ל-user-session management אלא גם לפרסונליזציה. פרסונליזציה מתבססת על cookies, כך שהשרת שולח את ה-cookie לדפדפן של המשתמש, וכאשר הוא מבצע בקשה חדשה יש את המידע כדי לבצע פרסונליזציה. זוהי פונקציונאליות קלאסית של .web app, אז אמנם ניתן לבטל את ה-cookies כדי לשמור על הפרטיות, אך יש לכך חיסרון. המשתמש יכול להגדיר מאילו אתרים מוכן לקבל cookies ומאילו לא.

Request a Web Page:

הדפדפן יוצר HTTP request, כאשר שם השרת מופיע ב-host header. הדפדפן עושה DNS resolving, פותח TCP connection שעל בסיסה שולח את בקשת ה-HTTP. השרת מקבל את הבקשה ויוצר HTTP response – בשורה הראשונה נקבע הפרוטוקול, status code, status description. לאחר מכן מופיע סט של headers ובתור body – תוכן הדף שביקש המשתמש. אחד ה-header שיכול השרת להחזיר לדפדפן הוא ה-set cookie header, שמשמעו הוא בקשה מהדפדפן לשמור אצלו תוכן מסויים. תוכן ה-cookie הוא לרוב key-value pair, כלומר name=value כאשר cookies יכולים להכיל רבים כאלו. הדפדפן אמור לשמור את ה-cookie אצלו, כאשר בבקשות הבאות אל אותו domain יופיע header חדש שהוא cookie header, שם מחזיר את ה-name=value שקיבל מקודם (או כמה כאלה).

Reset/Modify the HTTP Cookie:

בכל response יכול לחזור סט cookie. אם נקבל name=value עבור name חדש, זה יתווסף. אם נקבל name קיים עם ערך חדש, זה אומר לדפדפן לדרוס את הערך הקיים עם הערך החדש, כלומר מודיפיקציה לתוכן ה-cookie. ל-web server תמיד יש את האופציה לעדכן את ה-cookie שהוא שלח. הדפדפן שומר לכל domain את התוכן שהוא שלח.

באמצעות קוד JavaScript ניתן לקרוא את תוכן ה-cookie, למשל עבור עגלת קניות לעדכן את תוכן עגלת הקניות. כך השרת לא "עובד קשה", כי מאחסן ב-cookie של המשתמש את תוכן העגלה במקום לגשת לשרת ולמשוך משם את המידע. ל-JavaScript יכולת לשנות גם את תוכן ה-cookie, וגם למחוק.

Cookies Options:

ניתן להנחות את הדפדפן מתי לשלוח את ה-cookie, למשל לשלוח רק עבור בקשות מסויימות כמו של path מסויים, sub-domain מסויים וכו'. כך מנחים ומגבילים את הדפדפן מתי לשלוח את ה-cookie. ניתן להנחות את הדפדפן גם מתי לזרוק את ה-cookie, ע"י discard – למשל, כאשר המשתמש סוגר את הדפדפן, עליו לזרוק את ה-cookie. ניתן להגדיר גם max age – תוקף ל-cookie, וניתן גם לעשות שילוב בין max age ל-discard. אם לא ניתן max age אז הדפדפן שומר אותו בזיכרון ולא ב-file system. אם מופיע max age ולא discard, ה-cookie כן נשמר במערכת הקבצים. כאשר max age = 0 זה אומר לדפדפן לא לשמור כלל את ה-cookie אלא למחוק אותו לגמרי. בשלב ה-logout, אם יש session id cookie, נרצה לאמר לדפדפן למחוק אותו. זאת נעשה ע"י max age = 0.

Cookie Poisoning:

ניתן לתפוס cookies ביציאתם החוצה, וניתן לשנות את תוכן ה-header, בין היתר ה-cookie header, וניתן לשנות את תכני ה-cookies – וזו הרעלת תוכן ה-cookie. מה האלטרנטיבה למי שרוצה להשתמש במנגנון ורוצה להיות מוגן בפני poisoning? להשתמש בחתימה דיגיטלית שתתן data integrity, וזאת בכדי לדעת שמשתמש לא שינה את תוכן ה-cookies.

ניתן להשתמש בחתימה דיגיטלית סימטרית כי מי שמייצר אותה הוא זה שיבדוק אותה – השרת, ואין צורך באסימטרית. הוא יכול לחשב digest ולהצפין אותו או בדרך כלשהי להשתמש ב-crypto hash כדי לייצר חתימה דיגיטלית סימטרית. השרת יחשב את ה-digest יחד עם מפתח סודי על תוכן ה-cookie ויצרף אותו ב-key=value. כשיגיע ב-cookie header, השרת יאמת את המידע.

Cookies for user authentication:

שמירת אינפורמציה בשרת ושמירת session id בלבד בעלת חסרונות: אם השרת נופל, כל אינפורמצית השרת הלכה לאיבוד, כולל אינפורמציה הסשנים שנשמרה על אותו שרת. חסרון נוסף הוא שאם עובדים עם load balancer לנהל סשנים על שרתים, צריך להיות במצב בו אם בקשה אחת התחילה להיות מטופלת ע"י שרת מסויים, ההמשך צריך להתנהל מול אותו שרת – מגבלה על חלוקת העומסים בין השרתים, מה שנקרא server stickiness.

Cookie theft – through sniffers:

כאשר משתמשים ב-session id כדאי להשתמש במנגנון יצירה שכבר נבדק, אך צריך לבדוק את תכונותיו ולראות שהוא טוב מבחינה קריפטוגרפית. חשוב שה-session id הקריפטוגרפי המשמש ל-user session management יועבר מעל SSL לכל אורך הסשן האפליקטיבי מהיצירה עד אינוולידציה ב-logout. ל-cookies יש אופציית secure שאמור להגיד לדפדפן לשלוח את ה-cookie מוצפן. אבל, אם לא מגדירים באופן מפורש את מימוש ה- secure option, היא לא מופעלת.

Cookies Theft using cross-site scripting:

בהתקפה זו התוקף גורם ל-.web app לשלוח למשתמש המותקף דף HTML עם קוד JS עויין שיצר התוקף. כיוון שהדפדפן קיבל דף מהשרת, הוא מתייחס אליו כדף חוקי ולכן כל מה שמותר לדף חוקי לבצע יהיה מותר גם לאותו קוד עויין שהגיע אליו כחלק מאותו דף. לקוד ה-JS יש יכולת לקרוא ולשנות או לכתוב לתוך ה-cookies. כך מספיק שתהיה יכולת לקרוא את ה-cookie ואז לקוד של התוקף יש דרך לגנוב את ה-cookie – יוכל לשלוח את התוכן לתוקף.

Session hijacking by cookie theft: מפורט במצגת.

דוגמאות:

MMS spoofing and billing:

שיטת האותנטיקציה המתוארת כאן היא שליחת סיסמא חד פעמית ב-SMS – something you have. אבל, מימוש לא נכן של שמירת האינורמציה ב-cookies הוביל למצב בו ניתן היה לשלוח MMS ולחייב מישהו אחר.

התסריט: משתמש נכנס ל-web app. ובונה הודעת MMS. כדי לשלוח אותה הוא נדרש להזדהות כדי לחייב את החשבון. ע"י שרת ההזדהות והכנסת מספר הטלפון של המשתמש נוצר OTP שנשלח למשתמש, והוא צריך להכניס אותו כ-authentication factor. כעת המשתמש הזדהה, ולכן האפליקציה תשלח את ה-MMS. המשתמש מצליח לגרום למערכת לחייב מישהו אחר. כיצד?

הבעיה היא כאשר השרת עושה set cookie עם המספר לדפדפן, והשרת מבקש אישור חיוב. המשתמש מאשר את החיוב ולכן החשבון שלו מחוייב וה-MMS נשלח. ב-cookie השרת הכניס את המספר של המשתמש באופן לא חתום – ולכן פתוח ל-cookie poisoning ע"י שינוי האינפורמציה ב-cookie. כך השרת רואה את ה-MSISDN אותו אמור לחייב כאשר המשתמש שינה אותו למישהו אחר – והוא זה שיהיה מחוייב.

לסיכום:

הבעיות היא חוסר הגנה על ה-credentials או על ה-tokens, הכוללים session ids במשך חייהם.

...

Authentication relies on secure communication and credential storage:

ב-chalange response חובה להצפין את ה-login request, אחרת תוקף יקליט את ה-chalange, response וימנה את כל הססמאות עד שימצא. כיוון שססמאות משתמשים מוגבלות ל-6-8 תוים, קל למנות מעליהן. גם ה-credentials צריכים להיות מוצפנים.

How to protect against:

- חשוב שיהיה מנגנון אותנטיקציה יחיד, ואז ניתן לנתחו ולבדוק את החוזקות והחולשות שלו. כדי שניתן יהיה לבצע את הניתוח בצורה קלה, כדי לא להקשות על הניתוח.
- כל נושא השאלות הפרסונליות הוא בעייתי, ולהסתמך עליו בלבד לא רצוי. בכל אופן, הן צריכות להיות מוצפנות בדיוק כמו סיסמאות, כי באמצעותן ניתן להחליף סיסמא ולכן צריך להגן עליהם כמו שמגנים על סיסמאות.
- לא לסמוך על ip addresses, referrer headers כי הם חשופים.
- לא להכניס את ה-credentials ב-GET אלא ב-POST שם החשיפה הרבה יותר נמוכה.
- כל הדפים שמבצעים אותנטיקציה צריכים להיות מסומנים כ-non-cachable, בפרט לא לדפדפן – כי אם יעשה זאת, יעשה cache גם ל-credentials, וזהו פתח לפירצה. יש להנחות את הדפדפן גם לא לבצע auto-complete לדפים אלו, ואז הדפדפן שומר את ה-user/pass.
- ...

Logout and session timeout:

ככל שה-session id חשופים לזמן קצר יותר, כך בטוח יותר. לכן צריך להחליף את ה-session id כל פרק זמן, ולדאוג לבצע logout בסוף סשן אפליקטיבי. אם המשתמש שכח לבצע logout, חשוב לממש inactivity timeout – אחרי פרק זמן מסויים של חוסר פעילות, ייסגר הסשן האפליקטיבי. בסגירתו חשוב לבצע:

- לבצע invalidation ל-session id בשרת.
- לבצע אינוולידציה גם בלקוח, אם אפשר, ע"י max age = 0 ב-cookie.

...