אבטחת מערכות ויישומים ברשת / תרגיל בית 3#

אריאל סטולרמן

1)

להלן מספר סיבות מדוע web applications הם משמעותית פגיעים יותר מאשר enterprise applications שנבנו בטכנולוגיות client-server:

- Common availability and access: החיבור לרשת האינטרנט מאפשר ברמה העקרונית לכל אחד בעולם לגשת אליו – החשיפה מבחינת הסיכונים גדולה. מעבר לנגישות הגבוהה דרך האינטרנט שיש לאפליקציות רשת, ברשת ישנם כלי תקיפה רבים העומדים לרשות התוקפים. אם בעבר האקרים היו צריכים רקע טכנולוגי משמעותי כדי לפתח כלי תקיפה, כיום ישנם כלים קיימים שעם ידע טכנולוגי מינימלי ניתן לתקוף באמצעותם. כלומר הנגישות עלתה והידע הנדרש ירד.
- Exposure to client side code: באפליקציות שאינן אפליקציות רשת, כדי להבין את הקוד האפליקטיבי בשביל לתכנן התקפה מוצלחת, צריך לבצע reverse engineering על ה-executable המותקן על מחשב התוקף. פעולה זו היתה מורכבת מאוד שכן ישנה שליטה על התפוצה שלו, וגם עבור תוקף שמשיג אותו הבנת הקוד האפליקטיבי מתוכו מסובך מאוד בסדרי גודל של שבועות עד חודשים של עבודה. לעומת זאת באפליקציות רשת ישנה נגישות לקוד, כיוון שבקבלת דף web ניתן לצפות ב-source שלו ולראות בקלות את כל הקוד האפליקטיבי, והבנת קוד HTML או JavaScript אינו מסובך (ב-client side). ממנו ניתן ללמוד על מבנה הפרמטרים וה-requests – ידע רב שיעזור לתוקף לתכנן את ההתקפה.
- Developers think in terms of client-server security: אפליקציות רשת מבוססות על Client-server interaction בארכיטקטורה מורכבת יותר מאשר client-server קלאסי, והארכיטקטורה הזו הטרוגנית – מורכבת ממוצרים שונים, טכנולוגיות שונות ופרוטוקולים שונים, והמפתח צריך להכיר טכנולוגיות רבות ואת אופן השימוש המאובטח בהם. מפתחים המגיעים מעולם client-server קלאסי עלולים לקחת עמם גישות לא מתאימות לעולם ה-web application:
  - Client side validation: ב-client-server קלאסי ניתן לבצע ולידציה בצד הלקוח באופן יחסית מאובטח, שכן לפריצתו יש צורך בתהליך קשה של reverse engineering. לעומת זאת באפליקציות רשת נגיש וקל לראות ולהבין אם ישנה ולידציה בצד הלקוח (קוד הלקוח פתוח), ויש צורך בולידציה בצד השרת.
  - אפליקצית רשת מתבססת על פרוטוקול HTTP שהוא stateless, session-less – טוב לתמיכה בבקשות מרובות בו"ז, אך הצורך לחבר requests למשתמשים יוצר בעיית ניהול ומקשה על authentication ו-authorization. חיבור לא נכון למשתמש יבטל את רלוונטיות האותנטיקציה, ואז לא ניתן ליישם גם access control. העבודה מעל HTTP יוצרת מורכבות רבה בניהול sessions המשליכה באופן חמור על זיהוי והרשאות לקוחות.

2)

ההבדל העקרוני בין HTTP basic authentication ובין FORM based authentication הוא שבראשון בכל בקשת HTTP שנשלחת, נשלחים גם שם המשתמש והסיסמא, בעוד בשני פרטים אלו נשלחים רק בהתחברות הראשונית (לאחר מכן נשלחת רק ה-cookie, או פרטיה ליתר דיוק ב- cookie header כמזהה):

- ב-HTTP basic authentication כאשר המשתמש מתבקש להזדהות כלפי השרת, הדפדפן מקפיץ חלון login, 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 ובודק האם הוא רשאי לגשת לאותו משאב. אם כן – נותן את המשאב, אחרת שולח הודעת אי-אישור.
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, כלומר העברת ה-credentials מתבצע "מאחורי הקלעים".
- ב-Form based authentication, השרת (ה-Ticket server אליו מועבר המשתמש) מעלה HTML form בפני המשתמש המכניס את נתוניו, נפתח session אפליקטיבי ונשלח לדפדפן המשתמש cookie המכילה ticket – נתונים אודות ה-session שבעיקרם ה-session id. כעת בכל HTTP request מהמשתמש מצורף כחלק מה-headers ה-cookie header, שהוא המזהה בפני השרת את המשתמש וליתר דיוק את ה-session האפליקטיבי אליו משוייך. ניתן לאמר שבניגוד ל-basic, ה-credential שמועבר לאורך ה-session יהיה ה-session id ולא ה-user/pass שוב ושוב.
(כפי שניתן לראות בהמשך התרגיל, ה-session id הינו מידע רגיש וסודי שיש לשמור עליו כמו על סיסמא).

ישנו צורך להצפין את ה-HTTP request authentication כאשר תהליך הזדהות הוא ב-HTTP basic authentication, וזאת כיוון שכאמור בשיטה זו שם המשתמש והסיסמא מועברים מקודדים בבסיס 64bit ולא מוצפנים, ובכדי לשמור על המידע הסודי מפני תוקף המאזין על ערוץ התקשורת, שיוכל מהנתונים הללו לשחזר את שם המשתמש והסיסמא, יש לעבוד מעל ערוץ מאובטח, למשל ע"י הצפנת SSL.

3)

מנגנון ה-Challenge-Response הוא פרוטוקול אותנטיקציה בו נעשה שימוש בתהליך HTTP Authentication ובו המשתמש מבצע identification מול השרת. הוא מקבל כתגובה challenge מהשרת – איזושהי מחרוזת, והמשתמש מחשב תגובה מתוך אותו אתגר ומפתח סודי משותף ללקוח ולשרת. התגובה יכולה להיות למשל הפעלת cryptographic hash function או הצפנה סימטרית כלשהי. התוצאה נשלחת לשרת שיודע את האתגר שנשלח והמפתח הסודי של המשתמש. השרת יוכל לבצע בעצמו את החישוב על בסיס האתגר ששלח והמפתח הסודי שברשותו, ואם קיבל את אותה תוצאה כמו תשובת המשתמש – המשתמש אומת.

הסיבה לשימוש במנגנון ה-Challenge-Response היא כיוון שפתרון זה נותן הן סודיות, שכן לתוקף המאזין לתקשורת יהיה קשה (עד בלתי אפשרי בזמן סביר) לשחזר מתוך התגובה המועברת את המפתח הסודי בכדי שיוכל ליצור בעצמו תגובות לשרת ולהשתלט על session קיים או להתחזות ללקוח האמיתי. כמו כן הפתרון, תחת הנחה שהאתגרים יהיו בלתי צפויים ולא יחזרו על עצמם, מונע התקפת reply ע"י תוקף ש"מקליט" את תגובת הלקוח ומנסה לשלוח אותה שוב אל השרת, שכן כאשר ישלח התוקף תהיה תשובתו לא תקפה כי זו תשובה לאתגר ישן ולא תקף. (למשל, אם השרת שולח בכל פעם מחרוזת ראנדומית חדשה כאתגר).

4)

הטכניקה המקובלת באינטרנט לניהול session היא ע"י פתיחת session אפליקטיבי המשוייך למשתמש מסויים, כך שכל הבקשות מאותו משתמש משוייכות ל-session יחיד. שרתים משתמשים ב-session id כמזהה ייחודי ל-session אפליקטיבי, בכדי להחזיק בו מידע אודות ה-session: איפיונים יחודיים של המשתמש, היסטוריה, שמירת מצב הלקוח באפליקציה וכו'. השרת מקצה למשתמש session-id, שולח אותו אל המשתמש, ומבקש מהדפדפן לשלוח חזרה אל השרת בכל בקשה חדשה את אותו session-id, וכך ידע השרת לחבר כל בקשה ל-session האפליקטיבי בשרת המנוהל עבור אותו משתמש. ישנן שלוש שיטות עיקריות לניהול session בסביבת הרשת:

- שילוב ה-session id בתוך ה-URL, וכך הוא נשלח כחלק מה-URL: אמנם שיטה זו מיידית ונוחה לשימוש, ויש צורך להשתמש בה למשל במקרה שמנגנונים אחרים (כמו cookies) אינ ם זמינים (למשל בשל privacy settings של המשתמש), אך יש בה כמה בעיות: ה-URL נשמר בהרבה מטמונים לאורך הדרך מהלקוח לשרת, בלוגים של שרתים ו-firewalls ואף עלול להשלח ב-HTTP referrer לשרתים אחרים. הבעיה בחשיפת ה-session id היא שזהו מידע רגיש, וחשיפתו עלולה לחשוף את המשתמש להתקפת session hijack – התחזות למשתמש מעל session קיים.
- שילוב ה-session id כ-hidden parameter בתוך ה-form: יתרונות מנגנון זה הם שהוא עובד גם במקרה של הגבלות אבטחה (כמו איסור שימוש ב-cookies) כאשר ה-session token והמידע עליו נשלחים תחת hidden parameters. יתרון נוסף הוא שחשיפת ה-session id בשיטה זו הרבה יותר קטנה לעומת השיטה הקודמת – לא יהיה חשוף בהיסטוריה, לוגים, referrers. הטופס נשלח אל השרת באמצעות HTTP POST command, מכאן בא אחד החסרונות והוא תקורה גבוהה יחסית בשל הצורך לשלוח שוב ושוב forms ע"י POST commands וכך לפגוע ביעילות. חסרון נוסף הוא שישנה בכל זאת חשיפה ב-source בבקשות שנעשות ב-HTTP POST (לא ב-GET).
- שימוש במנגנון ה-cookies: זוהי השיטה הנוחה והבטוחה ביותר מבין השלוש. בשיטה זו כאשר הלקוח מבצע login, השרת שולח לדפדפן cookie. כעת בכל בקשה שתשלח מהלקוח, ה-cookie תשלח יחד איתה כדי לזהות את הבקשה כחלק מ-session אפליקטיבי אחד. ה-cookie מכיל ticket המחזיק בפרטים שונים, כגון שם המשתמש, זמן הנפקת ה-cookie, תוקפה, כתובת המקור של המשתמש, ובנוסף חתימה של כל הפרטים הללו באמצעות סוד כלשהו. ע"י digest קריפטוגרפי יכול השרת לבדוק את אמינות ושלמות ה-ticket ולמשוך משם את הנתונים – יתרון על פני השיטות האחרות. חיסרון עיקרי הוא שחלק מהדפדפנים (או ליתר דיוק – מהמשתמשים) חוסמים את האפשרות לשימוש ב-cookies ב-privacy settings של הדפדפן, וכך שיטה זו נחסמת לפעמים לשימוש.

יצויין כי השיטה המקובלת ביותר לפתיחת וניהול session היא form based authentication ו-cookies: שלב האותנטיקציה מתבצע תוך שימוש ב-form, כאשר פרמטרי ה-login עוברים מעל ערוץ מאובטח, וכתגובה (כאמור בנקודה השלישית לעיל) השרת שולח cookie אל הלקוח.

פרוטוקול ה-HTTP אינו כולל session-management מובנה כיוון שמהגדרתו פרוטוקול זה הוא stateless, כלומר כל HTTP request/response היא בפני עצמה. מעל פרוטוקול כזה מובן כי לא ניתן לנהל session, הדורש הסתכלות על scope רחב הכולל בתוכו כמה requests/responses (אלו תחת אותו session).

5)

משמעות הטענה: "ה-session token של המשתמש מהווה למעשה את הסיסמא הזמנית של המשתמש" היא שה-session token (או session id) הופך לאחר שלב האותנטיקציה הראשונית של המשתמש להיות אמצעי האותנטיקציה של המשתמש לאותו session אפליקטיבי שנפתח עבור אותו משתמש בשרת ע"י אפליקצית הרשת. כמתואר בתשובה לשאלה 4, מרגע שנפתח session עבור לקוח מסויים, בכל בקשה של הלקוח מצורף גם ה-session id בכדי לזהות את אותה בקשה כמשוייכת ל-session אפליקטיבי מסויים (אופן העברת ה-session id משתנה כאמור: דרך ה-URL, hidden form parameter או באמצעות מנגנון ה-cookies). הזמניות של אותו session id עומדת אל מול ה"קביעות" של סיסמת המשתמש בשרתי האפליקציה, באמצעותה מבצע המשתמש אותנטיקציה ראשונית שאחריה מקבל את ה-session id: ה-scope של ה-session id הוא מרגע ה-login עד רגע ה-logout (וגם זה לא תמיד נכון, תתכן החלפה במהלך session יחיד מאילוצי אבטחה).

המשמעות מבחינת אבטחת מידע היא שיש להגן על ה-session id באותה מידה כמו שמגינים על כל המידע הרגיש האחר של המשתמש כגון סיסמאות, וכי חשיפת מידע זה מהווה פירצת אבטחה ועלולה להוות בסיס לתקיפה, כמו session hijack – התחזות למשתמש בתוך session מסויים ע"י העתקת ה-session id שלו וצירופו לבקשות של התוקף. כך התוקף למעשה ייראה כמו המשתמש המקורי בתוך session מסויים. דוגמאות נוספות להתקפה: מניה מעל ה-session id השונים, קוד זדוני השולח cookie מהמותקף אל התוקף (cross-site scripting attack), session fixation וכו'. בשל רגישות ה-session id, יש לשמור עליו באמצעי אבטחה שונים: הצפנת התקשורת בכדי למנוע גילוי של ה-session id ולמנוע בכך interception (התחזות למשתמש), דאגה למספר גדול מספיק עבור session id ואקראיות בבחירתו כדי למנוע מניה, החלפת ה-session id בתדירות גבוהה מספיק כך שגם ע"י מניה לא ניתן יהיה להשיגו, לדאוג שה-session id לא יימצא בלוגים בצד שלישי שלא אמור להימצא בהם, שמירה שרק השרת יהיה קובע ה-session id בכדי למנוע חשיפה ל-session fixation ועוד.

6)

להלן תיאור שלוש דרכים בהם מנסים תוקפים להשיג session id חוקי, ואמצעי ההגנה כנגד כל אחת מההתקפות.

- Interception: אם session token לא מוגן באופן מאובטח (הצפנת התקשורת בה מועבר ב-SSL), תוקף יכול לבצע session hijack ולחטוף את ה-session id של משתמש חוקי ב-session חוקי, וכך להתחזות למשתמש מול השרת המנהל את ה-session. כנגד התקפה זו יש צורך בהצפנת התקשורת: כל ה-session צריך להיות מוצפן תחת SSL מרגע ה-login עד רגע ה-logout, וזאת בכדי למנוע חשיפה לא מוצפנת של ה-session id, למשל ע"י חשיפת ה-cookie הנשלחת בתוך כל בקשה תחת session מסויים. כמו כן בכל מעבר עיקרי בין מצבי התקשורת רצוי להחליף את ה-session id, למשל בעת מעבר לתקשורת SSL או בשלב אותנטיקציה. יש לדאוג שה-session id לא יופיע ב-URL כדי למנוע הופעתו במטמון של הדפדפן, להישלח ב-referrer headers או להישלח בטעות לגורם שלישי כלשהו. כמו כן יש למנוע logging של ה-session id, בדיוק כמו שמונעים תיעוד של סיסמאות בלוגים. כל זאת יפחית את סיכויי חשיפת ה-session id וחטיפתו ע"י התקפה מסוג זה.
- Prediction and brute-force attacks: תוקף יכול לבצע מניה על גבי מרחב ה-session id שמגנרט השרת ללקוחותיו, או להשתמש בידע שניתן לצפות על ידו את ה-session id או לפחות לצמצם את מרחב החיפוש, וכך להגיע ל-session id חוקי ולהשתלט עליו. על מנת לאבטח את ה-session id כנגד התקפה זו, יש לבחור session id גדול מספיק ולעשות שימוש ב-cryptographically strong pseudo-random generators על מנת להבטיח ראנדומליות מספקת כדי למנוע ניסיונות חיזוי ע"י תוקף. כמו כן יש לשמור על כל אמצעי הגנרציה, כגון seed באופן מאובטח בשרת בכדי למנוע חיזוי. דרך נוספת להתמודדות עם brute-force attack היא החלפת ה-session id גם במהלך session באופן תדיר מספיק ובכך לבטל את יעילות התקדמות מניה של תוקף על גבי מרחב ה-session id.
- Session fixation attack: בהתקפה זו התוקף קובע את ה-session id של המשתמש עוד טרם המשתמש התחבר לשרת היעד, וכך מונע את הצורך לנסות להשיגו לאחר שה-session כבר נפתח. כעת ב-session פתוח יכול בקלות התוקף להתחזות למשתמש אל מול השרת. בכדי להיות מאובטח מול התקפה זו, על השרת לעולם לא לקבל session id מהמשתמש, אלא תמיד להיות המקור עצמו ל-session id (וכך גם יש שליטה על אופן בחירתו – מספר גדול מספיק וראנדומלי כצורך).

7)

Cookie poisoning: הרעלת cookie היא שינוי מכוון של תכני ה-cookie ע"י המשתמש (תוקף) טרם שליחתם מהלקוח אל השרת. למשל, אם cookie מכילה את הסכום הכולל שעל הלקוח לשלם עבור קניות מקוונות, יכול תוקף לשנות את המחיר הזה ולגרום לשרת לדרוש תשלום נמוך יותר מהמשתמש מאשר עליו לדרוש. שינוי התכנים הוא "הרעלת" ה-cookie.

המושג persistent cookies מתייחס ל-cookies שיש להם תאריך תפוגה כך שאינן נמחקות ממחשב המשתמש כאשר נסגר הדפדפן, אלא נמחקות ע"י המשתמש או ע"י הדפדפן בתאריך התפוגה שלהן. כמו כן, אין עליהן הגנה מפני שינויים ע"י המשתמש מצד מערכת ההפעלה, כך שהמשתמש רשאי לשנותם כרצונו. למשל, במערכת ההפעלה MS Windows XP/2000 יכול המשתמש להיכנס לתיקיית ה-cookies הנמצאת ב-c:\documents and settings\<username>Cookies, שם מאוחסנות ה-cookies כקבצי טקסט פשוטים.

התקפת cookie poisoning אינה ייחודית עבור persistent cookies. תוקף בהתקפת man in the middle יכול להאזין לתקשורת, לתפוס את ה-cookie הנשלחת מהמשתמש החוקי, לשנותה וכך למעשה להרעיל כל סוג של cookie. כלומר, בין אם ה-cookie נשמרת לפרק זמן מסויים ובין אם נמחקת לאחר שליחתה, היא עדיין חשופה לשינוי ע"י תוקף.

על מנת להתגונן בפני cookie poisoning ניתן להשתמש במספר שיטות: ראשית, ניתן לשמור ב-cookie אך ורק את ה-session id ושום פרט אחר נוסף. כל שאר הפרטים יישמרו על השרת וכך לא יהיו חשופים לשינוי ע"י משתמש עויין, כיוון שלא קיימים מלכתחילה ב-cookie. שנית, יש להשתמש בחתימה דיגיטלית שתספק data integrity ותוכל לסייע בזיהוי ניסיונות שינוי וזיוף נתונים ב-cookie: אם שרת מקבל מהמשתמש cookie, מבצע בדיקה של התאמת התוכן לחתימה הדיגיטלית של ה-cookie ומזהה חוסר התאמה, אזי נעשה שינוי מכוון וכנראה עויין ב-cookie והבקשה אליה התלוותה לא תאושר. החתימה הדיגיטלית יכולה להיות סימטרית (יעילה יותר מאסימטרית), כיוון שמי שמייצר אותה ומשתמש בה הוא אך ורק השרת. דרך נוספת היא לבצע הצפנה על ה-cookie וכך גם למנוע שינוי ע"י משתמש עויין.

8)

חשוב שמשתמש יבצע תהליך logout מסודר, לעומת סגירה של הדפדפן ללא יציאה מסודרת, וזאת כיוון שאם לא ייעשה זאת, גם אם ייסגר הדפדפן ה-session האפליקטיבי עדיין פתוח מבחינת השרת, כך שתוקף שיש לו גישה כלשהי ל-session id יכול להשתלט עליו באופן עויין ולהתחזות למשתמש הלגיטימי. למשל, תוקף שיש לו גישה למחשב בו עבד המשתמש וסגר את הדפדפן ללא logout מסודר בתום השימוש, אם יפתח את הדפדפן שוב יוכל לגשת התוקף לאתר בו עבד המשתמש הלגיטימי ולהמשיך לעבוד על אותו session תוך התחזות למשתמש הלגיטימי (תחת הנחה שפרטי ה-session, כלומר ה-session token נשמר ב-persistent cookie שלא נמחקה עם סגירת הדפדפן). בכדי לסייע למשתמש לבצע logout באופן מסודר יש לבצע את הפעולות הבאות:

- יש לקבוע פרק זמן ל-timeout, שלאחר חוסר פעילות מצד המשתמש למשך פרק זמן זה יבוצע logout אוטומטי מצד השרת וייסגר ה-session גם ללא פקודה מפורשת מהמשתמש. כמובן שככל שזמן ה-timeout קטן יותר, כך בטוח יותר (אך גם עלול לפגוע ב-usability).
- יש לוודא שבכל עמוד אליו יכול להגיע המשתמש בזמן חיי ה-session יהיה לינק ל-logout במקום ברור ובולט לעין, ובנוסף תהליך ה-logout צריך להתחשב בגורמים אנושיים. למשל, לא לבקש וידוא מהמשתמש ל-logout שכן המשתמש עלול לאבד סבלנותו ופשוט לסגור את הדפדפן ללא שביצע logout מסודר. בעת ה-logout: בצד השרת יש לבצע invalidation ל-session id, ובצד הלקוח יש לבצע אינוולידציה באם ניתן ל-cookies, ע"י כיוון max-age=0 המבטל את ולידציית ה-cookie.

9)

מערכת Web SSO על בסיס SAML assertions עובדת באופן הבא: המשתמש מבצע זיהוי (אותנטיקציה) מול שרת אחד, הוא המערכת המרכזית שתפקידו להיות ה-SAML asserting party (identity provider), ולאחר מכן, ללא צורך בהזדהות נוספת, יכול המשתמש לגשת לאתרים/מערכות לוויניות המספקות שירות על בסיס אותה אותנטיקציה, כאשר מערכות אלו הן ה-SAML relying party (service provider).

ה-asserting party מנהל את המשתמשים במערכת, מחזיק את כל האינפורמציה לגביהם ולכן יכול לזהותם. תפקידו אל מול מערכות לוויניות הוא לייצר assertions הכוללים הצהרות אודות עובדות מסויימות על יישויות מסויימות (לרוב משתמש או תהליך) על בסיסן מחליטות המערכות הלוויניות, הן ספקיות השירותים עצמם, אילו שירותים הן מאשרות לשימוש המשתמשים ואילו לא. פרוטוקול ה-SAML מאפשר יצירת assertions אודות זהות נושא מסויים, מאפייניו או הרשאותיו ביחס ליישויות אחרות, המבוססים על XML בפורמט סטנדרטי המאפשר תקשורת אחידה בין מערכות שונות. ספק ה-assertions מעביר אותן לספק השירות, והוא זה שמחליט על בסיס המידע שמקבל (כאשר ישנו אמון כמובן בין ספק הזיהוי לספק השירות) האם הוא סומך על אותה יישות (לרוב משתמש או תהליך) והאם מספק לו את השירות אותו מבקש.

סטנדרט התקשורת בפרוטוקול SAML מוגדר ע"י פרופילים המתארים תקשורת ואופן עבודה גנרי בין id provider ל-service provider, כאשר הפרופילים משתייכים לשתי משפחות מרכזיות: pull model ו-push model . ההבדל בין אופן העבודה המתבסס על push ובין אופן העבודה המתבסס על pull קשור באופן בו מועברים ה-assertions אל ה-service provider: ב-pull model ה-assertion לא עובר דרך דפדפן המשתמש, אלא ה-id provider שולח איזשהו artifact, פיסת מידע המכילה את מזהה ה-assertion שיצר, אל ספק השירות, והוא מבקש באמצעות אותו artifact את ה-

assertion מה-id provider. ב-push model ה-assertion נדחף מה-id provider אל ה-service provider מעל HTTP request. נדגים זאת על תסריט source first scenario:

- Pull model: משתמש פונה לפורטל הארגוני שלא מכיר את המשתמש כרגע. מערכת ה-access control מזהה שאין session פתוח ולכן מערכת ה-login מבקשת מהמשתמש להזדהות. נתוני ההזדהות מגיעים ל- authentication authority ב-source site, והוא מבצע אותנטיקציה ומועבר חזרה לאפליקציה שאליה רצה המשתמש להגיע (redirect). בדף זה יש לינקים לאפליקציות אפשריות שיכול הפורטל הארגוני להעביר את המשתמש. לינקים אלו לא מצביעים ישירות לאותם אפליקציות אלא ל-inter site transfer service – העברה מאובטחת אל ה-destination site. ה-id provider מייצר assertion לאותו משתמש ויוצר artifact עם uid של ה-assertion וכתובת ה-destination. הוא מבצע redirect למשתמש ומעביר את ה-artifact ל-service provider. כאשר זה מגיע אליו, הוא מחפש את הפרמטר artifact, יודע לבצע עליו פענוח (כי אלו מקודדים), ומבצע SAML request ממנו אל הרכיב המתאים ב-source site ומבקש את ה-assertion התואם ל-uid ב-artifact. זה נשלח ל-responder ב-source site האחראי על קבלת הבקשות ושליחת התשובות. הוא מקבל את ה-assertion ב-SAML response. הוא יוצר כתגובה למשתמש session cookie ומבצע לו redirect לבקשה עם ה-cookie. כעת יש כבר פניה ל-remote application עם ה-cookie המתאים והמשתמש הגיע ליעדו ל-remote application מבלי להזדהות שוב.
- Push model: אין קשר ישיר בין ה-source site ל-destination site, בניגוד למה שיש ב-pull. כל השלבים הראשונים דומים ל-pull model. מהשלב בו ה-assertion מקודד כ-hidden ב-HTTP form, הוא מועבר לדפדפן של המשתמש. כעת המשתמש נדרש לבצע submit ב-post אל ה-assertion consumer, וכעת הוא מקבל את ה-assertion במלואו ולא פונה ל-id provider. הוא בודק את ה-assertion, ואם נמצא תקין הוא יוצר cookie ומפנה את המשתמש ל-remote application (הערה: לא ניתן לעבוד כאן ב-redirect אלא ב-post request בגלל מגבלות גודל – יש מגבלות על גודל query string ב-redirect ו-XML הוא מבנה לא קטן מספיק ולרוב יחרוג מאילוצי הגודל).

10)

להלן תיאור כללי לתהליך ההזדהות ב-OpenID (identity provider – IP, relying party – RP):

- המשתמש מתקשר עם ה-RP ומעביר לו את ה-URL שלו, הוא ה-OpenID של הזהות שלו. ה-RP מזהה מתוך ה-URL את ה-IP של המשתמש (נותן חסות לזהות המשתמש).
- ה-RP מפנה את המשתמש אל ה-IP שלו בכדי לקבל identification token, דהיינו לקבל אישור על זיהויו ע״י ה-IP.
- ה-IP מבצע תהליך הזדהות על המשתמש המופנה אליו בכדי לאמת שאכן המשתמש הלגיטימי הוא מבקש ההזדהות. לרוב ייעשה ע״י בקשת משתמש וסיסמא.
- אם המשתמש הכניס את ה-credentials שלו באופן תקני ואומת, ה-IP מעביר דרך המשתמש token בשביל ה-RP. אם ישנה היכרות מוקדמת בין ה-RP ל-IP, כאן נגמר תהליך האותנטיקציה. אחרת, ישנו תהליך הקורה מאחורי הקלעים להיכרות בין ה-RP ל-IP.

להלן תיאור התקפת Phishing על משתמש המזדהה באמצעות OpenID:

- המשתמש מגיע ל-Evil RP בדרכי phishing מוכרות וידועות, למשל דרך תוצאות חיפוש במנוע חיפוש כלשהו.
- המשתמש מעביר ל-Evil RP את ה-URL שלו, והוא מזהה מתוכו מי ה-IP של המשתמש.
- במקום להעביר את המשתמש ל-IP שלו, הוא מעביר אותו לצד נוסף, Evil scooper.
- ה-Evil scooper יוצר קשר עם ה-IP של המשתמש, כך שה-IP מעלה תהליך אותנטיקציה מולו. כעת המשתמש חושב שהוא מזדהה מול ה-IP שלו, אך למעשה מזדהה מול ה-Evil scooper, וכך המשתמש מכניס למעשה את פרטיו הסודיים (סיסמא למשל) ל-Evil scooper, וכעת יכול ה- Evil scooper למשוך tokens מה-IP של המשתמש. ה-Evil scooper הוא למעשה man-in-the-middle בין המשתמש ל-IP שלו.
- ע״י ה-tokens שיכול לגשת אליהם ה-Evil scooper יכול התוקף כעת להיכנס ל-RP אמיתיים ולגיטימיים.

הבעיה הטמונה בשיטה המאפשרת את ההתקפה הנ״ל היא שמי שמפנה את המשתמש אל ה-IP שלו הוא ה-RP שיכול להיות עויין, ולהעביר את המשתמש למעשה ל-Evil scooper במקום ישירות ל-IP האמיתי שלו.