אבטחת מידע - הרצאה 13 10/6/6

הלא נראים לעניין בבעיות אבטחה של תוכנה ב-input validation - הרעיון שהקלט שמקבל התוכנה רע ולכן לא מבצע אותו ומאפשר לתוקף לשנות לגמרי בקלט לא רצוי. בהרצאה הקודמת דיברנו על משפחת ה-injections - התוקף משתמש בטכניקה כדי לקבל מידע שהטכניקה משתמשת בה (למשל מידע ההפעלה או בסיס נתונים)

הלא נראים מהתקפות שמנצלות את הדפדפן ולא מתבססות. נדבר על 2 משפחות של התקפות:

Cross site scripting - התוקף מנצל את החוסר ב-input validation כדי לתקוף את הדפדפן של המשתמש (להעביר קוד לא חוקי בדפדפן של משתמש) ב-sql injection המטרה שימשה לתוקף להעביר התקפה בצד שרת, כאן האפליקציה נשמשת תווך להעביר התקפה בדפדפן של התוקף.

משפחה נוספת בה הקוד רץ בצד הדפדפן ותוקף את האפליקציה נוצרה עקיפה ב-input validation.

Cross Site Scripting

אם מאתרים את ה-injections בצד שרת ואלו בדפדפן מתבטלים את המסה הגדולה של פגיעות יבוא באפליקציות web, אבל משהו שקל יחסית לבצע אותן (אי כל בעיה, תוקפים ומו, לכן אם לא מטפלים בפגיעה תוקף שאתו כלי בדיקה יש סיכוי גבוה למצוא אותה ולנצל)

מקור הדפדפן הראשון של Netscape היה לבנות בפני html סקריפטים לחלונות. בהתחלה לא היו לו JavaScript כשפת תיכנות אולם אולם פרקטי מגבלות שמבטאות מניפולציה בדפי html. הקוד שלו מתקיים בצד לקוח יכול לבצע הרבה דברים (לפרק למערכת הקבצים וכו'). בעיה גם היא מובנת ולכן הרבה שקרי אתרים יהיה חשוב בהתחלה פיתוחי Activex מ"מ מיידע הקוד

המשתמש הוא יכול לראות את הקוד של הדפדפן גם.
בהרבה קבצי exe לוקח מעבר הפעלה שונות את המשתמש
הם מנשה לקבל לדוגמא איך הדיפולט של משתמש פשוט הוא
לבקש כן. לא מעט משחקים מגלים תוך מה משחק קוד פנים.

בתחילה הדינמי לא מאפשר לדעת מי מגיב מעדכני הקוד בלי
לדעת שהקוד לא השתנה.
עם JS לא מדובר כשפת תכנות, אלא כ enhancment
ל - web ולכן לא חשבו שצריך להכניס מנגנון שיאפשר את
פעולתו והסגנון ביה של ה JS.
לכן אין מנגנון לתרגום דינמי (המרבה שיוצרים קוד JS
בזמן מתפל ת הסאפשרות הזו)
html דינמי - אלוהים html ו- JS
JS נותן יכולת לקבל event מ של המשתמש ולהפעיל קוד
לדוגמא homesubmit מבלי קוד JS שיבדוק את הקלט של
המשתמש בטרם שולח submit. JS מאפשר הרבה בדיקות
handlers event שהדפדפן מחזיר ומבלי קוד JS
המשתמש לאלו html element שקשור ל- event.
כמו כן, בהתחלה לקח לא המשתמש נתן לשנות את ה html.
בעבר המקורי JS היה חלק מה- html ולכן כשה JS
רצה מהדפדפן על השרת, לא היה קשר בין הדפדפן לשרת.
AJAX מאפשר ל- JS לנהל תקשורת מ ה שרת בלי
לתן בתהליך עבודתו JS קיבל נתון ולא יכולת ל- JS
שתמש כ enhancment ל - html CSS כדי להכניס דינמיות
הפך לשפת תכנות של הדפדפן עוד של ה- לשון בעולם
client interaction. בפועל JS אלו הוא כשפת תכנות
לכן לא מדובר בבעיה מנגנון ורבות מזה מבחר שלמות
נגיש שפת הנתונה ל הדפדפן. נתונה מצו, הפכה לשפה
לכל דבר עם הרבה יכולות, יכולות תקשורת ועוד.

הדפדפן הפך להיות מאוד מורכב הפעלה כיוון שמותר
לכתוב עליו תוכניות. הפך להיות פלטפורמה שלמה ניתן
לכתוב תוכניות ב- JS. לתוכנות טובות יש הרבה בטווח
יכולות טוב מבחינת שיש לקח הרבה תכונה ל- usability, זה
הרבה פעמים מנצח.

ניתן להכניס פקודות שיהיו לקוד דרך הטג script וזו לחבין
מה ה- JS מקוד חיצוני ויש גם אפשרות לשים embedded
לסקריפט בלי להשתמש בפלל.

הוא מה שמריץ את ה- JS זה ה- engine JS שהוא
חלק מהדפדפן. כל דפדפן מישהו JS engine משל עצמו.
מה שקורה היום יותר מקובל זה html זה היה הדפדפנים
המודרניים מגיעים מה הפל. נדבר מכך שהדפדפנים מנסים להיות
developer friendly. התשתית הוא מאוד בינוני מה מקומות
כפול במקרי קצה ב- JS. ניתן לחשוב דוגמאות למקרי קצה
בסוף ההרצאה. (JS לא מקרן שהדפדפנים מבינים אותו)

בעולם ה- Cross Site Scripting יש scripting engine שרץ
בצד לקוח שמריץ קוד JS / קטעי JS שהם embeded
מתוך ה- html. הבעיה מקבל מתקפות שמוסיף לזה וזה
מוסיף על ה- html ועל ה- JS שהוא embeded בתוכו.
יש כמה מנגנון שדרכו כדי מוקל לשתול קוד מזיק בדפדפן
של המשתמש. הבעיות הוא בטכנולוגיה הלא מאוד טוב בעבר
מלקבים מה משתמש הקצה דרך הדפדפן שלו כי הוספה
מה הטכנולוגיה הלא מאוד למנגנון מאוד לשתול בתוך דפי
ה- html של הטכנולוגיה שקלחת לקח של המשתמש קוד
JS מין, לתן בעבר מלקבים מה משתמש הקצה.

בעולם ה- JS ישנם 2 מנגנוני טובות מידע שמתקדם
לבדוק הרשאות
- לבדוק input validation
- לבדוק output encoding

בעולם ה- cross site script התיקון להשתמש בתוך המנגנונים

u.multinet.co.il

ההתשובה אותה מנגן השרת היא המבדה שהדפדפן שומר בלי
קשר ה- html שהגיע מהאפליקציה הרגילה (הוא מניח שכל
הקוד ה- כך שהוא חלק . מצד ה- html הוא
תוקני)

כדי להימנע מ- cross site צריך לנסות לא- ה- engine כך בדפדפן
אלא השרת תהיה שכנעה מלוא-ד מהאתרים לבד אותה לא
בצורה אוטוריטרית, כלומר הכל לסלנדרי שלו אפשר בלאדון.
הדפדפן צריך לדאוג שלא כל קוד native של ה- form
(קוד מקורי) אות קוד מוצרים אלא לא יכול לבצע מה תקין ומה לא
כמו ה- engine sql שמבצע כל שאילתת sql שלא היא נכונה
סינטקטית.

לקוד המין יש גישה ל- cookie של האפליקציה. ראינו כי
cross site scripting היא אחת הבעיות העיקריות לגנבת session cookie
html יכול לגרום ל- http request.

same domain policy - אם יש קוד כלשהו מכל html שיורד
מאפליקציה, (הוא) צריך לקבל את ה- web server של האפליקציה.
לכן יש קוד כדי לאלץ לגרום בקשות רק לשרת מסוים.
אלא גם לגרום לדפדפן לאמת שהוא לשרת הנכון.
מלבד גניבת session cookies (הבעיה לא כולל session hijacking)
מצוין שה- כך יכול לבצע מניפולציה לכל ה- dom...
ניתן לשנות את תכני ה- web server גם בצד הדפדפן.
defacement משמש ל- phishing.

ראינו עד עכשיו בעיות של ה- לקוח מול האפליקציה, אבל יש
אפליקציות שהרעיון שקוד כך יכול לגרום את הרשת שבה
נמצאת ה- לקוח וגם לקבל מחשבים ברשת שבה נמצאים הדפדפן
של הרשת. לכן יש משמעות גדולה. עד עכשיו הקוד יכל
לבצע ברשת ולגרום לו את ה- session אפליקציה. כלומר
בעזרת של ה- לקוח מול האפליקציה (כמו phishing) מה לא
בעזרת של הוירוסטרנר. אבל מהעבודות יכולות משתמש בלי

הגיון שקרוב של הרבה לא יכול אפה משהו לעשות של הרפיסה.
הוא הדפדפן הפך ב- לפרוצדורה להריץ malwhare.
אם מצליחים לחדור לאלו האפליקציה ולשנות את מבני ה- html
ניתן לגרום לשמשים להריץ קוד אין בלי שיודעת. יש אתרים
שמאפשרים לה כדי שיכניסו קוד מין כזה. הם בהגדרה
מריצים קוד כך מין.
ה- web הפך לשדה המרכזי להריץ malwhare למשתמשי הקצה.
נחלק את משפחת ה- cors site ל- 3 קטגוריות:
1. reflected 2. stored 3. DOM
ההבדל הבסיסי בין 1 ל-2 הוא בשאלה האם קוד הכך הזה
נשמר ב- web server. ב- stored קוד הכך הזה נשמר
ב-server למשל ה- user מקבל הודעה שנשמרה ב- db של
השרת ומוצגת. ב-reflected ההודעה לא נשמרת ב- web server
אלא נשלחת מיד מה- web server שבולל את הקוד הזה
והאפליקציה תדפיס משהו מתוך הקוד בדף שמגיע למשתמש.
ב-reflected הרעיון שלה זה ה- req מול השרת מכיל
למשל אימייל בתוך שהמשתמש יקליק על הלינק ושלח את
הדף את הקוד המזיק מול ה- server.
יש כאן הפלקציה של קוד ה-JS מה-request של ה- response
בשניהם הרגישים תופסת ב-input validation הוא בקוד של האפליקציה שרץ בצד
שרת ב-DOM התופסת ב- input validation הוא בקוד האפליקציה שרץ
בדפדפן. כל שלוש הסוגים יכולים לגרום כל דבר, הרעיון יכול
לגבל מה הקוד המזיק כדי לפרוץ המשתמש בחוסר מזל.
ב-L. האפליקציה הזו מקבלת בקשה מ מישהו מין, לא שומר
בשרת אלא משהו לבד reflection לתוך דף ה- html שמחזיר
ב- response לדפדפן שאלה מגיע מה- request.
הדף לא מבצע ולידציה אלא לא מבצע ולידציה מספיק טובה
כפלקציה מה רק של המשתמש מול דף ה- html שמחזיר לו
מבצעים לוגיקה בעזרת מיפוש מה המחרוזת שהכניס נדבקים למשתמש
בדף הרגולה.

דוגמה: יש ספרים וירטואלי שמכילה שדה חיפוש כדי המשתמש
מכין קוד שנשלח ל-web server, משתמש קוד לינוקס חיפוש טוב ב
מודך למשתמש בכל הכלל. תוקף יכול להכניס testing for xss
ולראות שדה מוחבא בכל הרצוות.
כמו כן ניתן להכניס ואותו דבר כך שיפורים כמו כך ולכן יוכל
מתוצאה מנוע ה- כך בדפדפן. זה דרך לבדוק cross site
לי לבדוק נכון. זה נראה כי בכל שמוריד למשתמש הקסם
שיכין המשתמש שלה בשני מקומות.
הקוד האמן שולח מ' ה- web app. בכל ה- מה שנשלח
לדפדפן alert זה דוגמה. אם לא הקוד כך יכל לרוץ
כדי לגנוב session cookie נשלח לינק שמכיל סקריפט שמכיל 2 פקודות
כך שבאינו בעמוד. דרך המדבקה שב- image ה- אדם יכול
לפנות לשרת אחר ובכך מוכנה ל- כך לשלוח בקשת המדבקה
לשרת תיבנו. בנוסף מלבד sessionid ניתן לתת מודעה כמו
hidden parameter ב- form. הדרך הקלאסי להפוך reflected cross-site
זה לשלוח אימייל מ הלינק נשמר לה שגם אם ssl יכול להיות
תשלח ל- reflected cross...
הבעיה מגיעה יש אפליקציה בנקאית שמשפה ל- cross site scripting
בנוסף לינק משלח לאפליקציה ומשלח מרובה קוד מן שמוגדר שהוא
שלח בכל שאלה למשתמש שלח אל הלינק לאימייל של המומחה.
האפליקציה לא מגיבה שזה קוד מזין ולכן מעלה אל הקוד בכל
html שמרכיב למשתמש. הדפדפן מבין ה- כך בתוך הכל,
מגיבי מקשה מ session cookie ושולח לשרת של הבנק.
stored cross site scripting - כאן לא צריך לשלוח אימייל ולא מלוות ביטוי
למצב spam attack כדי לשלוח מייל מ הלינק להרבה משתמשים
שלחתם קוד של האפליקציה שנשלחת בפעם הראשונה (input validation)
לא מבינה אם הקוד מזין אל הקוד לשרת ה- DB. ואז הקוד יגיע למשתמש
אחרי מירקון מוצא המשתמש כמה הכל מקבל מ אפליקציה שמבין
והאפליקציה שלחה לו מ קוד כך שכל לו בדפדפן
הדוגמה כשהאפליקציה מקבלת הקוד ומעבדת ולבדוק ומבינה שזה קוד
שגוי

ב- reflected כל מה שמשתמש הקוד פנוי ב- error page...
הרבה פעמים מפרסמים שגיאה בקוד הזה. לשפר
ניצור ל- stored: ניתן בשם משתמש להכניס Joe סקריפט
שמבטל קוד מקומי חיצוני. הקוד בה- כך המין ישולב בדף
שמוצג לאדמיניסטרטור, ואם ניתן לגנוב לו את ה- cookies הסשן session
DOM Base - דומה ל- reflected אלא בנוסף שלוקח אפילו שגיאה
את הקוד כך המין בתוך ה- html. גם הקוד שרץ ב-
server side, הרי שמה שמגיע ב- post את הקוד המין
ב- html גם הקוד האפליקטיבי שרץ ב- client side. הרעיון
שלה אל המשתמש בטוחים קיים שיש בו קוד מין. הקוד
לא נשלח לשרת, נשאר בדפדפן אלא בשביל זה ה- html הוא
מהשרת לדפדפן...
אלמנט שנקרא fragment נשמר אצל הדפדפן ולא נשלח לשרת.
זה אינדיקציה לדפדפן איך להקפיץ את המשתמש בתוך הדף
שנשלח מהשרת ואינו נשלח לשרת. את הסקריפט
הסקריפט בשביל מה נשלח ל- user בתור לינק יביא לשרת
בקבוצה תוקף מחלונים אלא הקוד המין יכתב לדף ה- html
הזה לצלם בשמים מתפיל הזה ע"י view source.
כאן לא יעבוד input validation ב- server side כי הקוד המין
לא מגיע בכלל ל- server web. לכן אם זה קיים, זה מאוד חמור.
לסיכום cross site scripting כל אחד שכול לשלוח בקשה הוא
threat agent.

מבחן יעילה קובל שאלה לבדיקה במאגר שבור.
מבחן אמריקאי (בדיאה מה שאלה מסוגים לענות על 33).
בגוש הבנה והכרה של החומר בצורה טובה יש לדאוג על
כל החומר. אין צורך לשנן. מבחן בפתוח שונה מבגרות הפה.
יהיה 100 דקות. להיות איך הפרוטוקול מאבד. ניתן לשאול
לגבי SAML מה מבר ב- post data בין ה- IdP לבין
נשים בכיר לפני תרגיל אז לא ולזה שאלה בנקסוס ה- SSL,
מה מבר בכל שלב ב- handshake. לזה בוטל ה- handshake

ולגבי ה- ssl layer, מה ההבדל בין one base ל-
symetric nat בהרצאות.

הדרכים להתגונן בפרצת cross site scripting:

1) input validation - כל קלט שמגיע בשרת על html שנשלח מ-
ה- web app על ה- browser או קלט שה- צד
בשפת web page מעבדת לתוך ה- html חייב לעבור input validation.
מצד אחד שה- input validation יהיה positive כי באופן
קשה לבנות negative security logic אפקטיבי אז מנסים להימנע מ-
cross site scripting. ב- form ההגדרה בהגדרה פיזית free text
לא חייבים negative מתקדם מסוימים שם להימנע מבנה כשלעצמו
חוקים
כל מקרה נדרש להבין את האורך ובמקרה אם הפקודות אינן
לתוך ובנוסף לבדוק לבדוק את ה- characters ולוודא
character set תוך מחרוזת שמעבירים קונקטיביות.
נניח שאותו קלט אין מגיע לשלב שלו בצד html כי אם הקוד
יש לו מטרות והקוד יכול בדפדפן על המשתמש.
גם בקונפיג על stored אם מאפשרים את הקלט האין לא בהודעה
הפרטית, ולגבי נושאים לקנות עסק אם את הכלי שבנינו
באדמינסטרטור מבר לא קבע ה- עסק הוא browser based
אלא הקוד יכול אבל האדמינסטרטור.

שים לעסק לתקן את הקלט כי בה מאוד מסוכן.
בכלל הדרישה היא לא להשתמש ב- black list (negative security logic)
כי אם מתפשרים פרצה script tag (<) ובמה אני
ניתן לעשות רגיסטרים מחרוזת כמו כותרת וגם לא יהיה כ-
script tag פרצה ובן יכולים בכל זאת הדפדפן אבל ישנו
בעיות אם בתוך מאות בתים < כדי להבנות סקריפט שיושלם בתוך
בל ה- html ה- negative חייבים לבנות ולוודא שבן
שמתפשרים את ההגבלה (css) cascading styled shit יכול להכיל קוד JS.

נשים לב שבשונה מנקודות תורפה אחרות, הבעיה לא נמצאת אצל המתקפה "הנכונה" ולכן גם לא (בדרך כלל) בקוד. אז כמה חבל לבדוק content replacment כדי לראות שאם אינני על הקו, בגלל שאין לבדוק כמה מהר אותו input validation מספיק כדי אותם תוקפים ל- Content Replacment Attacks.

כל נושא שהיו לעיל כמעט

בשרשרת לשאר הקוד בתרום בל לשלח, בדיוק לסמן ל- browser שהדפדפן הבא הוא מילון בעל משמעות כפולה משמעות אוקסיוטיות. כדי לנקות output encoding אלו המשמעות של ט בדרך קידוד מלוכלך

תשים להשתמש ב- library שיודע לבצע Output Encoding בהתאם לסביבה שבה הולכים לשלח את הקוד בתוך הדף. כדי שהדפדפן לא יתבלבל משהו שהמפתח יצר מעלים בתוך הדף עדיף ה- encoding של הדף ב- ASP.NET יש ספריה שנוצרה ומיועדת לעבודה לבצע output encoding בצורה טובה מבלי פוגע שיקודדו את הקוד שרוצים לשלח בהתאם לסביבה בו רוצים לשלח אותו (בתוך HTMX element/ HTMX ודומה) בגלל פגיעה בט מיקום כי הקידוד שלנו צריך בכל מקום מתאים.

נשים לב ש- output encoding לא יחליף input validation אלא יעבוד בנפרד, בעיותיו כשהה- input validation הוא ה- negative.

התקפה אחרונה - Cross-site Request Forgery (CSRF/XSRF)

זהו הקפה שקורה בדפדפן ומטרתה לגרום פקודות מסוימות בשם המשתמש שלתו מה- browser. אל המשתמש מול ה- web app. נכנס ל- request אל האפליקציה שלה מה- browser של המשתמש מ- credentials של המשתמש ולכן מתבצע האפליקציה כאילו בקשה שנשלחה מהמשתמש.

3 הקבלה שרגילה בדפדפן:

XSS - מנצל את האמון הקיים של ה- browser בדף ה- html
שקיבל מהאפליקציה.
CSRF - מנצל את האמון של האפליקציה ב- browser ב
המשתמש ל- credentials שלו.
ה- server side מקבל פקודה מה- browser של ה- web app.
נראית תקינה - ומגיעה ל- credentials ב המשתמש.
לכן לאפליקציה אין דרך לדעת שלא ה- user יכה אותה.
גורם לו באמה של חוסר בvalidation input
המכון CSRF בלי CSS. אם האפליקציה משפה ל- CSS
לא תהיה מיקל להתגונן בפני CSRF אלא ב. אם מסיגה
מפני CSS, אינה מסיגה מפני ה- CSRF.
ה- cookie שמטל על ה- session id מגדל אוטומטי אי
הדפדפן ולכן במצב שבו נרמה ל- browser לשלוח את
הבקשה ל- web app. הוא יבקל אוטומטי את ה credentials
לא הבקשה. נניח למשל לבקש מבקשה לשלוח מיילים לכל
ה- contacts ל קוד הזן אל קוד כך שישלח לאפליקצי
המיילים ובקש את כל ה- contacts של המשתמש.
כל אפליקציה שלא מעבר אוריזציה אפקטיבי לכל הבקשות
תשלחה אלא כי כל אפליקציה שלא מבצע אוריזציה לכל
בקשה שמגיעה ל credentials.
מין דרך אוטומטי - להגן בפני CSRF. יש לקבל שלא בקשת
יצרנו אלא למשל בקשה מאוד להעביר כסף מחשבון למשבון
לבצע אוקיה של מיליון שלא יש request בדרך האפליקציה
בלא מיליון בעיה, עדיפה נגרום שה- user יצטרך את מילוי
CAPCHA - מאיפין קשה לקוד אוטומטי לבצע אותו, לכן אלגוריתם
שמגרה מיצב user מאחורה.