אבטחת מערכות ויישומים ברשת / תרגיל בית 2#

אריאל סטולרמן

1)

SSL Session: קישור שנעשה בין לקוח ושרת המגדיר סט פרמטרים , ונוצר בתהליך ה- handshake. הפרמטרים המגדירים SSL session הם: מזהה session הנוצר ע"י השרת כדי לזהות session עם לקוח מסויים , X.509 certificate ללקוח ולשרת לזיהויים , החלטה על מתודת דחיסה שתשמש לדחיסת מידע טרם הצפנתו , החלטה על אלגוריתמי הצפנה ו- hash (למשל DES ו-MD5), master-secret שהוא סוד בן 48 בתים בין הלקוח לשרת ודגל המציין האם ניתן להשתמש ב-session כדי להתחיל על גביו connection חדש.

SSL Connection: קישור לוגי בין לקוח ושרת המקושר לשירות מסויים . הפרמטרים המגדירים connection הם: server and client random - מחרוזת רנדומאלית הנוצרת ע"י הלקוח והשרת, write MAC ו-write key המשמשים את הלקוח והשרת להצפנת המידע ופענוחו (זוג מפתחות לחישוב digest וזוג מפתחות להצפנה ; זוג כיוון שיש אחד לכל ערוץ – שרת-לקוח ולקוח-שרת), ו-sequence number שנשמר בנפרד ע"י השרת והלקוח למעקב אחר ההודעות שנשלחו ונתקבלו במהלך העברת מידע.

הקשר ביניהם הוא כזה בו ניתן ליצור כמה connections על גבי אותו session, ובכך לחסוך תחלופת מידע ופרמטרים מחודשת בכל יצירת connection. תהליך ה-handshake כבד יחסית (נעשה על בסיס הצפנה אסימטרית), ולכן יעיל להרים כמה connections על גבי אותו session. כמו כן, המפתח בו נעשה שימוש ב-connection נגזר מה-master key ב-session, בתוספת ראנדומליות של הל קוח והשרת ושימוש ב- hash. כמובן שמ-master key בודד יכולים להגזר connections רבים (בעלי מפתח שונה אחד מהשני).

2)

אם תוקף הצליח לפרוץ את ערוץ התעבורה בין ה-client ל-server ולקרוא את התעבורה, הוא לא יוכל כפועל יוצא לקרוא גם את התעבורה בין ה-server ל-client שכן כל ערוץ (client-server ו-server-client) מוצפן באמצעות מפתח שונה – גם מפתח הצפנה שונה וגם מפתח digest שונה, כאשר כל אלו נוצרים בעת יצירת connection (בין היתר גם על בסיס מידע מה-session עליו נמצא ה-connection).

כחלק מתהליך הקמת connection ולצורך וידוא תיאום המפתחות, שולח אחד הצדדים מידע לצד השני באופן מוצפן , והצד השני מפענח זאת ובודק שזו הודעה אליה ציפה. כיוון שמוקמים שני ערוצי תקשורת נפרדים , הודעות כאלו, הן ה-finish messages, צריכות להשלח גם מהשרת ללקוח וגם מהלקוח לשרת באופן בלתי תלוי.

3)

להלן מספר דוגמאות לטעויות אופייניות בשימוש בהצפנה להגנה על סודיות של נתונים:

- שמירה לא מאובטחת של מפתחות בזיכרון: memory dump עלול לחשוף מהזיכרון לדיסק מידע סודי . תוקף שמשכפל את הדיסק (למרות שמלכתחילה לא אוחסן עליו המפתח), יכול להשיג את המידע הסודי. הפתרון לכך יהיה להחזיק בזכרון את המידע הסודי לפרק זמן מינימלי שנדרש – ומיד לאחר מכן למחוק אותו . צעד חשוב נוסף הוא ביצוע דריסה explicit של המידע הסודי בזיכרון בתום השימוש בו כדי לוודא שלא יהיו שאריות ממנו בזיכרון.
- מקורות לא טובים לראנדומיות בעת בחירת מפתחות: שימוש למשל בפונקציית rand ב-C אינה מקור טוב לראנדומיות כיוון שהיא פרדיקטיבית . בקריפטוגרפיה דורשים אקראיות סטטיסטית ואי יכולת חיזוי בין ערך אחד לבא אחריו . שימוש במקור פרדיקטיבי לראנדומיות בעת בחירת מפתחות יכולה להקל על התוקף למצוא את המפתחות הנכונים ולבצע תקיפה .
- בחירת אלגוריתמים מיושנים שאינם בטוחים יותר: ישנם אלגוריתמים שבעבר נחשבו בטוחים וכיום כבר אינם , למשל MD-5 או RC4 כאשר עבור ה-MD-5 הוכחה חולשה בתחום ה-collision resistance. שימוש באלגוריתמים לא בטוחים להצפנה מסוכן שכן תוקף יוכל בסבירות גבוהה לחשוף את המידע המוצפן באלגוריתם לא בטוח.
- שימוש באלגוריתם שפותח עצמאית: הנחת העבודה צריכה להיות שהאלגוריתם ידלוף , ואז המצב יהיה כנראה גרוע יותר מה -APIs המוכרים להצפנה קריפטוגרפית. הסיבה לכך היא שפיתוח אלג' הצפנה קריפטוגרפי דורש הרבה זמן ומשאבים לפיתוח נכון של כזה , השקעה שלא סבי רה להמצא בפיתוח עצמאי. בסופו של דבר פיתוח עצמאי בעל סיכויי פריצה גבוהים בהרבה מאשר crypto APIs מוכרים.
- שמירה לקויה על backups למערכות: אם ה-backup אינו מוגן, הגישה למידע שאמור להיות חסוי היא פשוטה – במקום לנסות לפרוץ למערכת העיקרית, תוקף ייגש לגיבויים שם סיכויי הצלחתו גבוהים יותר (אם אינם מוגנים).

4)

מפתח אפליקציה צריך לדאוג לא רק לכך שהאפליקציה תהיה מאובטחת , אלא גם שתשתית המחשוב עליה רצה האפליקציה תהיה מאובטחת , שכן שמירה על אבטחת האפליקציה בלבד אינה מספקת ותוקף יכול לגרום נזק או לחשוף מידע סודי בתנאים אלו . נקודה חשובה בעניין זה היא העובדה שלרוב מערכות התשתית לפני הרמה האפליקטיבית הן רכיבים מסחריים או public domain ולכן מוכרים ע"י גורמים עויינים המחפשים דרכים לתקוף, ולכן חשיבות ההגנה עליהם גבוהה . כמו כן ישנם כלים אוטומטיים לסריקת מערכות למציאת פגיעויות ופירצות , אך אלו מצויים גם בידי תוקפים שיכולים לנצלם למציאת פירצה בתשתית דרכה יכולים לתקוף את המערכת . להלן מספר דוגמאות לבעיות באבטחת תשתית המחשוב המאפשרת תקיפת מערכת:

- מוצרי תשתית המכילים פגיעויות ולא הותקנו עליהם patches שה-vendor שחרר לתיקון אותן פגיעויות: בכל מוצר יש באופן פוטנציאלי פגיעויות, וכל הזמן הם מתגלים ומשוחררים תיקונים . למי שתהיה גישה למערכת ההפעלה בגלל פגיעות שלא patched, אזי תהיה לו גישה למשאבים במערכת ההפעלה.
- קונפיגורציה לוקה של ה-web server: יש להגדיר לאילו משאבים תתאפשר גישה ע"י ה-web server כלפי פנים. הגדרת הרשאות לקויה יכולה להוות פירצת אבטחה שתנוצל ע"י תוקפים.
- חשיפה מתוך backups: יש צורך לדאוג לאבטחת גיבויים , שכן חשיפתם שקולה לחשיפת המערכת או המידע במערכת . דוגמא קלאסית לכך היא רכיבי תוכנה כמו demos שבד"כ לא מקפידים בהם על אבטחה , שכן המפתח יתמקד ברצון שהרכיב יעבוד , וישים פחות דגש על אבטחה . תוקפים יכולים לחשוף בהן פגיעויות אותם ינצלו כדי לפגוע במערכת.
- הרשאות מיותרות לאפליקציה: מערכת ההפעלה מכילה שירותים רבים , ומומלץ לא לאפשר לאפליקציה שירותים שאינה צריכה. אם האפליקציה לא תגדיר את השירותים הנדרשים לה בצורה מסודרת , אדמיניסטרטור המערכת לא יוכל להגביל אותה , וכך להפחית את הסיכונים . אם נושא זה לא מטופל, תוקפים יכולים להגיע לממשקים אדמיניסטרטיביים אפילו מתוך חיבור כמשתמש רגיל ולגיטימי.
- Default accounts: המערכת מגיעה עם משתמש דיפולטי וסיסמא דיפולטית . אם האדמין לא ידאג למחוק אותם ולהחליף את כל ה- default passwords, תוקפים יכולים לנצל זאת.
- הודעות שגיאה אינפורמטיביות יתר על המיד ה: חשוב לדאוג כי הודעות שגיאה למשתמש הבאות ממערכות התשתית לא יחשפו מידע מיותר שיכול לשמש תוקף להבנת המערכת וכך לעלות את סיכוי הצלחתו.
- קונפיגורצית SSL חסרה או בעייתית: חשוב לקנפג את ה-server side בו יש להגדיר cipher-suite מינימלי שתחתיות לא יהיה ניתן לעבור , וזאת כדי למנוע תקשורת SSL שהיא למעשה לא מאובטחת.

5)

להלן מספר תהליכי אבטחת מידע הנחוצים על מנת להגן על תשתית המחשוב עליה רצה האפליקציה בפני התקפות:

- Server Hardening: הקשחה צריכה להתבצע באופן אחיד על כל השרתים וכל ההתקנות של מוצרי התשתית בכל הארגון . הארגון צריך להגדיר guideline של הגנה מאובטחת לכל מוצרי התשתית והוא יהיה בסיס להתקנה אותו מוצר תשתית באותו הארגון . בתוך כך: קונפיגורציה של כל מנגנוני האבטחה, ניטרול וכיבוי כל השירותים שלא נעשה בהם שימוש , אתחול וקונפיגורציה של כל ה- roles, permissions, account בהתאם, ניטרול default accounts (או לפחות החלפת הסיסמא הדיפולטית) וקונפיגורצית מערך היומנים וההתראות . ניתן להשתמש בכלים לשכפול דיסק כדי לקחת image של שרת מוקשח אחד ולהעתיקו לשרת אחר בשביל להקשיח אותו באותה קונפיגורציה.
- טיפול ב-Third Party Product Vulnerabilities: שרתי צד שלישי ומוצר י web חיצוניים יכולים להכיל פגיעויות רבות ולכן מהוות סכנה אל מול תוקפים. הטיפול בכך הוא לדאוג לעקוב אחרי patches שיוצאים לכל תוכנות ה- third party ולעדכן לגרסאות חדשות מעת לעת . כמדיניות יש להתקין את כל עדכוני החומרה ותוכנה בכל מערכות הארגון , גם מידה והאפליק ציה אינה אקטיבית . על האדמיניסטרטורים לעקוב אחר שחרור עדכונים וגרסאות ולשאוף ל- 0-day vulnerability policy, דהיינו התקנה מיידית בכל שחרור עדכון , המבטיח הגנה מקסימלית אל מול האיומים בתחומים אלו.
- טיפול ב-Remote Administration Flaws: אתרים רבים מאפשרים remote administration בכדי לשלוט במערכת מרחוק. כלי זה אמנם חזק אך מהווה מקור לפגיעויות, בייחוד כאשר נשה שימוש בממשקי אדמין "חבויים" (דוגמא לכך היא עמוד html המאפשר פעילות אדמין שאין אליו לינק, תחת הנחה שבכך תימנע גישת משתמשים לא מאושרים / תוקפים לאותו עמוד ; כמובן שישנם כלים אוטומטיים למציאת עמודים כאלו בשימוש תוקפים). כדי להגן בפני פגיעויות אלו יש לנטרל את כל אפשרויות האדמיניסטרציה מעל האינטרנט , לבצע הפרדה בין אפליקציות אדמיניסטרציה

ובין האפליקציה הראשית , הגבלת הפעולות שניתן לבצע דרך remote administration והחלת אמצעי אותנטיקציה חזקים (למשל כרטיס חכם למתחבר לשירותי אדמיניסטרציה מרחוק).

- <u>טיפול במשתמשים משותפים או לא שמישים</u> : כאשר נעשה שימוש במשתמשים משותפים , רמת הזהירות של המשתמשים העובדים דרך אותו account נוטה לרדת, כיוון שמשתמשים נוטים להיות פחות זהירים כשלא ניתן לאתר אותם (לא ניתן לזהותם באופן מיידי). לטיפול בבעיה זו תוך שמירה על האפשרות להגדיר את אותן הרשאות לקבוצה של משתמשים ללא כפילות הגדרות יש להשתמש ב- roles עליהם מגדירים הרשאות , ולשייך כל משתמש יחיד ל- role אליו מתאים. יש לבצע review כל פרק זמן בכדי לעדכן במידת הצורך שיוך של משתמשים ל- roles בהתאם להגדרת תפקידם – יתכן למשל מצב בו משתמש עבר תפקיד ונוסף ל-role מסויים ללא הסרתו מהקודם (אגירת כוח לא נחוץ בידי משתמש).
  בנוסף לכך יש להתמודד עם חשבונות משתמשים לא אקטיביים העלולים להוות מוקד לפריצה , שכן לעומת משתמש אקטיבי בו הסיכוי לזהות שימוש לא חוקי גבוה , על משתמש רדום חסר ניטור . על כן יש למחוק משתמשים רדומים או לכל הפחות לחסום אותם , ובתוך כך להגדיר מדיניות review אחת לכמה זמן כדי לזהות מי מהמשתמשים אקטיבי ומי לא , ובכדי לזהות משתמשים לא ידועים . האחרון נועד לאיתור משתמשים שיתכן ונוצרו ע"י תוקף שפרץ לחשבון אדמין ויצר לעצמו משתמש חדש (כך סיכויי גילויו נמוכים לעומת אם ימשיך לעבוד דרך חשבון האדמין).
- <u>Separation of Duties</u> : לפי עקרון ה- least privileges כל תוכנית או משתמש צריכים לעבוד תוך שימוש בסט ההרשאות המצומצם ביותר המאפשר להם לבצע את עבודתם . אם כל משתמש / תהליך יעבוד תחת ההרשאות המצומצמות ביותר , הנזק שיכול להגרם בשימוש לקוי או ע"י תוקף שהצליח לחדור לתהליך או לפרוץ למשתמש יהיה מוגבל.
  הפרדת תפקידים מאפשרת מניעת חשיפה של מידע חסוי לגורמים שאינם מורשים לכך , גם שלא במתכוון , וכמו כן יכולה למנוע נזקים שעלולים להיעשות ע"י משתמשים . יישום ההפרדה צריך להיעשות בכל השכבות : מערכות הגיבוי , מערכת ההפעלה , מערכות בסיסי הנתונים , פיתוח ואפליקציה. ההפרדה מבטיחה שרק מי שמורשה ומומחה בתחומו יוכל לגשת , לשנות ולהשפיע בתחום אליו משוייך , ובכך ניתן למנוע נזקים וגישות למקומות בהם אותם משתמשים עלולים לגרום לנזקים (גם שלא במתכוון).

6)

חשיבות הרצת אפליקציה בהרשאות המינימליות באה לידי ביטוי בהרשאות במערכת ההפעלה לגבי הרצת תהליכים ובהרשאות מערכות בסיסי הנתונים לגבי שמירה על ה-data. אם מחילים מדיניות הרשאות מינימליות לכל המשתמשים והתהליכים הרצים תחת מערכת ההפעלה , ניתן למנוע פגיעה מכוונת או לא מכוונת ע"י תהליך או משתמש – מכוונת : תוקף / משתמש פנימי זדוני מנצלים הרשאות הגישה לשירותים שונים לניצול לצורכי התקיפה , או לא מכוונת : משתמש בעל הרשאות לתהליכים בתחום שאינו אחראי בו עלול לגרום לנזקים מחוסר ידע או טעות . דוגמא לניצול הרשאות ע"י תוקף : תוקף הפורץ לחשבון משתמש כלשהו בעל הרשאות ליצור משתמשים חדשים , יוצר לעצמו משתמש חדש בעל הרשאות מקסימליות תחתיו יכול לעבוד עם סיכויי גילוי נמוכים לפרק זמן ארוך (משתמש סמוי ולגיטימי מבחינת המערכת).
אם מחילים מדיניות הרשאות מינמליות במערכות בסיסי הנתונים , ניתן למנוע פגיעה מכוונת או לא מכוונת במידע הארגוני או חשיפתו. אי דאגה לכך מעלה את הסיכון לפגיעה במידע , למשל ע"י משתמש בעל הרשאות כתיבה בתחום מידע שלא נוגע לו , העלול לעשות נזק ולדרוש קבצים קריטיים . לחילופין יהיה התוקף, כאשר אין הרשאות מינימליות למשתמשים השונים בארגון , בעל סיכוי גבוה יותר להצליח להשיג מידע או לשנותו שכן לרשותו יותר משתמשים ש"כדאי" לו לפרוץ אליהם, ודרכם יוכל לגשת למידע סודי או לבצע פגיעה במידע.
לסיכום, ברמת מערכת ההפעלה הרשאות מינימליות באות להגן ולמנוע גישה לקבצים רגישים או הרצת תהליכים שעלולים לבצע שינויים מסוכנים במערכת, וברמת בסיס הנתונים הרשאות מינימליות באות להגן על אי חשיפת מידע לגורמים לא מורשים ומניעת שינוי ופגיעה במידע רגיש.

7)

עקרון ה- defense in depth הוא אסטרטגיית הגנת מידע לפיה שכבות שונות של הגנה פרוסות על גבי מערכת ארגונית תוך הגנה על כל שכבה : האפליקציה, מערכת ההפעלה, מערכות הגיבוי, מערכות בסיסי הנתונים וכו'. גישה זו באה להגן בפני פגיעויות אבטחה שמקורם במשתמשים , טכנולוגיות או תהליכים לכל אורך החיים ושכבות המערכת . מטרת ההגנה בשכבות היא לכסות כמיטב היכולת כל נקודת פירצה אפשרית לאורך שכבות המערכת הארגונית בכדי למנוע, או לפחות לעכב ולהפחית את הנזק שיכול להגרם בני סיונות פריצה למערכת.
להלן שתי דוגמאות להגנה בשכבות מדוגמת החנות הוירטואלית שנותחה בשיעור :

- שימוש במספר שכבות firewall כדי לאפשר סינון מבוסס IP, ports שיאפשר גישת סוגי משתמשים / שרתים שונים לשירותים ההכרחיים להם בלבד : firewall לוקאלי על המכונה (כמו iptables ב-Linux), firewall חיצוני בשימוש חברת ה- hosting. לדוגמא, יש לאפשר גישה על פורט 22 (SSH) למשתמשים המגיעים מכתובות מסויימות בלבד המזוהות עם אדמיניסטרטורים של המערכת או מפתחים , ובכך להקטין את חשיפת השירותים הללו לשאר העולם. שימוש במספר firewalls מספק הגנה טובה יותר כך שגם אם תוקף מצליח לעבור חומה אחת, הוא ייתקל בנוספת.
- מימוש access control בשכבות : מימוש access control גם ברמת ה-firewall, המבצעת סינון ראשוני על בסיס ip ופורט, ובנוסף מימוש access control ברמת שרת ה- apache. האופן הטוב ביותר לאפשר מימוש שכבתי כנ"ל בדוגמת החנות הוירטואלית הוא הפרדת שרת ה- apache לשני

שרתים שונים, כאשר לכל אחד מהם כתובת ip שונה וכך ה- firewall יכול לבצע את הסינון הראשוני המבוסס ip ופורט. בתוך כל שרת ימומש access control ברמת שרת ה- apache עצמו, כאשר שרת אחד כעת מחזיק את ה- store admin, CMS admin ושרת שני את ה- store וה- web site, כלומר מידע ציבורי שחשוף לכל העולם. ה-access control ב-apache servers יהיה מבוסס סוגי משתמשים ויגדיר את הרשאותיהם.

8)

עקרון ה-reduce attack surface גורס כי יש להפחית את אפשרויות גישת משתמשים לא מורשים או שלא התב צעה עליהם אותנטיקציה דרך נקודות חשופות במערכת, כגון קלטי משתמש, פרוטוקולים, ממשקים ושירותים שונים. לפי עקרון זה יש לנטרל פונקציונאליות שאינה הכרחית ובכך להקטין את כמות הפגיעויות והסיכון לפריצה למערכת. לפי עקרון זה, הקטנת הנגישות למינימום מסייעת באבטחת מערכות הארגון בכך שמקטינים את אזור התקיפה הפוטנציאלי.

להלן שתי דוגמאות ל-attack surface reduction מדוגמת החנות הוירטואלית שנותחה בשיעור:

- שימוש בשני שרתים נפרדים, כאשר האחד עבור ה-store admin, CMS admin והשני עבור החנות והקטלוג. השרת הראשון נגיש לכל העולם, ואילו השרת השני יהיה נגיש רק מכתובות IP ופורטים ספציפיים, אלו של מנהלי החנות ומפתחי האפליקציה. בשיטה זו גישה לשרת השני, המכיל מידע ושירותים רגישים יותר, לא תתאפשר אלא מטווח מצומצם יחסית של כתובות ופורט ים.
- שימוש בפרוטוקולים שונים מעל פורטים שונים לקבוצות שונות של משתמשים: גישת לקוחות נעשית מעל פרוטוקול HTTP בעוד גישת מפתחים או אדמיניסטרטורים של המערכות השונות נעשית מעל פרוטוקולים אחרים נוספים, כגון SSH או FTP/MYSQL מעל SSL. הפרוטוקולים נבדלים ביניהם בפור טים מעליהם רצים, וכך ניתן להגביל קבוצות משתמשים לגישה מעל פרוטוקולים ספציפיים. הגבלת גישה זו מקטינה את שטח התקיפה הפוטנציאלי בכך שמאפשרת גישה מעל מספר מצומצם של פרוטוקולים, ובכך פונקציונאליות מוגבלת עבור תוקף (למשל, תוקף ללא גישה ל-SSH לא יוכל לבצע דברים שתוקף עם גישה זו יוכל).

9)

עקרון ה-separation of duties and least privileges גורס כי יש להגדיר תפקידים ולכל אחד מהם לתת את מינימום ההרשאות הנדרשות לו על מנת לבצע את תפקידו ולא יותר מכך. חלוקת משתמשים לבעלי תפקידים והגדרת הרשאותיהם בהתאם יכולה למנוע חשיפת מי דע לא מורשית או לא מכוונת, שיכולה בכך למנוע טעויות משתמשים ונזק למערכת. הגבלת הרשאות בעלי התפקידים חוסמת את הנזק הפוטנציאלי שיכול להגרם ע"י משתמש או תהליך שמנוצל לרעה, למשל ע"י תוקף.

להלן שתי דוגמאות ל-separation of duties and least privileges מדוגמת החנות הוירטואלית שנותחה בשיעור:

- הפרדת בעלי התפקידים בארגון לקבוצות שונות בעלי roles שונים, כאשר לכל אחד מהם גישה לכלים הספציפיים הנדרשים לביצוע תפקידם. בדוגמת החנות הוירטואלית חולקו המשתמשים לשלוש: מפתחים, מנהלי שיווק ומשתמשים ציבוריים, ולכל קבוצה ניתנו הרשאות וגישה לשירותים הנדרשים לה בלבד. למשל, לאדמיניסטרטורי המערכת לא תהיה גישה ל בסיס הנתונים, ולמפתחים להם כן תהיה גישה ל בסיס הנתונים מתוקף תפקידם, לא תהיה גישה לתהליכים או קבצים של תשתיות הארגון.
- אחסון המידע במספר בסיסי נתונים והגדרות גישה מינימליות של כל מנגנון לבסיסי הנתונים הנדרשים לו בלבד, עם ההרשאות המצומצמות ביותר הנדרשות לו. בדוגמת החנות הוירטואלית ההפרדה באה לידי ביטוי באחסון כרטיסי האשראי בבסיס נ תונים נפרד מהאחרים ובאופן מוצפן, ואפשור גישה לבסיס נתונים זה למנגנון האחראי על ה- checkout בלבד ועם הרשאות כתיבה בלבד. באמצעות הגדרות אלו, לא יתאפשר ניצול תהליכים ע"י תוקפים על מנת לגשת לקרוא מבסיס הנתונים הרגיש, שכן אלו ללא הרשאות קריאה.

10)

משמעות המשפט "the web application is part of the enterprise security perimeter" היא שאפליקציית הרשת הינה חלק ממערך האבטחה של מערכות הארגון. החלק הראשון של מערך האבטחה הוא בשכבת הרשת, הכוללת מערכי אבטחה שונים המתבססים על Firewalls, הקשחה, הצפנה ע"י עבודה מעל פרוטוקול SSL וכו', אך חלק זה אינו נגיש ואינו רואה את שכבת האפליקציה. משתמשים חוקיים של המערכת, ביניהם יכולים להימנות גם תוקפים, יכולים לעקוף את כל מערך האבטחה בשכבת הרשת ולכוון את ההתקפה על שכבת האפליקציה, שכן כל גישת משתמש באופן מורשה נראית לגיטימית ברמת הרשת, ולא ניתן ברמה זו לאבחן פעולות לא תקינות או זדוניות. על כן, יש לאבטח ולהגן על אופן עבודת משתמשים עם אפליקציית הרשת בכדי למנוע פגיעויות ברמת האפליקציה. המשפט לעיל בא לאמר כי לא מספיק לאבטח ברמת הרשת, וכי אבטוח והגנה ברמה האפליקטיבית הכרחית וחשובה לכלל ההגנה על מערכות הארגון.

בכדי להדגים זאת על החנות הוירטואלית שניתחנו בהרצאה, ניתן להניח למשל גישת תוקף דרך משתמש חוקי במערכת, המשתמש ב-SQL injection במקומות מסויימים בקלטי משתמש בכדי להשיג מידע אודות רכישות של משתמשים אחרים (דוגמא לכך ניתנה בהרצאה הראשונה). שאילתת המשתמש היא לכאורה לגיטימית, אך רק ברמת הרשת, בעוד ברמה האפליקטיבית ברור כי אינה כזו. דרכי התמודדות לדוגמא עם התקפה זו היא ע"י

שמירה על הרשאות גישה מינימליות של האפליקציה למסד הנתונים (למשל, לכתיבה בלבד במקרה זה), ולידציה של קלט המשתמש ופילטור שאילתות מסוכנות, ניקוי היסטוריית שאילתות וכו'. ברמה יותר כללית, לאפליקציה יש גישה לבסיסי הנתונים ול-File System במערכת ההפעלה ולכן תוקף יכול דרך האפליקציה לגשת לאותן מערכות, ולגרום לאפליקציה לבצע בעבורו בשמה גישות ל-DB או פעולות קריאה/כתיבה/ביצוע מסוכנות או לא מורשות במערכת ההפעלה.