האוניברסיטה הפתוחה
המחלקה למתמטיקה ולמדעי המחשב

כריית תפקידים בהינתן אילוצי הפרדת אחריות

עבודת מסכמת זו הוגשה כחלק מהדרישות לקבלת תואר
"מוסמך למדעים" M.Sc. במדעי המחשב
באוניברסיטה הפתוחה
המחלקה למתמטיקה ולמדעי המחשב

על-ידי
ולדימיר פונומריוב

העבודה הוכנה בהדרכתו של פרופ' תמיר טסה

מרץ 2024

תוכן העניינים

רשימת איורים 1
תקציר 2
1. מודל RBAC 3
1.1 מוטיבציה ומושגי יסוד 3
1.2 השוואה עם מודלי בקרת גישה אחרים 5
2. הגדרת מודל RBAC והרחבות 7
2.1 מודל $RBAC_0$ 8
2.2 מודל $RBAC_1$ 11
2.3 מודל $RBAC_2$ 14
2.3.1 דוגמאות נפוצות של אילוצים ב-RBAC 14
2.3.2 הגדרה של מודל אילוצים ב-RBAC ושיטת רישום של האילוצים 16
2.3.3 אכיפה של אילוצים במודל $RBAC_2$ 19
2.4 מודל $RBAC_3$ 21
3. כרית תפקידים - בעיית RMP 23
3.1 Basic RMP ו-$\delta - Approx$ RMP 24
3.2 אלגוריתם לכריית התפקידים בעזרת תעדוף תת-קבוצות 28
3.2.1 עקרונות ייסוד עבור אלגוריתמים לכריית תפקידים 28
3.2.2 אלגוריתם CompliteMiner 29
3.2.3 אלגוריתם FastMiner 36
3.3 אלגוריתם היוריסטי לפתרון Basic RMP ו-$\delta - Approx$ RMP 41
4. כריית תפקידים בנוכחות אילוצים 48
4.1 מבוא והגדרות פורמאליות 48
4.2 בעיית $RMP - SoD$ 50
4.3 אלגוריתמים ליצירת SMER בגישת $post - processing$ 53
4.4 אלגוריתמי כריית תפקידים מונחי SoD 57
4.5 מטריצת UPA אינה עקבית עם אילוצי SoD 59
5. סיכום 63
רשימת מקורות 64

רשימת איורים

איור 1. יחס הכלה בין מודלי RBAC 7
איור 2. תיאור סכמתי של RBAC 9
איור 3. דוגמה לירושה בין תפקידים 12
איור 4. דוגמה ל-public & private permissions ..... **שגיאה! הסימניה אינה מוגדרת.**
איור 5. יחסים שימור בין סוגים שונים של אילוצים 20
איור 6. פסאודוקוד עבור $CompleteMiner$ 31
איור 7. דוגמה של UPA 34
איור 8. תוצאה של $CompleteMiner$ 34
איור 9. פסאודוקוד עבור $FastMiner$ 37
איור 10. פסאודוקוד עבור אלגוריתם לפתרון $\delta - Approx\ RMP$ 42
איור 11. תוצאות של $FastMiner$ כולל חישוב שטחים עבור UPA 43
איור 12. פסאודוקוד של $SMER2\ \ Post\ Processing$ 54
איור 13. פסאודוקוד של האלגוריתם $SMERt\ Post\ Processing$ 56
איור 14. פסאודוקוד של $\delta - Approx\ RMP\ SoD\ aware\ routine$ 62

תקציר

מודל בקרת גישה מבוסס תפקידים - RBAC הוא מודל נפוץ מאוד למימוש אבטחת מידע בהרבה מערכות מידע ארגוניות. המודל מאפשר לבנות מבנה תפקידים אשר מתאים למבנה של הארגון ובכך הופך את מימוש מדיניות אבטחת מידע לפשוט יותר ואינטואיטיבי יותר. בחלק הראשון של העבודה (פרקים 1,2) נסקור בהרחבה את המודל RBAC כולל הרחבות שלו, כגון היררכיית תפקידים ואילוצים.

אחד האתגרים הגדולים של מעבר למודל RBAC במערכות קיימות הוא בניה של מבנה תפקידים מתאים. אחד הפתרונות לבניה של מבנה התפקידים הוא להשתמש באלגוריתמים לכריית תפקידים. החלק השני של העבודה (פרק 3) מוקדש לתיאור פורמאלי של שתי בעיות אשר מגדירות את בעיית כריית תפקידים: בעיה בסיסית של כריית התפקידים - Basic RMP ובעיה מקורבת Approx RMP – δ .

בספרות מתוארות המון שיטות לפתרון הבעיות האלו. חלק מהשיטות מבוססות על האלגוריתמים שמחלצים רשימת תפקידים פוטנציאליים מתוך היחס הקיים בין משתמשים והרשאות. בעבודה הזאת נציג שני אלגוריתמים מהמשפחה הזאת: CompleteMiner, FastMiner.

בסוף החלק השני נציג אלגוריתם לפתרון בעיות Basic RMP ו-Approx RMP – δ אשר מבוסס על האלגוריתם FastMiner. האלגוריתם הזה מתאים רק למודל RBAC בסיסית – ללא אילוצים.

בחלק האחרון של העבודה (פרק 4) נעסוק בהרחבה של הפתרון לכריית תפקידים שמתאים למודל שכולל אילוצים סטטיים או דינאמיים. בחלק הזה נכלול תיאור של שתי שיטות להתמודדות עם אילוצים סטטיים. שיטה ראשונה היא post – processing, ז״א השיטה מתאימה את הפתרון למודל ללא אילוצים למודל עם האילוצים. מכיוון שהאילוצים אינם חלק של התהליך כריית התפקידים יכול ליוצר מצב שחלק האילוצים אינם בר אכיפה במבנה התפקידים שהתקבל מקודם, או לחלופין שחלק מהתפקידים אינם אפשריים לפתרון סופי עקב האילוצים.

על מנת להתגבר על הבעיה הזאת נתאר שיטה נוספת, אשר מאפשרת לשנות את התהליך כריית התפקידים כך שיתחשבו באילוצים תוך כדי כריית התפקידים עצמה. בנוסף נתאר שדרוג של FastMiner אשר מאפשר לקבל את מבנה התפקידים אשר מותאם למודל אשר כוללת תפקידים דינאמיים.

1. מודל RBAC [1]

1.1 מוטיבציה ומושגי יסוד

מאמצע המאה הקודמת מערכות מחשוב נכנסות יותר ויותר לתחומים שונים בחיי היום יום. בד בבד, גם כמות המידע האישי שהופך לדיגיטאלי הולכת ועולה. תהליך זה מלווה בהתפתחות מנגנוני הגנה על המערכות המחשוב, השירותים שאנו צורכים והמידע הדיגיטאלי שלנו. אחת מאבני היסוד של אבטחת מידע ומערכות מחשוב היא בקרת גישה. בקרת גישה היא מנגנון המאפשר להגביל גישה למידע או ביצוע פעולות רק ע״י משתמשים המורשים לכך. יש מגוון מודלים למימוש בקרת גישה. אחד המודלים הוא - בקרת גישה מבוססת תפקיד (Role Based Access Control – RBAC)

במודל RBAC תפקיד הוא יחידה בסיסית עבורה מוגדרות הרשאות גישה במערכת. אחראי אבטחת המידע בארגון יכול לבנות מדיניות גישה על בסיס התפקידים הקיימים בארגון. לאחר מכן, כל משתמש ישויך לתפקיד או מספר תפקידים. לכל תפקיד יוגדרו ההרשאות המתאימות והמשתמש יוכל לבצע פעולות ולגשת למידע בהתאם לתפקידו. חשוב להדגיש שאין קשר בין ידע או מיומנויות של המשתמש לפעולות שהוא יכול לבצע במערכת, ההרשאות נגזרות מתפקידו בלבד. לדוגמה, גם אם טייס יודע כיצד יש לבדוק את המטוס לפני טיסה, לרוב יהיה צוות טכנאים שזאת *אחריותם* ורק הם יהיו מורשים לאשר את תקינות המטוס.

בצורה הזאת תפקיד מגדיר גם קבוצה מסוימת של המשתמשים וגם מקבץ הרשאות שלהם. לאורך הזמן אנשים בארגון עוברים תפקידים, וגם ההרשאות במערכות משתנות כל הזמן. לעומת זאת, מיפוי תפקידים והרשאות בארגון הם יציבים לאורך הזמן, וגם אם יש צורך להחליף הרשאות לקבוצה מסוימת של המשתמשים, הרבה יותר קל לעשות זאת במונחי התפקידים ולא משתמשים ספציפיים.

בקרת גישה מבוססת תפקידים מתאימה למגוון רחב של מערכות, בין היתר בצבא, בגופי הממשלה ובחברות פרטיות. המודל מאפשר:

- לקבוע מדיניות הרשאות על סמך תפקידו של המשתמש בארגון
- לנהל מדיניות גישה בצורה ריכוזית ולאכוף דרישות או צרכי אבטחת המידע של הארגון
- שיוך משתמשים לתפקידים דורש פחות מיומנות באבטחה מאשר קביעת מיפוי ישיר בין משתמשים להרשאות המתאימות.
- להתייחס אל מדיניות של כל ארגון כאל ייחודית ולא להיות כפוף למוצרי מדף קיימים בשוק.

באופן כללי, המודל מאפשר שימוש במונחים של משתמש, תפקיד, הרשאה וביחסים ביניהם לקבוע את כלל המדיניות של הארגון מבחינת בקרת גישה למידע ופעולות. עם זאת המודל מאפשר גמישות כך שכאשר צרכים של ארגון ישתנו יהיה ניתן בקלות לשנות גם את המדיניות.

המודל של RBAC בנוסף למונחי בסיס (תפקידים, הרשאות ומשתמשים) מאפשר לאכוף בצורה ישירה שלושה חוקי אבטחת מידע:

- Least privilege – כל תפקיד מקבל ההרשאות המינימליות הנדרשות לו לביצוע תפקידו
- Separation of duty – דרישת מעורבות מספר משתמשים או תפקידים לביצוע פעולה רגישה
- Data abstraction - מנגנון ההרשאות יכול להיות הרבה יותר רחב מקריאה או כתיבה של נתונים, כגון ביצוע חיוב מחשבון או זיכוי חשבון וכו'

למרות שהמודל מאפשר אכיפת החוקים הנ"ל, הם אינם חלק מהמודל במובן שהם אינם מובנים בו. מנהל אבטחת המידע יכול לממש את המודל בצורה כזאת שסותרת את החוקים. לדוגמה, מתן הרשאות יתר לתפקיד או מתן תפקידי יתר למשתמש.

כיום מודל RBAC הוא חלק ממנגנוני האבטחה של מערכות הפעלה, שירותי ענן, אפליקציות web וכו'. למרות שהמודל נתמך על ידי רוב מערכות התשתית האלה, עדיין לא קיים סטנדרט יחיד למודל. קיימות מספר אינטרפרטציות של המודל, להלן רק שתי דוגמאות:

- מודל שמאפשר ירושה
- מודל שמאפשר לקבוע הגבלות (אילוצים) על קשרים בין משתמשים לתפקידים או בין תפקידים להרשאות.

למרות שיש מגוון רחב של תת-מודלים של RBAC, הם עדיין לא מאפשרים לכסות את כל האתגרים בתחום בקרת גישה, למשל המודל אינו מאפשר להגדיר סדרת אירועים אשר תקבע האם תפקיד מסוים יקבל גישה למידע או יקבל הרשאה לביצוע פעולה במערכת. עם זאת מודלים יותר מורכבים יכולים להתבסס על RBAC ולהרחיב אותו בהיבטים נוספים כגון הוספת לבקרת גישה תכונות של זמן ומיקום או קשרים בין המשתמשים. לדוגמה רק רופא המשפחה של המטופל רשאי לגשת למידע רפואי שלו בשעות העבודה של המרפאה ולא כל רופא משפחה במערכת בכל שעה.

מושגי יסוד של המודל RBAC

אובייקט – ישות פאסיבית אשר מכילה או מקבלת מידע.

סובייקט – ישות אקטיבית (אדם או תהליך) שגורמת למידע לזרום בין האובייקטים וע"כ מחליפה את המצב של המערכת.

גישה – סוג של פעולה בין סובייקטים ואובייקטים אשר גורמת למידע לזרום ביניהם.

בקרת גישה – תהליך הגבלת גישה למשאבים של המערכת רק לישויות מורשות.

משתמש – האדם שמקיים אינטראקציה ישירה עם מערכת מחשוב

הרשאה – מקבץ אינטראקציות אשר סובייקט רשאי לבצע עם האובייקטים

תפקיד – ישות במערכת מחשוב אשר מוגדרת על קבוצת הרשאות שלה

חלק מהמושגים מוגדרים בהרחבה בפרק הבא.

1.2 השוואה עם מודלי בקרת גישה אחרים

הרבה מנגנוני בקרת גישה מתייחסים אל קבוצה של המשתמשים כיחידה עבורה מוגדרת מדיניות גישה במערכת. אבל חשוב להדגיש שבדרך כלל יש הבדל מהותי בין קבוצת משתמשים לבין תפקיד. קבוצת משתמשים מגדירה רק אילו משתמשים שייכים אליה, בעוד שתפקיד לעומת זאת מגדיר גם את ההרשאות שלהם. אם מנגנון בקרת גישה מיישם את המונח "תפקיד" ומספק הרשאות למשתמש על בסיסו, אזי עליו לעמוד בשני תנאים.

- קביעת השתייכות של המשתמש לתפקיד וקביעת ההרשאות של התפקיד הינן פעולות באותו סדר גודל של מורכבות (רצוי שיהיו פשוטות)
- הגדרת התפקידים וההרשאות נעשית במקום מרכזי וניתנת רק למספר מצומצם של משתמשים.

במערכות מחשוב קיימות עוד שני מודלים לבקרת הגישה : MAC ו-DAC.

Mandatory Access Control (MAC) – מודל בקרת גישה מבוסס על תיוג של אובייקטים ומתן הרשאות לסובייקטים לפי רמת הגישה שלהם ותיוגו של האובייקט. קביעת תיוגים ורמות גישה הינם ריכוזיים ולא ניתנים להחלטת משתמש. לדוגמה אם משתמש יצר מסמך הוא לא יכול לקבוע מהי רמת הגישה אליו. כל מידע בארגון מתויג ברמת גישה מסוימת ורק בעל רמת גישה מספיקה יכול לגשת למידע. לדוגמה בצבא נפוץ לקטלג את המידע לפי תיוגים הבאים : בלמ"ס, שמור, סודי וסודי ביותר. התיוגים האלה מאפשרים לכל אחד לדעת מי רשאי לגשת למידע ומהן הגדרות השמירה של המידע (קיימת גם חלוקה נוספת ברמות גישה גבוהות ולא כל מי שיש לו רמת גישה "סודי" יכול להיחשף לכל המידע שמתויג כמידע "סודי"). המודל הזה יכול להזכיר את המודל RBAC, אבל הוא בסה"כ מימוש של מדיניות בודדת אשר מגדירה זרימה חד כיוונית של המידע לפי רמות הגישה. מנגנון RBAC, לעומת זאת, אינו מוגבל למדיניות אחת ויכול לממש מגוון מודלי גישה.

Discretionary Access Control(DAC)– מודל בקרת גישה בעל המאפיינים הבאים :

- לכל אובייקט במערכת יש משתמש יחיד שהוא בעליו, והוא קובע רמת גישה אל האובייקט
- במערכת יש משתמשי-על אשר יכולים לקבוע הרשאות לכל האובייקטים
- משתמש יכול להעניק את רמת הגישה שלו למשתמש אחר

המודל הזה מיושם במערכת הפעלה Unix כמודל בקרת גישה לקבצים. ב-Unix משתמשים וקבוצות מוגדרים בקבצים מסוימים ולכן קביעת השתייכות משתמש לקבוצה היא פעולה יחסית פשוטה. לעומת זאת הרשאות של קבוצות מוגדרות על כל הקבצים והתיקיות בנפרד. ולכן קביעת הרשאות הקבוצה היא פעולה מורכבת אשר בדרך כלל דורשת מעבר על כל הקבצים במערכת. הסיבה לכך היא שמתן הרשאות נעשה בצורה מבוזרת וכל משתמש יכול לקבוע הרשאות לכל הקבצים והתיקיות ששייכים לו.

שני המודלים האלה לא מתנגשים עם המודל RBAC. מערכת מסוימת יכולה לשלב ביניהם או גם לממש בעזרת RBAC את מנגנון MAC או DAC. מנגנון RBAC מאפשר גמישות: האם השליטה תהיה ריכוזית, מבוזרת או שילוב שלהן, הכל תלוי איך הוגדרו הרשאות ניהול של ה-RBAC.

2. הגדרת מודל RBAC והרחבות [1]

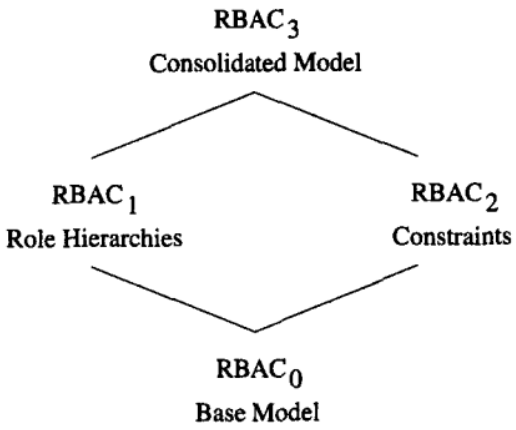
המודל RBAC הוא שם כללי למשפחה של מודלים לבקרת גישה אשר מבוססים על תפקידים. נהוג לציין ארבעה מודלים בסיסיים, עם זאת חשוב לציין שעם הזמן מתפתחים מודלים נוספים אשר עונים על צרכים מתקדמים יותר. להלן רשימה של מודלי RBAC הבסיסיים:

$RBAC_0$ - מודל יסוד אשר מגדיר את המושגים: משתמשים, תפקידים והרשאות

$RBAC_1$ – הוספת היררכיה של התפקידים ל-$RBAC_0$ (על בסיס ירושה)

$RBAC_2$ – הוספת הגדרת אילוצים ל-$RBAC_0$, בד"כ אילוצים חלים על קבוצות (תפקידים, הרשאות) או על קשרים ביניהם וקבוצת משתמשים

$RBAC_3$ – המודלים $RBAC_1$ ו-$RBAC_2$ אינם קשורים זה לזה ומוגדרים עצמאית, אך ניתן להגדיר מודל שכולל את שתי ההרחבות. למודל הזה נהוג לקרוא $RBAC_3$.



איור 1. יחס הכלה בין מודלי RBAC.

באיור 1 ניתן לראות את היחסים בין המודלים, כאשר כל מודל בסיסי $RBAC_0$ נמצא בתחתית ההיררכיה, מודלים $RBAC_1$ ו- $RBAC_2$ נמצאים מעליו וכוללים אותו, אך ביניהם אין שום קשר ומודל $RBAC_3$ מכיל את כל המודלים האחרים.

בהמשך הפרק נרחיב על כל אחד מהמודלים וניתן הגדרה פורמאלית עבורם.

2.1 מודל $RBAC_0$

משתמש– בן אדם אשר רשאי לבצע אינטראקציה עם המערכת, בדרך כלל קבוצת משתמשים תסומן כ- U ($Users$)

אימות (Authentication) – בדיקת זהות המשתמש אשר מבקש לבצע פעולה במערכת, בדרך כלל נעשית בזמן התחברות ראשונית של המשתמש למערכת. בשלב הזה המשתמש נדרש "להוכיח" שהוא באמת אותו סובייקט שבשמו הוא מתכוון לבצע פעולות במערכת. בדרך כלל משתמש מתבקש להכניס שם משתמש וסיסמא למערכת או לבצע תהליך אימות אחר.

הרשאה – אישור לסובייקט לבצע פעולה מסוימת על אובייקט כלשהו במערכת. במודל RBAC ההרשאה היא תמיד במובן חיובי, זאת אומרת שההרשאה מגדירה איזה פעולות וגישה לאיזה מידע במערכת **רשאי** לבצע בעל ההרשאה (גישת Whitelist). בספרות קיימת התייחסות למודלים בהם הרשאה מוגדרת באופן שלילי, ז"א בעל ההרשאה לא יכול לבצע פעולה כלשהי או לגשת למידע במערכת (גישת Blacklist). במודל RBAC מניעת גישה של הסובייקט למידע או ביצוע פעולה תתבטא כחוסר ההרשאה ולא כהרשאה שלילית.

מושג ההרשאה תמיד יהיה מוגדר ברמת המערכת. לדוגמה, במערכת ההפעלה אובייקטים יכולים להיות קבצים ותיקיות כאשר הפעולות הן קריאה, כתיבה והרצה. במקרה הזה ההרשאות הן - קריאה מקובץ, כתיבה לקובץ וכדומה. לעומת זאת, במסד נתונים האובייקטים הם טבלאות, קשרים בין טבלאות, תכונות של הטבלאות וכו', ופעולות הן הכנסה, שליפה, עדכון ומחיקה. במקרה הזה ההרשאות הן הכנסה לטבלה, מחיקה מטבלה וכו'. הרשאות יכולות להיות מוגדרות עבור אובייקט ספציפי או על קבוצת אובייקטים במערכת.

ניתן להגדיר בנוסף להרשאות ישות נוספת – חובות המשתמש. זה סוג של פעולות שמשתמש חייב לבצע על מנת שהמערכת תמשיך לתפקד כנדרש, אך הרחבה זאת יוצאת מחוץ למודל RBAC.

קבוצת הרשאות במודל RBAC בדרך כלל תסומן ע"י P (Permissions)

תפקיד – ישות אשר מוגדרת ברמת המערכת. בדרך כלל מגדירים תפקידים כך שמשתמשים באותו תפקיד יהיו בעלי אותה אחריות וסמכות במערכת, בהתאם לצרכי העבודה שלהם. לרוב קבוצת התפקידים של מודל $RBAC$ תהיה דומה (או אפילו זהה) למבנה התפקידים של הארגון. בדרך כלל תסומן כ-R (Roles)

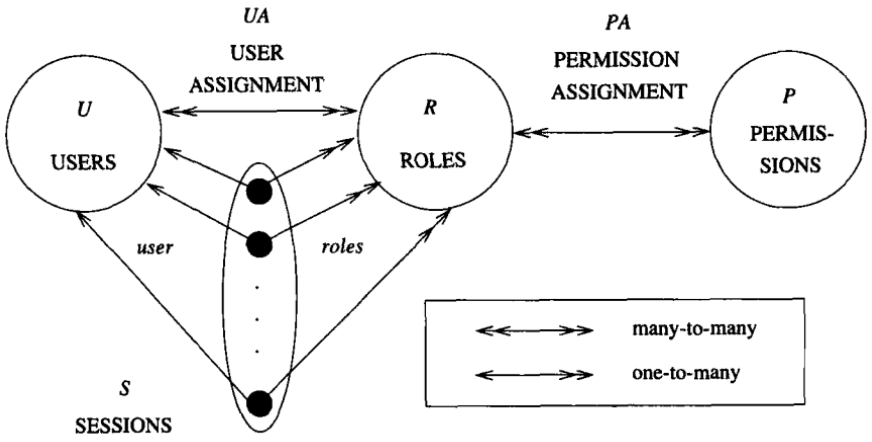
סמיכה (Authorization) – תהליך שיוך של המשתמש לתפקיד מסוים. בדרך כלל מתבצע ללא מעורבות של המשתמש, אלא על סמך הנתונים שיש במערכת.

למרות ששני השלבים של שיוך בן אדם לתפקיד, אימות וסמיכה, אינם תלויים אחד בשני (ז"א ששיטת האימות לא משפיעה על שיטת הסמיכה), שניהם קריטיים למערכות המשתמשות במודל RBAC, מכיוון שללא אימות לא ניתן לשייך אדם לאף משתמש וכיוצא בזה לשייך אותו לתפקיד במערכת. כמו כן, ללא שלב הסמיכה לא ניתן לקבוע את הרשאות המשתמש מכיוון שבמודל RBAC אין מיפוי ישיר בין משתמשים להרשאות.

התחברות – היא אינטראקציה בין משתמש למערכת, בדרך כלל היא מתייחסת לפרק זמן מוגבל. במהלך ההתחברות, משתמש מבצע אימות וסמיכה למספר תפקידים אשר רלוונטיים לאותה התחברות. קבוצת התפקידים יכולה להיות שונה בין התחברויות, אך אף פעם לא יקבל משתמש תפקיד שהוא לא משויך אליו כלל במערכת. בזמן התחברות ההרשאות שחלות על המשתמש הן איחוד כל ההרשאות של כל התפקידים שהוא משויך אליהם בהתחברות. במידה והמערכת מאפשרת *רב-חיבוריות*, משתמש מסוים יכול להיות מחובר במקביל בכמה התחברויות למערכת, אך בכל התחברות הוא יקבל הרשאות אשר שייכות לתפקידים של אותה התחברות ספציפית. המנגנון של ההתחברות מאפשר למודל RBAC לשמור על Least-privilege principle. משתמש בעל תפקיד עם הרשאות קריטיות למערכת יכול לעבוד עם המערכת ע"י התחברות עם תפקיד בעל הרשאות בסיסיות ורק כאשר הוא נדרש לבצע פעולה קריטית למערכת הוא יפעיל התחברות עם תפקיד בעל הרשאות מתקדמות/קריטיות. במודל $RBAC_0$ רק המשתמש מחליט איזה סוג של התחברות לבצע בכל גישה אל המערכת. קבוצת התחברויות במודל RBAC בדרך כלל תסומן כ-S (Sessions).

הקשרים בין הקבוצות במודל RBAC מוגדרים בעזרת מיפוים. מיפוי המשתמשים לתפקידים תסומן כ- UA (User Assignment) ומיפוי ההרשאות לתפקידים (Permission Assignment). היחסים האלה הם יחסים many-to-many. משתמש יכול להיות שייך למספר תפקידים ומספר משתמשים יכולים להשתייך לאותו תפקיד. באופן דומה, הרשאה יכולה להיות שייכת למספר תפקידים ותפקיד יכול להיות בעל מספר הרשאות. המבנה הזה הוא מהותי למודל RBAC. בפועל הרשאות חלות על המשתמשים, אבל הצבת מושג התפקיד כמתווך בין קבוצת משתמשים וקבוצת הרשאות מאפשרת גמישות רבה יותר ובקרה הדוקה יותר על היחסים בין משתמשים והרשאות. את המיפויים האלה נוח לייצג בעזרת מטריצות בוליאניות, כאשר כל כניסה במטריצה מיוצגת על ידי 1 מסמנת שיש שיוך בין משתמש לתפקיד או בין תפקיד להרשאה.

הקבוצות והיחסים של המודל RBAC מתוארות בצורה סכמתית באיור 2.



איור 2. תיאור סכמתי של RBAC

הגדרה פורמאלית של מודל $RBAC_0$

תהיה U קבוצת משתמשים, R קבוצת תפקידים, P קבוצת הרשאות, S – קבוצת התחברויות, אזי מתקיים:

$PA \subseteq P \times R$ – הוא יחס many-to-many בין קבוצת ההרשאות לקבוצת התפקידים, מיוצג כמטריצה בוליאנית

$UA \subseteq U \times R$ – הוא יחס many-to-many בין קבוצת המשתמשים לקבוצת התפקידים, מיוצג כמטריצה בוליאנית

$u: S \to U$ – פונקציית מיפוי של ההתחברות s_i למשתמש היחיד $user(s_i)$ (מוגדר לזמן חיים של ההתחברות)

$roles: S \to 2^R$ – פונקציית מיפוי של ההתחברות s_i לקבוצת R: $roles(s_i) \subseteq \{r | (user(s_i), r) \in UA\}$.
בזמן קיום התחברות משתמש יכול לקבל כל התפקידים שלו או רק חלק מהתפקידים, לכן $roles(s_i)$ היא תת-קבוצה של קבוצת התפקידים של המשתמש.
אפשר להגדיר גם יחס בין התחברות להרשאות $permissions(s_i)$: $\bigcup_{r \in roles(s_i)} \{p | (p, r) \in PA\}$

בדרך כלל לכל תפקיד תהיה משויכת לפחות הרשאה אחת, וכל משתמש יהיה משויך לפחות לתפקיד אחד, אך המודל אינו מחייב זאת.

דוגמה

נניח אנחנו מתבקשים להגדיר מודל RBAC עבור מערכת של קופה אוטומטית בסופר. קבוצת משתמשים U תהיה מורכבת מלקוחות הסופר ועובדים שלו, נניח בנוסף ללקוחות של הסופר, שהם משתמשים אנונימיים, בסופר יש שלושה עובדים: אבי ובן שהם עובדים כללים וגל מנהל הסופר. קבוצת הרשאות P היא הוספת מוצר לסל מוצרים, תשלום על סל מוצרים, הסרה של מוצר מהסל ושינוי מחיר של המוצר. קבוצת התפקידים R יכולה להיות מורכבת מהישויות הבאות: קונה, קופאי עזר וקופאי בכיר. אזי המודל RBAC המתאימה יכולה להראות כך:
מיפוי UA מיוצג ע"י מטריצה בוליאנית:

	קונה	קופאי עזר	קופאי בכיר
לקוח אנונימי	1	0	0
אבי ובן	1	1	0
גל המנהל	1	1	1

חשוב לשים לב, שלמרות שעובדי סופר משויכים לתפקידי קופאים, הם עדיין יכולים לרכוש מוצרים בסופר ולתפקד כקונים באינטראקציה עם הקופות האוטומטיות. אם זאת יכול להיות שלהם לא יכולים באותה התחברות להיות גם קונים וגם קופאים על מנת למנוע ניצול של הרשאותיהם לרע.

מיפוי PA:

	הוספת מוצר לסל	הסרת מוצר מסל	שינוי מחיר של המוצר בסל	תשלום על הסל
קונה	1	0	0	1
קופאי עזר	0	1	0	0
קופאי בכיר	0	1	1	0

אל כל סל רכישה בקופה ניתן להתייחס כאל התחברות בודדת, כאשר לקוח אנונימי לא נדרש לאימות ובשלב סמיכה הוא מקבל תפקיד של הקונה. עם זאת עובדי סופר, אבי, בן וגל, ידרשו לבצע הזדהות מול המערכת שכוללת שלב אימות (למשל ע"י כרטיס עובד) ושלב סמיכה וכך הם יכולים לקבל תפקידים שלהם. אם לדוגמה אבי מחליט לרכוש סל מוצרים בסופר הוא יצטרך לבצע התחברות כמשתמש אנונימי על מנת לקבל הרשאות של הוספת מוצרים ותשלום, ורק במידה והוא יצטרך לבצע הסרה של מוצר מהסל הוא יידרש להתחבר אל המערכת כאבי עובד סופר על מנת לקבל תפקיד של קופאי עזר.

2.2 מודל $RBAC_1$

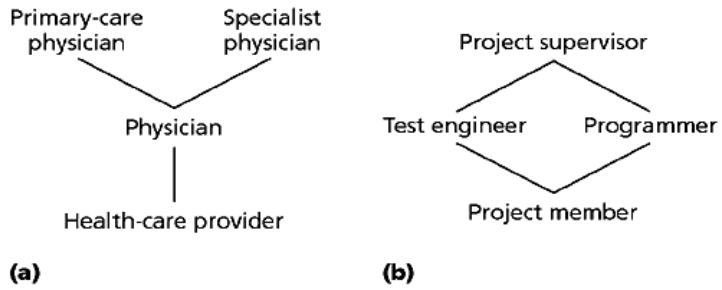
דוגמה שהבאנו בסעיף קודם יכלה לתת לנו מוטיבציה להגדרה של הרחבה ראשונה של מודל $RBAC_0$. בדוגמה ראינו שלושה תפקידים: "קונה", "קופאי עזר" ו"קופאי בכיר". "קונה" הוא ייחודי מבחינת הרשאות שלו, לעומת זאת קל לראות שיש קשר בין תפקידים של קפואים. "קופאי בכיר" מבחינת הרשאות הוא מעין הרחבה של תפקיד "קופאי עזר". זה מאוד מאפיין את המודל RBAC, כאשר חלק מהתפקידים הם הרחבה של תפקידים אחרים, מכיוון שבמבנה ארגוני הרבה פעמים תפקיד ניהולי או מקצועי יותר מקבל הרשאות נוספות להרשאות בסיסיות של תפקיד בסיסי. מבחינת ניהול הרשאות מתבקש ליצור קשר בין תפקידים בסיסיים (זוטרים) לבין תפקידים בכירים יותר, כך שהוספה או הסרה הרשאות לתפקיד בסיסי תשפיע על כל התפקידים הקשורים. לדוגמה, אם נחזור לקופה אוטומטית, נרצה להוסיף יכולת חדשה לקופה: הוצאת דו"ח קניות יומי. הדבר יגרום להוספת הראשה חדשה למערכת. הרבה יותר קל להוסיף הרשאה פעם אחד לתפקיד בסיסי של קופאי - "קופאי עזר" ובכך לחלחל אותו לכלל התפקידים שקשורים לתפקיד "קופאי עזר". הקשר כזה נקרא *ירושה בין תפקידים* (Role Hierarchy – *RH*)

מודל $RBAC_1$ מרחיב את המודל הבסיסי $RBAC_0$ ומגדיר את היחס *ירושה* בין התפקידים. הרחבה זו מאוד נפוצה וממומשת ברוב המערכות מבוססות RBAC. הירושה נדרשת על מנת לבנות מבנה תפקידים התואם את המבנה שקיים בארגון ולשקף את הקשר הין האחריות והסמכות בארגון. ניתן לראות דוגמאות נוספות לסוגי הירושה בין תפקידים באיור 3. נהוג למקם את התפקידים הבכירים יותר מעל התפקידים הזוטרים.

בדוגמה (a) התפקיד הזוטר ביותר הוא "אחות במרפאה" (Health-care provider), והתפקיד הבכיר יותר זה "רופא". התפקיד של ה"רופא" יורש מהתפקיד של ה"אחות" ובכך מקבל את כל ההרשאות שיש לאחות. בנוסף להרשאות של ה"אחות" לתפקיד "רופא" יכולות להיות לתפקיד הרופא הרשאות נוספות משלו. הירושה

הוא יחס טרנזיטיבי, ז"א התפקיד של "רופא מומחה" שיורש מהתפקיד "רופא" הוא גם יורש מהתפקיד "אחות". "רופא בכיר" גם כן יורש מתפקיד "רופא" ו"אחות", אך יהיו לו גם הרשאות נוספות שונות מאלו של "רופא מומחה".

בדוגמה (b) ניתן לראות ירושה מרובה. התפקידים "מתכנת" ו"מהנדס בדיקות" יורשים מהתפקיד "חבר בפרויקט", ולכל אחד מהם יש הרשאות משלו. התפקיד "מנהל הפרויקט" יורש גם מ"מתכנת" וגם מ"מהנדס בדיקות" (וגם מ"חבר הפרויקט" דרכם) ולכן יש לו את כל ההרשאות של "מתכנת" ושל "מהנדס הבדיקות".



איור 3. דוגמה לירושה בין תפקידים

הגדרה פורמאלית של $\mathbf{RBAC_1}$

מודל RBAC_1 מוגדרת ע"י קבוצות U, R, P, S ויחסים PA, UA כפי שהן מוגדרות ב-RBAC_0. בנוסף מודל RBAC_1 מגדירה יחס חדש בין איברים בקבוצה R שהוא RH.

$RH \subseteq R \times R$ – הוא יחס סדר חלש על קבוצת R ונקרא ירושה בין התפקידים (מסומן כ- $\geq$)

מתמטית, יחסי ירושה בין התפקידים זה יחס סדר חלש, מכיוון שהוא טרנזיטיבי, אנטי-סימטרי ורפלקסיבי:

- טרנזיטיבי – דרישת המודל, כפי שראינו קודם בדוגמאות של איור 3.
- אנטי-סימטרי – שני תפקידים שיורשים אחד מהשני, יהיו בעלי קבוצת הרשאות זהה ולכן הם בעצם אותו תפקיד ועל כן מיותר להגדיר שניהם בנפרד
- רפלקסיבי – קבוצת הרשאות של תפקיד היא ייחודית לו ולכן ניתן להגיד שתפקיד יורש את עצמו.

היחס RH מאפשר להגדיר מחדש את מיפוי תפקידים של התחברות

$$roles: S \rightarrow 2^R - \ roles(s_i) \subseteq \{r | (\exists r' \geq r)[(user(s_i), r') \in UA)]$$

ז״א התפקידים שמשתמש יכול לקבל בזמן התחברות הם התפקידים שהוא משויך אליהם ב-UA וכל התפקידים הזוטרים לתפקידים שלו. באופן דומה ניתן להגדיר מחדש את קבוצת הרשאות של המשתמש בזמן התחברות:

$$permissions(s_i): \bigcup_{r \in roles(s_i)} \{p | (\exists r'' \leq r)[(p, r'') \in PA]\}$$

מודל RBAC_1 מספיק גמיש על מנת לתת למענה למספר אתגרים שניתן להציב למודל ירושה בסיסית

2.3 מודל $RBAC_2$

הרחבה נוספת עבור מודל בסיסי $RBAC_0$ היא הוספת מושג ה"אילוצים" למודל. אילוץ הוא מושג מרכזי במודל RBAC ויכול לשמש כמניע עיקרי לשימוש ב-RBAC. הדוגמה הכי פשוטה לאילוץ במודל היא – *הפרדת אחריות*. נניח יש מערכת חשבונאית ובה שני תפקידים: "דורש תשלום" (D) ו"מאשר תשלום" (C). לא נרצה שאותו משתמש יוכל להפעיל את שני התפקידים ובכך לדרוש תשלום ולאשר אותו לעצמו. לכן נוכל להגדיר אילוץ מסוג הפרדת אחראיות כך שמשתמש מסוים לא יכול להיות ממופה גם לתפקיד C וגם לתפקיד D (או לא יכול להפעיל אותם באותה התחברות), תפקידי C ו-D יקראו גם *תפקידים סותרים*.

אילוצים הם כלי חזק אשר מאפשר להגדיר את המדיניות של הארגון ברמת על. במידה והמדיניות מגדירה תפקידים סותרים, אזי ניתן לבצע את ההשמה של המשתמשים לתפקידים בצורה מבוזרת ולא לדאוג שמשתמש מסוים יקבל הרשאות אשר יאפשרו לו לבצע פעולות אסורות בארגון.

באופן כללי כאשר השמת תפקידים והרשאות נעשות בצורה מרוכזת אזי אילוצים הם רק "כללי אצבע" עבור מנהל המערכת. אם נרצה להפוך את המודל למבוזר ולהעביר חלק מניהול המערכת לגורמים נוספים, אזי מנהל המערכת יכול להבטיח אי-הפרה של מדיניות הארגון על-ידי הגדרת אילוצים מתאימים.

אילוצים יכולים לחול גם על יחסים UA ו-PA וגם על פונקציות users ו-roles אשר מוגדרות על ה-sessions. כאשר אילוץ מופעל עליו להחזיר תשובה בינארית "מותר" או "אסור". חלק מהאילוצים יכולים להיות הגדרה פורמלית על הפונקציה או היחס וחלק יכולים להיות רק תיאור בשפה טבעית.

בהמשך העבודה נושא האילוצים יהווה חלק מרכזי, לכן בסעיף הזה נרחיב על הסוגי אילוצים שונים, שיטת רישום של אילוצים ודוגמאות למנגנוני הבקרה.

2.3.1 דוגמאות נפוצות של אילוצים ב-RBAC

תפקידים סותרים – Mutually exclusive roles

האילוץ הכי נפוץ במודל RBAC הוא – תפקידים סותרים. המשמעות של האילוץ היא שמשתמש לא יכול להיות ממופה לכל התפקידים בקבוצה של תפקידים סותרים. בדוגמה שראינו מקודם קבוצת תפקידים סותרים היא $\{D, C\}$, ולכן משתמש לא יוכל להיות משויך לשני התפקידים. האילוץ מסוג הזה מאפשר *הפרדת אחריות*, שזה בעצם ההגבלה על PA, כך שמשתמש לא יכול לקבל הרשאות אשר ביחד מאפשרות לו לעשות פעולות אסורות במערכת.

באופן כללי ניתן להגביל כל מיני שילובים של התפקידים, למשל אותו משתמש יכול להיות גם מתכנת וגם בודק אבל בפרויקטים שונים. אם מדובר באותו פרויקט אז זה אסור. באופן דומה ניתן להגביל כל מיני שילובים של ההרשאות.

הגבלת גודל של הקבוצות – Cardinality

אילוץ על ה-UA אשר מגדיר מספר מקסימאלי של משתמשים ממופים לאותו תפקיד, לדוגמה רק משתמש אחד יכול להיות מנהל המחלקה. באופן דומה ניתן להגביל מספר תפקידים אליהם משויך המשתמש. בנוסף אפשר להכיל את האילוץ על יחס PA, המשמעות שהרשאה לא יכולה להיות שייכת ליותר ממספר מסוים של תפקידים או מוגבל מספר ההרשאות שמקבל תפקיד מסוים. האילוץ הזה יכול למנוע התפשטות של הרשאות קריטיות בין המשתמשים.

האילוץ מדבר על מספר מקסימאלי של תפקידים/הרשאות. לעומת זאת מספר מינימאלי בדרך כלל קשה לאכוף מכיוון שאם משתמש נמחק, לא ברור איזה משתמש יכול להשתייך לתפקיד מסוים במקומו. עם זאת ניתן לממש מערכות אשר תומכות גם באילוץ הזה.

תפקידי קדם – Prerequisite roles

אילוץ חשוב נוסף הוא – *תפקיד קדם*. ברוב המקרים מודל תפקידים ב-RBAC מנסה לשקף את מבנה התפקידים בארגון. לדוגמה, תנאי מקדים להיות בודק QA בפרויקט מסוים זה להיות בעל ההרשאה לצפות בפרטי הפרויקט. האילוץ של תפקיד קדם משקף את הדרישה הזאת: רק משתמש אשר ממופה לתפקיד של צופה בפרויקט (PV - project viewer) יכול להיות ממופה לתפקיד בודק הפרויקט (PT- project tester). האילוץ הזה הוא על יחס UA.

באופן דומה ניתן להגדיר אילוץ על היחס PA: ניתן לשייך הרשאה p לתפקיד מסוים רק אם כבר שויכה אליו הרשאה q. לדוגמה, תפקיד יכול לקבל הרשאה לערוך את הקובץ, רק אם יש לו כבר הרשאה לקריאה של הקובץ, אחרת השמה של הרשאה לעריכה תהיה לא חוקית.

כל האילוצים על היחס UA הם תקפים רק אם קיים מנגנון הזדהות אשר מבטיח מיפוי חח"ע בין משתמשים לבני אדם אשר יוצרים התחברות עם אותו משתמש. במידה ואדם מסוים יכול ליצור התחברות עם מספר משתמשים, כל האילוצים מסוג הפרדת אחריות והגבלת גודל הקבוצות אינם אפקטיביים, מכיוון שגם אם יש חלוקה בין תפקידים היא אינה קיימת בין האנשים ולכן היא פורמאלית בלבד. באופן דומה נדרש מיפוי חח"ע בין הרשאות לפעולות אשר ניתן לבצע בפועל. במידה וקיימות מספר הראשות אשר מאפשרות לבצע פעולה קריטית כלשהי במערכת, אין טעם להפרדה לוגית ואילוצים על היחס PA מכיוון שתפקיד מסוים יכול לבצע את אותה פעולה בעזרת הרשאות אחרות שעליהן אולי אין אילוץ.

האילוצים לעיל מתייחסים בדרך כלל להתחברות בודדת, כך שמשתמש יכול להיות ממופה למספר תפקידים אך לא יכול להפעיל את כולם בהתחברות בודדת, אלא להפעיל רק תפקיד אחד. באופן דומה ניתן להגביל מספר התחברויות מקבילות פעילות שמשתמש יכול להחזיק. ניתן גם להגביל לכמה התחברויות בו זמנית יכולה להיות משויכת הרשאה ספציפית.

אל ההיררכיה של התפקידים ניתן להתייחס כאל אילוצים, כך לדוגמה את הירושה בין תפקידים ניתן להחליף באילוץ אשר דורש כי לפני השיוך של משתמש לתפקיד בכיר לוודא שהוא משויך לתפקידים זוטרים או לחלופין לפני שיוך של הרשאה אל התפקיד הזוטר לוודא שהיא משויכת אל כל התפקידים הבכירים יותר.

באופן הזה ניתן לומר ש-RBAC_2 מכיל את ה-RBAC_1. עם זאת עדיף להגדיר גם את הירושה וההיררכיה של התפקידים ולאכוף את התפשטות ההרשאות ישירות על ההיררכיה ולא בצורה עקיפה דרך אילוצים.

2.3.2 הגדרה של מודל אילוצים ב-RBAC ושיטת רישום של האילוצים [2]

בספרות נהוג להבדיל בין שלושה סוגים של אילוצים:

אילוצים סטטיים – אילוצים אשר מוגדרים בצורה סטטית, ללא התניה במצב הנוכחי של המערכת ובמצביה הקודמים. ז"א לדוגמה שהגבלה של תפקידים סותרים מתקיימת ברמת UA

אילוצים דינאמיים – אילוצים שהגדרתם מתייחסת למצב הנוכחי של המערכת. ז"א לדוגמה שמשתמש יכול להיות משויך אל תפקידים סותרים ברמת UA, אך לא יכול להפעיל שני תפקידים סותרים באותה התחברות.

אילוצים מבוססי היסטוריה – אילוצים מהסוג הזה מתייחסים לא רק למצב הנוכחי של המערכת, אלא גם למצבים שקדמו לו. ז"א לדוגמה שאם משתמש מסוים היה משויך לתפקיד כלשהו מקבוצה של תפקידים סותרים, אזי גם אם בהמשך הוא יוסר מאותו תפקיד, הוא בכל מקרה לא יוכל להיות משויך לאף תפקיד אחר בקבוצה של תפקידים סותרים.

לדוגמה נתאר אילוצים אפשריים במערכת ניהול מלאי של מחסן. בעזרת אילוצים סטטיים ניתן לקבוע כלל בו יש הפרדה בין פעולות מסוג דרישה למשיכת ציוד ואישור משיכה, כאשר יש אילוץ אשר לא מאפשר לאותם תפקידים לקבל הרשאות לשתי הפעולות. לעומת זאת במונחים של אילוצים דינאמיים ניתן לקבוע כלל כאשר אותו תפקיד לא יכול לבצע פעולת דרישה עבור משיכה וגם אישור משיכה על אותו אובייקט, עם זאת אין מניעה שאותו תפקיד גם יבצע פעולת דרישה וגם אישור עבור אובייקטים שונים. בהרחבה לאילוצים מבוססי היסטוריה ניתן לקבוע כלל בו דרישת משיכת ציוד חייבת להתבצע לפני אישור המשיכה וכל אחת מהפעולות חייבות להתבצע על ידי תפקידים שונים.

הגדרה פורמאלית של RBAC_2 ושיטת רישום של האילוצים

מודל RBAC_2 מוגדרת ע"י קבוצות U, R, P, S ויחסים PA, UA כפי שהן מוגדרות ב-RBAC_0. בנוסף מודל RBAC_2 מגדירה קבוצה חדשה C (Constraints) אשר מכילה אילוצים של המודל.

אילוץ הוא משפחה של קבוצות ההרשאות ה"פסולות" במערכת, כאשר עבור כל קבוצה פסולה אסור שיתקיימו כל איבריה. לדוגמה אחד האילוצים מסוג הפרדת אחריויות סטטי הוא דרישה שאותו משתמש לא יכול להיות משויך גם לתפקיד r_1 וגם לתפקיד r_2. אם כך האילוץ הוא משפחה של הקבוצות ה"פסולות" הבאות: $\{(u_1, r_1), (u_1, r_2)\}, \dots, \{(u_n, r_1), (u_n, r_2)\} \mid U = \{u_1, \dots, u_n\}$

הדרישה היא שמנגנון בקרת גישה ימנע קיום של אף קבוצה במלואה במערכת.

תיאור פורמאלי:

אילוץ הוא שלישיה מסודרת מהצורה הבא - (area , constraint , type)

area - תחום ההגדרה של האילוץ

constraint - משפחה של קבוצות האילוץ

type - סוג האילוץ: סטטי (s), דינאמי (d) ומבוסס היסטוריה (h).

כאשר area ו- constraint הם תת קבוצות של U, R, P

האילוץ מגדיר משפחה של קבוצות באופן הבא: עבור האילוץ $c = (A, B, type)$ משפחת הקבוצות הפסולות היא:

$$Q_c = \bigcup_{a \in A} (\{a\} \times B)$$

כאשר כל קבוצה $Q \in Q_c$ תקרא "קבוצת אילוץ" וכל איבר $q \in Q$ יקרא "דרישת הרשאה". לדוגמה אם משפחת אילוץ היא $(U, \{p_1, p_2\}, h)$, אזי לכל $u \in U$ תהיה קבוצת אילוץ $\{(u, p_1), (u, p_2)\}$ ודרישת הרשאה (u, p_i)

היחס של האילוץ הוא תת קבוצה של $X \times Y$, כאשר $X, Y \in \{U, R, P\}$. בדוגמה של $(U, \{r_1, r_2\}, s)$ היחס של האילוץ הוא UA.

דוגמאות

אילוצי הפרדת אחריויות מבוססי משתמש (חל על קבוצת המשתמשים U):

- $(R, \{u_1, u_2\}, s)$ - זוג משתמשים (u_1, u_2) לא ישויכו לאותו תפקיד.
- $(\{r_1, \dots, r_n\}, \{u_1, u_2\}, s)$ - זוג משתמשים (u_1, u_2) לא ישויכו לאותו תפקיד בקבוצת תפקידים $\{r_1, \dots, r_n\}$

אילוצי הפרדת אחריויות מבוססי תפקיד (חל על קבוצת התפקידים R):

- $(U, \{r_1, r_2\}, s)$ - אף משתמש לא יקבל גם תפקיד r_1 וגם r_2 (Mutually exclusive role)

- $(U, \{r_1, r_2\}, d)$ - הגרסה הדינאמית של האילוץ הקודם: משתמש לא יכול להפעיל תפקיד מהזוג (r_1, r_2) אם כבר הופעל תפקיד שני באותה התחברות (session)
- $(U, \{r_1, \dots, r_n\}, s)$ - אף משתמש לא יקבל את כל התפקידים מתוך הקבוצה $\{r_1, \dots, r_n\}$
- $((\{u_1, \dots, u_n\}, \{r_1, r_2\}, s))$ - אף משתמש מהקבוצה לא יכול לקבל את שני התפקידים. יש דרך להגדיר קבוצות דרך תפקיד דמה. נניח יש תפקיד r שאין לו אף הרשאה במערכת, אז ניתן להגדיר קבוצת משתמשים $U(r) = \{u \in U: (u, r) \in UA\}$. בצורה הזאת ניתן להגדיר קבוצה ללא חשש לזליגת הרשאות והביטוי עבור האילוץ יראה כך - $(U(r), \{r_1, r_2\}, s)$
- $(\{u_1, \dots, u_n\}, \{r\}, s)$ – אף משתמש מהקבוצה לא יכול לקבל תפקיד r. אילוצים מהסוג הזה לא נפוצים, אך הם יכולים לשמש בסיס לתיאור אילוצים יותר שימושיים. לדוגמה אילוץ מסוג $(u, \{r\}, s)$ מתאר את הדרישה שמשתמש u לא יכול לקבל מ-r ובכירים לו, אם במודל יש היררכיית תפקידים. האילוץ כזה בעצם מגדיר מעין סף עליון r בהיררכית תפקידים ש-u יכול לקבל.
- $(U, \{r_i, r_l\}, s)$ כאשר $1 \leq i < j \leq n$ – משפחת $\binom{n}{2}$ אילוצים, אשר דורשים שמשתמש $u \in U$ לא יקבל יותר מתפקיד אחד בקבוצה $\{r_1, \dots, r_n\}$.
- $(\{p\}, \{r_1, r_2\}, s)$ – דרישה שהרשאה p לא תשויך ליותר מתפקיד אחד בקבוצה $\{r_1, r_2\}$. דרישות כאלה יכולים לתאר אילוצים למודל חומה סינית.

אילוצי הפרדת האחריויות מבוססי הרשאות (חל על קבוצת ההרשאות P):

- $(U, \{p_1, \dots, p_n\}, c|c \in \{s, d, h\})$ – דרישה שאף משתמש לא יוכל לקבל את כל ההרשאות מהקבוצה, כאשר דרישה יכולה להיות סטטית, דינאמית או היסטורית
- באופן דומה עבור תפקידים - $(R, \{p_1, \dots, p_n\}, c|c \in \{s, d, h\})$

הגבלות של שיטת תיאור

כפי שהוזכר קודם, שיטת התיאור המוצגת בעבודה זאת היא מוגבלת רק לאילוצים פשוטים. נבחן לדוגמה את האילוץ הבא: בהינתן שני משתמשים u ו-v ושני תפקידים r ו-s:

- אסור שמשתמש מסוים ישויך לשני התפקידים
- במידה והמשתמש הראשון ישויך לתפקיד אחד, על המשתמש השני אסור להיות משויך אל התפקיד השני.

אין דרך ישירה בשיטת מתוארת כאן לתאר את האילוץ הזה בצורה פשוטה. האילוץ המבוקש דורש לאסור על ארבע קבוצת הבאות של שיוך בין התפקידים למשתמשים:

$$\{(u, r), (u, s)\}, \{(v, r), (v, s)\}, \{(u, r), (v, s)\}, \{(u, s), (v, r)\}$$

והאילוצים מסוג $c = (A, B, x)$ הם משפחה של האילוצים $\bigcup_{a \in A}\{a\} \times B$. ולכן על מנת לאחד את הארבע הקבוצות האלה נדרש להגדיר תתי-קבוצות של התפקידים וההרשאות.

בנוסף השיטה לא מאפשרת בצורה פשוטה לתאר אילוצים מבוססי סדר פעולות כרונולוגי. הדרך לממש אילוצים כאלה היא בניית רשימות של הבקשות שמגדירות סדר פעולות אסורה וכאשר בקשה ראשונה מאושרת יש להכניס את שאר הבקשות ברשימה לרשימה שחורה.

2.3.3 אכיפה של אילוצים במודל $RBAC_2$ [2]

קונפיגורציה של המערכת מוגדרת על ידי רביעייה - (UA, PA, RH, C), כאשר C זה קבוצת אילוצים במערכת.

המצב של המערכת מוגדר ע״י - $(UA, PA, RH, C, UA_I, P_I)$ כאשר $UA_I \in UA$ זה קבוצת התחברויות ותפקידים שמשתמשים מפעילים במערכת בזמן נתון ו- $P_I \subseteq U \times P$ זאת קבוצת הרשאות שמופעלות ע״י משתמשים דרך אותם התחברויות. לכל זוג $(u, p) \in P_I$ קיימות זוגות מתאימות $(u, r) \in UA_I, (p, r) \in PA$

בקשה (request) - אינטראקציה עם המערכת, כאשר המטרה היא לשנות את הקונפיגורציה או המצב הנוכחי של המערכת, כגון התחברות לתפקיד חדש, הפעלת הרשאה או שינויי היררכיית תפקידים במערכת. בזמן בקשה האילוץ נאכף אם עבור כל $Q \in Q_c$ לא ניתן לספק כל ה-$q \in Q$. הנחה היא שקיים מנגנון חיצוני למערכת, מבוסס על reference monitor, מעין constraint monitor. המימושים האפשריים מתוארים בהמשך.

כפי שתואר לעיל, אילוצים יכולים להיות מסוגים שונים. נבחן לדוגמה את האילוץ $(U, \{r_1, r_2\}, x)$, זה אילוץ טיפוסי אשר מונע ממשתמשים להיות משויכים גם לתפקיד r_1 וגם לתפקיד r_2

אילוץ סטטי – חל על היחס UA, ז״א $\{(u, r_1), (u, r_2)\} \nsubseteq UA$, אילוץ דינאמי – חל על היחס UA_I, ז״א $\{(u, r_1), (u, r_2)\} \nsubseteq UA_I$

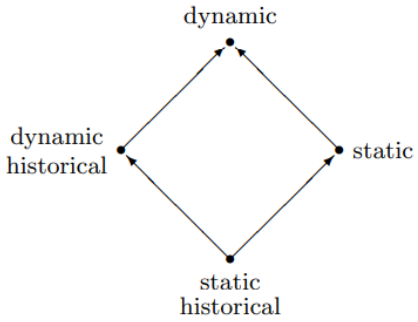
קשה יותר להגדיר אילוץ היסטורי $(U, \{r_1, r_2\}, h)$. האם מדובר באילוץ היסטורי שמוגדר על UA או UA_I.

במילים אחרות, האם הכוונה היא לכך שאם משתמש היה פעם משויך לתפקיד r_1 , אזי הוא לעולם לא יהיה משויך ל-r_2, או לחלופין מדובר בהתחברות בודדת. בעבודה הזאת מוצע לחלק את האילוצים היסטוריים לאילוצים *היסטוריים סטטיים* ואילוצים *היסטוריים דינמיים*.

נקודה נוספת היא יחס בין סוגי האילוצים. אם קיים אילוץ סטטי, אז ברור שמתקיים אילוץ דינמי. לעומת זאת היחס בין אילוץ סטטי והיסטורי הוא פחות חד משמעי. אם לדוגמה רצף האירועים במערכת היה כדלקמן: שיוך המשתמש אל תפקיד r_1 $->$ הסרת שיוך $->$ שיוך המשתמש אל תפקיד r_2.

במקרה הזה אילוץ סטטי אינו מופר, אך מופר אילוץ היסטורי סטטי, ואפילו יכול להיות מופר אילוץ היסטורי דינמי, אם מדובר בפעולות שנעשו באותה התחברות. באופן דומה לא ניתן לקבוע שאילוץ היסטורי דינמי

משמר את האילוץ היסטורי סטטי, אבל אילוץ היסטורי סטטי משמר את האילוץ הסטטי הרגיל. איור 5 מסכם את היחסים בין האילוצים מבחינת אי הפרה הדדית, כאשר חץ מגדיר שאילוץ מסוג מסוים משמר את האילוץ אליו הוא מצביע



איור 5. יחסים שימור בין סוגים שונים של אילוצים

מימוש מנגנון אכיפת אילוצים ע"י רשימה שחורה - Blacklist

על מנת לבצע אכיפה של האילוצים, ניתן לבצע בדיקה עבור כל בקשה ולקבוע האם היא מאושרת. לדוגמה עבור אילוץ $(U, \{p_1, p_2\}, h)$ בקשה ראשונה (u, p_1) תאושר, אבל בקשה נוספת (u, p_2) תדחה.

המימוש האפשרי עבור מנגנון האכיפה הוא רשימה שחורה. למשל, בדוגמה הקודמת, כל בקשה נבדקת מול הרשימה השחורה. עבור בקשה ראשונה הרשימה היא ריקה ולכן הבקשה (u, p_1) תאושר, אך הזוג (u, p_2) יכנס לרשימה, ובמידה תעלה כזאת בקשה בעתיד היא תדחה. על פי סוג האילוץ, בקשות אסורות יוסרו מהרשימה או בסוף התחברות עבור אילוצים דינמיים, או בעת בקשות הסרת השמה של תפקידים\הרשאות עבור אילוצים סטטיים, או שהם תמיד יישארו ברשימה במידה ומדובר באילוץ היסטורי סטטי.

אילוצי משתמשים מבוססי הרשאות

נחזור לדוגמה $(U, \{p_1, p_2\}, h)$ והבקשות (u, p_1) ו- (u, p_2) : אפשרות נוספת למימוש רשימה שחורה היא ליצור רשימה של הרשאות אסורות לכל משתמש. בדוגמה הזאת, כאשר בקשה (u, p_1) תאושר, תיווצר רשימה של הרשאות אסורות עבור u והרשאה p_2 תצורף לרשימה האסורה.

הרשימה הזאת היא מעין "אנטי תפקיד", כך שלכל משתמש יש תפקיד ρ_u אשר מכיל הרשאות שאסורות כרגע על אותו משתמש. ניתן להגדיר תהליך שמסיר את ההרשאות שמשתמש מפסיק להשתמש בהם מהתפקידים שלו ובאותה עת הרשאות סותרות נמחקות מה-"אנטי-תפקיד" שלו. אומנם התהליך לא הכרחי, אך רשימות הרשאות קצרות יקלו על התהליך בדיקת הרשאות אסורות ובכך טיפול בבקשות הרשאות בכללי.

אילוצי תפקידים מבוססי הרשאות

בספרות קיימות דוגמאות לשימוש באילוצים מסוג - $(R, \{p_1, p_2\}, h)$. המשמעות של האילוץ היא שאם משתמש כלשהו קיבל הרשאה p_1 דרך תפקיד r, אזי אף משתמש לא יכול יותר לקבל הרשאה p_2 דרך r. המימוש האפשרי של האילוץ הוא השמה של ההרשאה ההופכיות - $\neg p_2$ אל התפקיד r, כך שכל פעם שתגיע בקשה מסוג (p_2, r) היא תדחה מכיוון שכבר קיים שיוך $(\neg p_2, r)$ והעדיפות של ההרשאות השליליות הן גבוהות יותר מההרשאות הרגילות.

הפירוש הסטנדרטי של הזוג $(\neg p, r)$ במודל RBAC הוא שההרשאה משויכת לתפקיד r וכל התפקידים היורשים ממנו, בצורה הזאת ניתן להדגיר $\neg r$ כך ש-r יורש ממנו וכל ההרשאות השליליות לשייך אליו במקום אל ה-r.

הגבלות מימוש

בנוסף להגבלות תיאור קיימות הגבלות מימוש. מימוש מבוסס רשימה שחורה (יוצג בהמשך) מאפשר אכיפה יעילה עבור אילוצים עם זוג בקובצת האילוץ. לדוגמה נבחן אילוץ -$(U, \{p_1, p_2, p_3\}, h)$, נניח קודם כל מגיע בקשה (u, p_1). בשלב הזה עדיין לא ברור אם להכניס את ההרשאות p_2, p_3 לרשימה שחורה עבור המשתמש, אם להכניס את שתיהן אזי האילוץ שנקבל הוא הדוק מדי ולא מאפשר לאכוף רק מקרה בו מתקיימות שלושת השיוכיים. על מנת לממש אילוצים מסוג הזה נדרש להוסיף היסטוריית בקשות על מנת ליצור רשומה ברשימה שחורה רק אחרי שתי הבקשות מהסוג הזה. או לחלופין בעת קבלה של הבקשה ראשונה ליצור אילוץ נוסף במערכת $(U, \{p_2, p_3\}, h)$ ועל ידו לוודא שאילוץ מקורי לא מופר.

2.4 מודל $RBAC_3$

המודל $RBAC_3$ הוא בעצם שילוב של ה-$RBAC_1$ וה-$RBAC_2$, כך שהוא מכיל גם הגדרת היררכיה של התפקידים וגם אילוצים על היחסי UA, PA והתחברויות. השילוב של המודלים מעלה מספר נקודות נוספות.

אילוצים על היררכיית תפקידים

במודל $RBAC_3$ ניתן להגדיר אילוצים לא רק על היחס UA אלא גם על ההיררכיה עצמה. לדוגמה, ניתן להגביל את מספר התפקידים שיכולים לרשת מתפקיד מסוים או את מספר התפקידים שתפקיד יכול לרשת מהם (cardinality). בנוסף, ניתן להגדיר אילוץ אשר אוסר על שני תפקידים להיות בעל אב/בן משותף (mutual exclusion). אילוצים כאלה יכולים לעזור כאשר היררכית תפקידים ניתנת לשינוי בצורה מבוזרת, אך מנהל המערכת רוצה להגביל את האפשרויות לשינוי שלה ולמנוע פגיעה במדיניות הארגון.

התנגשות בין המודלים

לפעמים שילוב בין המודלים יכול ליצור התנגשות. נניח למשל שקיימים שני תפקידים p, q . התפקידים האלה יכולים להיות מוגדרים כתפקידים סותרים מבחינת אילוצים דינאמיים, ולכן משתמש לא יכול להפעיל אותם בו-זמנית. עם זאת נניח קיים תפקיד r אשר יורש משניהם וכך מקבל את כל ההרשאות שלהם. תפקיד r בעצם מאפשר להתגבר על האילוץ mutual exclusion. ניתן לאפשר או לא לאפשר מצב כזה, אך זה אמור להיות מוגדר באופן מפורש ביישום של המודל RBAC_3.

באופן דומה, גם אילוץ ה- cardinality יכול להיות מופר - בקבוצה שמוגבלת למספר מסוים של תפקידים מופיע תפקיד אשר יורש ממספר יותר גדול של תפקידים. האם להגביל את התפקיד לפי מספר תפקידים מהם הוא יורש או לא? גם זה נתון להגדרה של יישום מסוים של המודל.

3. כרית תפקידים - בעיית RMP

כפי שראינו, מודל RBAC הפך למודל סטנדרטי לבקרת גישה ברוב הארגונים. אחד האתגרים הגדולים במעבר ל-RBAC בארגון הוא בניית מבנה תפקידים (role engineering) מתאים למבנה הארגוני ולצרכי האבטחה באופן מרבי. קיימות שתי גישות עיקריות לבניית מבנה התפקידים – "מלמעלה למטה" (top – down) ו-"מלמטה למעלה" (bottom – up).

בגישה top – down התפקידים מוגדרים על ידי ניתוח מבנה של הארגון ותהליכים ארגוניים, אחרי זה מתבצעת חלוקה של המבנה ליחידות פונקציונליות. כאשר כל יחידה מקושרת לקבוצת הרשאות במערכות המידע של הארגון לפי צרכיה. הגישה הזו מתאימה לארגונים קטנים, אך במצבים שבהם מדובר בעשרות תהליכים עסקיים, מאות משתמשים ומיליוני התחברויות, הגישה הזאת הופכת להיות פחות רלוונטית ובעצם מהווה מחסום בפני מעבר למודל RBAC .

מעבר לכך, ברב המקרים, קבוצות של משתמשים והרשאות הן נתון שלא ניתן לשנות. קבוצת הרשאות היא חלק מובנה של המערכת וקבוצת המשתמשים היא נתון של הארגון. לעומת זאת, קבוצת התפקידים היא קבוצה חדשה שנדרש להגדיר לטובת מודל RBAC. לכן ניתן לבנות קבוצת התפקידים המתאימה בגישה bottom – up. בגישה הזאת מתבצע זיהוי וניתוח של הקשרים הקיימים בארגון בין משתמשים והרשאות. ואז על סמך הקשרים האלה ניתן לבנות קבוצת התפקידים המתאימה.

לדוגמה, במידה וקיימת היסטוריית גישות של המשתמשים למידע ו/או ביצוע פעולות, אזי ניתן לייצג את הקשרים הקיימים בארגון בין המשתמשים להרשאותיהם כמטריצה בינארית. כל עמודה במטריצה מייצגת הרשאה, כל שורה מייצגת משתמש וכניסה במטריצה תהיה 1 אם משתמש מקבל את ההרשאה או 0 אחרת. על בסיס המטריצה הזאת היינו רוצים לקבל קבוצת תפקידים אשר ישקפו את התוכן של המטריצה. בניית קבוצת תפקידים מתוך המטריצה כזאת נקראת Basic Role Mining Problem (Basic RMP) (הבעיה הבסיסית של כריית תפקידים).

בספרות מתוארים מספר סוגים של בעיית RMP ואלגוריתמים מתאימים לפתרון עבורם. מבנה התפקידים שמתקבל כפלט של האלגוריתמים האלה לא תמיד יהיה מושלם ותואם באופן מוחלט את המבנה הארגוני. אף על פי כן, מכיוון שהאלגוריתמים הקיימים מאפשרים לקבל פתרון לבעיות RMP בצורה ממוחשבת, הגישה bottom – up מאפשרת לגלות תפקידים פוטנציאלים באופן מהיר ויעיל. התפקידים הפוטנציאליים שיתקבלו לאחר הרצת אלגוריתם יכולים להיות סוג של "טיוטה", כאשר בשילוב עם גישת top – down ניתן לבנות מודל תפקידים סופית. אם אין ברשות הארגון היסטוריית גישות, אפשר להפעיל מנגנון ניטור אשר יפיק את לוג הגישות בארגון, אך חשוב שהניטור יפעל זמן מספיק כך שהלוג ידמה את אופי הפעולות של המשתמשים באופן כללי ולא בחלון זמן קצר שעלול להיות לא מייצג.

הפרק הזה של העבודה יעסוק בבעיות RMP ואלגוריתמים לפתרון הבעיות האלה. קודם כל נציג שתי בעיות ממשפחת RMP : BasicRMP ו-$\delta -$ Approx RMP. אחרי זה נציג אלגוריתם לכריית תפקידים פוטנציאליים בעזרת תעדוף תת-קבוצות בשם CompleteMiner ו-FastMiner. פלט האלגוריתמים לא מספק רשימה סופית של התפקידים הנדרשים לפתרון של הבעיות BasicRMP ו-$\delta -$ Approx RMP. הם מספקים רק "מועמדים לתפקידים". לאחר קבלת התפקידים הפוטנציאליים אלו, נדרשת עבודה נוספת כדי לקבל את הרשימה הסופית. בסוף הפרק נציג פתרון לבעיה $\delta -$ Approx RMP אשר מבוסס על ה- FastMiner .

עבור כל אחד מהאלגוריתמים שמתוארים בעבודה יוצגו ניתוח סיבוכיות הזמן שלהם ותוצאות אמפיריות המדגימות את ההשפעה של הפרמטרים השונים על הדיוק והמהירות של האלגוריתמים.

3.1 Basic RMP ו-$\delta -$ Approx RMP [5]

תהיינה מטריצה בינארית A בגודל $m \times n$ שמייצגת קשרים בין משתמשים להרשאות, כך ש- m מספר המשתמשים בארגון ו-n מספר ההרשאות. נדרש לפרק את A לשתי מטריצות B, C כך ש :

- המטריצה B בגודל $m \times k$ מייצגת קשרים בין משתמשים לתפקידים,
- המטריצה C בגודל $k \times n$ מייצגת את הקשרים בין תפקידים להרשאות,
- $A = B \otimes C$, עבור כפל מטריצות בוליאניות כפי שנגדיר בהמשך, ו-
- k הוא מספר תפקידים שיידרש על מנת לספק את הקשרים האלה.

הבעיה הזאת נקראת - BRMP. חשוב לציין שיש מספר פתרונות טריוויאליים של הבעיה. לדוגמה, כאשר כל משתמש מיוצג ע"י תפקיד משלו אזי מתקבל $k = m, B = I, C = A$. לחלופין, אם כל הרשאה היא תפקיד בפני עצמו אזי $k = n, B = A, C = I$. אך הפירוקים הטריוויאליים האלה אינם מועילים לבניית מבנה תפקידים של הארגון. לכן BRMP דורש פתרון כך ש-k יהיה קטן ככל האפשר. ללא ידע נוסף על המבנה הארגוני, פתרון עם מספר מינימאלי של התפקידים יכול לתת קירוב הכי טוב למבנה התפקידים הסופי של הארגון.

יש מספר רב של וריאציות של BRMP שמתמקדות בהיבטים שונים של הבעיה המקורית כגון : דיוק של התוצאות, עמידה של הפתרון במדדים נוספים. ווריאציות של הקלט והפלט של הבעיה. אחת מהווריאציות היא $\delta -$ Approx RMP. בזמן ש-BRMP דורשת פתרון מדויק לבעיה, כך שלאחר פירוק A ל- B ו- C יתקבל $A = B \otimes C$, הבעיה המקורבת $\delta -$ Approx RMP מאפשרת לקבל תוצאות לא מדויקות, אך קרובות מספיק. ההקלה הזאת על דיוק התוצאות מאפשרת לצמצם משמעותית את מספר התפקידים. במובן הזה בעיית $\delta -$ Approx RMP מייצגת פשרה בין דיוק של התוצאות ליעילות של הפתרון. פקטור δ מציין את רמת השגיאה המותרת, ולכן הפתרון לבעיה המקורבת יכול לשמש גם כפתרון בעיית BRMP כאשר $\delta = 0$.

על מנת לתת הגדרה פורמאלית לבעיות BRMP ו-$\delta - \text{Approx RMP}$ נדרש להגדיר מהו כפל מטריצות בוליאניות, נורמה L_1 ופקטור δ.

כפל מטריצות בוליאניות:

בהינתן $C \in \{0,1\}^{k\times n}$ ו-$B \in \{0,1\}^{m\times k}$ תהיינה מטריצה A מכפלה שלהם $A = B \otimes C$, כאשר $A \in \{0,1\}^{m\times n}$ ו- $a_{ij} = \bigvee_{l=1}^{k}(b_{il} \wedge c_{lj})$

נורמה L_1 :

נורמה L_1 של וקטור $v \in X^d$ מעל שדה X מוגדרת כך:

$$\|v\|_1 = \sum_{i=1}^{d} |v_i|$$

הנורמה L_1 מאפשרת להגדיר מרחק בין שני הווקטורים:

$$\|v - w\|_1 = \sum_{i=1}^{d} |v_i - w_i|$$

באופן דומה ניתן להגדיר גם מרחק בין שתי מטריצות במרחב $X^{m\times n}$, כאשר המרחק בין המטריצות הוא בעצם סכום המרחקים בין השורות:

$$\|A - B\|_1 = \sum_{i=1}^{m} \|a_i - b_i\|_1 = \sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij} - b_{ij}|$$

בצורה הזאת מאפשרת הנורמה L_1 לחשב מרחק בין שתי מטריצות. עבור שתי מטריצות זהות המרחק הוא 0.
על סמך הגדרת המרחק ניתן להגדיר את סביבת δ של מטריצה A כאוסף כל המטריצות B עבורן מתקיים:

$$\|A - B\|_1 \leq \delta$$

בהמשך נוכל להשתמש בפקטור δ כפרמטר לקביעת קבוצה של המטריצות שהן קרובות מספיק למטריצת הייחוס A. הקבוצה הזאת תשמש כקבוצת פתרונות אפשריים עבור בעיית $\delta - \text{Approx RMP}$

בהינתן קבוצת משתמשים U, קבוצת הרשאות P והיסטוריית פעולות שנעשו ע"י משתמשים ניתן להגדיר מיפוי UPA (User Permission Assignment) . מיפוי UPA הוא מטריצה בוליאנית שבנויה באופן הבא: שורות המטריצה הן משתמשים מקבוצת U , עמודות המטריצה הן הרשאות מקבוצה P, וכניסות המטריצה

מציינות את הקשר בין משתמשים להרשאות. אם למשתמש יש הרשאה מסוימת, אזי הערך של הכניסה מתאימה הוא 1, אחרת 0.

דוגמה:

$$U = \{u_1, u_2, u_3, u_4, u_5\}$$

$$P = \{p_1, p_2, p_3, p_4, p_5, p_6\}$$

$$UPA = \begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{array} \begin{array}{cccccc} p_1 & p_2 & p_3 & p_4 & p_5 & p_6 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array}$$

אם קיימת קבוצה של תפקידים R, אזי ניתן להגדיר מטריצות UA (User Assignment) ו-PA (Permission Assignment). UA היא מטריצה בוליאנית שמסמנת קשר בין המשתמשים לתפקידים, כאשר השורות הן משתמשים, העמודות הן תפקידים וכניסות מסמנות האם תפקיד שייך למשתמש. PA היא מטריצה בוליאנית שמסמנת קשר בין התפקידים להרשאות, כאשר השורות הן תפקידים, העמודות הן הרשאות וכניסות המטריצה מסמנות האם הרשאה שייכת לתפקיד. עבור הדוגמה לעיל של מטריצת UPA, מטריצות UA ו- PA אפשריות הן:

$$UA = \begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{array} \begin{array}{cccc} r_1 & r_2 & r_3 & r_4 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{array} \qquad PA = \begin{array}{c} \\ r_1 \\ r_2 \\ r_3 \\ r_4 \end{array} \begin{array}{cccccc} p_1 & p_2 & p_3 & p_4 & p_5 & p_6 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array}$$

הגדרה פורמאלית של Basic RMP:

בהינתן קבוצת משתמשים U, קבוצת הרשאות P ומטריצה UPA יש למצוא קבוצת תפקידים R, ומטריצות UA ו-PA, כך שיתקיים $\|UPA - UA \otimes PA\|_1 = 0$ ו-$k = |R|$ הוא מינימאלי.

הגדרה פורמאלית של δ − Approx RMP:

בהינתן קבוצת משתמשים U, קבוצת הרשאות P, מטריצת UPA ופקטור δ יש למצוא קבוצת תפקידים R כך שמספר התפקידים $|R| = k$ הוא מינימאלי, ומטריצות UA, PA עבורן מתקיים:

$$\|UA \otimes PA - UPA\|_1 \leq \delta$$

קל לראות שבעיית Basic RMP היא בעיית δ − Approx RMP כאשר $\delta = 0$.

יש מספר סיבות להגדיר בעיה שדורשת הקלה ברמת הדיוק. למרות שהפתרון המדויק הוא תמיד טוב, לפעמים על ידי הקלה קטנה ברמת הדיוק ניתן להשיג הפחתה משמעותית במספר התפקידים שיענו על דרישות הבעיה.

במקרים בהם ישנם ארגונים גדולים עם מספר רב של משתמשים, רובם מחזיקים בהרשאות מועטות. כתוצאה מכך, המטריצה UPA היא די דלילה, כלומר, רוב הערכים בה הם 0. לדוגמה אם $|U| = 1000$ ו- $|P| = 100$, אזי במטריצת UPA המתאימה תהינה 100,000 כניסות, אבל יכול להיות שרק 5000 מתוכם יהיו עם הערך 1. נניח שעל מנת לקבל פירוק מדויק של ה-UPA נדרשת קבוצת תפקידים R, כך ש- $|R| = 200$. אם נקל קצת בדרישות ונקבל פירוק שמקיים

$$\|UA' \otimes PA' - UPA\|_1 \leq 0.01|UPA|$$

כלומר, אם נצליח למצוא קבוצת תפקידים R' כזאת שהפירוק המתאים של $UA' \otimes PA'$ יכסה 99% מהמטריצה UPA המקורית, יכול להיות שנקבל $|R'| = 120$, שזה מוריד ב-40% את מספר התפקידים בפתרון המדויק. במידה ולא נוסיף הרשאות נוספות בפתרון המקורב (לא יהיו כניסות 1 חדשות במטריצה), אז הפתרון השני הוא הטוב יותר, מאחר ומספר תפקידים קטן יותר יקל על ניהול כל המערכת של RBAC, וישאיר מספר קטן של מקרים ללא כיסוי שניתן לטפל בהם באופן ידני. בהמשך העבודה יתואר אלגוריתם שמספק פתרון מקורב ללא הוספת הרשאות.

יתרון נוסף של הפתרון המקורב הוא שהוא יכול לתת מענה יותר טוב לאורך זמן. מטריצות UPA מייצגות, בדרך כלל, רק תמונת מצב זמנית של הקשר בין המשתמשים להרשאותיהם, אבל האופי של מודל התפקידים ב-RBAC הוא דינאמי, מכיוון שבארגון כל הזמן יש שינוים בקבוצות משתמשים והרשאות. לדוגמה בבניית מודל RBAC המטריצה המתאימה היא UPA_1, ולאחר פרק זמן מסוים, בעקבות שינוי בהרשאות של המשתמשים הבודדים מטריצה מתאימה היא UPA_2. אבל יכול להיות שעבור פתרון מקורב מתקיים:

$$\|UA' \otimes PA' - UPA_1\|_1 \leq \delta \;\&\; \|UA' \otimes PA' - UPA_2\|_1 \leq \delta$$

במקרה כזה קבוצת תפקידים מקורבת נשארת תקפה לאורך הזמן, ולא מושפעת משינוים נקודתיים ב-UPA. פקטור δ הוא מדד חשוב אשר מאפשר לתחום את המרחק בין הפתרון המקורב לפתרון המדויק.

3.2 אלגוריתם לכריית התפקידים בעזרת תעדוף תת-קבוצות [4]

3.2.1 עקרונות ייסוד עבור אלגוריתמים לכריית תפקידים

בחלק הזה של העבודה נציג שני אלגוריתמים שבהינתן מטריצת UPA מחזירים רשימה של התפקידים הפוטנציאלים ממוינים לפי התעדוף. האלגוריתם הראשון נקרא CompleteMiner והאלגוריתם השני נקרא FastMiner . CompleteMiner מחזיר רשימה מלאה של המועמדים לתפקידים פוטנציאליים, אך הוא איטי. האלגוריתם FastMiner הוא יעיל יותר מבחינת זמן ביצוע, אך מחזיר רשימה חלקית של המועמדים לתפקידים. האלגוריתמים המוצעים מתייחסים למודל RBAC_0 בלבד, ללא התחשבות באילוצים והיררכיה בין התפקידים. בהמשך העבודה נראה גם התייחסות לכריית התפקידים למודל RBAC_2.

ההגדרה של התפקיד בהקשר של כריית התפקידים היא "קבוצת הרשאות". כמו כן נעשות הנחות הבאות על המונח "תפקיד":

- תפקידים שונים יכולים להחזיק באותה הרשאה, ז"א הרשאה לא ייחודית לתפקיד.
- משתמש יכול לקבל מספר תפקידים ולהחזיק באותה הרשאה דרך מספר התפקידים שמוקצים לו.

התעלמות מההנחות האלה יכולה להפוך את הבעית כריית התפקידים לפשוטה יותר, אך הפלט שלה יהיה פחות מדויק. זה בניגוד לבעיה הקלאסית של כריית הנתונים שדורשת אשכולות שאינם חופפים. הדבר נובע מכך שמודל RBAC מניח שמשתמש יכול לקבל תפקידים שונים ואותה הרשאה יכולה להשתייך לתפקידים שונים. הגמישות הזאת מאפשרת למודל RBAC להתמודד עם אתגרים רבים בתחום בקרת הגישה. בהתאם לכך, שימוש באלגוריתם פשוט לבניית אשכולות שאינם חופפים לא יעניק את התוצאות הרצויות וזה הופך את כריית התפקידים לבעיה לא טריוויאלית.

מכיוון שמבחינת המודל "תפקיד" זה פשוט תת-קבוצה של קבוצת ההרשאות, אז בהינתן k הרשאות ניתן למנות 2^k תפקידים, כאשר רק חלק מתוכם זה תפקידים שהם בעלי משמעות מבחינת הארגון. ברור שלא ניתן לבחון את כל ה-2^k תפקידים, לכן נדרשת טכניקה יותר יעילה לגילוי התפקידים. האלגוריתם CompleteMiner משתמש בטכניקה שמתבססת על שתי עקרונות.

עקרון 1:

הגדרת התפקידים מובנית בתוך ה-UPA. אם יש קבוצת הרשאות $\{p_a, p_b, \dots, p_n\}$ שאינה משויכת לאף משתמש, אזי כנראה התפקיד הזה לא קיים בארגון. יכול להיות שמבחינה ארגונית יש תפקיד שמקבל קבוצת הרשאות $\{p_a, p_b, \dots, p_n\}$ ועדיין אף משתמש לא קיבל את התפקיד הזה. אבל תהליך אוטומטי אינו יכול להבדיל בין תפקיד ללא משתמשים לקבוצת הרשאות שלא מגדירות תפקיד על סמך UPA בלבד וללא מידע נוסף. לכן האלגוריתם שמוצע בעבודה הזו מתייחס רק אל התפקידים שבפועל משויכים לפחות למשתמש אחד.

עקרון 2:
יש לזהות את התפקידים על פי המספר המרבי של ההרשאות המשותפות בין המשתמשים. ז״א אם קיימים שני תפקידים שמשתמשים מקבלים אותם תמיד ביחד, יש לזהות אותם כתפקיד אחד עם כל ההרשאות שלהם. לדוגמה, אם קיים זוג הרשאות $\{p_a, p_b\}$ שעבור כל המשתמשים מתקיים שמשתמש מקבל או את שתי ההרשאות ביחד או אף הרשאה מהזוג, אזי ניתן להסיק ש-$\{p_a, p_b\}$ מגדירים תפקיד. גם אם קיים תפקיד רק עם הרשאה p_a, לפי עקרון 1, לא ניתן להבדיל בין התפקיד ללא משתמש וקבוצת הרשאות שלא מגדירה אף תפקיד.

העקרון הזה דורש גם להתייחס לקבוצת הרשאות המשותפת המקסימאלית בין המשתמשים כתפקיד בפני עצמו. לדוגמה, אם סט הרשאות של משתמש אחד - $\{p_a, p_b, p_c, p_d\}$ ועבור משתמש אחר סט ההרשאות הוא $\{p_a, p_b, p_e\}$, אזי קבוצת הרשאות $\{p_a, p_b\}$ צריכה להיספר כתפקיד. לספור תפקיד רק עבור הרשאה p_a או עבור הרשאה p_b חסר משמעות במקרה הזה וללא מידע נוסף. אבל, אם לדוגמה ימצא משתמש עם סט הרשאות $\{p_b, p_f\}$, אז קבוצה $\{p_b\}$ צריכה להיכנס לרשימה של התפקידים הפוטנציאלים גם כן.

3.2.2 אלגוריתם CompliteMiner

באופן אידיאלי, תהליך כריית תפקידים צריך גם לזהות תפקידים פוטנציאליים וגם לבסס את הבחירה כך שרשימת התפקידים הסופית תהיה אמינה. לכן האלגוריתם CompleteMiner מורכב משני מרכיבים: זיהוי של כל התפקידים הפוטנציאלים ובחירה של הרשימה הסופית (תעדוף של התפקידים) לאחר מכן.

הטכניקה שמוצעת על ידי Vaidya et al היא זיהוי של התפקידים ע״י חישוב חיתוכים בין קבוצות של ההרשאות. הליבה של האלגוריתם היא חיפוש קבוצות הרשאות שמכסות את מטריצת הקלט (UPA). אלגוריתם מגלה את הקבוצות החדשות של התפקידים מתוך כל החיתוכים האפשריים בין קבוצות שכבר נמצאו בצורה רקורסיבית.

אלגוריתם CompleteMiner מורכב משלושה שלבים:

זיהוי ראשוני של קבוצת התפקידים הפוטנציאליים

בשלב הזה יש לאחד את כל המשתמשים עם הרשאות זהות. ניתן לעשות זאת ע״י מעבר אחד על הקלט. השלב הזה יכול להקטין משמעותית את הקלט מכיוון שכל המשתמשים עם אותם תפקידים יהיו בעלי אותו סט של הרשאות. בשלב הזה נשאר עם קבוצת משתמשים עם הרשאות ייחודיות. כפי ציינו קודם, תפקיד במובן כריית התפקידים זה קבוצת הרשאות. לכן לכל משתמש שנשאר בקלט נצמיד תפקיד פוטנציאלי שמורכב מהרשאות שלו וכך נקבל את קבוצת התפקידים הפוטנציאליים הראשונית, נקרא לה - InitRoles.

מספור תתי-קבוצות

בשלב הזה ניתן לזהות את כל התפקידים בעלי משמעות שמתקבלים מתוך החישוב של קבוצות החיתוך של התפקידים שהתקבלו בשלב המקדים. נקרא לקבוצה הזאת GenRoles. כל חיתוך ייחודי יש להוסיף אל קבוצת GenRoles. למרות שבאופן תיאורטי גודל של GenRoles יכול להגיע ל-$2^{InitRoles}$, בפועל הוא יהיה הרבה יותר מצומצם, מכיוון שחלק ניכר מהחיתוכים האפשריים הן קבוצות ריקות או חופפות עם חיתוכים אחרים. התפקידים שמתקבלים מחישוב החיתוכים הם קבוצה מקסימאלית של התפקידים בעלי משמעות. באופן כללי, כל קבוצה ייחודית של ההרשאות מזוהה כתפקיד, אין משמעות לחלוקת יתר של ההרשאות. לדוגמה, אם מספר משתמשים מקבלים קבוצת הרשאות $\{p_a, p_b, p_c\}$, אבל אין אף משתמש שיש לו רק תת קבוצה של ההרשאות האלה, אזי רק הקבוצה הזאת מזוהה כתפקיד.

ספירת משתמשים

בשלב הזה עבור כל תפקיד שזוהה בשלב השני יש לספור מספר המשתמשים שמזוהים עם התפקיד הזה. בפועל מנוהלות שתי ספירות:

orig_count(i) – מספר המשתמשים שתפקיד i מתאים במדויק לקבוצת ההרשאות שלהם (ניתן לספור בשלב 1)

final_count(i) – מספר משתמשים שתפקיד i הוא תת-קבוצה של קבוצת הרשאות שלהם

קל לראות שעבור התפקידים שזוהו בשלב השני ע"י חיתוכים orig_count הוא 0. כמו כן, עבור כל תפקיד שהוא תת קבוצה של הרשאות של התפקיד אחר, ה-final_count יהיה יותר גדול מ-orig_count.

Require: Dataset $D \equiv (U, P)$;
$P(u)$ gives the set of permissions assigned to user u;
$R(x)$ represents a role consisting of the set of permissions x;
$Count(i)$ gives the count of users associated with role i;
{Cluster users into initials roles based on exact match of the set of permissions}
$InitRoles \leftarrow \{\emptyset\}$
$GenRoles \leftarrow \{\emptyset\}$
for each user $u \in U$ **do**
    if $R\big(P(u)\big) \notin InitRoles$ **then**
        Set $orig\_count$ of $R\big(P(u)\big)$ to 1
        $InitRoles \leftarrow InitRoles \cup R\big(P(u)\big)$
    else
        Increment $orig\_count$ of $R\big(P(u)\big)$
    end if
end for
{Enumerate all possible interesting roles}
for each Role $i \in InitRoles$ **do**
    $InitRoles \leftarrow InitRoles - i$
    for each Role $j \in InitRoles$ **do**
        $GenRoles \leftarrow GenRoles \cup (i \cap j)$
    end for
    for each Role $j \in GenRoles$ **do**
        $GenRoles \leftarrow GenRoles \cup (i \cap j)$
    end for
end for
{Count number of users belonging to each candidate role}
for each Role $i \in GenRoles$ **do**
    for each Role $j \in InitRoles$ **do**
        if $i \subset j$ **then**
            $Count(i) \leftarrow Count(i) + orig\_count(j)$
        end if
    end for
end for

איור 6. פסאודוקוד עבור $CompleteMiner$

באיור 6 מובא פסאודו-קוד של אלגוריתם CompleteMiner. בחלק הראשון (שורות 2-11) של האלגוריתם נוצרת רשימה של קבוצות הרשאות ללא חזרות InitRoles ורשימת מונים שסופרים מספר משתמשים עבור כל קבוצה (שלב זיהוי ראשוני של קבוצת התפקידים הפוטנציאליים). בחלק השני (שורות 13-21) עבור כל קבוצה מתוך InitRoles מחושבים חיתוכים בין קבוצות שנמצאו עד כה (כולל חיתוכים אחרים) והם נכנסים לרשימה GenRoles (שלב מספור תתי-קבוצות). החלק האחרון (שורות 23-29) אחראי על ספירת משתמשים עבור כל חיתוך שנוצר בחלק השני (שלב ספירת משתמשים)

האלגוריתם המוצע מחשב את כל התפקידים האפשריים שהם בעלי משמעות מבחינת ה-RBAC. לאחר מכן התפקידים מקבלים תעדוף על סמך ספירת משתמשים שמקבלים את התפקיד, בכך מתקבלת רשימה של התפקידים הפוטנציאליים. הגישה הזאת נותנת תוצאות מדויקות מאוד כאשר היחס בין המשתמשים לתפקידים הוא לפחות 10, שלרוב זה הוא היחס הטבעי לארגונים גדולים. עבור יחס גבוה מ-5, התוצאות פחות מדויקות, אבל עדיין טובות.

הגישה הזאת בכריית התפקידים מבוקרת רבות בכך שהיא עלולה לגלות תפקידים (קבוצת הרשאות) שגויים מכיוון שהקלט שלה יכול להכיל נתונים שגויים. עם זאת, מחברי האלגוריתם מאמינים שגילוי תפקידים ע״י בחינת החיתוכים בין קבוצות של הרשאות הוא עמיד לרעש. הרעש במקרה זה הן הרשאות שבקלט נרשמו כשלילת גישה או מתן הרשאת יתר, במילים אחרות כניסה 1 ב-UPA הפכה להיות 0 או להפך. התוצאות האמפיריות ברמות הרעש השונות שמוצגות בהמשך מראות שהאלגוריתם עמיד לרעש שנמוך מ-20%. הסיבה העיקרית לכך היא שהאלגוריתם מזהה, ע״י שימוש בחיתוכים, תבניות דומות בין המשתמשים, ואז אפילו אם חלק מהקלט שגוי, רוב הקלט עדיין אמין ולכן התפקידים שמתקבלים ע״י חיתוכים מכסים את הרוב המוחלט של התפקידים מהקבוצה הסופית הרצויה.

תעדוף של התפקידים הפוטנציאלים

אלגוריתמי זיהוי תפקידים פוטנציאליים, בדרך כלל, מחזירים מספר גדול יותר של תפקידים מאשר מספר התפקידים הסופי שדרוש בפועל. כדי ליעל את תהליך הסינון ומציאת רשימה סופית, יש צורך לבחון את איכות וחשיבות התפקידים הפוטנציאליים שהוצגו.

בבעיות כלליות שקשורות לכריית נתונים ושמשתמשות בשיטות ניתוח אשכולות, ניתן להשתמש במדדי קרבה בתוך אשכול מסוים או בין אשכולות שונים. בבעיית זיהוי התפקידים, לעומת זאת, אין לנו ידע סמנטי נוסף על המבנה של הארגון, ולכן לא ניתן לדבר על ״קרבה״ ללא התייחסות למידע סמנטי נוסף.

במחקר זה, Vaidya et al מציעים שימוש בשיטות תעדוף לזיהוי התפקידים. דרך פשוטה היא למיין את התפקידים לפי מספר המשתמשים שמקבלים את ההרשאות של התפקיד (final_count). אך שימוש פשוט כזה יכול להביא לרשימה ארוכה מאוד של תפקידים שהם בעצם תפקידים ״משניים״ של התפקיד ה-״מקורי״.

לכן הם מציעים מספר הנחות שמשפרות את התעדוף של התפקידים:

- רשימה סופית של התפקידים טמונה בתוך מטריצת UPA מקורית, ולכן צריך לתת עדיפות לתפקידים שמייצגים את קבוצת ההרשאות המלאה של משתמש כלשהו במטריצה המקורית. ניתן להשתמש במדד orig_count ולהכפילו במקדם boost כדי להעניק עדיפות לתפקידים שמייצגים את קבוצת ההרשאות המלאה של משתמש מסוים במטריצת ה-UPA המקורית.
- תפקידים עם מספר גדול של הרשאות עדיפים על תפקידים "מצומצמים" יותר. לפחות בהיבט של ניהול התפקידים. לכן ניתן להוריד את התעדוף של התפקידים ה"מצומצמים". אלגוריתמים לזיהוי תפקידים על ידי חיתוכים באופן טבעי מחזירים המון תפקידים קטנים שהם בעצם קבוצות הרשאות נפוצות שכמה תפקידים מקבלים אותם, זאת סיבה נוספת להוריד תעדוף של התפקידים ה"מצומצמים".
- תפקידים שמשתייכים למשתמש בודד, גם אם הם מאגדים המון הרשאות, צריכים להיות בתעדוף נמוך, אחרת תפקידים כאלה יוצרים רעש ברשימה סופית. מנהלי אבטחת מידע יכירו שיש משתמש עם תפקיד מיוחד גם ללא רשימה של התפקידים הפוטנציאליים.

מדד אפשרי שיכול לשמש על מנת לתעדף תפקידים הוא זה:

$$orig_{count} * boost + final_{count} * discount$$

לדוגמה, אם המקדמים הם $boost = 0$ ו-$discount = 1$, התעדוף פשוט יימין את הרשימה לפי final _count. Vaidya et al. מציעים את הערכים הבאים עבור המקדמים:

$$boost = \begin{cases} 1 & for\ \ 1 \le |r| \le 5 \\ 20 & for\ \ |r| > 5 \end{cases} \qquad discount = \begin{cases} 0.1 & for\ \ 1 \le |r| \le 3 \\ 0.5 & for\ \ 4 \le |r| \le 5 \end{cases}$$

החיסרון של הערכים האלה הוא שהם מקשים על מציאת תפקידים ששייכים למעט משתמשים. אך על ידי discount נמוך עבור תפקידים קטנים, ניתן לעלות את ההסתברות למצוא תפקידים גדולים ולא שכיחים, במיוחד אם יש משתמשים שסט ההרשאות שלהם במטריצה המקורית תואם לתפקיד, כך שמשתמש מקבל רק את התפקיד הזה. לדוגמה, אם לחמישה משתמשים יש קבוצה של שש הרשאות שמזוהה כתפקיד פוטנציאלי, אזי הערך של התעדוף שלו יהיה 100. לעומת זאת תפקיד פוטנציאלי עם הרשאה אחת יקבל תעדוף של 100 רק אם ימצא אצל 1000 משתמשים.

דוגמה של CompleteMiner

נניח שקיים ארגון עם 4 סוגי הרשאות, מכיוון שתפקיד מוגדר כקבוצת הרשאות, אזי במקרה הזה לארגון יש 16 תפקידים פוטנציאליים. נבחר מטריצת UPA לדוגמה עבור 15 משתמשים ו4 סוגי הרשאות (איור 7) ונבצע עליה את האלגוריתם CM.

בשלב ראשון נקבל רשימה InitRoles - $\{\{p_1,p_2,p_4\},\{p_2,p_3,p_4\},\{p_2,p_3\},\{p_4\},\{\emptyset\}\}$. זאת הרשימה של כל קבוצות ההרשאות מתוך UPA ללא חזרות. באיור 8 מופעים מונים עבור כל אחת מהקבוצות האלה (Original Count). בשלב שני אנחנו מחפשים כל החיתוכים הלא ריקים בין הקבוצות שנמצאו בשלב ראשון. נקבל רשימת GenRoles - $\{\{p_2,p_3\},\{p_2,p_4\},\{p_2\},\{p_4\},\{\emptyset\}\}$. כאשר תפקידים פוטנציאליים חדשים שיתווספו הם $\{p_2,p_4\},\{p_4\}$. איטרציות נוספות של גילוי חיתוכים לא יחזירו תוצאות חדשות ולכן שלב שני מסתיים. בשלב שלישי עבור כל קבוצת הרשאות i סופרים מספר משתמשים שעבורם מתקיים $i \subseteq P(u)$. התוצאות מופיעות באיור 8 (Total Count). בנוסף ניתן לחשב תיעדוף עבור כל תפקיד. לפי נוסחה שמופיע בתת-סעיף קודם, מכיוון שכל התפקידים בעלי בין הרשאה לשלוש הרשאות, ניתן לקבוע

$$boost = 1, discount = 0.1$$

התוצאה מופיע באיור 8 בעמודה Priority

User	p_1	p_2	p_3	p_4
u_1	0	0	0	0
u_2	1	1	0	1
u_3	0	1	1	0
u_4	1	1	0	1
u_5	1	1	0	1
u_6	0	1	1	1
u_7	0	1	1	1
u_8	0	1	1	0
u_9	0	1	1	0
u_{10}	0	0	0	1
u_{11}	0	0	0	1
u_{12}	0	0	0	0
u_{13}	1	1	0	1
u_{14}	1	1	0	1
u_{15}	0	1	1	1

איור 7. דוגמה של UPA

Role	Original Count	Total Count	Priority
p_1,p_2,p_4	5	5	5.5
p_2,p_3,p_4	3	3	3.3
p_2,p_3	3	6	3.6
p_2,p_4	0	8	0.8
p_2	0	11	1.1
p_4	2	10	3

איור 8. תוצאה של $CompleteMiner$

סיבוכיות של CompleteMiner

קלט של האלגוריתם: מטריצה ריבועית $n \times m$, כאשר n מספר משתמשים במערכת ו-m מספר הרשאות.

חלק ראשון – מעבר אחד על רשימת המשתמשים, לכן הסיבוכיות שלו $O(n)$

חלק שני – כפי שניתן לראות בפסאודו-קוד, החלק השני בנוי מלולאה חיצונית ושתי לולאות פנימיות. לולאה חיצונית מבצעת איטרציה עבור כל איבר בקבוצה InitRoles (כ-n איטרציות סה"כ). כאשר כל איטרציה כזאת מכילה שתי לולאות. לולאה פנימית ראשונה גם עוברת על האיברים ב-InitRoles ומשנה את הרשימה GenRoles. לולאה פנימית שניה עוברת על כל האיברים ב-GenRoles שנוספו עד כה. קשה להעריך את האורך של הלולאה הזאת, מכיוון שהוא תלוי במספר החיתוכים שימצא איטרציות קודמות.

דרך אפשרית לאמוד את הסיבוכיות של החלק השני של האלגוריתם הוא להעריך כמה חיתוכים אנחנו נדרש לבדוק סה"כ. בהינתן n איברים בקלט, אם נרצה לגלות את כל החיתוכים בין כל זוג של האיברים אנחנו נצטרך לעבור על סדר גודל n^2 חיתוכים אפשריים. במידה ונרצה לבדוק כל החיתוכים בין כל השלישיה האפשרית של n איברים, אזי עבור כל איבר נצטרך לבדוק את חיתוך עם כל חיתוך בין זוגות שנמצא לפני זה, וזה סדר גודל של n^3. וכן הלאה עבור כל רביעייה, חמישייה וכו. מהצד השני, גודל קבוצת החיתוכים של כל ה-n איברים הוא 1, וגודל של קבוצת כל החיתוכים האפשריים של קבוצות $n-1$ איברים הוא n. ניתן להסיק שמספר החיתוכים כפונקציה של גודל קבוצות האיברים ביניהם מחפשים חיתוך היא מעין "פעמון". השטח של הפעמון הזה הוא מספר חיתוכים אפשריים בין כל הקבוצות. את המספר המקסימאלי של החיתוכים ניתן לאמוד ב-2^n, שזה מספר אפשרויות לתפקידים פוטנציאליים וגם חסם עליון על אורך רשימת GenRoles.

מכאן נקבל שסיבוכיות הזמן של החלק השני היא מעריכת - $O(2^n)$

חלק שלישי – עבור כל חיתוך מתוך GenRoles$(\sim 2^n)$ האלגוריתם עובר על התפקידים ב-InitRoles וסופר בכמה תפקידים החיתוך הזה מופיע ($O(n2^n)$) .

בסך הכל קיבלנו שהאלגוריתם CM הוא בעל זמן ריצה מעריכי באורך הקלט שלו.

3.2.3 אלגוריתם FastMiner

החיסרון העיקרי של CompleteMiner הוא האיטיות שלו. סיבוכיות הזמן שלו כפי שראינו מעריכית ביחס למספר המשתמשים במטריצת UPA. האיטיות של האלגוריתם הופכת אותו ללא שימושי, למעט קלטים ממש קטנים. אבל ניתן לשפר את האלגוריתם. האיטיות של CM נובעת מכך שבחלק השני הוא מחפש חיתוכים לא רק בין התפקידים ב-InitRoles אלא גם בין חיתוכים עצמם, בנוסף אין סיבה שהחלק השלישי יתבצע בנפרד, אלא ניתן לבצע אותו תוך כדי החלק השני. השיפורים האפשריים ל-CM הם חיפוש חיתוכים רק בין תפקידים מתוך InitRoles ובמקביל ספירה של המשתמשים שאותו חיתוך מופיע בקבוצת ההרשאות שלהם. בכך ניתן לאחד את החלק השני והשלישי של CM לחלק אחד שסיבוכיות הזמן שלו היא ריבועית ביחס לאורך ה-InitRoles. החלק הראשון נשאר ללא שינוי. הפסאודוקוד של האלגוריתם FastMiner מובא באיור 9.

היתרון של ה-FM שהוא הרבה יותר מהיר ושימושי, אבל זה בא על חשבון התפקידים הפוטנציאליים שהוא מגלה. האלגוריתם יכול לגלות רק תפקידים שהם תפקיד משותף הכי גדול לכל שני משתמשים. אבל אם יש תפקיד שהוא משותף לשלושה משתמשים, אבל לא תפקיד הכי גדול משותף לאף זוג ביניהם, האלגוריתם לא יגלה אותו. על מנת למצוא תפקידים שמשותפים לשלושה משתמשים נדרש לחפש חיתוכים לא בין זוגות של התפקידים ב-InitRoles, אלא בין שלשות.
במקרה הזה תעלה סיבוכיות האלגוריתם מ-$O(n^2)$ ל-$O(n^3)$, כפי שראינו בניתוח סיבוכיות של CompleteMiner. אבל גם במקרה הזה לא יהיה ניתן לגלות תפקידים שהם משותפים לארבעה משתמשים. ניתן להמשיך להרחיב את האלגוריתם למציאת תפקידים שהם משותפים ליותר ויותר משתמשים, אך בסוף האלגוריתם יהפוך ל- CompleteMiner המקורי.

למרות שנראה שככל שעולה סיבוכיות האלגוריתם כך גם הדיוק שלו עולה, וזה נכון לבעיות כריית נתונים רגילות, הדבר לא תמיד נכון לבעיית כריית תפקידים. ככל שמספר התפקידים הפוטנציאלים גבוה יותר, כך יותר קשה לבחור סט תפקידים מינימאלי שיענה על צרכי הארגון. יתר על כן, בהיבט של כריית התפקידים יש קושי להגדיר באופן חד משמעי מהו פתרון מדויק או נכון, מכיוון שלשם כך נדרש לדעת מהו סט תפקידים שיבחר בסוף. אך הסט הסופי הוא לא קבוע ותלוי במה שיבחרו האחראים על אבטחת מידע בארגון. המדד הפורמאלי לבחינת רשימת התפקידים הפוטנציאליים חייב להתייחס לטיב התפקידים או יעילותם לארגון. בהינתן המדד הפורמאלי, ניתן להרחיב פתרון לחיפוש חיתוכים בין שלשות של המשתמשים וכו׳. מחברי האלגוריתם טוענים שחיפוש תפקידים על ידי חיתוכים בין זוגות של המשתמשים עונה על הצרכים המעשיים ומראים זאת בהמשך ע״י תוצאות אמפיריות. עבור דוגמה מופשטת של CompleteMiner, שני האלגוריתמים מביאים אותו פתרון.

שיטת התעדוף המוצעת פשוטה למדי, אך ללא מידע נוסף עבור התפקידים ובהינתן מטריצת UPA בלבד אין דרך לשפר את התעדוף באופן מהותי. עבור בעיות מורכבות יותר, כאשר יהיה יותר מידע אודות קשרים בתוך קבוצות ההרשאות, המשתמשים והקשרים ביניהם ניתן לבנות שיטות תעדוף יעילות יותר.

Require: Dataset $D \equiv (U, P)$; $P(u)$ gives the set of permissions assigned to user u;
$R(x)$ represents a role consisting of the set of permissions x;
$Count(i)$ gives the count of users associated with role i;

{Cluster users into initial roles based on exact match of the set of permissions}
$InitRoles \leftarrow \{\emptyset\}$; $GenRoles \leftarrow \{\emptyset\}$
for each user $u \in U$ **do**
  if $R\big(P(u)\big) \notin InitRoles$ **then**
    Set $orig_{count}$ of $R\big(P(u)\big)$ to 1; $InitRoles \leftarrow InitRoles \cup R\big(P(u)\big)$
  else
    Increment $orig_{count}$ of $R\big(P(u)\big)$
  end if
end for
{Enumerate all intersecting roles between pairs of users}
for each Role $i \in InitRoles$ **do**
  $InitRoles \leftarrow InitRoles - i$
  for each Role $j \in InitRoles$ **do**
    $NewRole \leftarrow i \cap j$
    if $NewRole \notin GenRoles$ **then**
      $Count(NewRole) \leftarrow Count(NewRole) + orig_{count(i)} + orig_{count(j)}$
      Add i, j to the list of contributors for $NewRole$
      $GenRoles \leftarrow GenRoles \cup NewRole$
    else
      if i has not contributed before to $NewRole$ **then**
        $Count(NewRole) \leftarrow Count(NewRole) + orig\_count(i)$
        Add i to the list of contributors for $NewRole$
      end if
      if j has not contributed before to $NewRole$ **then**
        $Count(NewRole) \leftarrow Count(NewRole) + orig\ \_count(j)$
        Add j to the list of contributors for $NewRole$
      end if
    end if
  end for
end for

איור 9. פסאודוקוד עבור $FastMiner$

באיור 9 מופיע פסאודו-קוד של FastMiner. החלק הראשון (שורות 1-9) דומה לחלק הראשון של האלגוריתם CM. במהלך החלק הזה נבנית רשימה InitRoles, רק הפעם היא כוללת עדכון של מונה $orig_{count}$. בחלק השני של האלגוריתם (שורות 11-30), בשונה מ-CM, מחפשים חיתוכים רק בין התפקידים מתוך InitRoles. עבור כל חיתוך בין התפקידים בודקים אם הוא עדיין לא שייך ל-GenRoles אז מוסיפים אותו ומעדכנים את המונים ($final_{count}$ ו-$orig_{count}$) עבור התפקיד החדש (שורות 15-18). לעומת זאת, אם הוא כבר קיים מעדכנים רק את המונים עבור כל אחד מהתפקידים בחיתוך (שורות 19-28).

היתרונות של האלגוריתמים CM ו-FM:

- תאימות למערכות המלצה

השיטה מאפשרת לא רק לגלות את התפקידים הפוטנציאליים, אלא גם לספק המלצות עבור התפקידים, לדוגמה אחוזון מסוים מתוך רשימת העדיפויות הסופית. האחוז יכול להתייחס גם לאחוז המשתמשים הרלוונטיים לתפקיד מכלל המשתמשים או אחוז ההרשאות המכוסות ע"י תפקיד מכלל ההרשאות.

- אי תלות בסדר הקלט

בחלק מהשיטות לגילוי תפקידים תלוי הפלט באופן בו הקלט מסודר. עבור אלגוריתמים CM ו-FM הפלט תמיד זהה ללא תלות בסדר הופעתם של המשתמשים או ההרשאות במטריצה המקורית.

- שלב התעדוף אינו תלוי בגילוי

האלגוריתמים מורכבים משני שלבים לא תלויים – שלב גילוי ושלב תעדוף התפקידים. ניתן בקלות להחליף את השלב של תעדוף התפקידים בשיטה אחרת שיותר מתאימה לארגון.

- אופי מצטבר

לאחר ביצוע האלגוריתם על קלט מסוים ניתן להוסיף לאלגוריתם קלט חדש ולבצע את האלגוריתם רק על החלק הנוסף ורשימות InitRoles, GenRoles ומונים שהתקבלו מהקלט המקורי. בצורה הזאת ניתן לבצע כריית התפקידים עבור מחלקה מסוימת בארגון ולאחר כך להוסיף מחלקות חדשות בלי צורך להתחיל את כל התהליך מחדש.

3.2.4 תוצאות אמפיריות ועמידות לרעש

כותבי המאמר מביאים תוצאות אמפיריות עבור אלגוריתם FastMiner. ע"י הבדיקות האלה הם רוצים לחקור התנהגות של האלגוריתם גם עבור גודל שונה של הקלט וגם איך פרמטרים שונים משפיעים על התוצאות. קודם מובאות תוצאות עבור קלטים ללא רעש, ולאחר מכן מרחיבים את הדיון למודל הרעש האפשרית, רמת הרעש והשפעה של הרעש על התוצאות.

כקלט הם משתמשים במטריצת UPA שהם מייצרים באופן רנדומלי, מכיוון שקשה למצוא קלטים אמיתיים אשר יאפשרו לראות השפעה של הפרמטרים השונים.
על מנת ליצור את הקלט הם משתמשים באלגוריתם הבא:

קלט: מספר תפקידים - NRoles, מספר משתמשים - NUsers, מספר הרשאות - NPermissions, מספר מקסימאלי של התפקידים עבור המשתמש - MRolesUsr, מספר מקסימאלי של ההרשאות לתפקיד – MPermissionsRole

בשלב הראשון נוצרים NRoles תפקידים, כאשר כל תפקיד הוא רשימה באורך MPermissionsRole של ההרשאות שנבחרים רנדומאלית מתוך ה-NPermissions אפשרויות. בשלב השני נוצרים NUsers, כאשר כל משתמש מקבל מספר רנדומאלי של התפקידים עד NRolesUsr

על מנת לבחון את הדיוק של התוצאות, מוצע לבדוק כמה תפקידים מהרשימה המקורית נמצאות ב-$1 \times$ NRoles התפקידים הפוטנציאלים ראשוניים וכמה ב-$2 \times$ NRoles.

סה"כ בדיקות מראות תוצאות טובות. הדיוק משתפר ככל שהיחס בין המשתמשים ומספר התפקידים שלהם גדל, כי יותר משתמשים מקבלים את אותו תפקיד וסיכוי לגלות אותו גדל. כמו כן תוצאות משתפרות כאשר גדל מספר הרשאות לכל משתמשים ומספר התפקידים תפקידים נשאר קבוע.
זמני הריצה עבור קלטים שונים הם סבירים ונעים בין דקות בודדות לעשרות כתלות בגודל הקלט.

מודל הרעש

על מנת לבצע כריית התפקידים בארגון נדרש להכין מיפוי בין המשתמשים אל ההרשאות אותם הם מקבלים – מטריצת UPA. בפועל תמיד יש פער בין המיפוי למצב ריאלי, מכיוון שמבנה ארגוני תמיד דינאמי וקשה לתחום אותו בזמן על מנת לבנות UPA בצורה אמינה. לדוגמה משתמשים יכולים להחליף תפקיד ולקבל סט חדש של ההרשאות או לפעמים משתמשים לא מקבלים (או מקבלים) הרשאות מסוימות בטעות, למרות שהם חלק (או לא) מהתפקיד שלהם.

באופן הזה ניתן להתייחס לרעש כאל הבדל בין המטריצה שמייצגת המצב האמיתי לבין מטריצה שהתקבלה בקלט. במקרה כזה ניתן לזהות שלוש אפשרויות לרעש:

רעש כללי – ערך 1 מוחלף בערך 0 או להפך. מדובר בטעות נקודתית בקביעת הרשאה ספציפית למשתמש.

רעש מתווסף – מדובר במקרה בו 0 הפך ל-1, אך לא להפך. בדרך כלל זה קורה בגלל שמשתמש קיבל הרשאה באופן נקודתי להשלמת המשימה ולאחר מכן הרשאה לא הוסרה ממנו.

רעש מחסיר – מדובר במקרה שמשתמש עוד לא קיבל הראשות שנדרשות לו לפי התפקיד שלו. בדרך כלל זה קורה עבור משתמש שמתחיל תפקיד חדש ועוד לא קיבל את כל ההרשאות. במקרה כזה 1 הופך ל-0.

הרעש הכללי מכיל גם רעש מחסיר וגם רעש מתווסף, עם זאת היחס שלהם לא יהיה זהה. המטריצות UPA לרוב מאוד דלילות ולכן, בדרך כלל, יהיה הרבה יותר רעש מתווסף במטריצה, מאשר רעש מחסיר. המצב הזה גם תואם את המציאות, כי בדרך כלל לאורך הזמן למשתמשים יש יותר הרשאות מיותרות ממצבים בהם חסר להם הרשאות.

כותבי המאמר בוחנים את העמידות של האלגוריתם FM לרעש כללי, מכיוון שלדעתם זה ישקף את הטיב של האלגוריתם, גם אם במציאות אופי של הקלטים יהיה קצת שונה.

רמות של הרעש

בנוסף לקביעת מודל של הרעש צריך לקבוע איך מודדים את רמת הרעש בקלט. מכיוון שקלט בנוי כמטריצה בינארית, ניתן להתייחס אל כל כניסה במטריצה כביט. הגישה הנאיבית היא לקבוע אחוז הביטים במטריצת הקלט שערכם שונה מהערך האמיתי. אך הגישה הזאת יכולה להיות בעיתית מכיוון שהיא לא מבחינה בין משמעויות של ערכים 0 ו-1. נבחן לדוגמה מיפוי של 2000 משתמשים ו-500 הרשאות. במיפוי כזה יהיו 1000000 כניסות. בעקבות הדלילות של מטריצות UPA , יכול להיות שיש רק 4000 ערכי 1 וכל השאר יהיו ערכי 0. במקרה הזה, אם נבחר רמת הרעש להיות 1%, אזי המספר הכניסות השגויות הוא 10000, שזה מספר גדול ביחס ל-4000 ערכי 1 במטריצה האמיתית. הדבר יכול לגרום למטריצה UPA' שכוללת רמת הרשע של 1% להיות לא אמינה לחלוטין. יתרה מכך, אם מנהל אבטחה יסיר את כל ההרשאות מהמשתמשים, אז הוא ישנה רק 4000 כניסות במטריצה וזה יהיה דומה לרעש של 0.4%. לכן על מנת לקבוע את הרעש בצורה מתאימה לאופי של מטריצת UPA צריך להתייחס רק לאחוז של ערכי 1 שגויים במטריצה.

בנוסף צריך לקבוע באיזה צורה מתווסף הרעש לקלט. אם קובעים את רמת הרעש להיות 10%, יש שתי אפשרויות איך זה יכול לבוא לידי ביטוי. האפשרות הראשונה היא להפוך 10% מתוך הנתונים (ערכי 1), ואז לפי דוגמה קודמת יתחלפו כ-400 כניסות, או האפשרות השנייה היא להפוך כל ביט בהסתברות 0.04 אחוז (400 מתוך 1000000). לדעת Vaidya et al הגישה השנייה יותר מדמה את המציאות.

אחוז שגיאה כללית p:

בהינתן UPA מקורית, תהיה מטריצת UPA' שכוללת רמת הרעש עם אחוז שגיאה p בנויה בצורה הבא:

אם $x \in UPA$, אזי $x \in UPA'$ בהסתברות $1-p$, ואם $x \notin UPA$, אזי $x \in UPA'$ בהסתברות p.

מדד הדיוק בנוכחות הרעש ותוצאות אמפיריות

לשם בדיקת הדיוק של התוצאות FM בנוכחות הרעש ניתן להשתמש בטכניקה קודמת ולבדוק אחוז התפקידים שמתגלים מתוך התפקידים ה-"אמיתיים" (שבעזרתם נבנה הקלט). יכול להיות שהפעם המדד הדיוק הזה פחות מתאים מכיוון שבגלל הרעש חלק מהתפקידים שיתקבלו יהיו תפקידים מקורבים ולא יתאימו במדויק. לכן במקרה הזה צריך להשתמש במדד קירוב מסוים ולבדוק "פסאודו" דיוק של התוצאות. המדד המקורב יכול לשנות את הרמת הדיוק, עם זאת Vaidya et al בעבודה הזאת משתמשים במדד דיוק רגיל ומשאירים את הנושא של "פסאודו" דיוק למחקר עתידי.

מחברי העבודה חוזרים על הניסויים עם קלט שמיוצר באופן רנדומאלי, אך הפעם הם מוסיפים רמות שונות של הרעש אל הקלט (1%, 5%, 10% ו-20%). תוצאות אמפיריות של ה-FM עבור קלט עם הרעש מראות שאלגוריתם חסין לרעש ברמות הנמוכות. התוצאות עבור רמת הרעש של 1% או 5% די זהות לתוצאות המקוריות.

ברמה של 10% תוצאות כבר פחות טובות, אך עדיין נותנות אפשרות להשתמש באלגוריתם (כ-60% דיוק עבור $1 \times \text{NRoles}$ תפקידים פוטנציאליים הראשונים, וכ-80% דיוק עבור $2 \times \text{NRoles}$).

ברמת הרעש של 20% התוצאות כבר לא מספיק מדויקות (כ-40% דיוק עבור $1 \times \text{NRoles}$ תפקידים פוטנציאליים הראשונים, וכ-60% דיוק עבור $2 \times \text{NRoles}$). תוצאות הן דומות גם עבור שינוי יחס בין מספר התפקידים לבין מספר ההרשאות.

3.3 אלגוריתם היוריסטי לפתרון Basic RMP ו-δ − Approx RMP [5]

האלגוריתם מאפשר לפתור גם את Basic RMP וגם את δ − Approx RMP, כאשר הפרמטר δ קובע את הסוג של הבעיה (Basic RMP היא בעצם δ − Approx RMP עם $\delta = 0$).

הקלט של האלגוריתם הוא מטריצת UPA ופרמטר δ. האלגוריתם מורכב משני שלבים:
בשלב הראשון מתוך מטריצת UPA בקלט מוציאים רשימה של התפקידים הפוטנציאליים. ניתן להיעזר בכל אלגוריתם שמחלץ תפקידים פוטנציאליים מתוך UPA. בעבודה הזאת נשתמש ב-FastMiner. על מנת להכין רשימת התפקידים לחלק השני, יש למיין אותם לפי שטח שהם מכסים במטריצת UPA, ז״א לפי מספר כניסות במטריצה שתפקיד מכסה. כאשר משתמשים ב-FastMiner אנו מקבלים מספר משתמשים שתפקיד מכסה (מספר שורות במטריצה), לכן עבור כל תפקיד פוטנציאלי יש להכפיל $final_{count}$ בגודל של תפקיד (מספר הרשאות שלו) על מנת לקבל מספר כניסות במטריצה שהוא מכסה, וזה יהיה שטח כיסוי של תפקיד בחלק השני של האלגוריתם.
בשלב השני מוציאים רשימה סופית של התפקידים ע״י אלגוריתם חמדן, כאשר בכל איטרציה נבחר מועמד הכי טוב עד שרשימת התפקידים מאפשרת לבנות UA, PA כך ש-$UA \otimes PA = UPA$ (אם מדובר ב-BRMP). במהלך איטרציה עבור כל תפקיד שנשאר ברשימה של התפקידים הפוטנציאלים נבחן שטח שלא מכוסה ע״י אותו תפקיד. השטח הלא מכוסה של התפקיד ניתן לחשב על ידי מציאת כניסות 1 ב-UPA שעוד לא מכוסות ע״י תפקידים שכבר נבחרו לרשימה הסופית.

על מנת להשתמש באלגוריתם לפתרון δ − Approx RMP נדרש לעצור כאשר מתקיים:

$$\|UA \otimes PA - UPA\|_1 \leq \delta$$

Require: User-Permission assignment UPA; the approximation threshold δ

1: {Create candidate set of roles}
2: Create a candidate set of roles, $CROLES$, using the $FastMiner$ algorithm
3: Sort $CROLES$ according to the area of each role
4: $ROLES \leftarrow \emptyset$
5: **while** UPA is not covered within δ **do**
6: $BestRole \leftarrow \emptyset$
7: $BestArea \leftarrow 0$
8: **for** each role C in $CROLES$ **do**
9: **if** $area(C) < BestArea$ **then**
10: Break out of the FOR {We have already found the best possible role}
11: **end if**
12: $carea \leftarrow Uncovered\_Area(C, UPA, ROLES)$ {Compute uncovered area of candidate role}
13: **if** $carea > BestArea$ **then**
14: $BestArea \leftarrow carea$
15: $BestRole \leftarrow C$
16: **end if**
17: **end for**
18: $ROLES \leftarrow ROLES \cup C$ {Add C to the set of roles $ROLES$
19: Remove C from $CROLES$
20: **end while**
21: Return $ROLES$

איור 4. פסאודוקוד עבור אלגוריתם לפתרון $\delta - Approx\ RMP$

באיור 10 מתואר פסאודו-קוד של האלגוריתם. בשורות 2-4 מפעילים אלגוריתם FastMiner וממיינים רשימת תפקידים פוטנציאליים לפי שטח כיסוי. בשלב השני (שורות 5-20) מנסים לכסות את המטריצה UPA באופן חמדני. בלולאה בשורות 8-17 עוברים על כל תפקיד ברשימה של תפקידים פוטנציאליים ומחפשים איזה תפקיד יכסה שטח מרבי ואז מוציאים אותה ורשימה של תפקידים פוטנציאליים ומוסיפים לרשימה סופית. השלב השני ממשיך עד שמספר כניסות לא מכוסות ב-UPA גדול מ- δ.

ניתן להיעזר בדוגמה של CompleteMiner (התוצאות של ה-FastMiner עבור הדוגמה הזאת זהות), על מנת להדגים את האלגוריתם. באיור 11 מופיעים תוצאות של השלב הראשון של האלגוריתם (שורות 2-3), כולל חישוב שטחים שתפקידים פוטנציאליים מכסים.

	p1	p2	p3	p4
u1	1	1	0	1
u2	0	1	1	0
u3	1	1	0	1
u4	1	1	0	1
u5	0	1	1	1
u6	0	1	1	1
u7	0	1	1	0
u8	0	1	1	0
u9	0	0	0	1
u10	0	0	0	1
u11	1	1	0	1
u12	1	1	0	1
U13	0	1	1	1

Candidate Roles	Associated Users	Orig Count	Gen Count	Total Count	Area	uc
{p2,p4}	{u1,u3,u4,u5,u6,u11,u12,u13}	0	8	8	16	
{p1,p2,p4}	{u1,u3,u4,u11,u12}	5	0	5	15	
{p2,p3}	{u2,u5,u6,u7,u8,u13}	3	3	6	12	
{p2}	{u1,u2,u3,u4,u5,u6,u7, u8,u11,u12,u13}	0	11	11	11	
{p4}	{u1,u3,u4,u5,u6,u9,u10,u11, u12,u13}	2	8	10	10	
{p2,p3,p4}	{u5,u6,u13}	3	0	3	9	

איור 5. תוצאות של $FastMiner$ כולל חישוב שטחים עבור UPA

כפי שהוזכר קודם, ניתן לראות באיור 11 שבסוף שלב ראשון מוסיפים לכל תפקיד את $area$ שתפקיד יכול לכסות ב-UPA, כאשר $area = |r| * final_{count}$ (באיור $final_{count}$ מסומן כ-$total\ count$).

בנוסף עבור כל תפקיד יש את פרמטר uc, עבור כל איטרציה הוא מסמן כמה שטח בפועל תפקיד יכול לכסות. הרי בכל איטרציה תפקיד שנבחר לרשימה סופית מכסה שטח שיכול להיות חופף עם התפקידים האחרים, ולכן נדרש בכל איטרציה לעדכן את uc עבור תפקידים שעדיין נשארו ברשימה של תפקידים פוטנציאליים. בסוף השלב הראשון, מכיוון שעדיין אין תפקידים ברשימה סופית, אזי מתקיים:

$$area(c) = UncoveredArea(c, UPA, ROLES)$$

אחרי השלב ההכנות (השלב הראשון של האלגוריתם) ניתן לעבור לשלב השני של חיפוש תפקדים שיכנסו לרשימה סופית ומהן יורכבו UA ו-PA.

איטרציה 1:

	p1	p2	p3	p4
u1	1	1	0	1
u2	0	1	1	0
u3	1	1	0	1
u4	1	1	0	1
u5	0	1	1	1
u6	0	1	1	1
u7	0	1	1	0
u8	0	1	1	0
u9	0	0	0	1
u10	0	0	0	1
u11	1	1	0	1
u12	1	1	0	1
u13	0	1	1	1

Candidate Roles	Associated Users	Area	uc
{p2,p4}	{u1,u3,u4,u5,u6,u11,u12,u13}	16	16
{p1,p2,p4}	{u1,u3,u4,u11,u12}	15	X
{p2,p3}	{u2,u5,u6,u7,u8,u13}	12	X
{p2}	{u1,u2,u3,u4,u5,u6,u7, u8,u11,u12,u13}	11	X
{p4}	{u1,u3,u4,u5,u6,u9,u10,u11, u12,u13}	10	X
{p2,p3,p4}	{u5,u6,u13}	9	X

באיטרציה ראשונה נבחר תפקיד $\{p_2, p_4\}$ מכיוון שהוא מכסה הכי הרבה כניסות ב-UPA בשלב הזה. בתמונה שמאלית ניתן לראות את הכנוסות שהוא מכסה.

איטרציה 2:

	p1	p2	p3	p4
u1	1	1	0	1
u2	0	1	1	0
u3	1	1	0	1
u4	1	1	0	1
u5	0	1	1	1
u6	0	1	1	1
u7	0	1	1	0
u8	0	1	1	0
u9	0	0	0	1
u10	0	0	0	1
u11	1	1	0	1
u12	1	1	0	1
u13	0	1	1	1

Candidate Roles	Associated Users	Area	uc
{p2,p4}	{u1,u3,u4,u5,u6,u11,u12,u13}	16	0
{p1,p2,p4}	{u1,u3,u4,u11,u12}	15	5
{p2,p3}	{u2,u5,u6,u7,u8,u13}	12	9
{p2}	{u1,u2,u3,u4,u5,u6,u7, u8,u11,u12,u13}	11	3
{p4}	{u1,u3,u4,u5,u6,u9,u10,u11, u12,u13}	10	2
{p2,p3,p4}	{u5,u6,u13}	9	X

באיטרציה שניה נבחר $\{p_2, p_3\}$. למרות ששטח הכללי שלו קטן מ-$\{p_1, p_2, p_4\}$, אבל השטח השולי שלו (uc) גדול יותר מכל שאר התפקידים הפוטנציאליים.

איטרציה 3:

	p1	p2	p3	p4
u1	1	1	0	1
u2	0	1	1	0
u3	1	1	0	1
u4	1	1	0	1
u5	0	1	1	1
u6	0	1	1	1
u7	0	1	1	0
u8	0	1	1	0
u9	0	0	0	1
u10	0	0	0	1
u11	1	1	0	1
u12	1	1	0	1
u13	0	1	1	1

Candidate Roles	Associated Users	Area	uc
{p2,p4}	{u1,u3,u4,u5,u6,u11,u12,u13}	16	0
{p1,p2,p4}	{u1,u3,u4,u11,u12}	15	5
{p2,p3}	{u2,u5,u6,u7,u8,u13}	12	0
{p2}	{u1,u2,u3,u4,u5,u6,u7, u8,u11,u12,u13}	11	0
{p4}	{u1,u3,u4,u5,u6,u9,u10,u11, u12,u13}	10	2
{p2,p3,p4}	{u5,u6,u13}	9	0

באופן דומה באיטרציה שלישית נבחר $\{p_1, p_2, p_4\}$

איטרציה 4:

	p1	p2	p3	p4
u1	1	1	0	1
u2	0	1	1	0
u3	1	1	0	1
u4	1	1	0	1
u5	0	1	1	1
u6	0	1	1	1
u7	0	1	1	0
u8	0	1	1	0
u9	0	0	0	1
u10	0	0	0	1
u11	1	1	0	1
u12	1	1	0	1
u13	0	1	1	1

Candidate Roles	Associated Users	Area	uc
{p2,p4}	{u1,u3,u4,u5,u6,u11,u12,u13}	16	0
{p1,p2,p4}	{u1,u3,u4,u11,u12}	15	0
{p2,p3}	{u2,u5,u6,u7,u8,u13}	12	0
{p2}	{u1,u2,u3,u4,u5,u6,u7, u8,u11,u12,u13}	11	0
{p4}	{u1,u3,u4,u5,u6,u9,u10,u11, u12,u13}	10	2
{p2,p3,p4}	{u5,u6,u13}	9	0

באיטרציה רביעית נבחר $\{p_4\}$. בתום האיטרציה הרביעית כל הכניסות עם הערך 1 ב-UPA מקורי מכוסים, ולכן קיבלנו רשימה סופית של התפקידים: $\{p_4\}, \{p_1, p_2, p_4\}, \{p_2, p_3\}, \{p_2, p_4\}$. אם היינו בוחרים פקטור δ להיות 2, אזי האלגוריתם היה עוצר באיטרציה שלישית והרשימה הסופית של התפקידים הייתה נראית כך: $\{p_1, p_2, p_4\}, \{p_2, p_3\}, \{p_2, p_4\}$.

ניתן לשפר את היעילות של האלגוריתם אם בכל איטרציה של ה-while נמיין את התפקידים שנותרו לפי סדר יורד של השטח שכל תפקיד יכול לכסות. בגישה ראשונה (שמופיעה בפסאודו-קוד) עוצרים כאשר מגלים את התפקיד שהשטח שהוא מכסה קטן יותר מהשטח המקסימאלי שעוד לא מכוסה שנראה עד אז, ובכך מסיקים שהוא פחות טוב מהקודם (בתום שלב ראשון התפקידים מסודרים לפי השטח הכללי שהם מכסים), ולכן נבחר תפקיד קודם שהוא הכי מתאים באיטרציה הנוכחית. במידה והתפקידים ממוינים לפי שטח שהם עתידים לכסות, ניתן לקחת תפקיד ראשון בכל איטרציה.

סיבוכיות הזמן של האלגוריתם

שלב ראשון - כפי שהראנו קודם, סיבוכיות של FastMiner היא $O(|U|^2)$

שלב שני – במקרה הגרוע ביותר נצטרך לשלוב $|U|$ תפקידים לרשימה הסופית, כאשר לכל משתמש יהיה תפקיד משלו. לכן בשלב השני תהיו לכל היותר $|U|$ איטרציות. בכל איטרציה יש לולאה שרצה על התפקידים פוטנציאליים ($O(|U|^2)$) ומחשבת את השטחים לא מכוסים שניתן לכסות ב-UPA. סה"כ מתקבל שסיבוכיות הזמן של השלב השני היא $O(|U|^3)$

הסיבוכיות הכוללת היא - $O(|U|^3) + O(|U|^2) = O(|U|^3)$

בפעול כאשר משתמשים בעצירה מוקדמת של האלגוריתם ומיון תפקידים פוטנציאליים לפי שטחים לא מכוסים, ניתן לקבל סיבוכיות טובה מכך.

תוצאות אמפיריות

בדומה ל-FastMiner, על מנת לבחון את האלגוריתם ולחקור השפעה של הפרמטר δ Vaidya et al. מביאים תוצאות אמפיריות של האלגוריתם על קלט שהם מייצרים באופן רנדומאלי. האלגוריתם ליצירת הקלט זהה לאלגוריתם שהם השתמשו בו עבור FastMiner ומאפשר לשלוט על מספר משתמשים, תפקידים, הרשאות, מספר משתמשים לתפקיד ומספר הרשאות לתפקיד. לצורך בדיקת השפעה של פרמטר δ הם בחרו ערכים: $0, 1, 5, 10, 20$ וביצעו שני ניסויים: מספר משתמשים קבוע עם מספר הרשאות משתנה ומספר הרשאות קבוע עם מספר משתמשים משתנה. מהתוצאות ניתן להסיק שהגדלת δ גורמת לירידה בזמני ביצוע, במיוחד עבור מספר הרשאות גבוה לתפקיד. זה צפוי מכיוון שככל שעולה מספר הרשאות לתפקיד כך האלגוריתם מצליח לכסות שטחים גדולים יותר בשלבים מוקדמים יותר וכל איטרציה רצה פחות זמן. כמו כן ככל ש-δ גדול יותר כך האלגוריתם צריך פחות איטרציות מכיוון שהוא מגיעה לאי-דיוק מותר מוקדם יותר. הגדלת הפרמטר δ

מצד שני משפיעה על הדיוק של האלגוריתם כולו. אחוז הדיוק של האלגוריתם ביחס ל-δ:

ערך של δ	אחוז תפקידים מקורים שהתגלו
0	~100%
1	~90%
5	~80%
10	~70%
20	~60%

חשוב לציין, גם עבור ערכי δ גבוהים, התפקידים שמתגלים בהתחלה הם תפקידים משמעותיים יותר עם מספר הרשאות גבוה יותר, ולכן גם מציאת חלק מהתפקידים תתרום לבניית מודל התפקידים הסופי.

עבור שינוי מספר משתמשים עם מספר ההרשאות הכללי קבוע תוצאות דיוק הן דומות. לעומת זאת זמני ביצוע הרבה יותר טובים, לכן שיפור שנובע משינוי δ פחות מורגש, אך עדיין קיים.

4. כריית תפקידים בנוכחות אילוצים [6]

4.1 מבוא והגדרות פורמאליות

בפרק קודם תואר אלגוריתם מאפשר לבצע כריית תפקידים עבור מודל $RBAC_0$. אבל כפי שראינו בתחילת העבודה מודל $RBAC_0$ הינו מודל מצומצם יחסית ולא נותן מענה לכל האתגרים של בקרת גישה. לכן נרצה להרחיב את הנושא של כריית התפקידים למודל מורחב $RBAC_2$.

החלק הזה של העבודה מתייחס לאילוצים מסוג SoD. האילוצים האלה מאפשרים להגביל את המשתמשים כך שלא יוכלו לבצע פעולות קריטיות במערכת ללא מעורבות של גורם נוסף.

דוגמה:

אם נדרשים k משתמשים כדי להפעיל n הרשאות על מנת לבצע פעולה רגישה, אזי האילוץ קובע שאף קבוצה של $k-1$ משתמשים לא תוכל להפעיל את הקבוצה של n ההרשאות ולבצע את הפעולה. לדוגמה אף משתמש לא יהיה מורשה גם לדרוש תשלום וגם לאשר אותו, אלא צריך לפחות שני משתמשים לביצוע של התשלום. ההרשאות בקבוצת הרשאות כזאת נקראים – *הרשאות סותרות*.

על מנת לאפשר הגבלה כזאת ע"י אילוץ SoD צריך לקבוע קבוצת הרשאות שמאפשרות לבצע פעולה קריטית ולקבוע מספר מינימאלי של משתמשים שנדרש לערב על מנת להפעיל את קבוצת ההרשאות.

בתחילת העבודה תיארנו בהרחבה את הנושא של האילוצים וקבענו שיטת רישום. בשלב הראשון של הניתוח כריית תפקידים עם אילוצי SoD נתייחס רק לאילוצים מסוג סטטיים. בהמשך נרחיב את הדיון גם לאילוצים דינאמיים.

האילוץ שמוצג בדוגמה הוא אילוץ סטטי שחל על כל תת-קבוצה של k משתמשים מתוך U. לכן ניתן לרשום את האילוץ בצורה הבא:

$$\left(\bigcup(U' = \{\forall U' \subseteq U, |U'| = k-1\}), \{p_1, p_2, \dots, p_n\}, s\right)$$

אבל מכיוון שתמיד מדובר על אילוץ SoD סטטי נוכל לרשום בצורה מופשטת:

$sod < \{p_1, \dots, p_n\}, k >$ - כאשר $\{p_1, \dots, p_n\}$ היא קבוצת הרשאות ו-k הוא מספר שלם $(2 \le k \le n)$,

הסימון המקוצר $SoD_{n,k}$

אילוצי הפרדת אחריויות מוגדרים על היחס בין המשתמשים להרשאות, אבל מודל RBAC אינו מקשר ישירות בין המשתמשים להרשאות, אלא עושה זאת דרך תפקידים. לכן על מנת לאפשר אכיפה של האילוצים נדרש להמיר אותם לאילוצים על היחס בין התפקידים להרשאות. האילוץ המתאים נקרא RSSoD Role level Static Separation of Duty. מודל RBAC מתייחס אל התפקידים כאל קבוצות הרשאות, לכן על מנת לבטא אילוצי SoD באמצעות אילוצי RSSoD נדרש להמיר קבוצת הרשאות לקבוצת תפקידים מתאימה, ז״א למצוא רשימה של תפקידים שכוללת את ההרשאות הרלוונטיות ולקבוע קבוצות תפקידים שעבורן נדרש מספר מינימאלי של משתמשים.

באופן דומה לאילוץ SoD , לפי שיטת רישום מלאה אילוץ RSSoD בצורה פורמאלית ניתן לרשום כך:

$$\left(\bigcup(U' = \{\forall U' \subseteq U, |U'| = k-1\}), \{r_1, r_2, \dots, r_m\}, s\right)$$

או בצורה מופשטת:

$rssod < \{r_1, \dots, r_m\}, k >$ - כאשר $\{r_1, \dots, r_m\}$ היא קבוצת תפקידים ו- k הוא מספר שלם $(2 \leq k \leq m)$.

המשמעות היא - דרושים לפחות k משתמשים על מנת לקבל קבוצה מסוימת של m תפקידים. הסימון המקוצר הוא $\mathrm{RSSoD_{m,k}}$. לדוגמה אם יש לנו אילוץ SoD ושלושה תפקידים הבאים:

$$SoD_{4,2} = sod < \{p_1, p_2, p_3, p_4\}, 2 >;\ \ r_1 = \{p_1, p_2, p_3\}, r_2 = \{p1, p_4\}, r_3 = \{p_3, p_4\}$$

אזי אכיפה של האילוץ $\mathrm{SoD_{4,2}}$ דורשת להגדיר את שני אילוצי RSSoD הבאים:

$$RRSoD_{2,2} = rssod < \{r_1, r_2\}, 2 > \ \ RSSoD_{2,2} = rssod < \{r_1, r_3\}, 2 >$$

אילוץ RSSoD יקר מאוד מבחינה חישובית, מכיוון שבזמן ההשמה של התפקיד אל המשתמש נדרש לבדוק את כל המשתמשים על מנת לוודא שלא הופר אילוץ על המספר המינימאלי של המשתמשים. לכן עדיף להמיר את אילוצי RSSoD לאילוצים על משתמש בודד.

אילוצים כאלה נקראים SMER (Statically Mutually Exclusive Role). אילוץ מסוג $\mathrm{SMER_{t,m}}$ אומר שאף משתמש לא יכול להיות משויך ל-t או יותר תפקידים מתוך קבוצה של m תפקידים. ניתן לרשום באופן פורמאלי כך:

$$(U, \bigcup(R' = \{\forall R' \subseteq \{r_1, \dots, r_m\}, |R'| = t\}), s)$$

או בצורה מופשטת:

$smer < \{r_1, \dots, r_m\}, t >$ - כאשר r_i הוא תפקיד $(1 \leq i \leq m)$ ו-t הוא מספר שלם $(2 \leq t \leq m)$

בדרך כלל נהוג להגדיר את האילוצי $\mathrm{SMER}_{t,m}$ כאשר $t = m$ (או בקיצור SMER_t). המשמעות היא שאף משתמש לא יכול לקבל את כל התפקידים בקבוצה מסוימת של t תפקידים. אם נחזור לדוגמה קודמת, אזי את האילוץ $\mathrm{RSSoD}_{2,2}$ ניתן לרשום באופן הבא:

$$SMER_2 = smer < \{r_1, r_2\} >, SMER_2 = smer < \{r_1, r_3\} >$$

אם יש קבוצת תפקידים שעבורה קיים אילוץ SMER אז התפקידים בקבוצה נקראים - *תפקידים סותרים*.

4.2 בעיית $RMP - SoD$

אלגוריתמים בסיסיים לכריית תפקידים מאפשרים לבנות מודל תפקידים מתוך המיפוי בין משתמשים להרשאות. את המיפוי נהוג לבנות כמטריצה בוליאנית - UPA. האלגוריתמים הבסיסיים האלה לא נותנים מענה עבור האילוצים של המודל $RBAC$. בחלק הזה של העבודה מוצג אלגוריתם אשר מבצע כריית תפקידים ובנוסף מחשב קבוצת אילוצים על התפקידים אשר מאפשרים לאכוף את האילוצים מסוג SoD. הקלט של האלגוריתם כולל מיפוי UPA בדומה לאלגוריתמים שראינו קודם ובנוסף קבוצה של אילוצים מסוג $\mathrm{SoD}_{n,k}$, והפלט שלו כולל פירוק למטריצות UA ו-PA וקבוצה מתאימה של האילוצים מסוג SMER_t כך שמספר התפקידים הוא מינימאלי. ניתן לחלק את האלגוריתמים לשתי קטגוריות: $\mathrm{SoD} - \mathrm{aware}$ ו-$\mathrm{post} - \mathrm{processing}$, תלוי אם אילוצי SMER מתקבלים תוך כדי כריית תפקידים או אחריה.

הסיבה לשימוש באילוצי SMER היא סיבוכיות. בספרות הוכח שבהינתן מטריצות UA ו-PA, אז בדיקה האם הם ביחד עומדים בדרישות אילוצי SoD היא בעיה $coNP$ שלמה. בשונה מאילוצי SoD שמגבילים את קבוצת המשתמשים, אילוצי SMER מגבילים משתמש בודד מבחינת התפקידים, ולכן ניתן לבדוק בזמן פולינומיאלי האם אילוץ מתקיים מול המטריצה UA. המטריצה PA איננה רלוונטית לאילוצי SMER. אבל על מנת לאכוף אילוצי SoD ע"י אילוצי SMER נדרש קודם לגזור אותם מתוך האילוצי SoD בעזרת מטריצה PA.

הגדרה פורמאלית של בעיית $RMP-SoD$

בהינתן קבוצת U (משתמשים), קבוצת P (הרשאות), מיפוי UPA וקבוצה E (אילוצי SoD) נדרש למצוא קבוצת תפקידים R, מטריצות UA ו-PA וקבוצה C (אילוצי SMER) , כך שיתקיים:

1) $UA \otimes PA = UPA$,

2) הקבוצה C אוכפת את E,

3) $|R|$ מינימאלי.

נניח לדוגמה שמתקבל הקלט הבא:

$$U = \{u_1, u_2, u_3, u_4, u_5\}$$

$$P = \{p_1, p_2, p_3, p_4, p_5, p_6\}$$

$$E = \{e_1, e_2\} \quad e_1 = sod < \{p_3, p_5\}, 2 > \quad e_2 = sod < \{p_1, p_5, p_6\}, 2 >$$

$$UPA = \begin{array}{c|cccccc} & p_1 & p_2 & p_3 & p_4 & p_5 & p_6 \\ u_1 & 0 & 1 & 0 & 0 & 1 & 0 \\ u_2 & 0 & 1 & 0 & 0 & 1 & 0 \\ u_3 & 1 & 1 & 0 & 1 & 1 & 0 \\ u_4 & 1 & 1 & 1 & 0 & 0 & 0 \\ u_5 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}$$

פתרון אפשרי לבעיה הוא:

$$UA = \begin{array}{c|cccc} & r_1 & r_2 & r_3 & r_4 \\ u_1 & 0 & 0 & 1 & 0 \\ u_2 & 0 & 0 & 1 & 0 \\ u_3 & 0 & 1 & 1 & 0 \\ u_4 & 1 & 0 & 0 & 0 \\ u_5 & 0 & 0 & 0 & 1 \end{array} \qquad PA = \begin{array}{c|cccccc} & p_1 & p_2 & p_3 & p_4 & p_5 & p_6 \\ r_1 & 1 & 1 & 1 & 0 & 0 & 0 \\ r_2 & 1 & 1 & 0 & 1 & 0 & 0 \\ r_3 & 0 & 1 & 0 & 0 & 1 & 0 \\ r_4 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}$$

$$C_1 = \{smer\langle\{r_1, r_3\}\rangle\}; \quad C_2 = \{smer\langle\{r_1, r_3, r_4\}\rangle, smer\langle\{r_2, r_3, r_4\}\rangle\}; C = C_1 \cup C_2$$

נראה שהפתרון מקיים את הדרישות (1), (2) ו-(3):

1) $UA \otimes PA = UPA$

ע"פ הגדרה של כפל מטריצות בוליאניות שהגדרנו קודם: $upa_{ij} = \bigvee_{l=1}^{4}(ua_{il} \wedge pa_{lj})$. ולכן נקבל:

$$UA \otimes PA = \begin{matrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{matrix} = UPA$$

2) הקבוצה C אוכפת את E

עבור e_1 מתקיים - p_3 שייכת רק לתפקיד r_1 , ו-p_5 שייכת רק ל-r_3, ולכן C_1 אוכף את e_1

עבור e_2 מתקיים - p_5 ו-p_6 שייכות לתפקידים r_3 ו-r_4 בהתאם, אבל p_1 שייכת לשני התפקידים r_1 ו-r_2 ולכן נדרשים שני אילוצים מהקבוצה C_2 על מנת לאכוף את ה-e_2.

איחוד $C_1 \cup C_2$ אוכף את E.

3) הקבוצה R שגודלה 4 היא מינימאלית

נבחן את מטריצה UPA על מנת לקבל הערכה למספר מינימאלי של התפקידים. נבדוק את ההרשאות שייחודיות למשתמשים מסוימים. p_6 ייחודית ל-u_5, p_3 ל-u_4, p_4 ל-u_3. שלוש ההרשאות האלו חייבות להשתייך לתפקידים שונים ע"מ לאפשר למשתמשים לקבל אותן באופן בלעדי. בנוסף, קבוצת ההרשאות $\{p_2, p_5\}$ אמנם לא שייכת באופן ייחודי למשתמש מסוים, אבל קבוצת המשתמשים $\{u_1, u_2\}$ מקבלת רק את ההרשאות מהקבוצה הזאת, לכן נדרש תפקיד נפרד רק עם ההרשאות האלו. סה"כ קיבלנו שמספר מינימאלי של תפקידים שיכול לספק את ה-UPA הוא 4.

4.3 אלגוריתמים ליצירת SMER בגישת post − processing

כפי שצוין קודם, יש שתי גישות לקבל את אילוצי SMER מתוך אילוצי SoD. אחת הגישות נקראת post − processing. בגישה זו קודם מקבלים מתוך המיפוי UPA קבוצת תפקידים מינימאלית ומטריצות UA, PA בעזרת אלגוריתם כלשהו לבעיית RMP, ורק אחרי זה בעזרת PA ואילוצי SoD יוצרים את הקבוצה C. להלן מוצגים מספר אלגוריתמים שמיישמים את הגישה הזאת.

פתרון נאיבי

האפשרות הראשונה היא להשתמש באלגוריתמים קיימים ליצירת אילוצי SMER. בספרות (Li et al. [2007]) קיים תיאור של האלגוריתם אשר מאפשר לבצע המרה של אילוצים מסוג SoD לאילוצים מסוג $\mathrm{SMER_{t,m}}$. האלגוריתם אינו מתייחס לכריית התפקידים, אלא מתאים למקרים בהם כבר קיים מבנה תפקידים של הארגון אשר לא נובע ישירות מהמיפוי בין הרשאות למשתמשים (מטריצה UPA). האלגוריתם פועל בשני שלבים. בשלב ראשון ממירים את אילוצי ה-SoD לאילוצי RSSoD. בשלב השני יוצרים אילוצי SMER מתוך אילוצי RSSoD בעזרת פונקציית $SMER - Gen$. בצורה הזאת ניתן לקבל קבוצה מינימאלית של $\mathrm{SMER_{t,m}}$ אשר אוכפים את אילוצי SoD המקורים. החיסרון של השיטה הוא שמציאת דרישות RSSoD כרוכה במציאת קבוצות מינימליות של תפקידים שעומדים בדרישות SoD, וזה מאוד יקר חישובית. בהמשך אנו מציגים שתי שיטות נוספות שמטרתן להתמודד עם החיסרון הזה. השיטות האלה מתבססות על העובדה שקבוצת תפקידים נובעת אך ורק מקשר בין המשתמשים להרשאות ולכן ניתן לבצע מעבר ישיר בין אילוצים מסוג SoD לאילוצים מסוג SMER. הדבר מתאפשר מכיוון שאנחנו יכולים להתעלם מכל קשר בין התפקידים, כגון היררכיית תפקידים, ולהתייחס לתפקיד כאל קבוצת הראשות בלבד.

יצירת אילוצי $\mathrm{SMER_2}$

השיטה המשופרת הראשונה היא ליצור אילוצי $\mathrm{SMER_2}$ ישר מתוך אילוצי SoD. כל אילוץ מסוג $\mathrm{SMER_2}$ ניתן לרשום באופן הבא:

$$c = \langle \{r_i, r_j\} \rangle$$

המשמעות היא שהתפקידים r_i ו-r_j הם תפקידים סותרים, ז״א אף משתמש לא יכול לקבל את שני התפקידים. אמנם, האילוצים מהסוג הזה מאוד מגבילים, אך עדיין ניתן לאכוף את אילוצי SoD בעזרתם בצורה יעילה. הקלט של האלגוריתם כולל מטריצות PA, UA וקבוצה $E = \{e_1, \dots, e_m\}$ של אילוצי $\mathrm{SoD_{n,k}}$. הפלט של האלגוריתם הוא קבוצה $C = \{c_1, \dots, c_q\}$ של אילוצי $\mathrm{SMER_2}$.

הרעיון המרכזי של האלגוריתם הוא לבדוק כל אילוץ e האם הוא ניתן לאכיפה ואם כן ליצור קבוצת אילוצים C מתאימה. על מנת ליצור אילוץ $\mathrm{SMER_2}$ מתאים עבור אילוץ e קודם כל צריך למצוא קבוצת תפקידים S כך

שכל תפקיד בקבוצה מקבל לפחות הרשאה אחת מתוך e. אחר כך יש לבדוק האם קיים תפקיד ב-S שמקבל את כל ההרשאות ב-e, אם כך אזי אילוץ e לא ניתן לאכיפה. אחרת ניתן לקבוע שכל הזוגות החוקיים ב-S הם תפקידים סותרים. שני תפקידים $\{r_i, r_j\}$ הינם תפקידים סותרים חוקיים אם מתקיים:

$$r_i \backslash r_j \neq \emptyset \wedge r_j \backslash r_i \neq \emptyset$$

דוגמה

נניח שקיים אילוץ $e = \langle\{p_a, p_b, p_c\}, 2\rangle$, ז"א נדרשים לפחות שני משתמשים על מנת להפעיל את ההרשאות $\{p_a, p_b, p_c\}$, וקיימים שני תפקידים: $r_1 = \{p_a, p_b\}$ ו-$r_2 = \{p_b, p_c\}$, אזי ניתן להגדיר את התפקידים כתפקידים סותרים מכיוון שאף אחד מהם לא תת קבוצה של השני ולכל אחד יש לפחות הרשאה סותרת אחת שאין לשני. ניתן לקבוע את התפקידים הסותרים בצורה הזאת ע"י בדיקה:

$$assign\_perms[r_i] \nsubseteq assign\_perms[r_j] \wedge assign\_perms[r_j] \nsubseteq assign\_perms[r_i]$$

את הבדיקה הזאת ניתן לעשות בעזרת מטריצה PA.

Algorithm 1. 2-2_SMER_Post_Processing

```
1: Required: UA, PA, a set E of k − n SoD constraints
2: C = φ
3: for each SoD e_i in E do
4:    Using PA, find a set S of roles having at least one permission in e_i
5:    if any role in S has all the permissions in e_i then
6:       Declare e_i as not enforceable
7:       continue
8:    end if
9:    for each pair of roles (r_i,r_j) in S do
10:      if assign_perms[r_i] ⊈ assign_perms[r_j] ∧ assign_perms[r_j] ⊈
         assign_perms[r_i] then
11:         if UA matrix satisfies ⟨{r_i,r_j},2⟩ then
12:            C = C ∪ {⟨{r_i,r_j},2⟩}
13:         else
14:            Declare e_i as not enforceable
15:         end if
16:      end if
17:   end for
18: end for
```

איור 6. פסאודוקוד של $SMER_2$ *Post Processing*

הפסאודו-קוד של האלגוריתם SMER_2 Post Processing למציאת אילוצי SMER_2 מתוך מטריצת PA וקבוצת E של אילוצי $\mathrm{SoD}_{\mathrm{n,k}}$ מובא באיור 12. הקבוצה S מציינת את קבוצת התפקידים המינימאלית שהרשאותיהם שייכים אל אילוצי E. עבור כל אחד מהאילוצים ב-E האלגוריתם מבצע את הפעולות הבאות:

- בשורות 5-8 האלגוריתם בודק עבור כל אילוץ האם אילוץ הוא בר אכיפה או לא.
- אם אילוץ אכן ניתן לאכיפה אזי עבור כל זוג תפקידים בשורה 10 נבדק האם צריך לסמן את הזוג כתפקידים סותרים.
- בשורות 11-15 – נבדק האם המטריצה UA לא מפרה את התפקידים הסותרים. אם המטריצה אכן מפרה את התפקידים הסותרים, אזי אילוץ מסומן ככזה שאינו בר אכיפה, אחרת זוג התפקידים מצורף לקבוצה C.

ניתן לייצר את האילוצים SMER_2 בזמן פולינומיאלי. זמן בדיקה האם אילוץ ניתן לאכיפה הוא $O(|S|)$ וזמן מציאת תפקידים סותרים הוא $O(|S|^2)$. לכן הזמן הכולל הוא $O(|E||S|^2)$.

השיטה הזאת הרבה יותר יעילה מהפתרון הנאיבי, אך החיסרון שלה הוא שיותר מדי אילוצי SoD יכולים להפוך ללא ברי אכיפה מכיוון שמטריצת UA מפרה את אילוצים SMER_2 המתאימים.

יצירת אילוצי SMER_t

השיטה הנוספת ממשפחת Post Processing היא ליצור אילוצי SMER_t מתוך אילוצים SoD. האלגוריתם עושה הערכה לקבוע t מקסימאלי שמאפשר לאכוף את אילוצים SoD בעזרת אילוצים SMER_t. האלגוריתם הזה פחות מגביל מהאלגוריתם שמשתמש רק באילוצי SMER_2, אך עדיין יעיל יותר מהפתרון הנאיבי.

האלגוריתם מקבל כקלט את מטריצת PA וקבוצה $E = \{e_1, \dots, e_m\}$ של אילוצי $\mathrm{SoD}_{\mathrm{n,k}}$, ובפלט שלו מתקבלת קבוצה $C = \{c_1, \dots, c_q\}$ של אילוצים מסוג SMER_t, כך שאילוצי C אוכפים את אילוצי E.

עבור כל אילוץ $e \in E$ האלגוריתם מבצע את הפעולות הבאות:

בשלב הראשון נדרש למצוא קבוצת תפקידים S כמו ב-SMER_2 Post Processing , כך שעבור כל תפקיד ב-S מתקיים שלפחות הרשאה אחת של התפקיד שייכת אל e.

בשלב השני נדרש לבדוק האם האילוץ הוא בר אכיפה. אם מתקיים לפחות תנאי אחד מהבאים, אזי האילוץ אינו בר אכיפה:

- עבור אילוץ מהצורה $e =< \{p_1, \dots, p_n\}, k >$ מתקיים $|S| < k$, ז״א יש פחות תפקידים ממספר מינימאלי של המשתמשים, ולכן לא ניתן להפריד את הרשאות בין המשתמשים ע״י תפקידים.
- לפחות תפקיד אחד ב-S מקבל את כל ההרשאות ב-e.

בשלב השלישי אם e הוא בר אכיפה נדרש לבדוק האם יש תפקיד ב-S שהוא תת-קבוצה של תפקיד אחר ב-S. אם כן, אזי האילוץ $SMER$ היחיד שניתן לקבוע יהיה מסוג SMER_2. אחרת ניתן למצוא קבוע t מקסימאלי שעבורו מתקיים $|S| > (k-1)(t-1)$. הסיבה לשימוש בתנאי הזה תובהר בהמשך במשפט 1.

בשלב האחרון יש ליצור אילוץ SMER_t עבור כל תת-קבוצה של S בגודל t ולהוסיף אותו לקבוצה C.

הפסאודו-קוד של האלגוריתם מובא באיור 13. בשורות 5-8 ו-9-12 נבדק האם אילוץ e_i הוא בר אכיפה. בשורות 13-14 מיוצר אילוץ SMER_2 אם לא ניתן ליצור אילוצי SMER_t, אחרת בשורה 16 מוצאים את הקבוע t ובשורות 17-18 נבדק שאילוצי SMER_t לא מופרים על ידי מטריצת UA. אם אכן קיים אילוץ שהופר, אזי אילוץ e_i מסומן כלא בר אכיפה, אחרת אילוצי SMER_t מצטרפים ל-C.

את זמן חישוב אילוצי SMER_t ניתן לאמוד באופן הבא. זמן הבדיקה שהאילוץ e_i ניתן לאכיפה הוא $O(|S|)$. זמן בדיקה האם ניתן לייצר SMER_t או רק SMER_2 הוא $O(|S|^2)$, וזמן יצירה של כל הקומבינציות של t תפקידים מתוך $|S|$ הוא $O(|S|^n)$, כאשר n הוא מספר ההרשאות באילוץ e_i. מכאן שהזמן הכולל הוא $O(|S|^n)$.

Algorithm 2. t-t_SMER_Post_Processing

```
1: Required: UA, PA, a set E of k − n SoD constraints
2: C = φ
3: for each SoD e_i in E do
4:    Using PA, find a set S of roles having at least one permission in e_i
5:    if number of roles in S is less than k of e_i then
6:       Declare e_i as not enforceable
7:       continue
8:    end if
9:    if any role in S has all the permissions in e_i then
10:      Declare e_i as not enforceable
11:      continue
12:   end if
13:   if permission set of any role is a subset of another (permission set considers only
      the permissions of e_i that are in the role) then
14:      Generate 2-2 SMER constraints similar to Algorithm 1
15:   else
16:      Find the largest value of t such that |S|>(t-1)(k-1)
17:      for each subset of roles R of size t from S do
18:         if UA satisfies ⟨R, t⟩ then
19:            C = C ∪ {⟨R, t⟩}
20:         else
21:            Declare e_i as not enforceable
22:         end if
23:      end for
24:   end if
25: end for
```

איור 7. פסאודוקוד של האלגוריתם $SMER_t\ Post\ Processing$

משפט 1

בהינתן אילוץ $\mathrm{SoD}_{\mathrm{n,k}}$ וקבוצת תפקידים S, הערך המקסימאלי של קבוע t שעבורו ניתן לייצר אילוץ מסוג $\mathrm{SMER}_{\mathrm{t}}$ שאוכף את האילוץ SoD הוא הקבוע השלם t הגבוה ביותר המקיים $|S| > (k-1)(t-1)$.

הוכחה:

התפקידים בקבוצה S הם תפקידים שמכילים את ההרשאות הנדרשות לביצוע משימה בהפרדת האחריויות. על מנת לאכוף את האילוץ $\mathrm{SoD}_{\mathrm{n,k}}$ נדרשים לפחות k משתמשים על מנת לקבל n הרשאות. לכן אם נמצא ערך של t כך שאם נשייך $(t-1)$ תפקידים שונים ל-$(k-1)$ משתמשים שונים, אזי $(t-1)(k-1)$ תפקידים לא יהיו שווים ל-$|S|$, ז״א $|S| \neq (t-1)(k-1)$. מכאן שאחרי השמה של $(t-1)$ תפקידים ל- $(k-1)$ משתמשים יישאר לפחות תפקיד אחד עבור המשתמש ה-k -י. מכאן שמתקיים $|S| > (t-1)(k-1)$.

הערך של t שנמצא בדרך הזאת הוא לא חסם מדויק, אך על מנת למצוא גבול מדויק עבור t דרוש זמן חישוב הרבה יותר גדול. מכיוון שבסוף נצטרך ליצור אילוץ $\mathrm{SMER}_{\mathrm{t}}$ עבור כל תת-קבוצה של S בגודל t, היינו רוצים ערך t גבוה כמה שיותר על מנת לקבל פחות תתי קבוצות. נניח לדוגמה $|S| = 8, k = 3$. זאת אומרת יש קבוצה של 8 תפקידים סותרים, כך שנדרשים לפחות 3 משתמשים על מנת לקבל אותם. לפי משפט 1 נקבל שעבור $t = 4$ אנחנו בטוח נוכל לקבל אילוצי SMER_4 אשר יאכפו את אילוץ SoD המקורי, מכיוון שמתקיים: $8 > (3-1)(4-1)$. האם ניתן לאכוף את ה-SoD עם אילוצי SMER_5 אי אפשר לדעת רק על סמך משפט 1 כי $8 \not> (3-1)(5-1)$ ולשם כך נדרש חישוב הרבה יותר יקר מבחינת הזמן.

4.4 אלגוריתמי כריית תפקידים מונחי SoD

אלגוריתמים לאכיפת אילוצי SoD ע״י חישוב אילוצי SMER_2 ו-$\mathrm{SMER}_{\mathrm{t}}$ פותרים את הבעיה של העלות החישובית הגבוהה של הפתרון הנאיבי. אבל מכיוון שהאלגוריתם הוא $\mathrm{post}-\mathrm{processing}$, ז״א אילוצים מחושבים אחרי שמתקבלות המטריצות PA ו-UA, כפי שראינו, יכול להיות שהאילוצים אינם ישימים מכיוון שהם מופרים ע״י מטריצת UA שהתקבלה מקודם.

הדרך להתגבר על הפער הזה היא להתחשב באילוצים בזמן כריית תפקידים ובניית UA. אם היינו משתמשים בפתרון הנאיבי, בשלב הראשון היינו מחפשים קבוצה מינימאלית של תפקידים שביחד מקבלים n הרשאות שתואמות את אילוץ $\mathrm{SoD}_{\mathrm{n,k}}$ על מנת לייצר אילוצי RSSoD מתאימים. בהתחלה מתוך רשימת התפקידים מוצאים קבוצה של כל התפקידים שמקבלים לפחות הרשאה אחת מתוך קבוצה של n הרשאות. אבל כל התפקידים לא יכולים להיכנס לקבוצה מינימאלית. התפקידים שלא נכללים בקבוצה מינימאלית הם בעלי הרשאות שמכוסות ע״י התפקידים בקבוצה מינימאלית. מציאה של הקבוצה המינימאלית שעונה על הדרישות

היא משימה יקרה מאוד מבחינה חישובית. אם לדוגמה נבחר תפקיד r_i שנכלל בקבוצה מינימאלית ותפקיד r_j שלא נכלל, אזי יש שלוש אפשרויות ליחס בין ההרשאות שלהם:

1. r_i ו-r_j הם בעלי אותם הרשאות, ונדרש ליצור שני אילוצי RSSoD, אילוץ אחד עבור r_i ושני עבור r_j.
2. הרשאות של r_j הן תת-קבוצה של הרשאות של r_i
3. הרשאות של r_j כבר מכוסות ע"י תפקידים אחרים בקבוצת התפקידים המינימאלית

אבל אם מדובר בפתרון בעיית כריית התפקידים עם אילוצים, זאת אומרת שקבוצת התפקידים לא נכפת על ידי קלט אלא אנחנו בונים אותה במהלך האלגוריתם, אזי ניתן להימנע משלושת המקרים האלה במהלך כריית התפקידים ובכך למזער משמעותית את עלות החישוב של הקבוצות המינימאליות של תפקידים. נתייחס אל כל אחת מהאפשרויות של היחס בין התפקידים כתנאי שאם הוא מתקיים, אזי התפקיד החדש לא יתווסף ל-R כמו שהוא, אלא נדרש לעשות בו שינוי.

כל אחד מהאלגוריתמים שפותרים RMP ניתן להרחיב בצורה מתאימה על מנת שהאלגוריתם יתחשב בתנאים האלה. בפרקים קודמים של העבודה תיארנו אלגוריתם לפתרון בעיית $\delta - \text{Approx RMP}$ על ידי חישוב שטחים שתפקיד מכסה במטריצת UPA. עכשיו נוכל לבחון האם האלגוריתם הזה הוא $SoD\ aware$ מבחינת התנאים האלה.

תנאי 1- אם במהלך האלגוריתם מתקבל r_j כך שהרשאות שלו תואמות לגמרי את ההרשאות של r_i, אזי השטח שעוד לא מכוסה על ידי r_j ב-UPA הוא 0 ($carea = 0$), ולכן תפקיד כזה לא יכנס ל-R בכלל, ולא ידרוש יצירת RSSoD.

תנאי 2 - במידה והרשאות של r_j הן תת-קבוצה של קבוצת הרשאות r_i, אזי יקרו שני דברים. r_i יימצא קודם על ידי האלגוריתם, מכיוון שיש לו יותר הרשאות, ולכן שטח פוטנציאלי שהוא יכול לכסות ב-UPA הוא גדול יותר. אחרי זה כאשר יימצא תפקיד r_j כמו מקודם השטח הלא מכוסה על ידו יהיה 0 ו-r_j לא יכנס ל-R ולא ידרוש אילוצי RSSoD.

תנאי 3 - במידה והרשאות של r_j מכוסות ב-UPA על ידי תפקידים אחרים, אזי r_j לא יכנס ל-R ולא יידרשו אילוצי RSSoD עבורו.

כפי שניתן לראות, אלגוריתם לפתרון $\delta - \text{Approx RMP}$ הוא SoD aware מהבחינה הזאת. המשמעות היא שניתן להשתמש בו בתהליך כריית התפקידים עם אילוצי SoD בשלב ראשון (בניית UA ו-PA) ואחרי זה להשתמש באחד מהאלגוריתם של $\text{post} - \text{processing}$.

עבור הפתרון הנאיבי מציאת קבוצה מינימאלית של התפקידים תהיה בזמן ליניארי, מכיוון שלא יהיו תפקידים עם הרשאות חופפות כמו שתואר בתנאים לעיל. ועבור אלגוריתמי $SMER_2$ ו-$SMER_t$ המטריצה UA לא תפר את האילוצי SMER שיתקבלו.

4.5 מטריצת UPA אינה עקבית עם אילוצי SoD

הדיון הקודם נכון רק כאשר מטריצת UPA בקלט אינה מפרה את אילוצי SoD. מטריצת UPA יכולה להפר את האילוצי SoD בשני המקרים הבאים: מטריצת UPA מכילה שגיאות או אילוצי SoD הינם דינאמיים. נבחן כל אחד משני המקרים האלה ודרכי התמודדות איתם בנפרד.

מטריצת UPA עם שגיאות

כפי שראינו קודם, לפעמים מטריצת הקלט UPA מכילה שגיאות מסוגים שונים. כאשר במטריצה יש שגיאות מתווספות, ז״א משתמש מקבל הרשאות יתר, יכול לקרות מצב שמשתמש קיבל הרשאות סותרות בטעות. בדרך כלל השגיאות האלה יהיו נקודתיות, במספר מועט של משתמשים, ולכן התפקידים הפוטנציאליים עם הרשאות סותרות יקבלו תיעדוף מאוד נמוך וישתייכו למשתמשים בודדים. במקרה כזה מנהלי אבטחת מידע בארגון יכולים בסוף תהליך כריית התפקידים לשים לב למשתמשים עם התפקידים החריגים ולבצע תיקון ידני. כמו כן, במקרה הזה אלגוריתם לבעיית $\delta - \text{Approx RMP}$ עם $\delta \neq 0$ מתאים יכול לסנן את התפקידים כאלה בכלל ולהשאיר את הטיפול במשתמשים האלה למנהלי אבטחת מידע.

אילוצי SoD דינאמיים

מטריצת UPA נבנית לאורך הזמן ע״י ניטור הרשאות שמשתמשים מקבלים. במידה ובארגון קיימים אילוצי SoD מסוג דינאמי, אזי משתמש u יכול לקבל לפרק זמן מסוים קבוצת ההרשאות P_1 ולפרק זמן אחר P_2. גם אם $P_1 \cup P_2$ מכילה הרשאות סותרות, הדבר אינו מפר אילוצי SoD מכיוון שמשתמש קיבל את ההרשאות הסותרות ב-sessions שונים. אבל במטריצת UPA, שאין בה התייחסות לזמן, $P_1 \cup P_2$ יהיה סט הרשאות הדרוש עבור המשתמש, במילים אחרות קבוצת הרשאות $P_1 \cup P_2$ תהיה מזוהה כתפקיד פוטנציאלי.

אם נחזור לדוגמה קודמת:

$$U = \{u_1, u_2, u_3, u_4, u_5\}$$

$$P = \{p_1, p_2, p_3, p_4, p_5, p_6\}$$

$$UPA = \begin{array}{c|cccccc} & p_1 & p_2 & p_3 & p_4 & p_5 & p_6 \\ u_1 & 0 & 1 & 0 & 0 & 1 & 0 \\ u_2 & 0 & 1 & 0 & 0 & 1 & 0 \\ u_3 & 1 & 1 & 0 & 1 & 1 & 0 \\ u_4 & 1 & 1 & 1 & 0 & 0 & 0 \\ u_5 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}$$

יכול להיות שיש אילוץ $e = sod < \{p_2, p_3\}, 2 >$ ומשתמש u_4 קיבל את ההרשאות p_2 ו-p_3 ב-sessions שונים. בתהליך מציאת התפקידים יזוהה התפקיד $r_1 = \{p_1, p_2, p_3\}$ ובפועל יתקבל תפקיד שמכיל הרשאות סותרות וזה לא יאפשר לאכוף את האילוץ במודל RBAC שיתקבל.

על מנת להתמודד על הבעיה הזאת נדרש להתאים את אלגוריתם כריית התפקידים שראינו מקודם ($\delta - \text{Approx RMP}$) כך שיאפשר לפתור בעיית RMP שכוללת את אילוצי SoD. חשוב לשים לב שהאלגוריתם לא יתמודד עם אילוצי SoD סטטיים, ז״א משתמש לא יכול לקבל אף פעם תפקיד עם הרשאות הסותרות את ההרשאות שהוא קיבל קודם. במקרה הזה המטריצה UPA לא תכיל משתמשים עם הרשאות סותרות וכל התפקידים שיתקבלו יהיו מתואמים לאילוצי SoD סטטיים.

כפי שראינו יש שתי שיטות להתאים את האלגוריתם להתמודדות עם SoD. השיטה הראשונה היא הוספת שלב נוסף לבניית אילוצים מתאימים אחרי שמתקבלות מטריצות UA ו-PA (post processing), והשיטה השנייה היא לבצע התאמות בתהליך כריית התפקידים עצמם (SoD aware). במידה ונרצה להשתמש בשיטה הראשונה להתאמת $\delta - \text{Approx RMP}$, אנחנו יכולים לקבל תפקידים שמכילים הרשאות סותרות עוד בשלב בניית UA, לכן לא נוכל לאכוף אותם בהמשך. לכן נשתמש רק בשיטה השנייה.

השלב הראשון של $\delta - \text{Approx RMP}$ הוא להריץ את FastMiner על מנת לקבל תפקידים פוטנציאליים. כפי שראינו קודם, FM קודם עובר על כל שורה ב-UPA ומבצע איחוד משתמשים עם הרשאות זהות, על מנת לקבל את קבוצת InitRoles. אם ב-UPA יש משתמש עם הרשאות סותרות, אזי כבר ב-InitRoles נקבל תפקידים בעייתיים. לכן לפני שנתקדם לשלב של מציאת חיתוכים נדרש לפצל את התפקידים כך שלא יכילו הרשאות סותרות. אם נשתמש בדוגמה קודמת אזי נקבל

$$\text{InitRoles} = (\{p_2, p_5\}, \{p_1, p_2, p_4, p_5\}, \{p_1, p_2, p_3\}, \{p_6\})$$

אם נניח קיים אילוץ $e = sod < \{p_2, p_3\}, 2 >$, אזי התפקיד הפוטנציאלי $\{p_1, p_2, p_3\}$ לא חוקי, במובן שלא מאפשר לאכוף את האילוצים. קיימות שלוש אפשרויות לפיצול תפקידים פוטנציאליים לתפקידים מותאמים ל-SoD.

אפשרות ראשונה לפיצול היא לצרף את ההרשאה p_1, שהיא בעצמה לא סותרת אף הרשאה, לאחת ההרשאות הסותרות ולבדל הרשאה סותרת אחרת. לפיצול כזה יש שתי אפשרויות:

$$\{p_1, p_2\} \cup \{p_3\} \ \vee \{p_1, p_3\} \cup \{p_2\}$$

פיצול כזה לא טוב, מכיוון שאנחנו לא יכולים לדעת עם איזה הרשאה בצימוד אמור להיות p_1 , ולכן בסופו של דבר אנחנו יכולים לקבל תפקידים חסרי משמעות. אם לדוגמה הרשאה p_1 חייבת לבוא ביחד עם p_2 ו-p_3

היא הרשאה עצמאית, אזי פיצול $\{p_1, p_3\} \cup \{p_2\}$ יגרום למצב שמשתמש לא יכול לקבל הרשאות p_1 ו-p_2 ביחד, מכיוון שהם שייכים לתפקידים סותרים.

האפשרות השנייה לפיצול היא להצמיד לכל הרשאה סותרת את ההרשאות לא סותרות, בדוגמה שלנו נקבל:

$$\{p_1, p_2\} \cup \{p_1, p_3\}$$

פיצול כזה הוא לגיטימי, אך הוא יכול להביא למצב שחלק מהתפקידים מקבלים הרשאות יתר, אמנם לא הרשאות סותרות, אבל לא נחוצות למשתמש בעת ביצוע משימה. נניח שבעת ביצוע משימה שדורשת הרשאה p_3 משתמש לא זקוק להרשאה p_1, עם זאת התפקיד שהוא יקבל יכלול גם את ההרשאה p_1, מכיוון שהפיצול לא מאפשר תפקידים עם קבוצת הרשאות במדויקת יותר.

אפשרות שלישית של פיצול היא לאחד את כל ההרשאות הלא סותרות לקבוצה נפרדת, ועבור כל הרשאה סותרת ליצור קבוצה נפרדת. בדוגמה שלנו נקבל:

$$\{p_1\} \cup \{p_2\} \cup \{p_3\}$$

הפיצול הזה הוא הכי טוב. אמנם אנחנו מגדילים את מספר התפקידים הסופי בצורה מקסימאלית, אבל הדרך הזאת מאפשרת להתאים תפקידים למשתמשים בצורה מיטבית.

באיור 14 מופיע פסאודוקוד של שלב ביניים לטיפול בקבוצת InitRoles. קבוצת SoDInitRoles שמתקבלת בפלט יכולה שוב להכיל כפילויות, מכיוון שחלק מהתפקידים ב-InitRoles יכולים להכיל אותן הרשאות ואחרי הפיצול נקבל תפקידים פוטנציאליים זהים. לכן בסוף השלב הזה צריך לעבור על הרשימה ולהוריד כפילויות שוב. בתום ניקוי כפילויות, הרשימה SoDInitRoles יכולה לשמש להמשך האלגוריתם FM במקום InitRoles. קל לראות שבהמשך, בשלב מציאת חיתוכים, לא ייווצרו תפקידים שכוללים הרשאות סותרות, מכיוון שאף תפקיד פוטנציאלי לא יכיל כאלה.

בניתוח סיבוכיות של אלגוריתמים CompliteMiner ו-FastMiner החסם העליון על מספר התפקידים ב-InitRoles היה מספר משתמשים - n, מכיוון שבמקרה הגרוע כל משתמש היה מקבל תפקיד משלו, אך לא היה נוצר תפקיד שלא שייך לאף משתמש. במקרה של פיצול, עבור כל שורה (משתמש) ב-UPA יכולים להתקבל מספר תפקידים, והחסם העליון על מספר התפקידים הוא מספר הרשאות m ב-UPA. לכן סיבוכיות הכללית של FM תשתנה מ-$O(n^2)$ ל-$O(m^2)$.

Input: $InitRoles, E$

Output: $SoDInitRoles$

1: $SoDInitRoles \leftarrow \{\emptyset\}$
2: **for** each constraint $e \in E$ **do**
3: **for** each role $r \in InitRoles$ **do**
4: **if** $e.perms \cap r \neq \emptyset$ **then**
5: $r' = r \backslash e.perms$
6: $SoDInitRoles = SoDInitRoles \cup r'$
7: **for** each $p \in e.perms$ **do**
8: $SoDInitRoles = SoDInitRoles \cup p$
9: **end for**
10: **else**
11: $SoDInitRoles = SoDInitRoles \cup r$
12: **end if**
13: **end for**
14: **end for**

איור 8. פסאודוקוד של $\delta - Approx\ RMP\ SoD\ aware\ routine$

5. סיכום

כפי שראינו מודל RBAC הוא מענה פשוט וגמיש שמאפשר לענות על רב הצרכים של ארגונים בהיבטים של בקרת גישה ואבטחת המידע. ההרחבות של המודל בסיסי (היררכיה של תפקידים ואילוצים) מאפשרות להתאים את המודל למבנה ארגוני ולאכוף מדיניות אבטחת מידע של מערכות הארגון.

האתגר העקרי שמציב מעבר למודל RBAC הוא בניית המודל המתאים והגדרה של קבוצת תפקידים. בזמן שקבוצת משתמשים והרשאות הן נתון של המערכת מידע עצמה, קבוצת התפקידים היא חלק "חיצוני" של המודל שלא מוגדר מראש. הצגנו בהרחבה את הבעיה RMP ופתרונות ל-"כריית תפקידים" אשר מאפשרים לבנות טיוטה עבור רשימת התפקידים, כאשר הטיוטה עצמה נגזרת מהמערכות של הארגון ומקשרים שקיימים בין המשתמשים והיסטוריית הרשאות שהם קיבלו בעבר.

אלגוריתמים פשוטים יחסית כגון CompleteMiner ו-FastMiner מוצאים את הרשימה של התפקידים הפוטנציאלית והם יכולים להוות בסיס טוב לאלגוריתמים לכריית התפקידים. האלגוריתמים האלה ניתנים להתאמה פשוטה גם למודל שכולל אילוצים. סקרנו מספר פתרונות שקיימים בספרות להתאמה לאילוצים סטטיים ותארנו איך ניתן להתאים את CompleteMiner ו-FastMiner גם לאילוצים דינאמיים.

אמנם תהליך כריית תפקידים לא תמיד יספק מודל RBAC מוכנה להטמעה מיידית. אבל הוא יכול לספק טיוטה שתאפשר לאנשי אבטחת מידע של הארגון בקלות לבנות מודל RBAC מותאם לארגון.

רשימת מקורות

1. Sandhu, R.S., 1998. Role-based access control. In *Advances in computers* (Vol. 46, pp. 237-286). Elsevier.
2. Crampton, J., 2003, June. Specifying and enforcing constraints in role-based access control. In *Proceedings of the Eighth ACM symposium on Access control models and Technologies* (pp. 43-50).
3. Mitra, B., Sural, S., Vaidya, J. and Atluri, V., 2016. A survey of role mining. *ACM Computing Surveys (CSUR)*, *48*(4), pp.1-37.
4. Vaidya, J., Atluri, V., Warner, J. and Guo, Q., 2008. Role engineering via prioritized subset enumeration. *IEEE Transactions on Dependable and Secure Computing*, *7*(3), pp.300-314.
5. Vaidya, J., Atluri, V. and Guo, Q., 2010. The role mining problem: A formal perspective. *ACM Transactions on Information and System Security (TISSEC)*, *13*(3), pp.1-31.
6. Sarana, P., Roy, A., Sural, S., Vaidya, J. and Atluri, V., 2015. Role mining in the presence of separation of duty constraints. In *Information Systems Security: 11th International Conference, ICISS 2015, Kolkata, India, December 16-20, 2015. Proceedings 11* (pp. 98-117). Springer International Publishing.

Abstract

Access control models based on roles (RBAC) are widely used for implementing information security in various information systems. This model enables the construction of a role structure that corresponds to the organization's hierarchy, simplifying the implementation of information security policies in a straightforward and intuitive manner. In the first part of the work (Chapters 1, 2), we extensively review the RBAC model, including its extensions such as role hierarchies and constraints.

One of the significant challenges in transitioning to the RBAC model in existing information systems is the construction of an appropriate role structure. One solution for building a role structure is to use role mining algorithms. The second part of the work (Chapter 3) is dedicated to the formal description of two problems that define the role mining problem and presents a solution based on the $FastMiner$ algorithm.

The solution presented in the second part is suitable for a basic RBAC model without constraints. In the final part of the work (Chapter 4), we delve into the expansion of the solution for role mining suitable for a model that includes static or dynamic constraints. This chapter will describe two methods for dealing with static constraints. Additionally, we will describe an improvement to $FastMiner$ that allows obtaining a role structure adapted to a model that includes dynamic constraints on roles.

Table of content

List of figures 1
Abstract 2
1. Model RBAC 3
1.1 Motivation and Basic defenitions 3
1.2 Comparing with other access control models 5
2. Defenition of RBAC and extensions 7
2.1 Model $RBAC_0$ 8
2.2 Model $RBAC_1$ 11
2.3 Model $RBAC_2$ 14
2.3.1 Common examples of constraints in RBAC 14
2.3.2 Constraints specification 16
2.3.3 Constraints enforcing model 19
2.4 Model $RBAC_3$ 21
3. Role Mining Problem 23
3.1 Basic RMP and $\delta -$ Approx RMP 24
3.2 Role Mining via Subset Priortization 28
3.2.1 Principles of Role Mining Algorithms 28
3.2.2 CompliteMinerAlgorithm 29
3.2.3 FastMinerAlgorithm 36
3.3 Heuristic approach for Basic RMP and $\delta -$ Approx RMP 41
4. Role Mining given Constaints 48
4.1 Introduction and formal defenitions 48
4.2 Problem RMP $-$ SoD 50
4.3 $SMER$ based algorith with post-processing approach 53
4.4 SoD driven algorithm for Role Mining 57
4.5 UPA matrix is inconsistent with SoD 59
5. Summary 63
Biblioagraphy 64

The Open University of Israel
Department of Mathematics and Computer Science

Role Mining given Separation of Duty Constraints

Final Paper submitted as partial fulfillment of the requirements
towards an M.Sc. degree in Computer Science
The Open University of Israel
Department of Mathematics and Computer Science

By
Vladimir Ponomarev

Prepared under the supervision of Prof. Tamir Tassa

March 2024