باسمه تعالی

دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

فرم تعریف **پروژه**
فارغ التحصیلی دوره کارشناسی

دانشکده مهندسی کامپیوتر
و فناوری اطلاعات

تاریخ: .....
شماره: .....

عنوان پروژه: طراحی و پیاده‌سازی ابزاری به منظور اِعمال خط مشی امنیتیِ عدم تداخل مبتنی بر روش بازنویسی برنامه
استاد راهنمای پروژه: دکتر مهران سلیمان‌فلاح **امضاء:**
مشخصات دانشجو: **نام و نام خانوادگی: سید محمدمهدی احمدپناه**[1] **گرایش: نرم‌افزار** **شماره دانشجویی: ۹۰۳۱۸۰۶** **ترم ثبت نام پروژه: دوم ۹۳-۹۴**
داوران پروژه: **۱-** **امضاء داور:** **۲-** **امضاء داور:**
شرح پروژه (در صورت مشترک بودن بخشی از کار که بعهده دانشجو می باشد مشخص شود): **به پیوست آمده است.**
وسائل مورد نیاز: **- امکان دسترسی به مقالات مرتبط** **- یک دستگاه کامپیوتر دارای دسترسی به اینترنت**
محل انجام پروژه: دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر **تاریخ شروع: اردیبهشت ۱۳۹۴**

این قسمت توسط دانشکده تکمیل می‌گردد:

تاریخ تصویب در گروه: **اسم و امضاء:** **تاریخ تصویب در دانشکده:** **اسم و امضاء:** **اصلاحات لازم در تعریف پروژه:**

توجه: پروژه حداکثر یک‌ماه و نیم پس از شروع ترمی که در آن در درس پروژه ثبت نام به عمل آمده است باید به تصویب برسد.

نسخه۱- دانشکده	نسخه ۲- استاد راهنما	نسخه ۳- دانشجو

[1] Email: smahmadpanah@aut.ac.ir

تعریف مسئله:

امروزه و با گسترش سیستم‌ها و نرم‌افزارها، امنیت انتقال اطلاعات در برنامه‌های نوشته شده به زبان‌های مختلف برنامه‌نویسی، بیش از پیش اهمیت پیدا کرده است. از این رو، می‌توان با افزودن ابزارهایی به یک زبان برنامه‌نویسی، باعث تولید نرم‌افزارهای مقاوم‌تری در برابر حملات و نفوذها شد. لذا باید مفهوم امن بودنِ یک سیستم یا برنامه به طور دقیق تعریف و مشخص شود که همین مسئله چالشی برای متخصصان این حوزه است. در مرحله بعدی، نحوه و رویکرد اِعمال آن مفهوم از امنیت مهم خواهد بود که وابستگی زیادی به تعریف ارائه شده دارد.

به طور کلی، خط مشی[۲] امنیتی، امن‌بودنِ یک سیستم یا برنامه را تعریف می‌کند. خط مشی امنیتی، قیود روی توابع و جریان‌های بین آن‌ها را مشخص می‌کند؛ مثل قیود دسترسی بر روی برنامه‌ها و سطوح دسترسیِ داده‌های بین کاربران که مانع از بروز مشکلات امنیتی از طریق سیستم‌های خارجی و حمله‌کنندگان شود.

یک خط مشی امنیتی را می‌توان به عنوان یک زیرمجموعه از مجموعه‌ی توانی همه اجراها، که هر اجرا یک دنباله دلخواه از حالت[۳]ها است، تعریف کرد. ضمناً می‌توان آن را به عنوان مجموعه‌ی برنامه‌هایی که آن خط مشی را برآورده می‌کنند، در نظر گرفت. بعضی از خط مشی‌های امنیتی، *خاصیت*[۴] هستند؛ به‌خاطر این‌که قابل دسته‌بندی توسط مجموعه اجراها می‌باشند.

به زبان ساده، خط مشی عدم تداخل[۵] بیان می‌کند که یک مشاهده‌گرِ[۶] سطحِ پایین که فقط به برنامه و مقادیر عمومیِ زمانِ اجرا دسترسی دارد، نتواند ورودی‌های سطح بالا یا خصوصیِ برنامه را بفهمد. به عبارت دیگر، این خط مشی بیان می‌کند که در هر جفت اجراهای برنامه که ورودی‌های عمومی یکسان دارند، مستقل از ورودی‌های خصوصی متفاوت، باید خروجی‌های عمومی یکی باشند.

نکته مهم این است که خط مشی عدم تداخل، یک *خاصیت* نیست؛ زیرا توسط اجراهای جداگانه که این خط مشی را برآورده می‌کند، قابل تعریف نیست. این نکته باعث ایجاد محدودیت‌هایی برای اِعمال این خط مشی در برنامه‌ها می‌شود.

از طرفی، خط مشی عدم تداخل را می‌توان به دو دسته *حساس به پیشرفت*[۷] و *غیرحساس به پیشرفت*[۸] تقسیم کرد. در عدم تداخلِ غیرحساس به پیشرفت، مشاهده‌گرِ سطح پایین، تنها می‌تواند خروجی‌های میانیِ سطح پایین را ببیند؛ در حالی‌که یک مشاهده‌گرِ سطح پایین در عدم تداخلِ حساس به پیشرفت، علاوه بر دسترسی‌های قبلی، به *وضعیت پیشرفتِ*[۹] برنامه نیز دسترسی دارد. این باعث می‌شود تا بتواند تفاوت بین واگرایی[۱۰] برنامه با موقعیتی که برنامه خاتمه می‌یابد یا در حال محاسبه مقادیر قابل‌مشاهده‌ی بعدی است، را تمیز دهد.

نکته مهم دیگر، بررسی جریان‌های غیرمجاز صریح و ضمنی است که می‌تواند مسئله‌ی داده شده را پیچیده کند.

در این پروژه، هدف این است که با بهره‌گیری از روش بازنویسی برنامه[۱۱]، برنامه‌های نوشته‌شده توسط یک زبان برنامه‌نویسیِ مدل به نام WL به نحوی تغییر داده شوند که خط مشی امنیتیِ عدم تداخل را برآورده سازند.

---

[2] Policy
[3] State
[4] Property
[5] Noninterference
[6] Observer
[7] Progress-Sensitive
[8] Progress-Insensitive
[9] Progress Status
[10] Divergence
[11] Program Rewriting

راه‌حل‌های فعلی و مشکلات آن‌ها:

ونکتکریشنن و دیگران [?] یک روش تبدیل برنامه‌ی ترکیبی برای اِعمال عدم تداخل ارائه داده‌اند. برنامه تغییر داده شده، سطوح امنیتی انتساب[۱۲] را دنبال می‌کند و زمانی‌که یک جریان غیرمجاز در حال وقوع باشد، خاتمه می‌یابد. این روش، تنها در فرمول‌بندی‌هایی از عدم تداخل که بدون توجه به رفتار خاتمه‌ی برنامه‌ها مطرح می‌شود، قابل استفاده است.

مگزینیوس و دیگران [?] یک چارچوب برای ناظر[۱۳]های امنیتی پویای درون‌برنامه‌ای- در حالی‌که برنامه در حال اجراست- ساخته‌اند. این روش، عدم تداخلِ غیرحساس به خاتمه را تضمین می‌کند و قابل به‌کارگیری در زبان‌های Perl و JavaScript است که از ارزیابی پویای کد پشتیبانی می‌کنند. ضمناً این روش نیاز دارد که تغییردهنده‌ی برنامه در زمان اجرا در دسترس باشد که یک ناظرِ مناسب بتواند در کدی که به صورت پویا تولید می‌شود، ورود کند.

چادنوف و نومن [?] یک ناظرِ ترکیبی برای عدم تداخلِ حساس به جریان در زبان‌های با ارزیابی پویای کد پیشنهاد داده‌اند. این روش ممکن است باعث وجود یک سربارِ غیرقابل قبول در زمانِ اجرا شود. همچنین این روش، اجازه وقوع مجراهای خاتمه[۱۴] را نمی‌دهد. سانتوس و رزک [?] نیز این روش را برای یک هسته JavaScript گسترش دادند.

این موضوع اثبات شده است که هیچ روش کاملاً پویایی برای اِعمال عدم تداخل حساس به جریان وجود ندارد. این موضوع باعث می‌شود که پروژه‌هایی که محدودیت‌های نَحوی[۱۵] بر روی کد دارند، از اطلاعات ایستا در ناظری بر اجراهای چندگانه‌ی برنامه‌ها استفاده کنند.

بلو و بونلی [?] یک ناظرِ اجرایی[۱۶] پیشنهاد دادند که از یک تحلیل وابستگیِ زمانِ اجرا بهره می‌برد. برای یافتن یک جریان غیرمجاز، همان‌طور که در طرح پیشنهادی آن‌ها و کارهای مشابه دیگر آمده است، ممکن است نیاز به چندین اجرا از برنامه مورد نظر داشته باشد که در بسیاری از کاربردها این امکان وجود ندارد.

لِگوئرنیک و دیگران [?] یک ماشین طراحی کرده‌اند که رخدادهای انتزاعی[۱۷] در زمان اجرا را دریافت می‌کند و اجرا را توسط بعضی از اطلاعات ایستا، ویرایش می‌کند. این روش نیز جالب است اما اجازه وقوع مجراهای خاتمه را می‌دهد.

به طور کلی، مسئله‌ی تشخیص برنامه‌هایی که عدم تداخل را برآورده می‌کنند، تصمیم‌ناپذیر[۱۸] است. پس در حالت کلی، عدم تداخل توسط روش‌های ایستا قابل اعمال نیست؛ به همین دلیل است که نوع‌سامانه[۱۹]های ارائه‌شده برای این مسئله، محافظه‌کارانه[۲۰] هستند و ممکن است بعضی برنامه‌های امن را نیز رد کنند. از طرفی، این مسئله هم‌بازگشتی شمارش‌پذیر[۲۱] نیز نیست. بنابراین، قابل اِعمال توسط ناظرهای اجرایی که نقضِ خط مشی عدم تداخل در یک برنامه در حال اجرا را بررسی می‌کنند، نیز نیست.

---

[12] Assignment
[13] Monitor
[14] Termination Channels
[15] Syntactic
[16] Execution Monitor
[17] Abstract Events
[18] Undecidable
[19] Type System
[20] Conservative
[21] Co-recursively Enumerable

راه‌حل پیشنهادی:

راه‌حل مورد توجه در این پروژه، از گراف‌های وابستگی برنامه[۲۲] استفاده می‌کند. گراف وابستگی برنامه، ابزاری است که توسط آن، یک برنامه با یک گراف جهت‌دار که گره‌های آن گزاره‌ها یا عبارت‌های برنامه است و یال‌های آن، وابستگی‌های کنترلی یا داده‌ای بین گره‌ها را نشان می‌دهد، بیان می‌شود. گراف وابستگی یک برنامه، همه وابستگی‌های بین گزاره‌های آن برنامه را منعکس می‌کند، ولی برعکس آن لزوماً درست نیست؛ یعنی ممکن است مسیری بین گره‌ها در این گراف وجود داشته باشد ولی جریانی بین آن دو برقرار نباشد.

برخلاف سایر روش‌های امنیتی مطرح‌شده، روش این پروژه، یک برنامه مغایر با خط مشی موردنظر را، چه قبل یا چه در طول زمان اجرا، رد نمی‌کند؛ بلکه آن‌ها بازنویسی می‌شوند و از برنامه‌های ناامن به برنامه‌های امن تبدیل می‌شوند. در این روش، هر دو نوع عدم تداخل- حساس به پیشرفت و غیرحساس به پیشرفت- در برنامه‌ها به همراه مقادیر قابل‌مشاهده میانی، مورد توجه هستند. ضمناً قابل اثبات است که بازنویس[۲۳]های مورد نظر، در اعمال عدم تداخل، سالم[۲۴] و شفاف[۲۵] هستند. شفاف بودن یک روش بدین معناست که تا حد ممکن، مجموعه اجراهای ممکنِ برنامه‌ی تبدیل‌شده مشابه برنامه‌ی ورودی باشد، خواه امن باشد یا نباشد.

با توجه به مشکلات اخیر مطرح‌شده در روش‌های قبلی، روش بازنویسیِ برنامه تغییراتی را به ذاتِ عدم تداخل وارد نمی‌کند؛ بلکه به جای آن، یک برنامه جدید با حداقل تغییرات ممکن نسبت به برنامه اصلی، که عدم تداخل را برآورده می‌کند، تولید می‌کند. در واقع، می‌توان روش بازنویسی برنامه را روشی بین روش‌های ایستا و روش‌های پویا دانست.

توصیف نرم‌افزار:

ورودی: کد برنامه (که ممکن است خط مشی عدم تداخل را برآورده نکند)

خروجی: کد برنامه تغییر داده شده (که خط مشی مورد نظر را برآورده می‌کند)

پردازش: با استفاده از کد برنامه ورودی، گراف وابستگی برنامه از روی آن ساخته می‌شود. سپس با توجه به این‌که خط‌مشی مورد نظر حساس به پیشرفت یا غیرحساس به پیشرفت است، پردازش مربوط به هر کدام انجام می‌شود.
در غیرحساس به پیشرفت، از روی گراف وابستگی برنامه، مسیرهای حاوی ورودی سطح بالا که در ادامه آن‌ها خروجی سطح پایینی نمایش داده خواهد شد، مورد بررسی قرار می‌گیرند. با استفاده از شرط‌های اجرا[۲۶] و شرط‌های مسیر[۲۷]، بسته به وجود جریان صریح یا ضمنی، دستور نمایش یک مقدار سطح پایین، به عبارتی شرطی تبدیل می‌شود که در صورت برقراری آن شرط، مقدار خروجی نشان داده می‌شود و در غیر این صورت، آن مقدار قابل مشاهده برای مشاهده‌گر سطح پایین نخواهد بود.

در حساس به پیشرفت، به دلیل این‌که مشاهده‌گر سطح پایین دسترسی‌های بیشتری دارد، لذا رفتار برنامه نیز اهمیت بیشتری پیدا می‌کند؛ یعنی باید بازنویس به گونه‌ای کد برنامه را تغییر دهد که وضعیت پیشرفت در برنامه‌ی تغییر پیدا کرده، به مقادیر سطح بالا وابستگی نداشته باشد. به همین دلیل، واگرایی یا خاتمه برنامه از دیدِ مشاهده‌گر سطح پایین نیز حاوی اطلاعاتی است که باعث نقضِ خط مشی عدم تداخل در این حالت می‌شود. در این حالت، روش پردازش و

[22] Program Dependence Graph (PDG)
[23] Rewriter
[24] Sound
[25] Transparent
[26] Execution Conditions
[27] Path Conditions

بازنویسی برنامه، به یک تحلیل‌گرِ حلقه[۲۸] وابسته خواهد شد. در ابتدا، مراحل حالت غیرحساس به پیشرفت بر روی کد برنامه ورودی انجام می‌شود و در صورتی که برنامه دارای مسیرهایی باشد که در آن قبل از رسیدن به یک حلقه، مقدار ورودی سطح بالایی وجود داشته باشد، مطابق با نتیجه‌ی تحلیل‌گر حلقه در هر کدام از حلقه‌ها، کدهای مربوط به آن بازنویسی و اصلاح می‌شوند. این‌گونه کد برنامه خروجی نیز خط مشی عدم تداخل حساس به پیشرفت را برآورده می‌سازد.

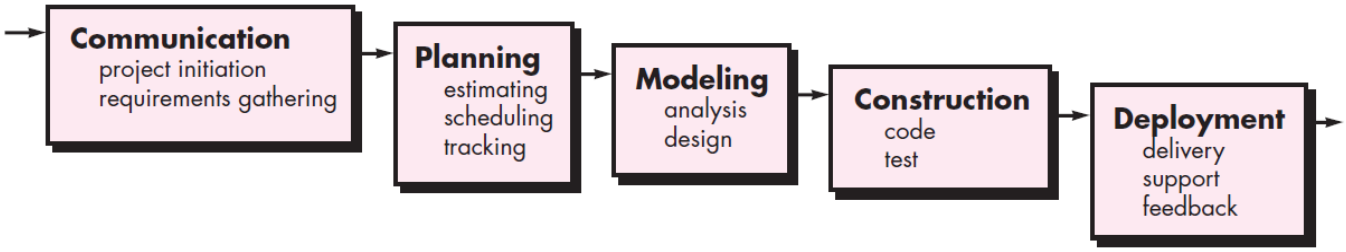
ضمناً نیاز است که برای زبان مورد نظر، مفسّری نیز نوشته شود تا کدهای برنامه قابل اجرا باشند.

همچنین برای آشنایی بیشتر کاربر با نرم‌افزار، بخش راهنمای استفاده از برنامه تهیه خواهد شد.

روش تصدیق[۲۹]: برای تصدیق این ابزار، علاوه بر اثباتِ صوریِ[۳۰] سالم بودن و شفافیت روش ارائه شده، آزمایش‌هایی بر روی ابزار پیاده‌سازی شده انجام خواهد شد.

تحلیل نرم‌افزار:

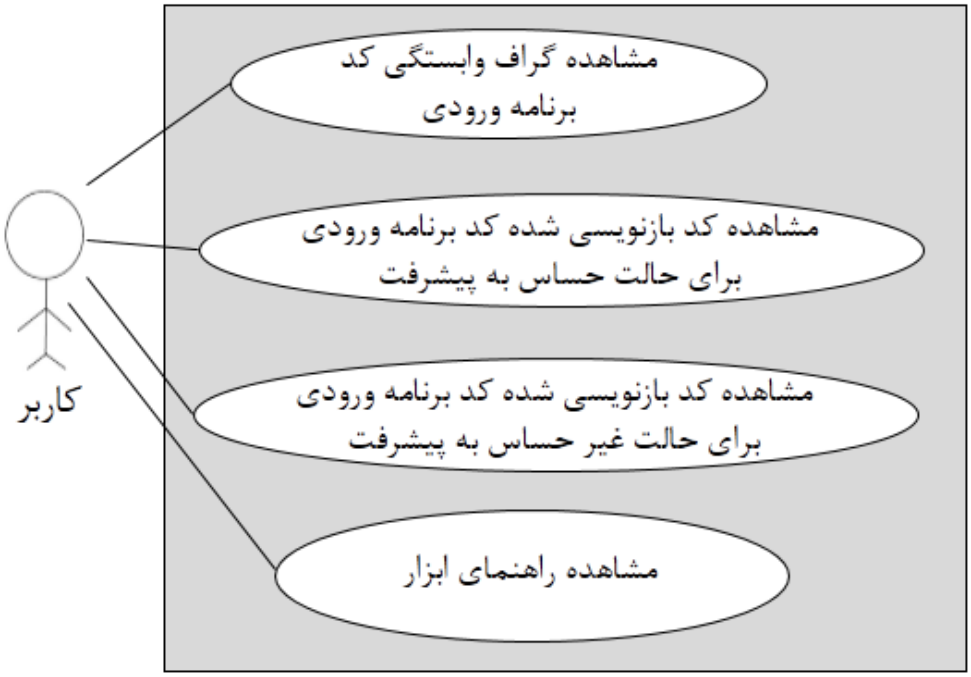
با توجه به مشخص و ثابت بودن نیازهای این نرم‌افزار در ابتدای تعریف پروژه، می‌توان از مدل فرآیندی آبشاری[۳۱] یا چرخه حیات کلاسیک[۳۲] استفاده کرد. این مدل فرآیندی شامل پنج مرحله ارتباط[۳۳]، برنامه‌ریزی[۳۴]، مدل‌سازی[۳۵]، ساخت[۳۶] و استقرار[۳۷] است.



شکل ۱ مدل فرآیندی آبشاری

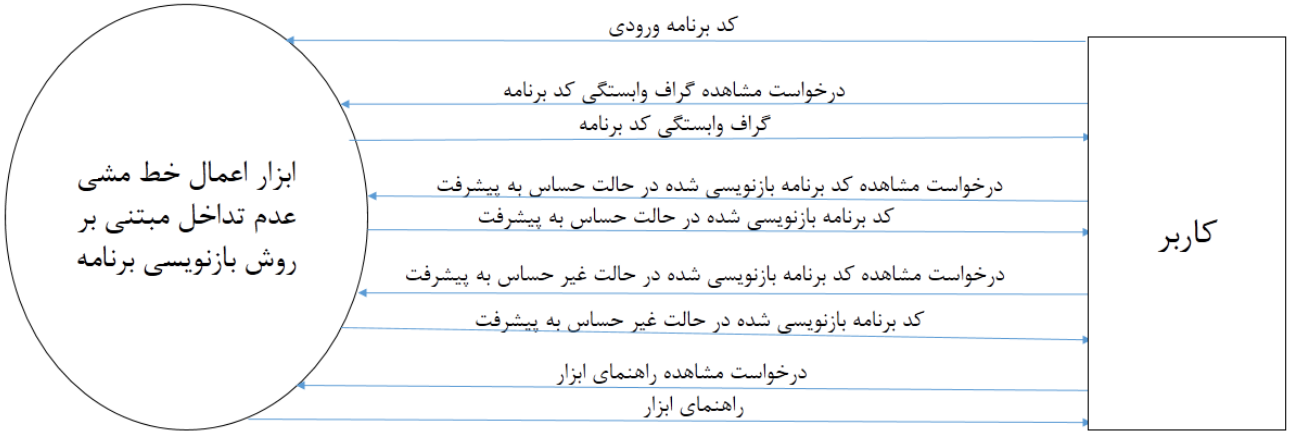
[28] Loop Analyzer
[29] Verification
[30] Formal
[31] Waterfall
[32] Classic Life Cycle
[33] Communication
[34] Planning
[35] Modeling
[36] Construction
[37] Deployment

نمودار Use Case:



شکل ۲ نمودار Use Case

نمودار Context:



شکل ۳ نمودار Context

مراجع:

مقالات مورد استفاده در این پروژه:

[1] A. Lamei and M.S. Fallah, “Rewriting-based Enforcement of Noninterference in Programs with Observable Intermediate Values”, To appear in the International Journal of Computer Mathematics, Vol. XX, No. XX, Month 20XX, 1-24.

[2] J.A. Gougen and J. Meseguer, “Security Policies and Security Models”, in Proceedings of IEEE Symposium on Security and Privacy, Vol. 12, IEEE, 1982, pp. 11-18.

[3] F.B. Schneider, “Enforceable Security Policies”, ACM Transactions on Information and System Security (TISSEC) 3, 2000, pp. 30-50.

[4] F.B. Shneider, J.G. Morrisett, and R. Harper, “A Language-based Approach to Security, in Informatics – 10 Years Back. 10 Years Ahead”, Springer-Verlag Berlin, Heidlberg, 2001, pp. 86-101.

[5] D. Wasserrab, D.Lohner, and G. Snelting, “On PDG-based noninterference and its modular proof”, in Proceedings of the ACM SIGPLAN Fourth Workshop on Programming Languages and Analysis for Security, PLAS ’09, ACM, 2009, pp. 31-44.

[6] J. Ferrante, K.J. Ottenstein, and J.D. Warren, “The program dependence graph and its use in optimization, ACM Transactions on Programming Languages and Systems 9, 1987, pp. 319-349.