ارائه یک پایگاه داده توزیع شده خود اصلاح با استفاده از عامل های نرم افزاری

امین مرادی، دانشجوی کارشناسی ارشد، دانشگاه آزاد اسلامی واحد نجف آباد
amin7moradi@gmail.com
محمد داورپناه جزی، استادیار، دانشگاه صنعتی اصفهان
محمد علی منتظری، استادیار، دانشگاه صنعتی اصفهان

چکیده:

امروزه با وجود در نظر گرفتن راهکارهای مختلف برای ایجاد پایگاه داده‌های امن، حمله‌های موفق صورت گرفته به این پایگاه داده-ها نشان داده است که همواره امکان شکست راهکارهای پیشگیری از حمله وجود دارد؛ بنابراین بازیابی داده‌های آسیب دیده در اثر اجرای تراکنش‌های ناشی از یک حمله به مقادیر صحیح پیش از حمله یکی از مراحل مهم در جهت حفظ جامعیت و دسترس پذیری یک پایگاه داده امن می‌باشد. به پایگاه داده‌هایی با قابلیت بازیابی خودکار داده‌های آسیب دیده، پایگاه داده‌های خود اصلاح می‌گویند. از آنجاییکه اغلب تلاش‌های صورت گرفته در زمینه طراحی پایگاه داده‌های خود اصلاح مربوط به پایگاه داده‌های متمرکز می باشد ما در این مقاله با استفاده از قابلیت‌های عامل‌های نرم افزاری روشی برای طراحی پایگاه داده‌های توزیع شده خود اصلاح ارائه کرده‌ایم که ضمن در نظر گرفتن پیچیدگی‌های پایگاه داده‌های توزیع شده، داده‌های آسیب دیده در اثر حمله به پایگاه داده را شناسایی کرده و به صورت خودکار آنها را به مقدار صحیح پیش از حمله بر می‌گرداند. روش ارائه شده فرآیندی توزیع شده می باشد که نقطه شکست منفردی در آن وجود نداشته و در برابر مشکلات احتمالی محیط‌های توزیع شده مانند قطعی خطوط ارتباطی یا گم شدن پیام‌ها پایدار می‌باشد.

واژه‌های کلیدی: امنیت، پایگاه داده توزیع شده خود اصلاح، عامل نرم افزاری.

۱. مقدمه

یکی از اصول اساسی در طراحی و پیاده سازی سیستم‌های مدیریت پایگاه داده[1](DBMS) که وظیفه‌ی ذخیره سازی داده‌های حساس را بر عهده دارند فراهم دیدن امنیت داده‌های ذخیره شده در آن پایگاه داده‌ها می‌باشد. حملاتی که تاکنون نسبت به DBMS ها در سراسر جهان رخ داده است نشان داده که استفاده از روش‌های سنتی تأمین امنیت پایگاه داده‌ها از قبیل سیستم احراز هویت، تأمین اعتبار کاربران و رمز نگاری داده‌های ذخیره شده در پایگاه داده نمی‌تواند در اختیار داشتن پایگاه داده ای کاملاً ایمن را تضمین کند و امکان شکست این راهکارهای پیشگیری از حمله همواره وجود دارد[۷]. در واقع فرآیند تضمین امنیت یک DBMS شامل سه مرحله‌ی اصلی است[۴] که تأمین امنیت داده‌های ذخیره شده در پایگاه داده از طریق روش‌های سنتی ذکر شده مرحله اول این فرآیند است. مرحله‌ی دوم عبارت است از شناسایی و کشف حملات صورت گرفته به پایگاه داده توسط سیستم‌های کشف نفوذ[2] (IDS). پس از آن که بروز حمله ای به پایگاه داده شناسایی شد نوبت به مرحله سوم فرآیند تأمین امنیت پایگاه داده می‌رسد؛ در این مرحله داده‌های ذخیره شده در پایگاه داده که طی حمله‌ی شناسایی شده توسط IDS آسیب دیده اند مشخص شده و هر کدام از آنها به حالت سالم قبل از بروز آن حمله بازگردانده می‌شود تا صحت و جامعیت آن داده‌ها و در نتیجه صحت و جامعیت کل پایگاه داده برقرار شود، این مرحله را فرآیند کشف و بازیابی آسیب‌ها[3](DAR) می‌نامیم. پس از کامل شدن مرحله‌ی سوم پایگاه داده به یک حالت پایدار رسیده و می‌تواند به کاربرانش خدمت رسانی کند. به سیستم پایگاه داده‌ای که هرسه مرحله‌ی فرآیند تأمین امنیت را به طور کامل و کارا اجرا کند پایگاه داده‌ی خود اصلاح گویند.

تا کنون چندین روش برای اجرای فرآیند DAR در پایگاه داده‌ها معرفی شده اند که از آن جمله می‌توان به تلاش‌های صورت گرفته در مراجع[۳،۹و۱۰] اشاره کرد؛ با این وجود اغلب این روش‌ها برای پایگاه داده‌های متمرکز ارائه شده‌اند و تعداد کمی از آنها برای استفاده در سیستم‌های پایگاه داده‌ی توزیع شده[4](DDBS) عملی هستند، در واقع اجرای فرآیند DAR برای یک سیستم پایگاه داده متمرکز ساده تر از یک سیستم پایگاه داده توزیع شده است چرا که در DDBSها به دلیل وجود تراکنش‌های توزیع شده که در میزبان‌های مختلف زیر تراکنش‌هایی دارند، نمی‌توان عمل کشف آسیب را در هر میزبان بصورت محلی انجام داد بلکه باید روابط بین تراکنش‌های عمومی در میزبان‌های مختلف را نیز در نظر گرفت[۵]. در مراجع [۱،۲و۱۲] راهکارهایی برای اجرای فرآیند DAR در DDBS ها ارائه شده است. به دلیل اینکه اساس راهکارهای ارائه شده در مراجع [۲و۱۲] فرآیندهایی با کنترل متمرکز است، این راهکارها دارای مشکلاتی از قبیل وجود نقطه تکی شکست و بار پردازشی زیاد بر روی هماهنگ کننده مرکزی هستند اما مشکل راهکار ارائه شده در مرجع[۱] که بر اساس الگوریتمی توزیع شده می‌باشد ارتباطات شبکه‌ای بسیار و در نتیجه عدم کارایی می‌باشد.

امروزه استفاده از برنامه نویسی عامل گرا جهت ایجاد برنامه‌های پیچیده گسترش روزافزونی یافته است. عامل‌ها را می‌توان بر اساس توانایی‌هایشان از قبیل خود مختاری، هدف گرایی، قابلیت انطباق، پیش فعال بودن، هوشمندی و استنتاج مشخص کرد[۱۱]. با استفاده از این توانایی‌ها می‌توان مسائل پیچیده موجود در محیط‌های پویا و توزیع شده را به خوبی کنترل کرد. امروزه بسیاری از

1. Database Management Systems
2. Intrusion Detection Systems
3. Damage Assessment and Recovery
4. Distributed Database Systems

کاربردهای موجود در حوزه‌هایی مانند امنیت شبکه، تجارت الکترونیک، ارتباطات و دیگر حوزه‌های فناوری اطلاعات بوسیله سیستم‌های چند عامله پیاده سازی شده اند. در این مقاله ابتدا الگوریتم‌هایی جهت استفاده در فرآیند کشف و بازیابی آسیب‌ها ارئه داده‌ایم که ضمن برخورداری از بار پردازشی مناسب نیازمند ارتباطات شبکه ای محدودی می‌باشند، سپس با استفاده از عامل‌های نرم افزاری راهکار ارائه شده را به گونه ای گسترش می‌دهیم که با در نظر گرفتن مشکلات احتمالی محیط‌های توزیع شده در برابر بروز این مشکلات پایدار بوده، فرآیند توزیع بار پردازشی را به خوبی انجام دهد.

۲. پیش زمینه‌ها و تعاریف

یک پایگاه داده‌ی توزیع شده عبارت است از مجموعه ای از پایگاه داده‌های محلی که بر روی میزبان‌های مختلف قرار داشته و از طریق شبکه ارتباطی به یکدیگر دسترسی دارند[۸]. یک تراکنش تشکیل شده است از مجموعه ای از اعمال خواندن و نوشتن بر روی داده‌های ذخیره شده در پایگاه داده که به عنوان یک فرآیند تجزیه ناپذیر[۵] در نظر گرفته می‌شوند. در یک پایگاه داده‌ی توزیع شده یک تراکنش عمومی چندین زیر تراکنش دارد که در میزبان‌های مختلف آن سیستم اجرا می‌شوند. اگر زیر تراکنشی از یک تراکنش عمومی به عنوان تراکنش مخرب شناسایی شود کل آن تراکنش عمومی تراکنشی مخرب در نظر گرفته شده و باید تأثیر همه‌ی زیر تراکنش‌های آن در میزبان‌های مختلف از بین برده شود[۱۲]. در ادامه منظور از نماد T_i تراکنش عمومی i و منظور از نماد $T_{j,i}$ یکی از تراکنش‌های محلی T_i است که بر روی میزبان j اجرا می‌شود. با توجه به نکات ذکر شده تعاریف رسمی از وابستگی تراکنش‌ها در ادامه می‌آید.

تعریف ۱) زیر تراکنش $T_{k,j}$ وابسته است به زیر تراکنش $T_{k,i}$ اگر:

۱- داده ای مانند x وجود دارد به طوریکه پس از آنکه $T_{k,i}$ آن را بروزرسانی[۶] کرد، $T_{k,j}$ آن داده را بخواند.

۲- قبل از آنکه $T_{k,j}$ آیتم x را بخواند $T_{k,i}$ شکست نخورده است (abort نشده است).

۳- قبل از آنکه آیتم x توسط $T_{k,j}$ خوانده شود، $T_{k,i}$ آخرین تراکنش موفقی (commit شده) است که آن را بروزرسانی کرده است.

در صورت وجود شرایط بالا رابطه‌ی وابستگی بین این دو زیر تراکنش بصورت $T_{k,j} \rightarrow T_{k,i}$ نشان داده می‌شود. این شرایط یک وابستگی مستقیم بین دو زیر تراکنش را تعریف می‌کند اما امکان وجود وابستگی غیر مستقیم بین دو زیر تراکنش نیز وجود دارد چرا که رابطه‌ی وابستگی یک رابطه‌ی متعدی است، در واقع اگر داشته باشیم $T_{k,j} \rightarrow T_{k,t}$ و $T_{k,i} \rightarrow T_{k,j}$ آنگاه خواهیم داشت $T_{k,i} \rightarrow T_{k,t}$.

تعریف ۲) تراکنش عمومی T_p وابسته است به تراکنش عمومی T_t اگر زیرتراکنشی از T_p به زیر تراکنشی از T_t به صورت مستقیم یا غیرمستقیم وابسته باشد. این وابستگی بصورت $T_p \rightarrow T_t$ نشان داده می‌شود.

5. Atomic
6. Update

تعریف ۳) زیر تراکنش $T_{k,i}$ متاثر[7] می‌کند $T_{k,j}$ را اگر اولاً $T_{k,j}$ وابسته باشد به $T_{k,i}$، ثانیاً $T_{k,i}$ مخرب باشد و در نهایت آنکه هم $T_{k,i}$ و هم $T_{k,j}$ با موفقیت به پایان رسیده باشند (commit شده باشند).

تعریف ۴) یک گراف وابستگی تراکنش محلی[8] در یک میزبان وابستگی‌های تراکنش‌ها در آن میزبان رانشان می‌دهد. هر گره یک زیر تراکنش و هر یال نشان دهنده‌ی ارتباط «وابستگی» بین دو گره‌ی آن یال می‌باشد.

گراف وابستگی محلی برای زیر تراکنش‌های یک میزبان را می‌توان براساس فایل ثبت وقایع[9] موجود در آن میزبان ایجاد کرد. باید توجه داشت که برای ساخت این گراف با استفاده از فایل ثبت وقایع علاوه بر ثبت اعمال نوشتن (write) نیاز داریم که اعمال خواندن (read) اجرا شده در آن میزبان نیز ثبت شوند و به همین دلیل در راهکار ارائه شده در این مقاله فرض شده که فایل ثبت وقایع علاوه بر اعمال نوشتن شامل اعمال خواندن نیز می‌باشد[۶]. با ترکیب گراف‌های وابستگی محلی همه‌ی میزبان‌های عضو DDBS گراف وابستگی تراکنش عمومی آن DDBS ساخته می‌شود که نمایانگر روابط وابستگی بین تراکنش‌های عمومی در آن پایگاه داده توزیع شده می‌باشد.

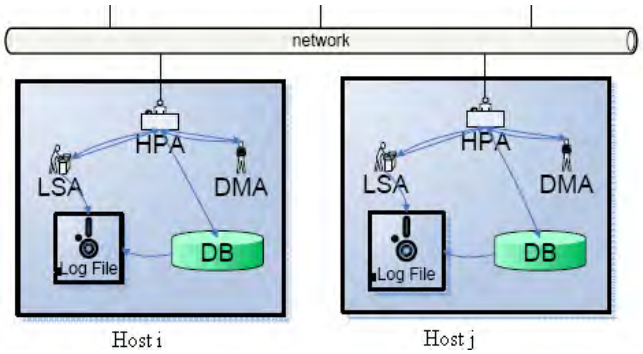
۳. راهکار ارائه شده

اساس راهکار ارائه شده تشکیل گراف وابستگی محلی در هر میزبان عضو پایگاه داده توزیع شده و سپس ترکیب دو بدوی آنها در یک ساختار درختی جهت ایجاد گراف وابستگی عمومی برای کل پایگاه داده توزیع شده می‌باشد. پس از کامل شدن گراف وابستگی عمومی و با استفاده از آن تمامی تراکنش‌های متاثر شناسایی می‌شوند. در پایان شناسایی و اصلاح داده‌های خراب شده در اثر اجرای مجموعه تراکنش‌های مخرب شامل تراکنش‌های مهاجم و تراکنش‌های متاثر بصورت مستقل در هر میزبان انجام می‌گردد.

برای پیاده سازی مدل ارائه شده در هر میزبان DDBS سه عامل بنام‌های Host Proxy Agent یا HPA، Dependency Manager Agent یا DMA و Log Scanner Agent یا LSA مستقر می‌شوند که در طی فرآیند DAR با یکدیگر و نیز با عامل‌های مستقر بر روی دیگر میزبان‌های DDBS همکاری می کنند. HPA عاملی است که در طی فرآیند DAR در هر میزبان از طریق ارسال و دریافت پیام با میزبان‌های دیگر تعامل دارد. علاوه بر این HPA مسئول کنترل و هماهنگی فعالیت‌های DMA و LSA محلی نیز می باشد. LSA وظیفه‌ی ساخت و بروز رسانی گراف وابستگی محلی زیر تراکنش‌ها در هر میزبان را بر عهده دارد. گراف وابستگی محلی در هر میزبان با استفاده از فایل ثبت وقایع آن میزبان و بر اساس الگوریتم تشکیل گراف وابستگی که در بخش ۳.۱ ارائه شده است ساخته می شود. طبق این الگوریتم LSA در هر میزبان ضمن بررسی فایل ثبت وقایع آن میزبان، ارتباطات موجود بین زیر تراکنش‌های آن میزبان را استخراج کرده و گراف وابستگی محلی در آن میزبان را می‌سازد. مسئولیت دیگر LSA ایجاد تراکنش‌های پاکسازی در جهت رفع اثر حملات صورت گرفته به پایگاه داده بر اساس الگوریتم ایجاد تراکنش‌های پاکسازی که در بخش ۳.۲ ارائه کرده ایم می‌باشد. با استفاده از این الگوریتم داده‌های آسیب دیده در اثر اجرای تراکنش‌های مخرب شناسایی شده و با استفاده از فایل ثبت وقایع تراکنش‌های پاکسازی جهت بازیابی مقدار آنها به حالت پیش از حمله ایجاد

7. Affect
8. Local Transaction Dependency Graph
9. Log File

می‌شود. DMA در طی فرآیند تشکیل گراف وابستگی عمومی مسئولیت ترکیب گراف‌های وابستگی محلی را بر عهده دارد. ساختار یک میزبان DDBS در راهکار ارائه شده مطابق شکل (۲) می باشد.



شکل (۱): ساختار میزبان‌های DDBS و عامل‌های مستقر بر روی آنها در راهکار پیشنهادی برای اجرای فرآیند DAR

۳.۱. الگوریتم تشکیل گراف وابستگی محلی

هدف از این الگوریتم ایجاد گراف وابستگی محلی در یک میزبان با بررسی فایل ثبت وقایع در آن میزبان می‌باشد. منظور از $W[T_i,x,v1,v2]$ در فایل ثبت وقایع یک میزبان بروزرسانی مقدار x از v1 به v2 توسط زیر تراکنشی از تراکنش عمومی T_i که در این میزبان اجرا شده است و منظور از $R[T_i,x]$ خوانده شدن مقدار x توسط زیر تراکنشی از T_i است. برای ارزیابی الگوریتم تشکیل گراف وابستگی و الگوریتم ایجاد تراکنش‌های پاکسازی که در بخش بعدی آمده است فرض می‌کنیم در یک میزبان خاص فایل ثبت وقایع از محل شروع بررسی تا انتها دارای n عنصر باشد، m تراکنش محلی در آن میزبان در حال اجرا باشند که حداکثر تعداد عملیات نوشتن توسط هر یک از آنها k باشد. در چنین شرایطی پیچیدگی هر سطر از الگوریتم مقابل آن نوشته شده است؛ سطرهایی که پیچیدگی آنها ذکر نشده است پیچیدگی از مرتبه ثابت دارند.

ورودی‌ها: فایل ثبت وقایع و لیست تراکنش‌های بداندیش (Bad_Transaction_List).

خروجی: آرایه Parents که $Parents[T_i]$ نشان دهنده تراکنش‌هایی است که تراکنش T_i به آنها وابسته است.

مقداردهی اولیه:

TempWirtedT = WriteItems = Parents = p = NULL,

Destroyer_Transaction_List = Bad_Transaction_List.

۱. با مقایسه محل اولین عملیات تراکنش‌های عضو Bad_Transaction_List زودترین نقطه شروع را یافته، p را به آن اشاره بده.

۲. تا زمانی که p به آخر log نرسیده ادامه بده (پیچیدگی o(n)).

۲.۱. اگر عملیات log[p] از نوع نوشتن است، یعنی log[p]=W[T_i,x,v1,v2] آنگاه،

۲.۱.۱. TempWirtedT[x] را برابر T_i قرار بده و x را به WriteItems[T_i] اضافه کن.

۲.۲. اگر عملیات log[p] از نوع خواندن است، یعنی log[p] = R[T_i,x] آنگاه،

۲.۲.۱. مقدار WritedT[x] را به Parents[T_i] اضافه کن.

۲.۲.۲. اگر T_i تراکنشی سالم است یعنی T_i در Destroyer_Transaction_List[T_i] برابر false می‌باشد و TempAssessParents[T_i] نیز خالی است، آنگاه،

۲.۲.۲.۱. اگر آخرین تراکنشی که x را بروز کرده تراکنشی مخرب است، مقدار WritedT[x] را به TempAssessParents[T_i] اضافه کن

۲.۳. اگر log[p] برابر است با Abort[T_i]:

۲.۳.۱ برای همه‌ی عناصر WriteItems[T_i] مانند L، TempWrite[L] را برابر WritedT[L] قرار داده سپس WriteItems[T_i] را خالی کن. (بدلیل استفاده از قرار داد قفل گذاری دو مرحله ای صریح نیازی به بروزرسانی لیست‌های Parents و TempAssessParents نیست). (پیچیدگی o(k))

۲.۴. اگر log[p] برابر است با Commit[T_i]:

۲.۴.۱. اگر لیست TempAssessPaterns[T_i] خالی نیست، تراکنش T_i را به Destroyer_Transaction_List اضافه کن.

۲.۴.۲. برای همه‌ی عناصر WriteItems[T_i] مانند L، WritedT[L] را برابر TempWrite[L] قرار بده. (پیچیدگی o(k))

۳. p را به رکورد بعدی log اشاره بده.

با توجه به پیچیدگی‌های ذکر شده برای هر سطر الگوریتم پیچیدگی کلی آن از مرتبه (m*k) چرا که غیر از خطوط ۲.۳.۲ و ۲.۴.۲ که پیچیدگی آنها از مرتبه k می‌باشد بقیه خطوط از مرتبه ثابت می‌باشند؛ دو خط ۲.۳.۲ و ۲.۴.۲ در هنگام commit یا abort یک تراکنش اجرا می‌شوند پس برای هر تراکنش یکی از آنها اجرا شده و در مجموع m بار اجرا می‌شوند.

۳.۲.الگوریتم ایجاد تراکنش‌های پاکسازی

هدف از این الگوریتم ایجاد تراکنش‌های پاکسازی جهت از بین بردن اثرات حمله به پایگاه داده به صورت بهینه است.

ورودی‌ها: فایل ثبت وقایع و لیست تراکنش‌های بداندیش(Total_Destroyer_TList) و خروجی: لیست تراکنش‌های پاکسازی

مقداردهی اولیه: LastValue = TempTransactionList = SubmitedList = TempList = NULL

۱. p را به آخر فایل log اشاره بده و با مقایسه محل اولین عملیات تراکنش‌های عضو Total_Destroyer_TList زودترین

نقطه شروع را یافته، q را به آن اشاره بده.

۲. تا زمانیکه p>=q است (پیچیدگی o(n)):

۲.۱. اگر عملیات log[p] از نوع نوشتن است، یعنی log[p]=W[T_i,x,v1,v2] آنگاه،

۲.۱.۱. اگر LastValue[x] برابر NULL است، LastValue[x] را برابر v2 قرار بده.

۲.۱.۲. اگر SubmitedList[x] برابر false است،

۲.۱.۲.۱. اگر عمل نوشتن جاری توسط یک تراکنش مخرب انجام می‌شود، TempTransactionList[x] را برابر تراکنش پاکسازی W[T_j,x,v2,v1].

۲.۱.۲.۲. اگر عمل نوشتن جاری توسط یک تراکنش سالم انجام می‌شود، SubmitList[x] را برابر true قرار داده و W[T_i,x,v1,v2] را به SubmitTransactionList[x] اضافه کن و اگر TempList[x] برابر true است آن را false کرده و TempTransactionList[x] را نیز حذف کن.

۲.۲. p را به عنصر قبلی log اشاره بده.

۳. تراکنش‌های TempTransactionList را به SubmitTransactionList اضافه کن و داده‌های TempList را نیز به SubmitList اضافه کن.

۴. برای هر عنصر عضو SubmitList مانند x اگر مقدار LastValue[x] مخالف مقداری است که تراکنش SubmitTransactionList[x] قصد نوشتن آن را دارد، تراکنش SubmitTransactionList[x] را اجرا کن وگرنه آن را حذف کن.

بدترین حالت در این الگوریتم زمانی اتفاق می‌افتد که همه تراکنش‌های اجرا شده تراکنش‌هایی مخرب بوده و هریک داده ای مجزا را خراب کنند که در این حالت پیچیدگی خطوط ۳ و ۴ و در نتیجه کل الگوریتم برابر m*k خواهد شد.

از آنجایی که در هر سیستم توزیع شده همواره امکان بروز مشکلاتی مانند خراب شدن پیام‌ها، قطعی خطوط ارتباطی و غیره وجود دارد، در ارائه راهکارهای مختلف برای استفاده در این نوع سیستم‌ها باید نحوه‌ی عملکرد راهکار مورد نظر در برابر این اتفاقات بررسی شود و گرنه راهکار مزبور راهکاری جامع و کاربردی نمی‌باشد. فرآیند DAR مطرح شده در این مقاله نیز از این قاعده مستثنی نبوده و باید عملکرد آن در برابر مشکلات متحمل در یک سیستم توزیع شده مشخص گردد. در ادامه این بخش ابتدا راهکار ارائه شده در حالتی که اجزای DDBS به درستی کار کنند را شرح داده و سپس به توضیح رفتار سیستم در برابر مشکلات احتمالی سیستم‌های توزیع شده می‌پردازیم.

۳.۳. فرآیند DAR در شرایط معمولی DDBS

با کشف حمله‌ی صورت گرفته به DDBS توسط IDS، پیامی به نام ASSESS که در بردازنده‌ی شناسه[10] تراکنش‌های مهاجم از سوی IDS به همه‌ی HPA های مستقر بر روی میزبان‌های DDBS فرستاده شده و بدین ترتیب فرآیند DAR در تمامی میزبان‌های DDBS شروع می‌شود. با آغاز این فرآیند و در اولین مرحله HPA مستقر بر روی هر میزبان گراف وابستگی محلی را از LSA درخواست می‌کند. در مرحله‌ی بعد و در میزبان شماره i پایگاه داده یکی از دو حالت یا میزبان i گراف محلی خود یعنی G_i را به سوی میزبانی دیگر (میزبان j) می‌فرستد تا عمل ترکیب G_i با G_j و تشکیل $G_{i,j}$ در میزبان j انجام شود و یا اینکه میزبان i منتظر دریافت G_j از میزبان j می‌ماند تا با ترکیب آن با گراف وابستگی محلی خود، $G_{i,j}$ را تشکیل دهد.

برای نیل به هدف توزیع بار پردازشی مناسب در راهکار ارائه شده از آرایه ای بنام HostMap استفاده می شود. این آرایه در هر میزبان توسط HPA نگهداری و مدیریت می شود. HostMap[i] در هر مرحله نشان دهنده ی موقعیت میزبان i برای فرآیند تشکیل گراف وابستگی عمومی در آن مرحله است که فرستنده یا گیرنده بودن آن میزبان در آن مرحله را مشخص می کند. بر اساس مقدار HostMap[i] در هر مرحله یکی از ۴ حالت زیر وجود دارد :

۱- HostMap[i] برابر m است که m عددی صحیح ، مثبت و زوج می‌باشد، در اینصورت میزبان i دریافت کننده ی گراف از میزبان j که HostMap[j] برابر m+1 است می باشد.

۲- اگر HostMap[i] برابر m است که m عددی صحیح ، مثبت و فرد می‌باشد، میزبان i گراف خود را به سوی میزبان j که HostMap[j] برابر m-1 است می‌فرستد.

۳- اگر HostMap[i] برابر ۲- باشد بدین معنی است که میزبان i در مراحل قبلی گراف خود را به سوی میزبانی دیگر فرستاده و از روند تشکیل گراف وابستگی عمومی خارج شده است .

۴- اگر HostMap[i] در یک گراف خاص مانند k ($k \neq i$) برابر ۱- باشد، بدین معنی است که بدلیل بروز مشکلی در میزبان i یا خطوط ارتباطی بین دو میزبان i و k ارتباط میزبان k با میزبان j قطع شده است.

برای مثال در حالتی که مقادیر آرایه‌ی HostMap بصورت زیر باشند:

HostMap[0] = 0, HostMap[2] = 1, HostMap[5] = 2, HostMap[9] = 3, HostMap[i] = -2

for i = 1,3,4,6,7,8

10. Identifier

بدین معنی است که میزبان ۲ باید گراف تشکیل شده در خود را به سوی میزبان ۰ بفرستد و میزبان ۹ نیز باید گراف تشکیل شده در خود را به سوی میزبان ۵ بفرستد.

با توجه به مطالب ذکر شده می توان فرآیند صورت گرفته در عامل‌های HPA و LSA و DMA را بصورت زیر شرح داد. پس از آنکه پیام ASSESS به HPA_i رسید HPA_i آرایه‌ی HostMap را بررسی کرده اگر HostMap[i] برابر m که عددی مثبت و زوج است باشد، مقدار j را که HostMap[j] برابر m+1 است یافته منتظر دریافت G_j می ماند امّا اگر m عددی مثبت و فرد باشد G_i را از LSA_i دریافت کرده و به سمت HPA_j که HostMap[j] برابر m-1 است می‌فرستد. با پایان این مرحله HostMap[k] برای تمامی Kها ی فرد که G_K را به سمت میزبان‌های دیگر فرستاده اند برابر ۲- شده و بدین ترتیب آنها از چرخه‌ی ساخت گراف وابستگی عمومی خارج می‌شوند. با تکرار این چرخه و ترکیب دو به دوی گراف‌های وابستگی حاصل از مراحل قبل در نهایت گراف وابستگی عمومی (G_T) در میزبان h ساخته می شود. با کامل شدن G_T نوبت به استخراج لیست شناسه‌ی تمامی تراکنش‌های مخرب شامل ترکنش‌های مهاجم و تراکنش‌های متاثر از آن‌ها به نام Destroy_transaction_list می‌رسد. این عمل با استفاده از G_T و توسط HPA_h انجام می‌شود. پس از کامل شدن Destroy_transaction_list عامل HPA_h آن را برای همه‌ی میزبان‌های دیگر DDBS باز پخش می کند. با دریافت Destroy_transaction_list در هر میزبان عملیات بررسی فایل ثبت وقایع و ایجاد تراکنش‌های پاکسازی در آن میزبان توسط LSA محلی آن میزبان انجام می گیرد.

۳.۴.عملکرد فرآیند DAR در برابر مشکلات احتمالی

می‌توان مشکلات ممکن در یک DDBS را که ممکن است بر فرآیند کشف و بازیابی آسیب‌ها در آن DDBS تاثیر بگذارند به دو دسته تقسیم کرد. دسته اول امکان تغییر دادن، جعل کردن، تکرار و یا گم شدن پیام‌های ارسالی در شبکه‌ی ارتباطی DDBS می‌باشد. دسته‌ی دوم مشکلاتی است که در پی از کار افتادن میزبان‌ها یا قطعی خطوط ارتباطی بین آنها رخ می‌دهند. در مورد مشکلات دسته‌ی اول با استفاده از تکنیک‌های ارتباطی امن مانند قرارداد لایه ارتباطی امن[11](SSL) می‌توان از عدم تغییر، گم شدن جعل شدن و تکرار پیام‌های ارسال شده اطمینان پیدا کرد. در مورد دسته دوم مشکلات با ارائه‌ی مثال‌هایی نحوه‌ی عملکرد سیستم در برابر آنها شرح داده می‌شود. فرض کنید DDBS ی داری ۱۰ میزبان بنام‌های H_0 تا H_9 باشد که به ترتیب از ۰ تا ۹ شماره گذاری شده اند، بنابراین در همه‌ی آنها به ازای i (که i بین ۰ تا ۹ است) HostMap[i] برابر i خواهد بود.

با بروز یک حمله به DDBS و کشف آن توسط IDS پیام ASSESS از سوی IDS به همه‌ی ۱۰ میزبان DDBS فرستاده می‌شود. حال فرض کنید به دلیلی مانند قطعی خط ارتباطی یا مشکل در میزبان‌های ۲ و ۷ پیام ASSESS مربوط بدست این دو میزبان نرسد امّا فرآیند DAR در بقیه‌ی میزبان‌ها با دریافت پیام ASSESS آغاز می‌شود. در این مرحله و بر اساس HostMap از یک سو میزبان ۶ منتظر دریافت G_7 می‌شود و از دیگر سو میزبان ۳ که G_3 را به سمت میزبان ۲ فرستاده منتظر دریافت ACK از سوی میزبان ۲ است. فرآیند DAR در دیگر میزبان‌ها بصورت عادی دنبال شده $G_{0,1}$ در میزبان ۰، $G_{4,5}$ در میزبان ۴ و $G_{8,9}$ در

11. Secure Socket Layer

میزبان ۸ تشکیل می‌شود. در صورتیکه G_3 ارسالی از سوی میزبان ۳ به میزبان ۲ برسد، HPA_2 با دریافت G_3 متوجه شروع فرآیند DAR شده و به آن می‌پیوندد امّا در غیر اینصورت HPA_3 با دریافت نکردن ACK از سوی HPA_2 پس از گذشت بازه‌ی زمانی مشخصی متوجه بروز مشکل در میزبان ۲ یا خطوط ارتباطی آن شده و با بازپخش کردن پیامی بنام Invalidate که حاوی مشخصات میزبان ۲ است این مطلب را به اطلاع میزبان‌های دیگر نیز می‌رساند که آنها نیز با دریافت این پیام HostMap[2] را برابر ۱- قرار داده آرایه‌ی HostMap را برای مرحله‌ی بعدی الگوریتم بروز رسانی می‌کنند. دقت کنید که درطی این بروز رسانی مقدار HostMap[3] از ۲- به ۱ تغییر می‌کند. اما در مورد میزبان ۷ از آنجاییکه HPA_6 منتظر دریافت G_7 می‌باشد پس از گذشت بازه‌ی زمانی مشخص و عدم دریافت آن با ارسال صریح یک پیام GraphRequest به سوی HPA_7، G_7 را از آن در خواست می‌کند و اگر پس از مدتی آن را دریافت نکرد عدم حضور او را نیز به بقیه اعلام می‌کند. پس در پایان این مرحله آرایه‌ی HostMap در هر ۱۰ میزبان به غیر میزبان ۲ و ۷ بصورت زیر می‌باشد.

HostMap: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Value: 0,-2,-1, 1, 2,-2, 3,-1, 4,-2

با ادامه یافتن الگوریتم در نهایت گراف متشکل از ترکیب ۸ میزبان معتبر در میزبان 0 تشکیل می‌گردد. در این مرحله HPA_0 با بررسی گراف حاصل و اطلاع از عدم مشارکت میزبان‌های ۲ و ۷ در ساخت آن یکبار دیگر با ارسال پیام GraphRequest به هر کدام از آنها گراف آنها را درخواست می‌کند که در صورت دریافت آنها ادامه روند بصورت عادی ادامه می‌یابد در غیر اینصورت دو انتخاب برای HPA_0 ممکن است؛ انتخاب خوش‌بینانه و انتخاب بدبینانه. در حالت خوش‌بینانه بدون توجه به مشکل میزبان‌های ۲ و۷ گراف حاصل را بعنوان G_T در نظر گرفته و روند کشف و بازیابی آسیب‌ها را بصورت عادی دنبال می‌کند. در آینده و بازگشت میزبان‌های مشکل دار به سیستم مجدداً فرآیند جدیدی برای کشف و بازیابی آسیب‌ها شروع می‌شود. اما در حالت بدبینانه تمامی تراکنش‌هایی که زیرتراکنشی در میزبان‌های مشکل دار دارند به عنوان تراکنش‌های مخرب در نظر گرفته می‌شوند. می‌توان انتخاب حالت خوشبینانه یا حالت بدبینانه را بر عهده‌ی مدیر سیستم گذاشت.

۴. نتیجه گیری

همانگونه که اشاره شد که برای ایجاد یک پایگاه داده امن نیازمند اجرای فرآیندی سه مرحله‌ای می‌باشیم، مرحله‌ی اول محافظت داده‌ها در برابر حملات، مرحله دوم کشف حملات صورت گرفته به پایگاه داده و مرحله سوم کشف داده‌های آسیب دیده در طی یک حمله و بازیابی آنها به مقادیر معتبر قبل از حمله. در این مقاله با استفاده از عامل‌های نرم افزاری راهکاری برای کشف و بازیابی داده‌های آسیب دیده در طی حملات به پایگاه داده‌های توزیع شده ارائه گردید که بر مبنای تشکیل گراف‌های وابستگی تراکنش محلی و ترکیب آنها برای ایجاد گراف وابستگی تراکنش عمومی کار می‌کند. نکته مهم برای داشتن یک پایگاه داده امن برخورداری از یک سیستم کشف نفوذ کارا می‌باشد. طراحی یک سیستم کشف نفوذ مختص پایگاه داده‌های توزیع شده مسئله‌ای است که در آینده به دنبال بررسی آن می‌باشیم.

۵. منابع

[1] Liu Peng; (Yu Meng). 2011. "Damage assessment and repair in attack resilient distributed database systems", Computer Stand.Interfaces, vol. 13, pp 55-67.
[2] Zuo Yanjun; (Panda Brajendra). 2004. "Damage Discovery in Distributed Database Systems", IFIP International Federation for Information Processing, Volume 144/2004,pp 111-123.
[3] Chiueh T; (Pilania D). 2004. "Design, implementation, and evaluation of a repairable database management system", Proc. Annual Computer Security Applications Conference, Washington, DC.
[4] Rahimi, Saeed;(Haug, Frank). 2010. "Distributed Database Management Systems (A Practical Approach)",Wiley,p171.
[5] Zuo Yanjun;(Panda Brajendra). 2006. "Distributed database damage assessment paradigm", Information Management & Computer Security,Vol. 14 No. 2, pp. 116-139.
[6] Zhu Hong; (Fu Ge). 2008. "Dynamic Data Recovery for Database Systems Based on Fine Grained Transaction Log", international symposium on Database engineering & applications, Portugal.
[7] Gertz, Michael;(Jajodia, Sushil). 2008. "Handbook of Database Security Applications and Trends,First Edition",Springer Science. p52.
[8] Zubi, Suliman Zakaria, 2009. "On Distributed Database Security Aspects", International Conference on Multimedia Computing and Systems, Ouarzazate.
[9] Lomet David; (Vagena Zografoula). 2006. "Recovery from bad user transactions", InternationalConference on Management of Data, New York.
[10] Ammann Jajodia; (Liu P). 2001. "Recovery from malicious transactions". IEEE Transactions on Knowledge and Data Engineering, Volume 14, Issue 5, pp 1167-1185.
[11] Bajio Javier; (Corchado M). 2010. "SCMAS A Distributed Hierarchical Multi-Agent Architecture for Blocking Attacks to Databases", International Journal of Innovative Computing Information and Control, Vol. 6, No. 9. pp. 128-140.
[12] Yu M. , (Liu P.). 2004. "Self healing workflow systems under attacks". 24th IEEE Int'lConf. on Distributed Computing Systems, USA