



دانشگاه صنعتی امیرکبیر  
(پلی تکنیک تهران)  
دانشکده مهندسی کامپیوتر و فناوری اطلاعات

پایان نامه کارشناسی ارشد  
گرایش امنیت اطلاعات

بهبود مکانیزم‌های مبتنی بر چنداجزایی برای اِعمال خط‌مشی‌های  
جریان اطلاعات

نگارش  
سید محمد مهدی احمدپناه

استاد راهنما  
دکتر مهران سلیمان فلاح

مهر ۱۳۹۶

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ



دانشگاه صنعتی امیرکبیر

دانشکده مهندسی کامپیوتر و فناوری اطلاعات

پایان نامه کارشناسی ارشد

گرایش امنیت اطلاعات

بهبود مکانیزم‌های مبتنی بر چنداجرایی برای اعمال خط‌مشی‌های جریان اطلاعات

نام و نام خانوادگی: سید محمدمهدی احمدپناه شماره دانشجویی: ۹۴۱۳۱۰۸۶ مقطع: کارشناسی ارشد

این پایان‌نامه توسط هیئت داوران زیر در تاریخ ۱۳۹۶ / ۷ / ۲۹ به تصویب رسیده است:

امضا:

استاد راهنما: دکتر مهران سلیمان‌فلاح

امضا:

داور داخلی: دکتر مهدی شجری

امضا:

داور خارجی: دکتر رامتین خسروی

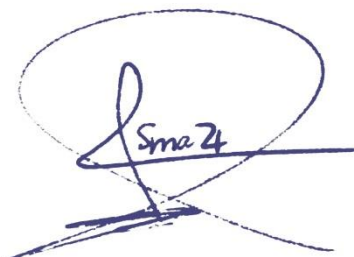
اینجانب سید محمدمهدی احمدپناه متعهد می‌شوم که مطالب مندرج در این پایان نامه حاصل کار پژوهشی اینجانب تحت نظارت و راهنمایی اساتید دانشگاه صنعتی امیرکبیر بوده و به دستاوردهای دیگران که در این پژوهش از آنها استفاده شده است مطابق مقررات و روال متعارف ارجاع و در فهرست منابع و مآخذ ذکر گردیده است. این پایان نامه قبلاً برای احراز هیچ مدرک هم‌سطح یا بالاتر ارائه نگردیده است.

در صورت اثبات تخلف در هر زمان، مدرک تحصیلی صادر شده توسط دانشگاه از درجه اعتبار ساقط بوده و دانشگاه حق پیگیری قانونی خواهد داشت.

کلیه نتایج و حقوق حاصل از این پایان نامه متعلق به دانشگاه صنعتی امیرکبیر می‌باشد. هرگونه استفاده از نتایج علمی و عملی، واگذاری اطلاعات به دیگران یا چاپ و تکثیر، نسخه‌برداری، ترجمه و اقتباس از این پایان نامه بدون موافقت کتبی دانشگاه صنعتی امیرکبیر ممنوع است. نقل مطالب با ذکر مآخذ بلامانع است.

سید محمدمهدی احمدپناه

امضا



تقدیم به پدرم

کوهی استوار و حامی من در سراسر زندگی

تقدیم به مادرم

سنگ صبوری که الفبای زندگی به من آموخت

تقدیم به خواهر و برادرم

همراهان همیشگی و پشتوانه‌های زندگیم

## تقدیر و تشکر:

از خانواده مهربانم که همواره و در تمام عرصه‌های زندگی یار و یاورم بوده‌اند و از حمایت‌های معنوی و مادی آن‌ها برخوردار بودم؛

از استاد گرانقدر جناب آقای دکتر مهران سلیمان فلاح که در کمال سعه صدر، با حسن خلق و فروتنی، راه و روش انسانیت و کسب دانش را به من آموختند؛

از اساتید محترم جناب آقای دکتر مهدی شجری و جناب آقای دکتر رامتین خسروی که زحمت داوری این پایان‌نامه را متقبل شدند؛

و از دوستان عزیزم، احسان عدالت، ریحانه شاه‌محمدی، احمد اسدی، امیرحسین ناصرالدینی، حمیدرضا رمضانی، محمد پزشکی، دانشجویان ورودی ۱۳۹۴ گرایش امنیت اطلاعات، اعضای آزمایشگاه امنیت صوری و سایر عزیزانی که طی کردن این دوره تحصیلی را برایم شیرین ساختند؛

کمال تشکر و قدردانی را دارم.

## چکیده

امنیت جریان اطلاعات مفهومی از امنیت برای تضمین محرمانگی و صحت داده‌ها بر اساس تعریف جریان‌های مجاز و غیر مجاز اطلاعات است. این مفهوم به ظهور نظریه عدم تداخل منجر شده است که به عنوان معناشناخت خط‌مشی‌های جریان اطلاعات نیز شناخته می‌شود. در سال‌های اخیر روش‌های ایستا و پویای متنوعی برای اعمال انواع خط‌مشی‌های جریان اطلاعات پیشنهاد شده‌اند که در بین آن‌ها روش موسوم به چنداجرایی امن توجه زیادی را به خود جلب کرده است. در این روش پویا و جعبه‌سیاه، به ازای هر سطح امنیتی رونوشتی از برنامه غیرقابل اعتماد اجرا شده و با محدود کردن خواندن از کانال‌های ورودی و نوشتن در کانال‌های خروجی، عدم تداخل حساس به خاتمه و زمان اعمال می‌شود. زمان‌بندی و همگام‌سازی اجرای رونوشت‌های مختلف برنامه چالش‌برانگیزترین موضوع در این روش است. به عبارت دقیق‌تر، زمان‌بندی و همگام‌سازی باید به گونه‌ای انجام شود که درستی و شفافیت را ایجاب کند. درستی به معنای امن بودن سامانه حاصل از چنداجرایی امن است و شفافیت به معنای آن است که اجراهای سامانه حاصل تا آن‌جا که ممکن است با اجراهای برنامه اصلی یکسان باشند. اگرچه با بهره‌گیری از مکانیزم‌های پیشنهاد شده مبتنی بر چنداجرایی امن، می‌توان به سطحی قابل قبول از شفافیت در اعمال عدم تداخل حساس به خاتمه دست یافت، این موضوع برای عدم تداخل حساس به زمان برقرار نبوده و یکی از نقاط ضعف مکانیزم‌های پیشنهادی محسوب می‌شود.

در این پایان‌نامه، مکانیزمی جدید مبتنی بر چنداجرایی امن برای اعمال خط‌مشی امنیتی عدم تداخل حساس به زمان پیشنهاد می‌کنیم که علاوه بر درست بودن، به سطحی از شفافیت دست می‌یابد که در آن هرگاه برنامه اصلی امن باشد، ترتیب خروجی‌های برنامه اصلی در کانال‌های مختلف نسبت به یکدیگر حفظ می‌شود. مکانیزم پیشنهادی را چنداجرایی امن بافردار می‌نامیم که در آن با بافر کردن رویدادهای حاصل از اجراهای رونوشت‌های مختلف برنامه و بهره‌گیری از نوعی زمان‌بند با گردش نوبت، در مورد قراردادن خروجی‌ها بر روی کانال‌های خروجی تصمیم‌گیری می‌شود.

## واژه‌های کلیدی:

چنداجرایی امن، عدم تداخل حساس به زمان، درستی، شفافیت

<b>فصل اول مقدمه.....</b>	<b>۱.....</b>
۱-۱ تعریف مسئله.....	۵.....
۲-۱ ایده اصلی در مکانیزم پیشنهادی.....	۸.....
۳-۱ ساختار مطالب.....	۸.....
<b>فصل دوم مفاهیم اولیه.....</b>	<b>۹.....</b>
۱-۲ تعریف خاصیت و فوق خاصیت امنیتی.....	۱۰.....
۲-۲ مکانیزم‌های امنیتی.....	۱۱.....
۳-۲ امنیت جریان اطلاعات.....	۱۲.....
۴-۲ عدم تداخل.....	۱۳.....
۵-۲ انواع خط‌مشی‌های عدم تداخل.....	۱۵.....
۶-۲ جمع‌بندی.....	۱۹.....
<b>فصل سوم کارهای مرتبط.....</b>	<b>۲۰.....</b>
۱-۳ روش‌های اعمال خط‌مشی امنیتی.....	۲۱.....
۱-۱-۳ مکانیزم‌های ایستا.....	۲۲.....
۲-۱-۳ مکانیزم‌های پویا.....	۲۳.....
۳-۱-۳ بازنویسی برنامه.....	۲۵.....
۴-۱-۳ تحلیل ترکیبی.....	۲۶.....
۲-۳ مقایسه روش‌های اعمال خط‌مشی امنیتی.....	۲۸.....
۳-۳ مکانیزم‌های مبتنی بر چنداجرایی امن.....	۳۲.....
۴-۳ جمع‌بندی.....	۴۳.....
<b>فصل چهارم مکانیزم پیشنهادی.....</b>	<b>۴۴.....</b>
۱-۴ چنداجرایی امن؛ مزایا و چالش‌ها.....	۴۵.....
۲-۴ شرح مکانیزم پیشنهادی.....	۵۱.....
۳-۴ جمع‌بندی.....	۶۵.....
<b>فصل پنجم صوری‌سازی و اثبات.....</b>	<b>۶۶.....</b>
۱-۵ نحو و معناشناخت زبان مدل.....	۶۷.....
۲-۵ معناشناخت مکانیزم چنداجرایی امن بافردار.....	۷۰.....
۱-۲-۵ مکانیزم چنداجرایی امن بافردار بدون گزارش نقض امنیت.....	۷۲.....
۱-۱-۲-۵ معناشناخت محلی.....	۷۲.....
۲-۱-۲-۵ معناشناخت سراسری.....	۷۵.....



۷۹.....	۳-۵ اثبات درستی و شفافیت کامل مکانیزم.....
۸۶.....	۴-۵ جمع‌بندی.....
۸۷.....	<b>فصل ششم جمع‌بندی و کارهای آینده.....</b>
۸۸.....	۱-۶ جمع‌بندی و نتیجه‌گیری.....
۹۱.....	۲-۶ کارهای آینده.....
۹۳.....	<b>منابع و مراجع.....</b>
۹۷.....	<b>پیوست.....</b>
۹۷.....	الف - صوری‌سازی معاشناخت مکانیزم چنداجرایی امن بافردارِ همراه با گزارش نقض امنیت.....
۹۸.....	الف - ۱ قواعد معاشناخت محلی.....
۱۰۰.....	الف - ۲ قواعد معاشناخت سراسری.....

شکل ۱ - برنامه دارای کانال خاتمه.....	۱۶
شکل ۲ - (الف) و (ب) نمونه‌هایی از برنامه‌های دارای کانال زمانی.....	۱۸
شکل ۳ - نحوه عملکرد مکانیزم ایستا، بازنویس برنامه و ناظر زمان اجرا به منظور اعمال خط‌مشی‌های امنیتی.....	۲۷
شکل ۴ - دسته‌بندی مکانیزم‌های اعمال خط‌مشی‌های امنیتی.....	۲۷
شکل ۵ - نمایی از روش چنداجرایی امن (SME).....	۳۳
شکل ۶ - چنداجرایی امن به همراه همگام‌سازی حصار.....	۳۷
شکل ۷ - چنداجرایی امن با حذف رده‌بندی.....	۳۸
شکل ۸ - نمایی از عملکرد مکانیزم چنداجرایی امن بافردار (الف) برای شبکه دوسطحی، (ب) برای شبکه چهارسطحی.....	۵۳
شکل ۹ - نمایی از عملکرد مکانیزم چنداجرایی امن بافردار به همراه گزارش نقض امنیت (الف) برای شبکه دوسطحی، (ب) برای شبکه چهارسطحی.....	۵۵
شکل ۱۰ - برنامه دارای کانال زمانی.....	۵۷
شکل ۱۱ - نمایی از محتوای بافرها در مکانیزم پیشنهادی در هنگام اجرای برنامه شکل ۱۰.....	۵۸
شکل ۱۲ - برنامه دارای کانال خاتمه.....	۵۸
شکل ۱۳ - نمایی از محتوای بافرها در مکانیزم پیشنهادی در هنگام اجرای برنامه شکل ۱۲.....	۵۹
شکل ۱۴ - برنامه دارای کانال جریان ضمنی.....	۶۰
شکل ۱۵ - نمایی از محتوای بافرها در مکانیزم پیشنهادی در هنگام اجرای برنامه شکل ۱۴.....	۶۰
شکل ۱۶ - برنامه امن طبق عدم تداخل حساس به زمان.....	۶۱
شکل ۱۷ - نمایی از محتوای بافرها در مکانیزم پیشنهادی در هنگام اجرای برنامه شکل ۱۶.....	۶۱
شکل ۱۸ - برنامه امن طبق عدم تداخل حساس به زمان.....	۶۲
شکل ۱۹ - نمایی از محتوای بافرها در مکانیزم پیشنهادی در هنگام اجرای برنامه شکل ۱۸.....	۶۲
شکل ۲۰ - برنامه امن طبق عدم تداخل حساس به زمان.....	۶۳
شکل ۲۱ - نمایی از محتوای بافرها در مکانیزم پیشنهادی در هنگام اجرای برنامه شکل ۲۰.....	۶۳
شکل ۲۲ - برنامه ناامن طبق خط‌مشی عدم تداخل حساس به زمان.....	۶۴
شکل ۲۳ - نمایی از محتوای بافرها در مکانیزم پیشنهادی در هنگام اجرای برنامه شکل ۲۲.....	۶۴
شکل ۲۴ - نحو دستورات موجود در زبان برنامه‌نویسی مدل.....	۶۷
شکل ۲۵ - معناشناخت کوتاه‌گام استاندارد زبان برنامه‌نویسی مدل.....	۷۰
شکل ۲۶ - عملیات خواندن از ورودی و نوشتن خروجی در بافر.....	۷۳
شکل ۲۷ - معناشناخت محلی چنداجرایی امن بافردار بدون گزارش نقض امنیت.....	۷۴
شکل ۲۸ - تابع تعیین اولویت زمان‌بندی سطوح امنیتی.....	۷۵

- شکل ۲۹ – معاشناخت قسمت بررسی و خروجی دادن در حالت عدم گزارش نقض امنیت ..... ۷۸
- شکل ۳۰ – معاشناخت سراسری برای چنداجزایی امن بافردار در حالت عدم گزارش نقض امنیت ..... ۷۸

صفحه

## فهرست جدول‌ها

- جدول ۱ - انواع استراتژی‌های زمان‌بندی و تضمین امنیت هر یک ..... ۳۵
- جدول ۲ - مقایسه مکانیزم‌های مبتنی بر چنداجرایی امن ..... ۴۲

## فصل اول

### مقدمه

## مقدمه

با گسترش روزافزون سامانه‌های کامپیوتری و ارتباطی، استفاده از نرم‌افزارهای گوناگون امری روزمره و فراگیر محسوب می‌شود. سهم بزرگی از این نرم‌افزارها را که اغلب برنامه‌های سیار هستند، برنامه‌های غیرقابل اعتماد تشکیل می‌دهد. منظور از غیرقابل اعتماد بودن برنامه آن است که کاربر به دلیل در اختیار نداشتن کد منبع برنامه، اطمینانی به امنیت آن نخواهد داشت. اگرچه برنامه غیرقابل اعتماد می‌تواند بدافزار باشد، اما ممکن است خطای ناآگاهانه برنامه‌نویس منجر به ایجاد آسیب‌پذیری در برنامه شود و مهاجم بتواند از آن سوء استفاده کند. در هر صورت، کاربران نیازمند روشی برای اطمینان از برآورده شدن امنیت این گونه برنامه‌ها در زمان اجرا هستند.

امنیت اطلاعات در سه محور محرمانگی<sup>۱</sup>، صحت<sup>۲</sup> و دسترس‌پذیری<sup>۳</sup> قابل بحث است. برنامه‌هایی که در یک سامانه اجرا می‌شوند، باید خواسته‌های امنیتی مورد نظر را برآورده کنند. این خواسته‌های امنیتی در قالب خط‌مشی‌های امنیتی<sup>۴</sup> بیان می‌شوند. دسته‌ای از خط‌مشی‌های امنیتی، خط‌مشی‌های جریان اطلاعات<sup>۵</sup> هستند. کنترل جریان اطلاعات در دو بُعد محرمانگی و صحت مطرح می‌شود. در بُعد محرمانگی، هدف از کنترل جریان اطلاعات، کسب اطمینان از عدم وجود جریان‌های اطلاعات ناامن از سطوح محرمانه (بالا) به سایر سطوح (پایین) است. از منظر صحت، یعنی اطمینان از صحیح بودن اطلاعات و عدم دستکاری غیرمجاز آن‌ها، هدف از کنترل جریان اطلاعات کسب اطمینان از عدم وجود جریان اطلاعات از سطوح غیرقابل اعتماد به سطوح قابل اعتماد است. به بیان دیگر، نباید جریان اطلاعات از سطح پایین به بالا باشد. لازم به ذکر است که سطح‌بندی امنیتی در دو بُعد محرمانگی و صحت متفاوت است.

تعریف جریان اطلاعات امن در یک سامانه اطلاعاتی توسط خط‌مشی جریان اطلاعات صورت

---

<sup>1</sup> Confidentiality

<sup>2</sup> Integrity

<sup>3</sup> Availability

<sup>4</sup> Security Policy

<sup>5</sup> Information Flow Policies

می‌پذیرد. در بیشتر این گونه خط‌مشی‌ها، عدم‌تداخل<sup>۶</sup> به عنوان معناشناخت<sup>۷</sup> خط‌مشی مورد نظر است. عدم تداخل در حالت کلی بیان می‌کند در صورتی که ورودی‌های عمومی یک برنامه تغییر نکنند، رفتار قابل مشاهده در سطح عمومی تغییر نمی‌کند. به عبارت دیگر، در این صورت جریان اطلاعات از سطح محرمانه به سطح عمومی وجود ندارد. اینکه کدام اطلاعات در سطح محرمانه و کدام یک در سطح عمومی هستند، توسط خط‌مشی جریان اطلاعات مشخص می‌شود.

با بهره‌گیری از مکانیزم‌های اعمال خط‌مشی‌های امنیتی، می‌توان از برقراری خط‌مشی‌های جریان اطلاعات اطمینان حاصل کرد. به طور کلی، مکانیزم‌های اعمال خط‌مشی‌های امنیتی به دسته‌های ایستا، پویا و بازنویسی برنامه<sup>۸</sup> تقسیم می‌شوند. در مکانیزم‌های ایستا قبل از اجرا، کد منبع برنامه تحلیل، و در خصوص برقراری یا عدم برقراری خط‌مشی مورد نظر تصمیم‌گیری می‌شود. این مکانیزم‌ها اجازه اجرای برنامه ناامن را نمی‌دهند. در تحلیل ایستا، با در اختیار داشتن کد منبع برنامه، اطلاعاتی درباره اجراهای مختلف برنامه به دست می‌آید. با داشتن تخمینی از اجراهای مختلف برنامه، می‌توان خط‌مشی جریان اطلاعات را اعمال کرد. بدیهی است که به دلیل عدم دسترسی به اطلاعات زمان اجرای برنامه‌ها، مکانیزم‌های ایستا به طور محافظه‌کارانه عمل می‌کنند و ممکن است بعضی از برنامه‌های امن نیز توسط این مکانیزم‌ها ناامن تشخیص داده شده و به مرحله اجرا نرسند. از این رو میزان کامل بودن<sup>۹</sup> مکانیزم‌های ایستا، به معنای دقیق بودن افزاز برنامه‌های امن و ناامن، در کم‌ترین مقدار ممکن است. بیشتر مکانیزم‌های تحلیل ایستا نوع مبنا<sup>۱۰</sup> [۱] هستند. در این مکانیزم‌ها، با طراحی یک نوع سامانه<sup>۱۱</sup> تضمین برقراری خط‌مشی جریان اطلاعات انجام می‌شود. انواع دیگری از مکانیزم‌های تحلیل ایستا عبارتند از استفاده از گراف‌های وابستگی کنترل<sup>۱۲</sup> و تحلیل جریان داده، تفسیر انتزاعی<sup>۱۳</sup> و واریسی مدل<sup>۱۴</sup>.

---

<sup>۶</sup> Noninterference

<sup>۷</sup> Semantics

<sup>۸</sup> Program Rewriting

<sup>۹</sup> Completeness

<sup>۱۰</sup> Type-based

<sup>۱۱</sup> Type System

<sup>۱۲</sup> Control Dependence Graph

<sup>۱۳</sup> Abstract Interpretation

دسته دیگر مکانیزم‌های اعمال خط‌مشی‌های امنیتی، مکانیزم‌های پویا هستند. برخلاف مکانیزم‌های ایستا، این مکانیزم‌ها در زمان اجرای برنامه، با نظارت بر اجرای آن و با مشاهده رویدادهای امنیتی برنامه، خط‌مشی امنیتی مورد نظر را اعمال می‌کنند. یکی از مهم‌ترین انواع مکانیزم‌های پویا، ناظرهای زمان اجرا<sup>۱۵</sup> هستند. از آنجایی که ناظرهای زمان اجرا تنها به یک اجرا از برنامه دسترسی دارند، برای اعمال خط‌مشی‌های جریان اطلاعات مناسب نیستند. زیرا، خط‌مشی‌های جریان اطلاعات به صورت گزاره‌هایی روی بیش از یک اجرا تعریف می‌شوند. از طرف دیگر، مکانیزم‌های پویا می‌توانند علاوه بر دسترسی به اطلاعات زمان اجرای برنامه، به اطلاعات تحلیل کد منبع برنامه نیز دسترسی داشته باشند. به این ترتیب، امکان تضمین درستی به اندازه مکانیزم‌های تحلیل ایستا برای خط‌مشی‌های جریان اطلاعات فراهم می‌شود [۲]. مزیت دیگر این مکانیزم‌ها، پذیرفتن برنامه‌هایی است که اگرچه طبق خط‌مشی مورد نظر امن بوده‌اند، اما مکانیزم‌های تحلیل ایستا به دلیل رویکرد محافظه‌کارانه، آن‌ها را امن نمی‌دانستند. به این ترتیب، بعضی از برنامه‌ها که توسط مکانیزم‌های ایستا پذیرفته نمی‌شوند، با بهره‌گیری از مکانیزم‌های پویا امن شناخته می‌شوند. تحلیل کد منبع برنامه و چگونگی آن در زمان اجرا، باعث ایجاد تفاوت در میزان کامل بودن این مکانیزم‌ها با یکدیگر می‌شود. با توجه به مزایای مکانیزم‌های پویا، در سال‌های اخیر توجه بیشتری به این روش‌ها شده است. با این حال، عیب اصلی مکانیزم‌های پویا تحمیل سربار اضافی در زمان اجرای برنامه است.

دسته سوم انواع مکانیزم‌ها، بازنویسی برنامه است [۳، ۴]. در این دسته از مکانیزم‌ها، مشابه روش‌های نظارت بر اجرا، همه برنامه‌ها به مرحله اجرا خواهند رسید. با این تفاوت که در این مکانیزم‌ها، برنامه غیرقابل اعتماد قبل از اجرا، با هدف اعمال خط‌مشی امنیتی مورد نظر تغییر داده می‌شود و کد منبع برنامه، با توجه به الگوریتم بازنویسی اصلاح می‌شود. از دیدگاهی می‌توان بازنویسی برنامه را به صورت تعمیمی از روش‌های نظارت بر اجرا دانست. در صورتی که مداخله ناظر زمان اجرا به صورت تغییر در برنامه باشد، به نوعی بازنویسی برنامه انجام می‌گیرد. اما از لحاظ این‌که بازنویسی برنامه در زمان کامپایل برنامه صورت می‌گیرد و با استفاده از تحلیل‌های ایستا، و نه اطلاعات زمان اجرا، کد منبع برنامه تغییر می‌کند و پس از آن، کد اصلاح‌شده به مرحله اجرا می‌رسد، می‌توان گفت زمان اعمال در این

<sup>14</sup> Model Checking<sup>15</sup> Runtime Monitors



مکانیزم‌ها، زمان اجرا است. به همین دلیل می‌توان این دسته از روش‌ها را در دسته‌ای خارج از روش‌های مکانیزم ایستا و پویا قرار داد.

اگرچه پیشرفت‌های قابل توجهی در رویکردهای ایستا و پویا صورت گرفته است، اما محدودیت‌های اساسی برای این روش‌ها وجود دارد. به عنوان مثال، خط‌مشی عدم تداخل، به معنای عدم وجود جریان اطلاعات از ورودی‌های محرمانه به خروجی‌های عمومی، تصمیم‌پذیر نیست و همچنین نشان داده شده است [۵] که نظارت اجرا نمی‌تواند عدم تداخل را با دقت اعمال کند. به همین دلیل، ارائه مکانیزمی برای اعمال خط‌مشی‌های جریان اطلاعات که درست<sup>۱۶</sup>، دقیق<sup>۱۷</sup> و شفاف<sup>۱۸</sup> باشد، کماکان از حوزه‌های پژوهشی محسوب می‌شود.

## ۱-۱ تعریف مسئله

همان‌طور که اشاره شد، خط‌مشی‌های جریان اطلاعات به عنوان دسته مهمی از خط‌مشی‌های امنیتی به شمار می‌روند. اغلب این خط‌مشی‌ها را می‌توان با صورتی از عدم تداخل تعبیر نمود. به این ترتیب، طراحی مکانیزم‌هایی برای اعمال خط‌مشی عدم تداخل که درست و شفاف باشند، اهمیت بسزایی دارد. از همین رو و با توجه به محدودیت‌های ذکر شده برای رویکردهای ایستا و پویای پیشین، روشی به نام چنداجرای امن (SME)<sup>۱۹</sup> [۶] پیشنهاد شده است. ایده اصلی این روش آن است که چندین رونوشت از برنامه را که هر رونوشت متناظر با یکی از سطوح امنیتی است، به طور همزمان با استفاده از زمان‌بندهای اختصاصی اجرا می‌کنند. در هر رونوشت تغییراتی در ورودی‌ها و خروجی‌ها در نظر گرفته می‌شود تا از عدم وجود جریان اطلاعات ناامن اطمینان حاصل شود. خروجی‌های مربوط به یک سطح امنیتی فقط در رونوشت متناظر با همان سطح امنیتی تولید می‌شوند و در کانال‌های خروجی قرار می‌گیرند. از طرفی، ورودی‌های مربوط به سطوح امنیتی غیر از سطح امنیتی رونوشت و پایین‌تر از آن، با مقداری پیش‌فرض جایگزین می‌شوند. همچنین، ورودی‌های سطوح پایین‌تر مجدداً توسط رونوشت‌های

<sup>16</sup> Sound

<sup>17</sup> Precise

<sup>18</sup> Transparent

<sup>19</sup> Secure Multi-Execution

سطوح بالا مورد استفاده قرار می گیرند. به این ترتیب، ورودی‌های هر رونوشت مطابق با خطمشی عدم تداخل، برای سطوح بالاتر از سطح امنیتی رونوشت همواره یکسان و معادل با مقادیری از پیش تعیین شده خواهند بود. با توجه به این که خروجی‌ها نیز تنها در رونوشت مربوط به همان سطح خود تولید می‌شوند، هیچ گونه ارتباطی بین ورودی‌های محرمانه و خروجی‌های عمومی وجود نخواهد داشت.

روش چنداجرایی امن مزایای زیادی دارد. اول آن که هر برنامه تحت چنداجرایی امن عدم تداخل را برآورده خواهد کرد. بنابراین، مکانیزم‌های چنداجرایی امن درست هستند. می‌توان به سادگی مشاهده کرد که یک اجرا در یک سطح امنیتی مشخص، تنها در همان سطح خروجی می‌دهد و به هیچ یک از مقادیر واقعی ورودی سطوح بالاتر دسترسی ندارد. بنابراین، این امکان وجود نخواهد داشت که خروجی‌ها به مقادیر ورودی‌های سطوح بالاتر وابسته باشند. مزیت دیگر مکانیزم‌های چنداجرایی امن، دقت آن‌ها است. اگر یک برنامه تحت اجرای عادی عدم تداخل حساس به خاتمه را برآورده کند، رفتار آن برنامه تحت چنداجرایی امن و اجرای استاندارد خاتمه‌پذیر یکسان خواهد بود.

پیشتر مطرح شد که روش‌های ایستا و پویا نمی‌توانند همزمان درستی و دقت را تضمین کنند. با وجود این، مکانیزم‌های چنداجرایی امن می‌توانند عدم تداخل حساس به خاتمه را درست و دقیق اعمال کنند. منظور از دقیق بودن آن است که مکانیزم اجراهای خاتمه‌پذیر از هر برنامه که عدم تداخل حساس به خاتمه را برآورده می‌کنند، بدون تغییر حفظ می‌کند.

یکی از معایب اصلی چنداجرایی امن، هزینه سربار زمان پردازنده و حافظه است که در پیاده‌سازی موازنه‌ای بین آن‌ها وجود دارد. از طرف دیگر، می‌توان گفت که چنداجرایی امن از خطمشی عدم تداخل یک برنامه، برای موازی‌سازی خودکار اجرای آن استفاده می‌کند. پس از معرفی چنداجرایی امن، پژوهشگران به بررسی نقاط قوت و ضعف آن پرداختند و سعی در برطرف کردن معایب آن داشتند که در فصل سوم، به بیان جزئیات و مقایسه آن‌ها خواهیم پرداخت.

یکی از تعابیر خطمشی عدم تداخل، عدم تداخل حساس به زمان<sup>۲۰</sup> است. این تعبیر عدم وجود کانال‌های نهان<sup>۲۱</sup> زمانی را تضمین می‌کند. به عبارت دیگر، زمان‌بندی برنامه نباید بر مشاهدات مهاجم تاثیر بگذارد. مطابق این خطمشی، برنامه‌ای امن محسوب می‌شود که داده‌ای با سطح امنیتی بالا روی

<sup>20</sup> Timing-Sensitive Noninterference

<sup>21</sup> Covert Channel

زمان اجرای بخشی از برنامه که در سطح عمومی قابل مشاهده است تأثیرگذار نباشد [۷]. در حالت کلی، کانال‌های نهان زمانی را می‌توان به دو دسته داخلی و خارجی دسته‌بندی کرد. در صورتی که اختلاف زمان اجرای برنامه که ناشی از مقادیر محرمانه است بر روی مقادیر عمومی تأثیر بگذارد، برنامه دارای کانال نهان زمانی داخلی است. اما اگر مهاجم با داشتن یک زمان‌سنج خارجی بتواند چیزی از اطلاعات محرمانه موجود در برنامه متوجه شود، برنامه دارای کانال نهان زمانی خارجی است.

اگرچه مکانیزم‌های مبتنی بر روش چنداجرایی امن برای خط‌مشی عدم تداخل حساس به زمان نیز راهکارهایی پیشنهاد داده‌اند، اما هیچ‌یک از آن‌ها شفاف کامل<sup>۲۲</sup> نیست. منظور از شفاف کامل آن است که اگر برنامه‌ای طبق تعبیر عدم تداخل حساس به زمان امن شناخته شود، ترتیب مشاهده رویدادهای خروجی در کانال‌های مختلف نسبت به یکدیگر مشابه ترتیب رویدادهای خروجی در اجرای استاندارد برنامه باشد. مکانیزم‌هایی که تاکنون پیشنهاد شده‌اند و از روش چنداجرایی امن برای اعمال امنیت جریان اطلاعات استفاده می‌کنند ممکن است ترتیب خروجی‌ها را بین کانال‌های مختلف در اجراهای یک برنامه امن تغییر دهند. به این ترتیب، مشاهده‌گر سطح بالا، که امکان مشاهده همه کانال‌های خروجی را دارد، بین اجرای یک برنامه امن تحت مکانیزم امنیتی و اجرای عادی آن تفاوت قائل خواهد شد. این موضوع به معنای عدم دستیابی به شفافیت کامل است.

مسئله پژوهشی این پایان‌نامه طراحی مکانیزمی مبتنی بر روش چنداجرایی امن است که بتواند شفافیت کامل را برای برنامه‌های امن، به تعبیر خط‌مشی عدم تداخل حساس به زمان، فراهم آورد. بنابراین، علاوه بر تضمین امنیت، باید مکانیزم پیشنهادی ترتیب رویدادهای خروجی یک برنامه امن را در کانال‌های مختلف نسبت به یکدیگر حفظ کند. در این پایان‌نامه، برای پاسخ به سوال پژوهشی مطرح‌شده، مکانیزم جدیدی ارائه می‌شود که عدم تداخل حساس به زمان را با شفافیت کامل اعمال می‌کند.

<sup>22</sup> Full Transparent

## ۱-۲ ایده اصلی در مکانیزم پیشنهادی

از آنجایی که در روش چنداجرایی امن هیچ ورودی‌ای با سطح امنیتی بالاتر از سطح امنیتی متناظر با یک رونوشت برنامه به آن رونوشت وارد نمی‌شود، امنیت تضمین می‌شود. اما برای فراهم آوردن شفافیت کامل، لازم است که رویدادهای خروجی حاصل از اجرای یک برنامه امن همان ترتیبی را در کانال‌های مختلف نسبت به یکدیگر داشته باشند که در اجراهای استاندارد برنامه دارند. با توجه به رویکرد چنداجرایی امن، در صورتی که اجرای یک رونوشت منجر به تولید رویداد خروجی شود، بلافاصله در کانال خروجی قرار می‌گیرد. از این رو، حفظ ترتیب خروجی‌ها در بین کانال‌های خروجی میسر نیست. مکانیزم ارائه شده در این پایان‌نامه از بافر برای نگه‌داری رویدادهای خروجی هر رونوشت، پیش از قرارگیری در کانال‌های خروجی استفاده می‌کند. به این ترتیب، می‌توان رویدادهای خروجی را با همان ترتیب اجرای اصلی در کانال‌های خروجی قرار داد. مکانیزم مطرح شده علاوه بر حل مسئله اصلی، با بهره‌گیری از ایده‌های موجود در سایر مکانیزم‌های مبتنی بر چنداجرایی، نقاط قوت آن‌ها را نیز در خود جای داده است.

## ۱-۳ ساختار مطالب

ساختار ادامه مطالب این پایان‌نامه به این شرح است: در فصل دوم مفاهیم و اصول اولیه مورد نیاز برای بیان خط‌مشی‌های جریان اطلاعات و مکانیزم‌های اعمال آن‌ها مطرح می‌شود. فصل سوم به مرور کارهای گذشته و مرتبط اختصاص دارد و پیشنهاد‌های موجود برای بهبود روش چنداجرایی امن مورد مقایسه قرار می‌گیرند. در فصل چهارم، مکانیزم پیشنهادی را مطرح کرده و با ذکر مثال‌هایی، نحوه عملکرد این مکانیزم را شرح می‌دهیم. فصل پنجم در خصوص صوری‌سازی معناساخت مکانیزم، بیان صوری خط‌مشی‌ها، و اثبات درستی و شفافیت خواهد بود. در فصل ششم، پس از جمع‌بندی مطالب، پیشنهاد‌هایی برای کارهای آینده مطرح می‌شود.

## فصل دوم

### مفاهیم اولیه

## مفاهیم اولیه

در این فصل، مفاهیم اولیه و تعاریف مورد نیاز در ادامه پایان نامه به طور مختصر شرح داده می شود. ابتدا با بیان تعریف خاصیت و فوق خاصیت امنیتی به تقسیم بندی خطمشی های امنیتی می پردازیم و به دنبال آن، مفهوم مکانیزم اعمال امنیت مشخص می شود. پس از آن، امنیت جریان اطلاعات و خطمشی عدم تداخل تعریف و انواع گوناگون آن معرفی می شود.

### ۲-۱ تعریف خاصیت و فوق خاصیت امنیتی

خطمشی امنیتی تعریفی از امن بودن یک سامانه یا برنامه را ارائه می دهد که رفتارهای مجاز و غیرمجاز در آن مشخص می شود. به عنوان نمونه، یک خطمشی کنترل دسترسی تعیین می کند که چه کاربری به چه منبعی از سامانه حق دسترسی دارد و نوع حق دسترسی نیز مشخص می شود. اگر در یکی از اجراهای برنامه دسترسی برخلاف آنچه خطمشی مشخص کرده است داده شود، آن برنامه خطمشی را نقض کرده است و آن اجرا ناامن تلقی می شود. با وجود این، همه خطمشی های امنیتی مانند کنترل دسترسی نیستند که بتوان برای آن ها اجرای امن و ناامن تعریف کرد.

در حالت کلی، یک سامانه به شکل مجموعه ای از اجراهای ممکن آن سامانه تعریف می شود و خطمشی امنیتی نیز گزاره ای روی مجموعه اجراها است. به عبارت دقیق تر، یک خطمشی به صورت خانواده ای از مجموعه های محاسبه پذیر از اجراها تعریف می شود. در صورتی که خطمشی بیانگر خاصیتی روی اجراهای منفرد برنامه باشد، می توان برای آن خطمشی مجموعه اجراهای امن تعریف کرد. به این ترتیب، برنامه ای که اجراهای آن زیرمجموعه ای از مجموعه اجراهای امن باشد، برنامه ای امن محسوب می شود. در این صورت خطمشی یک خاصیت<sup>۲۳</sup> را توصیف می کند.

دسته دیگری از خطمشی های امنیتی خواسته هایی هستند که به شکل یک خاصیت قابل تعریف نیستند. در بسیاری از خواسته های امنیتی لازم است که اجراهای مختلف برنامه رابطه مشخصی با

<sup>23</sup> Property

یکدیگر داشته باشند. این گونه خطمشی‌ها، فوق‌خاصیت<sup>۲۴</sup> نامیده می‌شوند [۸]. به طور خاص، در رابطه با امنیت جریان اطلاعات، مهاجم ممکن است با مشاهده یک اجرا و دانشی که در مورد اجراهای دیگر دارد، در مورد داده‌های محرمانه اطلاعاتی پیدا کند. بنابراین، امنیت جریان اطلاعات یک فوق‌خاصیت امنیتی است.

همان‌طور که پیشتر عنوان شد، خطمشی عدم تداخل به عنوان معناشناخت امنیت جریان اطلاعات مطرح می‌شود. طبق تعریف، یک برنامه عدم تداخل را برآورده می‌کند اگر هیچ دو اجرایی با مقادیر ورودی عمومی یکسان، که ممکن است در مقادیر ورودی محرمانه متفاوت باشند، خروجی‌های عمومی متفاوتی نداشته باشند. مشخص است که این خطمشی خاصیت نیست؛ یعنی با گزاره‌ای روی تک اجراهای برنامه قابل بیان نیست و با رابطه‌ای دوتایی روی اجراهای برنامه تعریف می‌شود. به همین خاطر، برای اعمال فوق‌خاصیت‌هایی مانند عدم تداخل باید روشی متفاوت از نحوه اعمال خاصیت‌ها پیشنهاد کرد.

## ۲-۲ مکانیزم‌های امنیتی

مکانیزم اعمال<sup>۲۵</sup> یا مکانیزم امنیتی عبارت است از روش، ابزار یا رویه‌ای برای اعمال یک خطمشی امنیتی داده شده [۹]. مکانیزم‌های اعمال خطمشی‌های جریان اطلاعات به دنبال دستیابی به اهداف زیر هستند [۱۰]. اولین هدف درست‌بودن مکانیزم است؛ به این معنا که جریان اطلاعات غیرمجاز در اجراهای برنامه‌ای که تحت این مکانیزم اجرا می‌شود وجود ندارد. دومین هدف شفافیت است. مکانیزمی که با کمترین تغییر در مجموعه اجراهای برنامه‌های اصلی خطمشی را اعمال می‌کند، شفاف نامیده می‌شود. همچنین، مکانیزم باید عملی<sup>۲۶</sup> بوده و هزینه اعمال آن قابل قبول باشد. این هزینه‌ها ممکن است مربوط به زمان توسعه<sup>۲۷</sup>، استقرار<sup>۲۸</sup> و یا اجرای برنامه باشد. اگرچه تلاش‌های بسیاری در دهه‌های

<sup>24</sup> Hyperproperty

<sup>25</sup> Enforcement Mechanism

<sup>26</sup> Practical

<sup>27</sup> Development

<sup>28</sup> Deployment

اخیر برای پاسخ به این مسئله انجام شده است، اما کماکان مکانیزم‌هایی که به طور همزمان به همه این اهداف دست‌یابند وجود ندارند.

## ۳-۲ امنیت جریان اطلاعات

جلوگیری از افشای اطلاعات محرمانه به موجودیت‌های (پردازه<sup>۲۹</sup>ها و حافظه‌ها) غیرمجاز طبیعی-ترین خواسته امنیتی است. افشای غیرمجاز اطلاعات بیشتر به صورت وجود جریان اطلاعات از موجودیت‌های حساس (با سطح امنیتی بالا) به موجودیت‌های عمومی (با سطح امنیتی پایین) تعبیر می‌شود. بنابراین، خط‌مشی‌های جریان اطلاعات، که خواسته‌های امنیتی را در قالب جریان‌های مجاز و غیرمجاز اطلاعات بیان می‌کنند، یک شبکه<sup>۳۰</sup> از سطوح امنیتی [۱۱] در نظر گرفته و به هر یک از موجودیت‌های سامانه یک سطح امنیتی نسبت می‌دهند. به این ترتیب، بخشی از توصیف خواسته‌ها صورت می‌گیرد و طبق آن جریان اطلاعات فقط در صورتی مجاز است که از موجودیتی پایین‌تر، مطابق با شبکه سطوح امنیتی، به موجودیتی بالاتر باشد. این توصیف زمانی کامل می‌شود که تعبیری روشن و دقیق از مفهوم جریان اطلاعات بین موجودیت‌ها وجود داشته باشد. چنین تعبیری به عنوان معناشناخت یک خط‌مشی جریان اطلاعات شناخته می‌شود. پژوهش‌های انجام‌شده در حوزه امنیت جریان اطلاعات حاکی از وجود تلاش‌های فراوان برای دستیابی به معناشناختی مؤثر است به گونه‌ای که از یک طرف مبتنی بر جریان‌های داده و کنترل در برنامه‌ها، و از طرفی مبتنی بر توانایی‌های مهاجم باشد.

اولین تعریف صوری از جریان اطلاعات در [۱۲] ارائه شده است. در این تعریف، مفهوم وابستگی قوی<sup>۳۱</sup> بین ورودی  $i$  و خروجی  $o$  چنین بیان می‌شود که در پردازش  $p$  یک جریان اطلاعات از ورودی  $i$  به خروجی  $o$  وجود دارد هرگاه اجرای  $p$  تغییر در  $i$  را به  $o$  منتقل کند. در سامانه‌های قطعی<sup>۳۲</sup>، وابستگی قوی بین ورودی  $i$  و خروجی  $o$  وجود دارد اگر و تنها اگر دو اجرا از  $p$  وجود داشته باشند که همه ورودی‌های آن‌ها به جز  $i$  با یکدیگر یکسانند ولی خروجی‌های آن‌ها در  $o$  با یکدیگر متفاوتند [۲].

<sup>29</sup> Process

<sup>30</sup> Lattice

<sup>31</sup> Strong Dependency

<sup>32</sup> Deterministic



از آن جا که محرمانگی و صحت دوگان یکدیگر هستند [۹]، می توان خواسته های مربوط به صحت را نیز در قالب خطمشی های جریان اطلاعات بیان نمود. با وجود این، بیشتر تلاش ها در توصیف و اعمال خطمشی های جریان اطلاعات به عنوان توصیفی از محرمانگی صورت گرفته است. همچنین، اگرچه به طور مرسوم برای بیان خواسته های امنیتی مربوط به محرمانگی اغلب از خطمشی های کنترل دسترسی استفاده می شود، این خطمشی ها تضمین نمی کنند که نحوه انتشار اطلاعات حساس منجر به نشت اطلاعات نشود و صرفاً دسترسی به اطلاعات حساس را محدود می کنند.

## ۲-۴ عدم تداخل

با الهام از وابستگی قوی، مفهوم عدم تداخل [۱۳] به عنوان تعبیری از امنیت جریان اطلاعات ارائه شده است. بر اساس این تعبیر یک گروه از کاربران با استفاده از مجموعه دستورات مشخص با گروه دیگری از کاربران عدم تداخل دارند هرگاه آن چه آن کاربران با آن دستورات انجام دهند، هیچ اثری بر روی آن چه کاربران گروه دیگر مشاهده می کنند نداشته باشد. به بیان صوری، گروه  $G$  از کاربران با گروه  $G'$  عدم تداخل دارد اگر هر دنباله ای از دستورات که به سامانه وارد می شود همان اثری را بر روی کاربران  $G'$  داشته باشد که دنباله پالایش شده از دستورات گروه  $G$  خواهد داشت. منظور از پالایش<sup>۳۳</sup> کنارگذاشتن دستورات صادر شده توسط کاربران گروه  $G$  از دنباله دستورات است. در صورتی که به جای دنباله دستورات دنباله ورودی ها به و خروجی ها از سامانه را در نظر بگیریم، عدم تداخل با استفاده از مفهوم دنباله اجرا<sup>۳۴</sup> چنین خواهد بود: اگر در یک دنباله اجرای سامانه، که به صورت دنباله ای از ورودی ها و خروجی های سطح بالا و پایین است، حذف ورودی های سطح بالا به دنباله اجرای دیگری از آن سامانه تبدیل شود، کاربران سطح بالا با کاربران سطح پایین عدم تداخل دارند.

روشن است که چنین تعبیری از امنیت بسیار سخت گیرانه است. طبق این تعبیر، رمزنگاری ایده آل نیز عدم تداخل را برآورده نمی کند. این موضوع به دلیل آن است که عدم تداخل هر گونه تأثیر ورودی های سطح بالا بر خروجی های سطح پایین را غیرمجاز می داند. بنابراین، تعبیرهای دیگری از عدم

<sup>۳۳</sup> Purge

<sup>۳۴</sup> Trace

تداخل ارائه شده‌اند که عدم قابلیت استنتاج<sup>۳۵</sup> [۱۴] اطلاعات محرمانه از مشاهدات عمومی را جایگزین عدم تأثیر کرده‌اند. عدم قابلیت استنتاج خواستار سازگاری مشاهدات کاربران سطح پایین با هر آن چیزی است که کاربران سطح بالا انجام می‌دهند. به بیان دیگر، برای هر دو دنباله اجرای  $T$  و  $S$ ، باید دنباله اجرای  $R$  وجود داشته باشد که شامل رویدادهای سطح پایین  $T$ ، با همان ترتیب، و رویدادهای ورودی سطح بالای  $S$ ، با همان ترتیب باشد و نیز در آن (احتمالاً) رویدادهای دیگری وجود داشته باشد که نه از رویدادهای سطح پایین  $T$  و نه از رویدادهای ورودی سطح بالای  $S$  هستند. لازم به ذکر است که گونه‌های مختلف عدم تداخل، چه اولین تعبیر [۱۳] و چه گونه‌های اصلاح‌شده و تعمیم‌یافته آن، همگی با نام کلی عدم تداخل شناخته می‌شوند.

انتظار می‌رود عدم تداخل معناساختی برای این موضوع باشد که یک مهاجم یا مشاهده‌گر سطح پایین<sup>۳۶</sup>، که رویدادهای عمومی اجراهای مختلف را مشاهده می‌کند، نتواند هیچ اطلاعی از ورودی‌های محرمانه و دارای سطح امنیتی بالای برنامه کسب کند. تفاوت در انواع چنین معناساختی به دو موضوع باز می‌گردد: تفاوت در معناساخت سامانه و تفاوت در توانایی‌های مهاجم. این که سامانه قطعی است یا غیر قطعی، این که سامانه در طول اجرا امکان تعامل با محیط را دارد یا ندارد، این که آیا سطوح امنیتی موجودیت‌ها ثابت است یا امکان تغییر آن‌ها در حین اجرا وجود دارد و نکاتی از این دست، منجر به ارائه انواع مختلفی از عدم تداخل شده است. همچنین، توانایی‌های گوناگون فرض‌شده درباره مهاجم اعم از این که آیا مهاجم می‌تواند مقادیر متغیرها را قبل از خاتمه‌یافتن برنامه مشاهده کند، این که آیا مهاجم بین خاتمه‌یافتن اجرا و واگرایی خاموش<sup>۳۷</sup> برنامه (بدون تولید خروجی) تمایز قائل می‌شود، این که آیا مهاجم می‌تواند تفاوت در زمان اجرای برنامه را درک کند، و مانند این‌ها نیز در ارائه انواع مختلف عدم تداخل تأثیرگذار بوده است.

آنچه در انواع خط‌مشی‌های جریان اطلاعات متفاوت است بیشتر مربوط به معناساخت خط‌مشی یا همان عدم تداخل در نظر گرفته‌شده است. به همین دلیل، اصطلاح خط‌مشی عدم تداخل اغلب به جای خط‌مشی جریان اطلاعات به کار می‌رود. موضوع دیگر آن است که در امنیت جریان اطلاعات فرض

<sup>35</sup> Nondeducibility

<sup>36</sup> Low Observer

<sup>37</sup> Silent Divergence

می‌شود مهاجم سامانه مورد نظر را کاملاً می‌شناسد. به عبارت دیگر، کد منبع برنامه در اختیار اوست. آن‌چه مهاجم نمی‌داند ورودی‌های (سطح بالای) برنامه است [۱۵]. همچنین، مشاهدات مهاجم ناشی از وجود جریان‌های صریح<sup>۳۸</sup> و ضمنی<sup>۳۹</sup> اطلاعات و نیز کانال‌های نهان است. به جریان حاصل از انتساب<sup>۴۰</sup> متغیری به متغیر دیگر، مانند دستور  $b:=c$ ، و نیز دستورات ورودی و خروجی جریان صریح گفته می‌شود. همچنین، جریان ضمنی جریانی است که اطلاعات را از طریق ساختارهای کنترلی برنامه منتقل می‌کند. به عنوان مثال، در صورتی که بخش شرط ساختاری مانند  $if$  به یک متغیر سطح بالا وابسته باشد، اجرا یا عدم اجرای دستورات در بدنه  $if$  به مقادیر سطح بالا وابسته خواهد بود. طبق تعریف موجود در [۱۱]، که در آن برای اولین بار اصطلاح امنیت جریان اطلاعات استفاده شده است، یک جریان ضمنی به متغیر  $b$  نتیجه اجرا یا عدم اجرای گزاره‌ای<sup>۴۱</sup> است که در آن گزاره، یک جریان صریح اطلاعات به متغیر  $b$  وجود داشته و گزاره به یک عبارت مشروط شده باشد. به عنوان نمونه، برنامه  $if a=0 \text{ then } b:=c$  دارای یک جریان ضمنی از  $a$  به  $b$  است خواه شرط  $a=0$  برقرار باشد یا نباشد. به این معنا که اجرا شدن یا نشدن انتساب  $c$  به  $b$  وابسته به مقدار متغیر  $a$  است.

## ۲-۵ انواع خط‌مشی‌های عدم تداخل

در ادامه، انواعی از خط‌مشی‌های عدم تداخل به طور مختصر معرفی می‌شوند.

- عدم تداخل حساس / غیرحساس به خاتمه<sup>۴۲</sup>: از نکات حائز اهمیت در خط‌مشی امنیتی، توانایی مهاجم در تشخیص واگرایی برنامه است. این توانایی می‌تواند کانال‌های نهان خاتمه را برای مهاجم آشکار کند. در صورتی که سامانه‌ای تک‌وهله‌ای<sup>۴۳</sup> باشد، که خروجی‌ها فقط در انتهای اجرای برنامه تولید می‌شوند، مفهوم خاتمه برنامه اهمیت می‌یابد. بنابراین، می‌توان خط‌مشی‌های

<sup>38</sup> Explicit

<sup>39</sup> Implicit

<sup>40</sup> Assignment

<sup>41</sup> Statement

<sup>42</sup> Termination-Sensitive/Insensitive Noninterference

<sup>43</sup> Batch-job

امنیتی را به دو گونه حساس به خاتمه (TSNI) و غیر حساس به خاتمه (TINI) دسته‌بندی کرد [۱۵]. در عدم تداخل حساس به خاتمه، باید دو اجرا از برنامه که ورودی‌های یکسان سطح پایین دارند، رفتار خاتمه‌ای یکسان داشته باشند؛ به این معنا که یا هر دو خاتمه یابند و خروجی‌های سطح پایین یکسان تولید کنند یا هر دو واگرا باشند و خاتمه نیابند. در عدم تداخل غیرحساس به خاتمه، مهاجم نمی‌تواند تمایزی بین واگرایی و خاتمه قائل باشد. به عنوان مثال، با این که برنامه شکل ۱ دارای کانال خاتمه است، اما عدم تداخل غیرحساس به خاتمه آن را امن می‌داند.

```
inH x ;    // high input : x
if x then while true do skip ;
else skip ;
outL y ;    // low output : y
```

شکل ۱ - برنامه دارای کانال خاتمه

- عدم تداخل آگاه از خاتمه<sup>۴۴</sup>: این تعبیر از عدم تداخل تعیین می‌کند که علاوه بر برنامه، وجود مکانیزم امنیتی باعث ایجاد کانال نشت اطلاعات نشود [۱۶]. عدم تداخل غیرحساس به خاتمه بین این که کانال خاتمه در برنامه وجود داشته یا به واسطه وجود مکانیزم ایجاد شده است، تفاوتی قائل نمی‌شود. حال آن که در عدم تداخل آگاه از خاتمه، مکانیزم باید رفتار یکسانی برای همه اجراهای با ورودی‌های سطح پایین یکسان، که اجرای برنامه اصلی خاتمه می‌یابد، داشته باشد. بنابراین، مکانیزم باید برای همه اجراها با تولید خروجی مشابه خاتمه یابد یا برای همه آن‌ها واگرا باشد. در حالی که اجرای برنامه اصلی همواره واگرا باشد، مکانیزم می‌تواند رفتار واگرایی را حفظ کند یا برای همه اجراها با تولید مقدار یکسانی خاتمه یابد. خط‌مشی آگاه از خاتمه (TANI) از نظر سخت‌گیرانه‌بودن در بین عدم تداخل حساس به خاتمه و غیرحساس به خاتمه قرار می‌گیرد.

<sup>44</sup> Termination-Aware Noninterference

- عدم تداخل حساس / غیرحساس به پیشروی<sup>۴۵</sup>: در برنامه‌های تعاملی<sup>۴۶</sup> که امکان تولید خروجی در هر گام از اجرا وجود دارد، مهاجم می‌تواند پیشروی محاسبه را نیز مشاهده کند. در خط‌مشی عدم تداخل حساس به پیشروی (PSNI) فرض بر این است که مشاهده‌گر سطح پایین قادر است بین این دو حالت تمایز قائل شود: حالتی که برنامه خاتمه نمی‌یابد و تولید خروجی هم ندارد (واگرای خاموش) و حالتی که برنامه در حال محاسبه مقدار خروجی بعدی است. در عدم تداخل غیرحساس به پیشروی (PINI)، مهاجم نمی‌تواند فرق بین واگرایی خاموش و خروجی‌های بعدی را تشخیص دهد.
- عدم تداخل حساس / غیرحساس به زمان<sup>۴۷</sup>: این دسته از خط‌مشی‌های امنیتی به کانال‌های زمانی توجه دارند که ممکن است باعث نشت اطلاعات حساس شوند. در صورتی که مقدار متغیر سطح بالا در زمان محاسبه متغیرهای سطح پایینی تأثیر داشته باشد، اصطلاحاً کانال نهان زمانی تشکیل شده است که اطلاعات حساس را به کانال‌های عمومی منتقل می‌کند. در عدم تداخل غیرحساس به زمان، همه اطلاعات مربوط به زمان و نحوه اجرای برنامه نادیده گرفته شده و تعبیر امنیت تعریف می‌شود. فرض موجود در این خط‌مشی آن است که مهاجم هیچ وسیله‌ای برای اندازه‌گیری زمان اجرای برنامه در اختیار ندارد. در عدم تداخل حساس به زمان، مهاجم می‌تواند زمان اجرای برنامه‌ها را اندازه‌گیری و با یکدیگر مقایسه کند. خط‌مشی حساس به زمان را می‌توان به دو نوع ضعیف<sup>۴۸</sup> و قوی<sup>۴۹</sup> دسته‌بندی کرد [۱۷]. در نوع ضعیف این خط‌مشی، زمان به صورت انتزاعی و بر اساس تعداد گام‌های اجرای دستورات برنامه محاسبه می‌شود. به این ترتیب، با هر گام از اجرای برنامه مقداری ثابت و مشخص به زمان برنامه افزوده می‌شود. در این حالت، جزئیات پیاده‌سازی و محیط اجرا در نظر گرفته نمی‌شوند. در عدم تداخل حساس به زمان قوی، مدت زمان واقعی اجرای برنامه مورد نظر است. هر دستور از برنامه می‌تواند مدت زمان متفاوتی داشته باشد. علاوه بر این، جزئیات پیاده‌سازی و به خصوص محیط اجرای برنامه

<sup>45</sup> Progress Sensitive/Insensitive Noninterference

<sup>46</sup> Interactive

<sup>47</sup> Timing Sensitive/Insensitive Noninterference

<sup>48</sup> Weakly Timing Sensitive Noninterference

<sup>49</sup> Strongly Timing Sensitive Noninterference

مانند حافظه نهان<sup>۵۰</sup> و خطلوله<sup>۵۱</sup> در مدت زمان اجرای برنامه محاسبه خواهند شد. این حالت سخت‌گیرانه‌ترین خطمشی کانال زمانی محسوب می‌شود و همه جزئیات محیط واقعی اجرا را در بر می‌گیرد. البته در مدل‌سازی‌های انجام‌شده اغلب از عدم تداخل حساس به زمان ضعیف استفاده شده است [۶، ۱۸]. در این پایان‌نامه نیز همین خطمشی مورد توجه قرار گرفته است.

برنامه‌های شکل ۲، بیانگر تفاوت بین دو حالت مطرح‌شده عدم تداخل حساس به زمان هستند. برنامه الف بنا بر تعریف نوع ضعیف، خطمشی امنیتی را برآورده نمی‌کند. چون در ساختار شرطی وابسته به مقدار سطح بالا، تعداد گام‌های اجرای هر شاخه متفاوت می‌شود. برنامه ب طبق خطمشی نوع ضعیف امن است. اگرچه از نظر تعداد گام‌های اجرای برنامه همواره تعداد ثابتی طی می‌شود اما به طور مثال، در صورتی که متغیر  $b$  در حافظه نهان وجود نداشته باشد، زمان اجرای برنامه در دو مسیر مختلف اجرا متفاوت خواهد شد. به این ترتیب، برنامه ب طبق تعبیر حساس به زمان قوی، برنامه امنی محسوب نمی‌شود.

<pre>inH x ;    // high input : x if x then a:=b ; else a:=c ; outL y ;   // low output : y</pre>	<pre>inH x ;    // high input : x if x then skip; skip ; else skip ; outL y ;   // low output : y</pre>
---	---

(ب)

(الف)

شکل ۲ – (الف) و (ب) نمونه‌هایی از برنامه‌های دارای کانال زمانی

- عدم تداخل حساس / غیرحساس به جریان<sup>۵۲</sup>: در خطمشی غیرحساس به جریان فرض بر این است که سطح امنیتی متغیرها بدون توجه به جریان برنامه مشخص می‌شود؛ یعنی سطح امنیتی متغیرها تا پایان اجرای برنامه ثابت می‌ماند. اما فرض موجود در عدم تداخل حساس به جریان این است که سطح امنیتی متغیرها با توجه به جریان برنامه تعیین می‌شود. در این نوع تحلیل،

<sup>50</sup> Cache<sup>51</sup> Pipelining<sup>52</sup> Flow Sensitive/Insensitive Noninterference

متغیرها می‌توانند سطوح امنیتی متفاوتی داشته باشند و با توجه به جریان برنامه، ارتقا<sup>۵۳</sup> و یا تنزل<sup>۵۴</sup> یابند که این رفتار ممکن است باعث ایجاد کانال نشت اطلاعات شود.

شایان ذکر است که انواع دیگری از عدم تداخل مانند تعمیم‌یافته<sup>۵۵</sup> (GNI) [۱۹]، احتمالاتی<sup>۵۶</sup> (PNI) [۲۰]، رابطه‌ای<sup>۵۷</sup> (RNI) [۸] و قطعیت مشاهده‌ای<sup>۵۸</sup> (OD) [۲۱] نیز در پژوهش‌های قبلی مطرح شده‌اند.

## ۲-۶ جمع‌بندی

در این فصل ابتدا با مروری بر مفهوم خاصیت امنیتی، یک تقسیم‌بندی در خصوص انواع خط‌مشی‌های امنیتی ارائه شد. پس از آن، مکانیزم‌های امنیتی به عنوان روش‌ها و ابزارهای اعمال یک خط‌مشی امنیتی تعریف شدند. درستی، شفافیت و عملی‌بودن از خواسته‌های اصلی مورد انتظار مکانیزم‌ها به شمار می‌روند. در ادامه امنیت جریان اطلاعات به عنوان یکی از مهم‌ترین دسته خط‌مشی‌های امنیتی مورد بحث قرار گرفت و مفهوم عدم تداخل به عنوان اصلی‌ترین خط‌مشی این حوزه تشریح شد. از انواع خط‌مشی‌های عدم تداخل به حساس به خاتمه، پیشروی، زمان و جریان اشاره شده و به طور خلاصه، تفاوت بین اعمال فوق‌خاصیت‌های عدم تداخل به عنوان مفهوم امنیت جریان اطلاعات و خاصیت‌های امنیتی مشخص شد.

<sup>53</sup> Upgrade

<sup>54</sup> Downgrade

<sup>55</sup> Generalized Noninterference

<sup>56</sup> Probabilistic Noninterference

<sup>57</sup> Relational Noninterference

<sup>58</sup> Observational Determinism

## فصل سوم

### کارهای مرتبط



## کارهای مرتبط

این فصل به مروری بر پژوهش‌های انجام‌شده مرتبط با پایان‌نامه اختصاص دارد. پس از معرفی روش‌های کلی اعمال خط‌مشی‌های امنیتی، انواع مکانیزم‌های پیشنهادشده مبتنی بر روش چنداجرایی امن مورد بررسی قرار می‌گیرند. در این بررسی، علاوه بر بیان کلی مکانیزم‌های موجود، نقاط قوت و ضعف هر یک تحلیل می‌شود.

### ۳-۱ روش‌های اعمال خط‌مشی امنیتی

یکی از معیارهای دسته‌بندی انواع مختلف روش‌های اعمال خط‌مشی‌های امنیتی، زمان اعمال‌شدن خط‌مشی است و این‌که یک مکانیزم می‌تواند از چه نوع اطلاعاتی برای اعمال استفاده کند. زمان اعمال یک خط‌مشی را می‌توان به قبل از اجرای برنامه یا زمان کامپایل<sup>۵۹</sup> و در طول اجرای برنامه یا زمان اجرا<sup>۶۰</sup> تقسیم‌بندی کرد. البته می‌تواند روشی ترکیبی وجود داشته باشد که در هر دو زمان برای اعمال خط‌مشی بهره گیرد. بدیهی است که در زمان کامپایل یا قبل از اجرای برنامه، نمی‌توان به اطلاعات زمان اجرای برنامه دسترسی داشت. پس مکانیزم تنها با استفاده از اطلاعات حاصل از تحلیل کد منبع برنامه درباره تضمین امنیت آن برنامه قضاوت می‌کند. در حالی‌که در طول اجرای برنامه، علاوه بر اطلاعات زمان اجرا ممکن است دسترسی به همه یا بخشی از کد منبع برنامه نیز وجود داشته باشد و به آسان‌گیری بیشتر مکانیزم در قبال برنامه‌های مختلف کمک کند.

با استفاده از مدل‌سازی برنامه به صورت ماشین برنامه<sup>۶۱</sup> (PM)، می‌توان مکانیزم‌های اعمال خط‌مشی‌های امنیتی را دسته‌بندی کرد [۳]. مطابق با این مدل‌سازی، روش‌های اعمال به دسته‌های تحلیل ایستا، تحلیل پویا، بازنویسی برنامه و تحلیل ترکیبی<sup>۶۲</sup> تقسیم می‌شوند. در ادامه، به طور خلاصه و براساس مدل مطرح‌شده هر یک از این روش‌ها معرفی می‌شوند.

<sup>۵۹</sup> Compile Time

<sup>۶۰</sup> Run Time

<sup>۶۱</sup> Program Machine

<sup>۶۲</sup> Hybrid Analysis

## ۳-۱-۱ مکانیزم‌های ایستا

مکانیزم‌های اعمالی که برنامه‌های غیرقابل اعتماد را پیش از اجرای آن‌ها رد<sup>۶۳</sup> یا قبول می‌کنند، در دسته مکانیزم‌های اعمال به کمک تحلیل ایستا قرار می‌گیرند. در این روش، مکانیزم اعمال باید در زمان متناهی برقراری خطمشی مورد نظر توسط برنامه را تشخیص دهد. پس از آن، برنامه‌های پذیرفته‌شده اجرا می‌شوند ولی از اجرای برنامه‌های ناامن جلوگیری می‌شود. به بیان صوری، در مدل ارائه‌شده [۳]، خطمشی امنیتی  $P$  قابل اعمال توسط مکانیزم تحلیل ایستا است اگر و تنها اگر ماشین تورینگ  $M_p$  وجود داشته باشد به طوری که ماشین برنامه  $PM$  را به عنوان ورودی دریافت کند و در صورت برآورده شدن  $P(M)$ ، برنامه را در زمان متناهی بپذیرد و در غیر این صورت، آن را در زمان متناهی رد کند. این مکانیزم‌ها می‌توانند تمامی خاصیت‌های تصمیم‌پذیر<sup>۶۴</sup> را اعمال کنند. البته می‌توان از این روش‌ها برای اعمال خطمشی‌های تصمیم‌ناپذیر - به شکل اعمال خطمشی‌های تصمیم‌پذیری که به طور محافظه‌کارانه تقریبی از آن‌ها هستند - نیز استفاده کرد. به عنوان نمونه، می‌دانیم که خطمشی «برنامه  $P$  سرانجام خاتمه خواهد یافت» تصمیم‌پذیر نیست. بنابراین، با مکانیزم‌های تحلیل ایستا قابل اعمال نخواهد بود. اما می‌توان تقریبی از این خطمشی را به صورت «برنامه  $P$  حداکثر پس از هزار عمل محاسباتی خاتمه خواهد یافت» در نظر گرفت که یک خاصیت تصمیم‌پذیر است. حال می‌توان این خاصیت تصمیم‌پذیر را با بهره‌گیری از روش‌های تحلیل ایستا اعمال کرد.

از جمله روش‌های تحلیل ایستا می‌توان به مکانیزم‌های نوع مبنا [۱] و مکانیزم‌های مبتنی بر راستی‌آزمایی<sup>۶۵</sup> [۲۲] اشاره کرد. این‌گونه مکانیزم‌ها درست هستند و هزینه‌ای در زمان اجرا یا استقرار تحمیل نمی‌کنند. با این حال مکانیزم‌های نوع مبنا شفاف نیستند و ممکن است به واسطه رویکرد محافظه‌کارانه آن‌ها، برنامه‌های امن زیادی توسط آن‌ها پذیرفته نشود. البته این دسته از روش‌های اعمال هزینه زمان توسعه زیادی خواهند داشت. از انواع دیگر این روش‌ها [۲۳] باید به تحلیل جریان داده، واریسی مدل و تفسیر انتزاعی اشاره کرد.

<sup>63</sup> Reject<sup>64</sup> Decidable<sup>65</sup> Verification-based

### ۳-۱-۲ مکانیزم‌های پویا

مکانیزم‌هایی که در طول زمان اجرای برنامه امنیت را تضمین می‌کنند، از جمله مکانیزم‌های پویا محسوب می‌شوند. قسمت مهمی از مکانیزم‌های این روش، مکانیزم‌های نظارت بر اجرا<sup>۶۶</sup> [۵، ۲۴] هستند. این دسته از روش‌ها عبارتند از مکانیزم‌هایی که در طول اجرای برنامه رویدادهای امنیتی برنامه را نظارت می‌کنند و در صورت نقض خط‌مشی امنیتی توسط اجرای فعلی برنامه، با مداخله در اجرا اقدام به اعمال خط‌مشی می‌کنند. در این روش‌ها فرض بر آن است که با اجرای هر گام از برنامه، انتزاعی از آن به صورت یک رویداد برای ناظر ارسال می‌شود و ناظر با بررسی این رویداد و در اختیار داشتن اطلاعات رویدادهای قبلی، در صورت عدم مغایرت رویدادهای مشاهده‌شده با خط‌مشی مورد نظر، آن را پذیرفته و به محیط اجرا ارسال می‌کند. توانایی ناظر در نگهداری رویدادهای قبلی، چگونگی ارسال رویدادها برای اجرا و میزان آگاهی از سایر اجراهای برنامه، به طور مستقیم با مجموعه خط‌مشی‌های قابل اعمال توسط آن مکانیزم در ارتباط است. اخیراً روش‌های دیگری مانند چنداجرای امن [۶] نیز مطرح شده است که با ایجاد تغییر در نحوه اجرای برنامه، در زمان اجرای برنامه امنیت را برآورده می‌کنند. با توجه به این که در تحلیل پویا فقط بخشی از رفتارهای برنامه در اختیار مکانیزم است، این روش‌ها نمی‌توانند کامل<sup>۶۷</sup> باشند؛ اما به خاطر دسترسی به رویدادهای زمان‌اجرا، دقیق هستند.

مکانیزم‌های نظارت بر اجرا را می‌توان در انواع مختلفی از جمله انتقال‌دهنده اجرا<sup>۶۸</sup> و تشخیص‌دهنده اجرا<sup>۶۹</sup> دسته‌بندی کرد [۲۵]. مکانیزم‌های نظارت بر اجرا به صورت تشخیص‌دهنده اجرا مکانیزم‌هایی هستند که فقط بر اجرای برنامه نظارت می‌کنند و در صورت نقض خط‌مشی یا بروز مشکل، اجرای برنامه را قطع<sup>۷۰</sup> می‌کنند. در واقع، مداخله این‌گونه مکانیزم‌ها تنها از نوع خاتمه‌دادن به برنامه است و تغییری در اجرای برنامه اعمال نمی‌کنند. این در حالی است که مکانیزم‌های از نوع انتقال‌دهنده اجرا، اجرای برنامه را به عنوان ورودی می‌گیرند، روی آن نظارت کرده و در صورت نیاز و مطابق با

<sup>۶۶</sup> Execution Monitoring

<sup>۶۷</sup> Complete

<sup>۶۸</sup> Execution Transformer

<sup>۶۹</sup> Execution Recognizer

<sup>۷۰</sup> Abort

خط‌مشی، تغییری به صورت جلوگیری از انتقال یک رویداد یا درج و حذف یک رویداد در اجرای فعلی ایجاد می‌کنند. به طور کلی، توانایی یک ناظر در اعمال خط‌مشی‌های امنیتی به دو عامل/اطلاعاتی که ناظر از/اجراهای دیگر برنامه در اختیار دارد و تنوع روش‌های مداخله در صورت تشخیص نقش امنیت بستگی دارد. مدل ماشین‌برنامه (PM) قادر است تا کلیه مکانیزم‌های نظارت بر اجرا، شامل هسته‌های امنیتی<sup>۷۱</sup>، ناظر مرجع<sup>۷۲</sup>، ناظر مرجع برخط<sup>۷۳</sup> و سایر مکانیزم‌های اعمال خط‌مشی امنیتی مبتنی بر سیستم‌عامل را مدل‌سازی کند.

همچنین، مکانیزم‌های تحلیل پویا بر اساس اطلاعات قابل دسترس آن‌ها برای تحلیل برنامه مورد مطالعه قرار می‌گیرند. در تعدادی از این مکانیزم‌ها، دسترسی به کد منبع برنامه وجود ندارد. حال آن‌که در برخی دیگر، این دسترسی به طور کامل وجود دارد و یا حتی ممکن است ناظر صرفاً در شرایطی به کد منبع برنامه دسترسی داشته باشد. پس از این منظر نیز می‌توان دسته‌بندی دیگری برای مکانیزم‌های تحلیل پویا تعریف کرد. از منظر اطلاعات در دسترس و زمان دسترسی، روش‌های پویا به سه نوع روش‌های تحلیل پویای محض<sup>۷۴</sup>، تحلیل پویای ترکیبی به صورت تحلیل کد در ابتدا و تحلیل پویای ترکیبی به صورت تحلیل کد در طول اجرا قابل دسته‌بندی هستند. در روش‌های تحلیل کد پویای محض، ناظر به کد منبع برنامه دسترسی ندارد. در دسته دوم این‌طور فرض می‌شود که کد منبع قبل از اجرا در اختیار ناظر قرار می‌گیرد و می‌تواند از این اطلاعات برای پیش‌بینی اجراهای دیگر برنامه استفاده کند. در روش‌های تحلیل پویای ترکیبی به صورت تحلیل کد در طول اجرا، ناظر قبل از اجرا کد منبع را در اختیار ندارد ولی در طول اجرای برنامه می‌تواند به بخش‌های مختلف کد منبع یا اجراهای دیگر برنامه دسترسی پیدا کند. از جهت شباهت دو دسته آخر به یکدیگر و فرض در اختیار داشتن کد منبع، می‌توان به طور کلی به آن‌ها روش‌های تحلیل پویای ترکیبی نیز اطلاق کرد.

همان‌طور که پیشتر گفته شد، در روش تحلیل پویای محض مکانیزم اعمال هیچ‌گونه اطلاعاتی از کد منبع برنامه در اختیار ندارد. پس در رویدادها نیز اطلاعاتی از کد منبع وجود ندارد. ناظرهای مرجع

<sup>71</sup> Security Kernel

<sup>72</sup> Reference Monitor

<sup>73</sup> Inlined Reference Monitor

<sup>74</sup> Pure Dynamic Analysis

و ناظرهای مرجع برخط [۲۶] در این دسته از مکانیزم‌های تحلیل پویا قرار می‌گیرند. زیرا، در آن‌ها مکانیزم اعمال فقط ناظر به رویدادهایی است که از برنامه دریافت می‌کند و در آن‌ها نیازی به تحلیل کد منبع برنامه برای تحلیل و پیش‌بینی رفتار آینده اجرای برنامه نیست.

مکانیزم‌های دسته دوم پیش از اجرای برنامه کد منبع برنامه را در اختیار دارند. پس می‌توانند تحلیل کد منبع را در همان مرحله انجام دهند. به این ترتیب، اطلاعات مورد نیاز برای تصمیم‌گیری در زمان اجرا در همین گام، از تحلیل کد منبع استخراج شده و در طول اجرای برنامه، در هنگام نظارت بر اجرای برنامه از آن‌ها استفاده می‌شود. پس طبیعی است که منجر به تحلیل و اعمال دقیق‌تری نسبت به روش‌های دسته اول می‌شود.

در دسته سوم روش‌های نظارت، اطلاعات همزمان با تولید رویدادها در اختیار ناظر قرار می‌گیرد. پس لازم است که بعضی از رویدادهای موجود در اجرا، از رویدادهای حاوی اطلاعاتی از کد منبع برنامه باشد تا ناظر بتواند بر اساس مدل مطرح‌شده امنیت را اعمال کند.

### ۳-۱-۳ بازنویسی برنامه

در این دسته از مکانیزم‌ها، مشابه روش‌های نظارت بر اجرا، همه برنامه‌ها به مرحله اجرا می‌رسد؛ با این تفاوت که در این مکانیزم‌ها برنامه غیرقابل اعتماد قبل از اجرا توسط مکانیزم اعمال، با هدف اعمال خط‌مشی امنیتی مورد نظر، دچار تغییر می‌شود و کد منبع برنامه با توجه به الگوریتم بازنویسی اصلاح می‌شود. باید این تضمین توسط مکانیزم بازنویسی برنامه داد شود که همواره برنامه بازنویسی‌شده از نظر معناساخت<sup>۷۵</sup> هم‌ارز برنامه ورودی اصلی باشد. به بیان صوری، خط‌مشی امنیتی  $P$  توسط مکانیزم بازنویسی برنامه قابل اعمال خواهد بود اگر و فقط اگر تابع بازنویسی برنامه  $R$  از ماشین برنامه به ماشین برنامه وجود داشته باشد به طوری که  $P(R(M))$  برقرار باشد و  $P(M) \Rightarrow M \approx R(M)$ .

از دیدگاهی می‌توان بازنویسی برنامه را به صورت تعمیمی از روش‌های نظارت بر اجرا دانست. در صورتی که مداخله ناظر زمان اجرا به صورت تغییر برنامه باشد، به نوعی بازنویسی برنامه انجام می‌گیرد. اما از لحاظ این‌که بازنویسی برنامه در زمان کامپایل انجام می‌شود و با استفاده از تحلیل‌های ایستا، و نه

<sup>75</sup> Semantics

اطلاعات زمان اجرا، کد منبع برنامه تغییر می‌کند و پس از آن، کد اصلاح شده به مرحله اجرا می‌رسد، می‌توان گفت زمان اعمال در این مکانیزم‌ها زمان اجرا است. به همین دلیل، می‌توان این دسته از روش‌ها را در دسته‌ای خارج از روش‌های مکانیزم ایستا و پویا قرار داد. به عنوان مثال، ناظر مرجع برخط را می‌توان یک بازنویس برنامه دانست. زیرا، در ناظر مرجع برخط فرض شده است که از یک تغییردهنده برنامه استفاده می‌شود و این تغییردهنده برنامه، کدهایی را به منظور اعمال خطمشی در لابه‌لای دستورات برنامه اصلی درج می‌کند. پس مداخله این ناظر اجرا نوعی بازنویسی برنامه محسوب می‌شود.

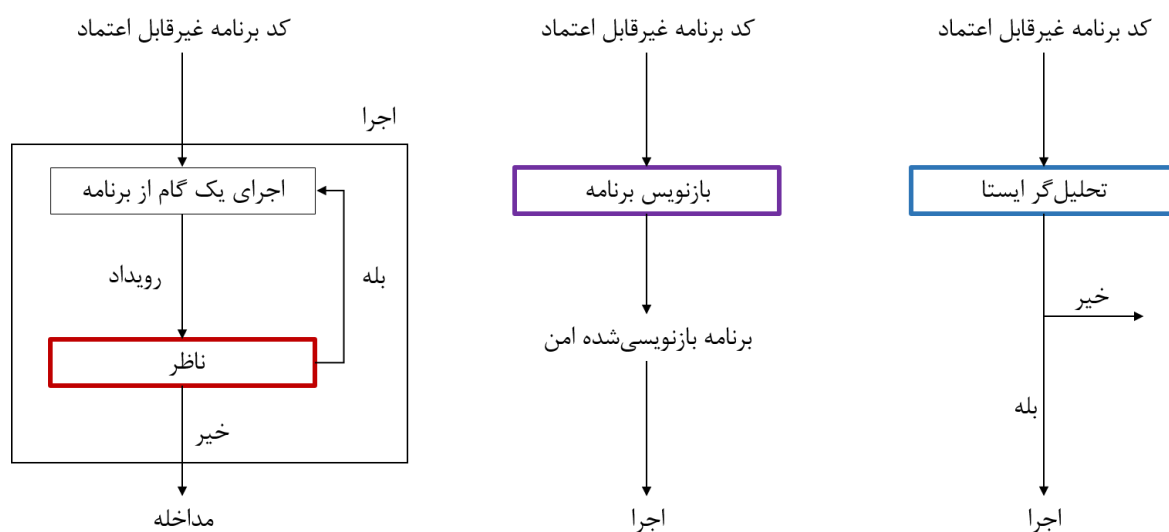
### ۳-۱-۴ تحلیل ترکیبی

در مکانیزم‌های مبتنی بر تحلیل ترکیبی، تلفیقی از روش‌های تحلیل ایستا و پویا مورد استفاده قرار می‌گیرد. مکانیزم اعمال در دو مرحله عمل می‌کند:

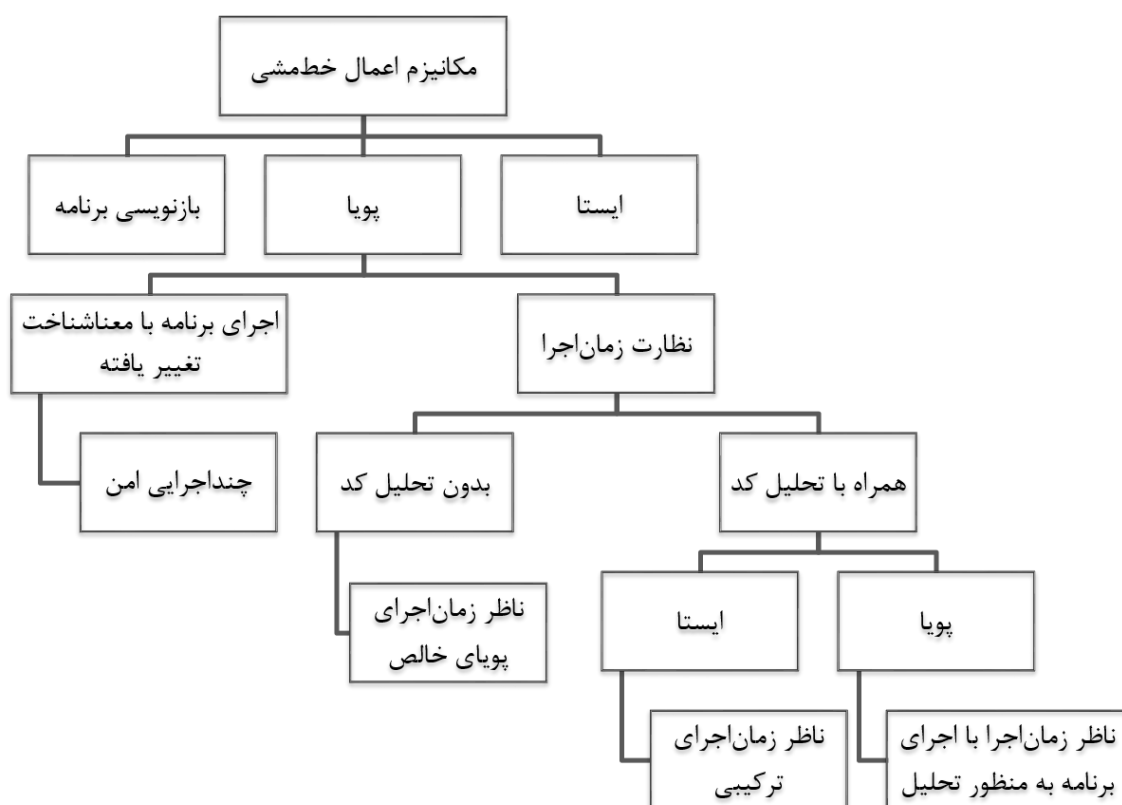
(۱) مرحله تحلیل ایستا: قبل از اجرای برنامه، کد منبع برنامه در این مرحله مورد بررسی و تحلیل قرار می‌گیرد. ممکن است در همین مرحله برنامه‌ای کاملاً ناامن تشخیص داده شده و اجازه اجرا شدن آن داده نشود. در موارد دیگر احتمال دارد که پس از اعمال تغییراتی، برنامه برای تحلیل پویا به مرحله بعدی فرستاده شود.

(۲) مرحله تحلیل پویا: پس از گذر از مرحله تحلیل ایستا، اجرای برنامه نظارت می‌شود و خطمشی مورد نظر توسط مکانیزم‌های مبتنی بر تحلیل پویا اعمال می‌شود. در این مرحله می‌تواند روش‌های مختلف تحلیل پویای مطرح شده در قبل وجود داشته باشد.

برای جمع‌بندی این قسمت، شکل ۳ نحوه عملکرد مکانیزم‌های ایستا، پویا و بازنویسی را به طور شماتیک نشان می‌دهد. به طور خلاصه در مکانیزم ایستا، تحلیل‌گر ایستا در خصوص امن بودن یا نبودن برنامه ورودی تصمیم‌گیری می‌کند. یک بازنویس برنامه، بر اساس تحلیل ایستا کد برنامه ورودی را به برنامه‌ای امن تبدیل می‌کند. از طرفی در حالت کلی، یک ناظر اجرا پس از هر گام از اجرای برنامه، امن بودن یا نبودن آن را تشخیص داده و در صورت لزوم مداخله می‌کند. این مداخله می‌تواند به روش‌های مختلفی صورت بگیرد. همچنین، در شکل ۴ نمودار دسته‌بندی روش‌های اعمال خطمشی‌های قابل مشاهده است. البته پژوهش‌هایی نیز به طور خاص به مرور نقادانه روی مکانیزم‌های ایستا [۲۳] و مکانیزم‌های پویا [۲، ۲۷] پرداخته‌اند.



شکل ۳ - نحوه عملکرد مکانیزم ایستا، بازنویس برنامه و ناظر زمان اجرا به منظور اعمال خطمشی‌های امنیتی [۴]



شکل ۴ - دسته‌بندی مکانیزم‌های اعمال خطمشی‌های امنیتی [۴]

## ۳-۲ مقایسه روش‌های اعمال خط‌مشی امنیتی

پیش‌تر تصور بر این بود که در مکانیزم‌های پویا از آن جهت که فقط یک اجرا تحلیل و بررسی می‌شود و خط‌مشی‌های جریان اطلاعات به صورت گزاره‌ای روی بیش از یک اجرا تعریف می‌شوند، نمی‌توان به درستی این خط‌مشی‌ها را توسط مکانیزم‌های پویا اعمال کرد و از همین رو، روش‌های تحلیل ایستا و نوع‌مبنا اغلب کارهای این حوزه را شامل می‌شد. اما از طرف دیگر استفاده از روش‌های پویا در اعمال خط‌مشی‌های امنیتی مزایایی دارد که در ادامه به برخی از آن‌ها اشاره می‌شود:

- آسان‌گیرتر بودن و دقت بیشتر در مقایسه با تحلیل ایستا: با توجه به این‌که روش‌های تحلیل پویا علاوه بر کد منبع برنامه، به اطلاعات زمان اجرا نیز دسترسی دارند، پس می‌توانند با اعمال خط‌مشی‌ها را با دقت بیشتری در مقایسه با تحلیل‌های ایستا انجام دهند. دقت اعمال از دو منظر قابل بحث است. اول آن‌که توانایی مکانیزم‌های پویا در تشخیص برنامه‌هایی که خط‌مشی را برآورده می‌کنند بیشتر است. زیرا ممکن است تحلیل ایستا به دلیل رفتار محافظه‌کارانه، از اجرای یک برنامه امن جلوگیری کند. به عنوان مثال، ممکن است همه اجراهای واقعی برنامه‌ای از نظر خط‌مشی امن باشد اما به واسطه وجود یک عبارت شرطی که هیچ‌گاه به اجرا نمی‌رسد و این احتمال که یک جریان ضمنی ناقض خط‌مشی به واسطه عبارت شرطی وجود دارد، برنامه توسط تحلیل‌گر ایستا امن تشخیص داده نشود. حال آن‌که همین برنامه در تحلیل پویا بدون مشکل امنیتی اجرا خواهد شد. در واقع، نقطه قوت روش‌های تحلیل پویا تحلیل اجراها و رفتار (معناشناخت) برنامه است، و نه صرف تحلیل نحوی<sup>۷۶</sup> و کد منبع آن.

از منظر دوم، زمانی مکانیزم‌های پویا در اجرای برنامه‌ها تغییر ایجاد می‌کنند که نشت اطلاعات رخ داده باشد. ممکن است در برنامه‌ای خط‌مشی مورد نظر برقرار نباشد، اما اجراهایی وجود داشته باشند که در آن هیچ‌گونه نشت اطلاعاتی وجود نداشته باشد. پس مکانیزم پویا قادر است در این برنامه اجرای امن را از اجرای ناامن تشخیص داده و تنها زمانی در اجرا مداخله کند که آن اجرا به نشت اطلاعات منجر شود. اما در تحلیل ایستا در صورتی که برنامه فقط دارای یک اجرای ناامن باشد، کل برنامه ناامن شناخته می‌شود. بنابراین با بهره‌گیری از تحلیل پویا می‌توان

<sup>76</sup> Syntactic Analysis



اجرای برنامه را تا زمانی که نشت اطلاعاتی صورت نگرفته است، ادامه داد. از این رو، مکانیزم‌های پویا را آسان‌گیرتر از تحلیل ایستا می‌دانند [۲۸].

- امکان سطح‌دهی پویا و تعریف جریان اطلاعات امن در هر اجرا: ممکن است در سامانه‌های واقعی، خط‌مشی امنیتی در حین اجرای برنامه دچار تغییر شود؛ یعنی نمی‌توان از قبل از اجرای برنامه سطح امنیتی عناصر موجود در برنامه را تعیین کرد. پس لازم است که از برچسب‌گذاری پویا<sup>۷۷</sup> استفاده شود. همچنین، تعریف شبکه سطوح امنیتی در مکانیزم‌های تحلیل ایستا باید حتماً قبل از کامپایل شدن برنامه صورت گیرد. ولی در مکانیزم‌های تحلیل پویا این امکان وجود دارد که پیش از هر اجرای برنامه، شبکه سطوح امنیتی تعریف شود. علاوه بر این، امکان سطح‌دهی پویا برای تعیین سطوح امنیتی متغیرها بر اساس اطلاعات زمان اجرای برنامه در طول اجرا وجود دارد.

- استفاده در زبان‌های برنامه‌نویسی پویا: بعضی از زبان‌های برنامه‌نویسی از جمله JavaScript و Perl ماهیت پویا دارند؛ یعنی عملیات پویایی در زبان تعریف شده است که تحلیل آن‌ها در زمان ایستا ممکن نیست. پس در این گونه زبان‌ها، تنها مکانیزم‌های تحلیل پویا کاربرد خواهند داشت. نوع‌دهی پویا<sup>۷۸</sup> در بعضی از زبان‌های پویا وجود دارد و قواعد نوع‌دهی<sup>۷۹</sup> به صورتی طراحی می‌شوند که برنامه‌های خوش‌نوع<sup>۸۰</sup>، خط‌مشی را برآورده کنند. طبق دسته‌بندی مطرح‌شده، می‌توان روش نوع‌دهی پویا را در دسته مکانیزم‌های تحلیل پویا قرار داد.

علی‌رغم دارا بودن مزایای عنوان‌شده بالا، مکانیزم‌های تحلیل پویا برخلاف تحلیل ایستا با توجه به این که همزمان با هر اجرای برنامه اعمال می‌شوند، سرباری به اجرای برنامه اضافه می‌کنند که از کارایی آن‌ها کاسته می‌شود. نشان داده شده است [۲۷] که این روش‌ها بین ۵۰٪ تا ۱۰۰٪ به زمان اجرا بار اضافه می‌کنند. نکته حائز اهمیت آن است که این سربار به ازای هر بار اجرای برنامه وجود خواهد داشت، در حالی که در تحلیل ایستا صرفاً پیش از اجرای برنامه سربار تحلیل وجود دارد و در زمان اجرای برنامه

<sup>77</sup> Dynamic Labeling

<sup>78</sup> Dynamic Typing

<sup>79</sup> Typing Rules

<sup>80</sup> Well-Typed

هیچ‌گونه سرباری به سامانه تحمیل نمی‌شود. به این ترتیب ممکن است در بعضی کاربردها تنها استفاده از مکانیزم‌های ایستا میسر باشد.

همچنین، مکانیزم‌های اعمال خط‌مشی امنیتی بر اساس دو معیار درستی و کامل‌بودن قابل مقایسه هستند. درستی مکانیزم اعمال به این مفهوم است که همه برنامه‌ها تحت اجرا توسط مکانیزم، مطابق خط‌مشی امن محسوب شوند. به عبارت دیگر، برنامه‌هایی که توسط مکانیزم اعمال امن تشخیص داده شده‌اند، واقعاً امن باشند و خط‌مشی را نقض نکنند. پس به این ترتیب نباید برنامه‌ای وجود داشته باشد که مکانیزم اعمال آن را امن تشخیص داده، ولی پس از اجرا خط‌مشی برآورده نشود. این معیار مهم‌ترین ویژگی برای مکانیزم‌های اعمال است. به طوری که چنانچه مکانیزمی وجود داشته باشد که درستی را تضمین نکند، بی‌استفاده خواهد بود. همان‌طور که پیش‌تر عنوان شد، مکانیزم‌های تحلیل پویا فقط زمانی می‌توانند خط‌مشی جریان اطلاعات را درست اعمال کنند که متن برنامه نیز در دسترس آن‌ها باشد [۲۹]. به این معنا که علاوه بر اطلاعات حاصل از اجرای فعلی برنامه، اطلاعاتی از سایر اجراهای ممکن برنامه را در اختیار داشته باشند. در غیر این صورت، این مکانیزم‌ها تنها قادرند که از بروز جریان‌های صریح جلوگیری کنند. برای تشخیص و جلوگیری از جریان ضمنی، باید مکانیزم به کد منبع برنامه دسترسی داشته باشد. حال آن‌که دسترسی به کد منبع برنامه و تحلیل آن در زمان اجرا، به اضافه‌شدن سربار این‌گونه مکانیزم‌ها می‌انجامد.

منظور از کامل‌بودن یک مکانیزم اعمال این است که برنامه‌ای که خط‌مشی امنیتی را نقض نمی‌کند، توسط مکانیزم نیز امن تشخیص داده شود. تفاوت بین درستی و کامل‌بودن با مثالی به خوبی قابل درک خواهد بود. فرض کنید که مکانیزم اعمال مورد استفاده اجرای همه برنامه‌ها را به اجرای برنامه چاپ عبارت سلام دنیا! تغییر می‌دهد. به این ترتیب، این مکانیزم، یک مکانیزم درست خواهد بود. زیرا برنامه‌ای که پس از اعمال توسط مکانیزم اجرا می‌شود، خط‌مشی امنیتی عدم تداخل را برآورده خواهد ساخت. اما این نحوه اعمال باعث می‌شود که هر برنامه امنی توسط این مکانیزم ناامن شناخته شده و رفتار برنامه به طور کلی تغییر کند. با استفاده از همین معیار، روش‌های مختلف اعمال خط‌مشی‌های امنیتی با یکدیگر مقایسه می‌شوند. البته هنوز روشی ارائه نشده است که کامل‌بودن در آن اثبات شده باشد. زیرا در مکانیزم‌های تحلیل ایستا، محافظه‌کاربودن این گونه روش‌ها محدودیتی جدی به شمار می‌رود. پس می‌توان گفت هیچ‌گاه با استفاده از تحلیل ایستا، یک مکانیزم کامل ارائه کرد. این مشکل در

مکانیزم‌های پویا نیز وجود دارد [۲]. اگرچه این مکانیزم‌ها نسبت به مکانیزم‌های مبتنی بر تحلیل ایستا آسان‌گیرتر و دقیق‌تر هستند اما کامل بودن برای آن‌ها نیز برقرار نیست. به عبارت دیگر، برای خط‌مشی‌های جریان اطلاعات، در عمل تقریبی از آن‌ها توسط مکانیزم‌ها اعمال می‌شود که باعث کامل نبودن این مکانیزم‌ها برای خط‌مشی‌های جریان اطلاعات می‌شود.

در خصوص ارزیابی و مقایسه مکانیزم‌های تحلیل پویا و ناظرهای زمان اجرا از اصطلاح‌های مختلفی مانند دقت، آسان‌گیری<sup>۸۱</sup> و شفافیت استفاده شده است که در ادامه، به تفکیک این مفاهیم از یکدیگر پرداخته می‌شود [۱۶]. به طور کلی، دقت یک مکانیزم پویا بیانگر این است که چه تعداد از برنامه‌های امن توسط مکانیزم ناامن تشخیص داده شده و متعاقباً مسدود یا دچار تغییر می‌شوند. همچنین در روش نظارت زمان اجرا، ناظرها توسط دو معیار درستی و شفافیت مورد مقایسه قرار می‌گیرند. شفافیت به این معنا که هرگاه اجرایی از برنامه خاصیت مورد نظر را برآورده می‌کند، ناظر بدون هیچ تغییری همان اجرا را به عنوان خروجی تولید کند. به طور مشابه می‌توان این معیارها را به همه مکانیزم‌های تحلیل پویا تعمیم داد.

در بیان تفاوت بین دقت و شفافیت باید اشاره کرد که اساساً دقت برای برنامه‌های امن تعریف می‌شود. منظور این است که مکانیزمی دقیق نامیده می‌شود که همه اجراهای همه برنامه‌های امن را عیناً حفظ کرده و تغییری در آن‌ها ایجاد نکند. این در حالیست که شفافیت روی حفظ/جر/های امن برنامه‌ها توسط مکانیزم تأکید دارد؛ یعنی ممکن است برنامه‌ای ناامن باشد اما باید تمامی اجراهای امن آن برنامه دست نخورده باقی بماند. به همین ترتیب، دو اصطلاح شفافیت حقیقی<sup>۸۲</sup> و شفافیت کاذب<sup>۸۳</sup> نیز قابل تعریف است [۱۶]. شفافیت حقیقی درباره حفظ اجراهای امن برنامه و شفافیت کاذب به حفظ اجراهای ناامن برنامه مطرح می‌شوند.

علاوه بر این، دلیل استفاده از لفظ شفافیت کامل در این پایان‌نامه آن است که در مکانیزم‌های ارائه شده مبتنی بر روش چنداجرایی امن، ترتیب رویدادهای خروجی فقط در هر کانال به طور مستقل از

<sup>81</sup> Permissiveness

<sup>82</sup> True Transparency

<sup>83</sup> False Transparency

دیگر کانال‌ها حفظ می‌شود. در حالی که ترتیب این رویدادها به طور کلی و با در نظر گرفتن کانال‌ها نسبت به یکدیگر نیز حائز اهمیت است و اصطلاح شفافیت کامل نیز به همین خاطر مطرح شده است.

همچنین، برای مقایسه ناظرها از اصطلاح آسان‌گیر بودن نیز استفاده می‌شود. ناظری از ناظر دیگری آسان‌گیرتر است اگر مجموعه اجراهای پذیرفته‌شده توسط آن ناظر شامل مجموعه اجراهای پذیرفته‌شده توسط دیگری باشد. بنابراین، طبق معیارهای معرفی‌شده می‌توان مکانیزم‌های مختلف را با یکدیگر مقایسه کرد.

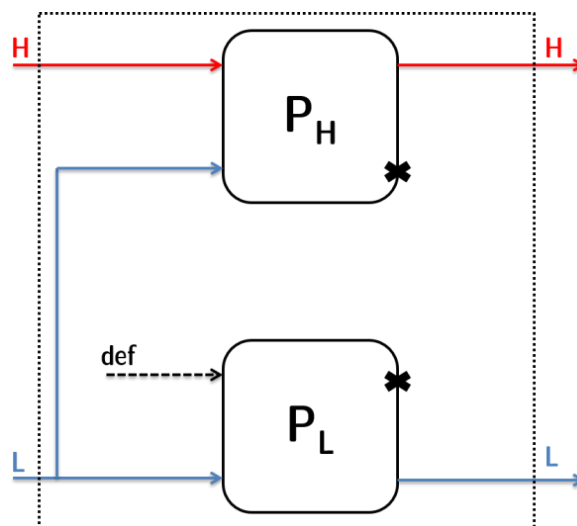
### ۳-۳ مکانیزم‌های مبتنی بر چنداجرایی امن

در این بخش، به مرور پژوهش‌های انجام‌شده درباره انواع مکانیزم‌های پیشنهادی پرداخته می‌شود که در آن‌ها از روش چنداجرایی امن استفاده شده است. اولین بار اصطلاح چنداجرایی امن (SME) توسط Devriese و Piessens [۶] مطرح شد. اگرچه کارهای مشابه با این روش قبل‌تر از آن هم انجام شده است [۳۰، ۳۱] اما تفاوت در آنجاست که برای نخستین بار صوری‌سازی مکانیزم، اثبات درستی و دقت آن، و نیز پیاده‌سازی برای برنامه‌های جاوااسکریپت مطرح شده است.

شِمای ایده چنداجرایی امن در شکل ۵ قابل مشاهده است. طرز کار مکانیزم این‌گونه است که به تعداد سطوح امنیتی شبکه، رونوشت‌هایی از برنامه تهیه می‌شود و هر رونوشت به یک سطح امنیتی اختصاص می‌یابد. با فرض محدود کردن شبکه امنیتی به دو سطح بالا (محرمانه یا H) و پایین (عمومی یا L) می‌توان چنین گفت که محتوای کانال ورودی سطح بالا فقط توسط اجرای همان سطح قابل مشاهده و استفاده است. به طور مشابه برای رونوشت سطح پایین نیز چنین فرضی وجود دارد با این تفاوت که با توجه به این که مشاهده‌گر سطح بالا قادر است محتوای کانال ورودی سطح پایین را نیز مشاهده کند، پس همان ورودی‌ها برای رونوشت سطح بالا قابل بازاستفاده<sup>۸۴</sup> خواهد بود. همچنین، در صورتی که رونوشت سطح پایین به دستوری از برنامه برسد که باید ورودی‌ای از کانال ورودی سطح بالا دریافت کند، همواره مقدار پیش‌فرض ثابتی به آن تخصیص داده می‌شود. با توجه به این که محتوای کانال‌های خروجی هر سطح توسط اجرای رونوشت همان سطح و با شرایط گفته‌شده تولید می‌شود، پس

<sup>84</sup> Reuse

هیچ‌گاه کانال خروجی سطح پایین تحت تأثیر مقادیر ورودی سطح بالا نخواهد بود. از این رو، می‌توان مشاهده کرد که چنین مکانیزمی همواره درست است.



شکل ۵ - نمایی از روش چنداجرایی امن (SME) [۶]

اگرچه ایده کلی مطرح‌شده منحصر به زمان‌بندی خاصی نیست اما اثبات‌های انجام‌شده [۶] برای یک مکانیزم مشخص با فرض استفاده از زمان‌بندی /اولویت با سطح پایین‌تر است؛ یعنی ابتدا رونوشت سطح پایین به طور کامل اجرا شده، و پس از آن نوبت به اجرای رونوشت سطح بالا می‌رسد. به همین دلیل، رونوشت سطح بالا از ورودی‌های خوانده‌شده توسط اجرای رونوشت سطح پایین مجدداً استفاده می‌کند.

شایان ذکر است که این روش برخلاف مکانیزم سندباکس داده [۳۱]، نیازی به در اختیار داشتن کد منبع برنامه ندارد و تنها ورودی‌ها و خروجی‌های رونوشت‌ها به نحو تعیین‌شده تأمین می‌شوند. از این جهت، روش چنداجرایی امن در دسته مکانیزم‌های جعبه‌سیاه<sup>۸۵</sup> قرار می‌گیرد. در مکانیزم سندباکس داده، برنامه به دو قسمت عمومی و خصوصی تقسیم‌بندی می‌شود که قسمت خصوصی شامل دستوراتی است که روی داده‌های حساس پردازش انجام می‌دهند. به این ترتیب، لازم است که کد منبع برنامه در اختیار مکانیزم باشد تا برنامه را به دو بخش عمومی و خصوصی افراز شود.

<sup>85</sup> Black-Box

پس از مطرح شدن ایده چنداجرایی امن، که اثبات شده است [۶] می‌تواند عدم تداخل حساس به خاتمه و زمان را درست و برای اجراهای خاتمه‌پذیر برنامه‌های امن دقیق اعمال کند، پژوهشگران [۱۰]، [۱۷]، [۳۲]–[۴۰] به توسعه مکانیزم‌های مبتنی بر این ایده پرداختند.

همان‌طور که پیشتر ذکر شد، اثبات درستی و دقت مکانیزم چنداجرایی امن با فرض استفاده از زمان‌بندی / اولویت با سطح پایین‌تر صورت گرفته است. Kashyap و همکاران [۱۷] انواع مختلف زمان‌بندی‌ها را برای اعمال عدم تداخل حساس به زمان و خاتمه مورد بررسی قرار داده‌اند. در واقع، عدم تداخل حساس به خاتمه را نوع خاصی از حساس به زمان دانسته‌اند؛ از این بابت که یک محاسبه سطح پایینی اتفاق افتاده و برنامه خاتمه نیافته باشد، به نوعی روی زمان محاسبه آن دستور تأثیرگذار است. همچنین، از دو اصطلاح حساس ضعیف<sup>۸۶</sup> و حساس قوی<sup>۸۷</sup> استفاده می‌شود. منظور از حساس ضعیف این است که خط‌مشی تنها به تعداد گام‌های محاسبه توجه دارد، نه زمان واقعی اجرا با در نظر گرفتن همه جزئیات محیط اجرا (حساس قوی). همچنین، فرض شده است که ریشه‌های ناشی از اجرای رونوشت‌ها می‌توانند در حالت‌های آماده<sup>۸۸</sup> (R)، مسدودشده<sup>۸۹</sup> (B) یا خاتمه‌یافته<sup>۹۰</sup> (T) باشند.

همان‌طور که در جدول ۱ مشاهده می‌شود، اثبات شده است که در صورت استفاده از زمان‌بندی متوالی<sup>۹۱</sup> و اولویت با سطح پایین‌تر برای ریشه‌های آماده برای اجرا، عدم تداخل حساس ضعیف برای سطوح قابل مقایسه و عدم تداخل غیرحساس برای سطوح غیرقابل مقایسه اعمال‌پذیر است. منظور از سطوح قابل مقایسه سطوحی است که در شبکه امنیتی با یکدیگر رابطه دارند. در صورتی که از همین زمان‌بندی با در نظر گرفتن ریشه‌های آماده و مسدودشده استفاده شود، می‌توان عدم تداخل حساس قوی را برای سطوح قابل مقایسه اعمال کرد.

<sup>86</sup> Weakly-sensitive

<sup>87</sup> Strongly-sensitive

<sup>88</sup> Ready

<sup>89</sup> Blocked

<sup>90</sup> Terminated

<sup>91</sup> Sequential

جدول ۱ - انواع استراتژی‌های زمان‌بندی و تضمین امنیت هر یک [۱۷]

استراتژی زمان‌بندی	نحوه انتخاب از بین ریشه‌ها	سطح تضمین امنیت
متوالی - نوع ۱	پایین‌ترین ریشه آماده	حساس ضعیف برای سطوح قابل مقایسه غیرحساس برای سطوح غیرقابل مقایسه
متوالی - نوع ۲	پایین‌ترین ریشه آماده/مسدودشده	حساس قوی برای سطوح قابل مقایسه غیرحساس برای سطوح غیرقابل مقایسه
تسهیم - نوع ۱	همه ریشه‌های آماده (یا آماده/مسدودشده)	غیرحساس برای همه سطوح
تسهیم - نوع ۲	همه ریشه‌های آماده/مسدودشده/خاتمه‌یافته	حساس ضعیف برای همه سطوح
مبتنی بر شبکه	انتخاب از بین تعداد مشخصی از ریشه‌ها که شامل همه ریشه‌های آماده/مسدودشده هستند که همه ریشه‌های پایین‌تر از آن‌ها خاتمه‌یافته باشند.	حساس قوی برای سطوح قابل مقایسه حساس ضعیف برای سطوح غیرقابل مقایسه

زمان‌بندی تسهیم مشابه زمان‌بندی با گردش نوبت<sup>۹۲</sup> است؛ به این معنا که یک سهم زمانی<sup>۹۳</sup> ثابت و مشخصی تعیین شده و هر یک از ریشه‌ها در هر نوبت به اندازه آن سهم زمانی قادر به اجرای رونوشت خود هستند و پس از آن، نوبت به ریشه بعدی تحویل داده می‌شود. در صورتی که این زمان‌بندی مورد استفاده قرار بگیرد و همه ریشه‌ها در همه حالت‌ها قابل انتخاب باشند، عدم تداخل حساس ضعیف برای همه سطوح قابل اعمال است. زیرا، هر یک از ریشه‌ها به مقدار ثابت و مشخصی منتظر می‌ماند و مطابق سهم زمانی می‌تواند اجرای خود را پیش ببرد. از آن‌جا که به عنوان مثال، ممکن است اجرای یک ریشه باعث پرشدن حافظه نهان شود، اجرای بعدی به دلیل تغییر در زمان واقعی اجرای

<sup>۹۲</sup> Round Robin<sup>۹۳</sup> Time Slot

خود می‌تواند اطلاعاتی از سطح بالا به دست آورد. پس این روش قادر به اعمال عدم تداخل حساس قوی نیست.

در استراتژی مبتنی بر شبکه، ابتدا ریشه پایین‌ترین سطح به طور کامل اجرا می‌شود و پس از آن، ریشه‌هایی که همه ریشه‌های پایین‌تر از آن‌ها خاتمه‌یافته باشند، در مجموعه ریشه‌های قابل انتخاب قرار گرفته و طبق زمان‌بندی با گردش نوبت اجرا می‌شوند. به این ترتیب، مشابه نمایش گرافیکی شبکه همه ریشه‌های موجود در هر سطر از شبکه در هر مرحله می‌توانند انتخاب شوند. در واقع، این استراتژی تلفیقی از دو استراتژی متوالی و با گردش نوبت محسوب می‌شود که بالاترین تضمین امنیتی به واسطه استفاده از این زمان‌بندی قابل حصول است.

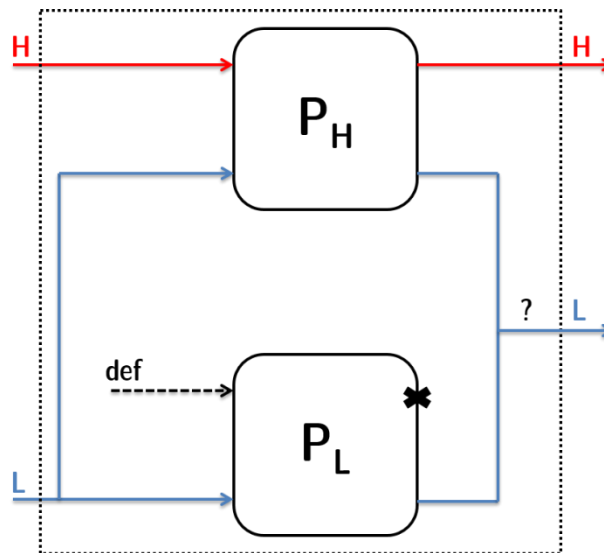
یکی از اصلی‌ترین نقاط ضعف چنداجرایی امن، عدم حفظ ترتیب رویدادهای خروجی برنامه‌های امن است. برای حل این مشکل، Zanarini و همکاران [۳۳] مکانیزمی به نام *ناظر چنداجرایی/امن* معرفی کردند. عملکرد این مکانیزم این گونه است که همزمان با اجرای برنامه، چنداجرایی امن برنامه نیز به طور موازی به کار گرفته می‌شود و در هر گام، خروجی‌های تولیدشده مطابقت داده می‌شوند. زمان‌بندی معرفی‌شده برای استفاده در چنداجرایی امن در این مکانیزم، ترتیب رویدادها را در کانال‌های خروجی حفظ می‌کند. پس در هر گام از اجرا، در صورتی که خروجی تولیدشده توسط برنامه اصلی و خروجی تولیدشده توسط چنداجرایی امن یکسان نباشد، به معنی وقوع یک نشت اطلاعات تلقی می‌شود. لازم به ذکر است که این مکانیزم فقط برای برآورده کردن عدم تداخل حساس به خاتمه طراحی شده است و تضمینی برای شفافیت حساس به زمان ارائه نمی‌کند.

برای برطرف کردن مشکل عدم حفظ ترتیب رویدادهای خروجی برنامه‌های امن، Rafnsson و Sabelfeld [۳۲] پیشنهاد دیگری را نیز مطرح کرده‌اند که از ایده همگام‌سازی حصار<sup>۹۴</sup> استفاده می‌کند. همان‌طور که پیشتر مطرح شد، چنداجرایی امن تنها قادر است ترتیب رویدادهای خروجی را در هر کانال حفظ کند، در حالی که برای برنامه‌های امن لازم است که رفتار خروجی‌ها در کانال‌های مختلف نسبت به یکدیگر نیز حفظ شود. در شکل ۶، نمایی از ایده مطرح‌شده قابل مشاهده است. نحوه کار مکانیزم به این شرح است که به جای در نظر نگرفتن خروجی‌های سطح پایین در اجرای رونوشت سطح بالا،

<sup>۹۴</sup> Barrier Synchronization



خروجی‌های تولیدشده توسط این ریشه با خروجی‌های تولیدشده توسط رونوشت سطح پایین مقایسه می‌شوند. در این حالت اگر برنامه امن بوده باشد، نباید هیچ مغایرتی در خروجی‌های تولیدشده توسط اجرای این دو رونوشت به وجود بیاید. پس اگر مغایرتی وجود داشته باشد، نشان از وجود ناامنی در برنامه دارد که می‌توان از همین موضوع برای بیان مثال نقض عدم تداخل در برنامه نیز استفاده کرد.



شکل ۶ - چنداجرائی امن به همراه همگام‌سازی حصار [۳۲]

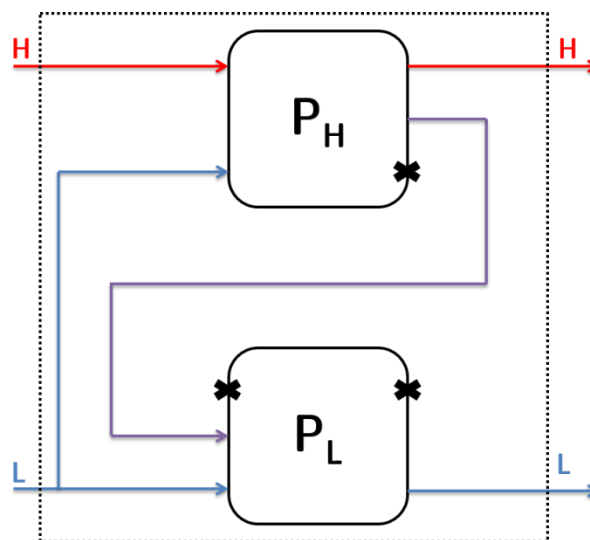
منظور از همگام‌سازی حصار این است که اجرای رونوشت سطح بالا تا زمانی که خروجی سطح پایینی تولید نشود، ادامه می‌یابد و به محض رسیدن به دستوری که رویداد خروجی سطح پایین را تولید می‌کند، حصاری تشکیل می‌شود و تنها زمانی رونوشت سطح بالا می‌تواند به ادامه اجرا بپردازد که رونوشت سطح پایین یک خروجی تولید کرده باشد. به طور مشابه، در صورتی که رونوشت سطح پایین زودتر خروجی تولید کرده باشد، اجرای آن متوقف می‌شود تا رونوشت سطح بالا نیز یک خروجی سطح پایین تولید کند یا از مدت زمان تعیین‌شده برای تولید خروجی سطح پایین توسط رونوشت سطح بالا گذشته باشد. در صورتی که در فرصت تعیین‌شده خروجی متفاوتی تولید شود یا زمان منقضی شود، می‌توان گفت که نقض امنیت رخ داده است.

اگرچه مکانیزم پیشنهادی می‌تواند ترتیب رویدادهای خروجی را حفظ کند اما تنها خط‌مشی امنیتی عدم تداخل غیرحساس به زمان توسط این مکانیزم قابل اعمال است. زیرا این‌که ریشه سطح

پایین باید مدت زمانی را در حالت انتظار برای تولید خروجی توسط ریشه سطح بالا باشد، منجر به ایجاد یک نشت زمانی می‌شود.

اشکال دیگر مکانیزم‌های مبتنی بر روش چنداجرایی امن این فرض است که مجموعه کانال‌های ورودی و خروجی باید ثابت در نظر گرفته شوند و هر کانال به یک سطح امنیتی مشخص نسبت داده شده باشد. Zandarini و Jaskelioff [۳۵] با بهبود مکانیزم ناظر چنداجرایی امن، مکانیزمی پیشنهاد کرده‌اند که در آن، مجموعه کانال‌ها و انتساب سطوح امنیتی به یک کانال قابل تغییر است.

طیف دیگری از پژوهش‌های انجام‌شده در رابطه با ارائه مکانیزم‌های مبتنی بر چنداجرایی به همراه پشتیبانی از خط‌مشی‌های حذف رده‌بندی<sup>۹۵</sup> [۴۱] است. Rafnsson و Sabelfled [۳۲] با تعمیم چنداجرایی امن به شرایطی که بین سطوح امنیتی حضور و محتوای پیام تمایز ایجاد شده است، مدلی از حذف رده‌بندی ارائه کرده‌اند که می‌توان مشخص کرد که چه اطلاعاتی قابل رهاسازی<sup>۹۶</sup> است. بنابراین، خط‌مشی‌هایی که در آن‌ها صرف وقوع رویدادهای ورودی محرمانه برای مشاهده‌گر سطح پایین قابل مشاهده است، توسط این مکانیزم قابل اعمال است. همان‌طور که در شکل ۷ دیده می‌شود، با ارسال ورودی محرمانه جعلی به اجرای سطح پایین، فقط وجود چنین ورودی‌ای قابل مشاهده خواهد بود.



شکل ۷- چنداجرایی امن با حذف رده‌بندی [۳۲]

<sup>۹۵</sup> Declassification

<sup>۹۶</sup> Release

Vanhoeef و همکاران [۳۶] مکانیزم دیگری مبتنی بر چنداجرایی امن پیشنهاد کرده‌اند که توانایی اعمال خط‌مشی‌های جریان اطلاعات را برای برنامه‌های رویدادمحور<sup>۹۷</sup> به همراه حذف رده‌بندی داراست که در آن، وضعیت سامانه در تابع رهاسازی نگهداری می‌شود و با استفاده از حاشیه‌نویسی<sup>۹۸</sup> مشخص می‌شود که کدام مقدار توسط این تابع محاسبه شده است.

همچنین، Bolosteanu و Garg [۳۸] با ارائه مکانیزمی تحت عنوان چنداجرایی امن نامتقارن<sup>۹۹</sup> توانستند دسته‌ای از خط‌مشی‌های حذف رده‌بندی را اعمال کنند که از بازخورد برنامه و حضور ورودی وابسته به حالت<sup>۱۰۰</sup> پشتیبانی می‌کند. عملکرد مکانیزم چنداجرایی امن نامتقارن به این صورت است که به جای اجرای یک نسخه از برنامه اصلی برای رونوشت سطح پایین، برش سطح پایینی<sup>۱۰۱</sup> از برنامه اصلی را اجرا می‌کند که مطابق خط‌مشی حذف رده‌بندی تعدیل شده است. این مکانیزم در نحوه ساخت این برش سطح پایین هیچ محدودیتی قائل نیست و اثبات می‌شود که این مکانیزم حتی برای برش سطح پایین ناصحیح نیز درست عمل می‌کند. البته در صورتی که برش سطح پایین صحیح باشد، مکانیزم دقیق خواهد بود. همچنین، معیاری از صحیح‌بودن معنایی بین برش سطح پایین و برنامه اصلی معرفی شده است.

از دیگر معایب روش چنداجرایی امن می‌توان به مشکلات پیاده‌سازی آن در محیط واقعی اشاره کرد. طبق ایده اولیه مطرح شده [۶]، لازم است تا محیط و زیرساخت اجرای برنامه‌ها مطابق با این روش تغییر کند. به این ترتیب، مرورگر یا سیستم‌عامل باید به نحوی تغییر داده شود که بتواند چنداجرایی را به درستی انجام دهد که همین می‌تواند باعث کاهش کارایی سامانه شود. برای افزایش کارایی، Austin و Flanagan [۴۰] روشی به نام وجه‌های چندگانه<sup>۱۰۲</sup> پیشنهاد کردند که ایده آن‌ها شبیه‌سازی چنداجرایی امن با تک اجرا و به کمک استفاده از مقادیر وجه‌دار<sup>۱۰۳</sup> است. در این روش، هر متغیر به ازای

<sup>97</sup> Event-driven

<sup>98</sup> Annotation

<sup>99</sup> Asymmetric SME

<sup>100</sup> State-dependent input presence

<sup>101</sup> Low Slice

<sup>102</sup> Multiple Facets

<sup>103</sup> Faceted Value

هر سطح امنیتی به یک مقدار وجه‌دار نگاشت می‌شود. هر یک از این مقادیر با دید متغیر از نقطه‌نظر مشاهده‌گر سطوح امنیتی متناسب است. مکانیزم به این ترتیب عمل می‌کند که اگر ارتقای حساسی وجود دارد، وجه قابل مشاهده از آن متغیر به‌روز نشود. در صورتی که هیچ ارتقای حساسی وجود نداشته باشد، روش وجه‌های چندگانه با توجه به معناساخت اصلی برنامه به‌روزرسانی‌ها را انجام می‌دهد. این روش اگرچه در ابتدا با هدف شبیه‌سازی چنداجرایی امن همراه با افزایش کارایی مطرح شد، اما نشان داده شده است [۴۲] که از نظر توانایی اعمال خط‌مشی‌ها توانایی‌های چنداجرایی امن را ندارد.

پیشنهاد دیگری که برای عدم تغییر محیط اجرا و بهره‌گیری از چنداجرایی امن مطرح شد، استفاده از روش بازنویسی (تبدیل) برنامه است [۱۰]. بازنویس برنامه این‌گونه عمل می‌کند که برنامه را به عنوان ورودی گرفته و برنامه جدید را از کنار هم قراردادن رونوشت‌های برنامه برای سطوح امنیتی مختلف تولید می‌کند. همچنین روش دیگری برای تبدیل برنامه‌های هم‌روند پیشنهاد شده است که بر طبق آن می‌توان سایر استراتژی‌های زمان‌بندی موجود را پیاده‌سازی کرد. پس با استفاده از این روش، نیازی به تغییر محیط زمان‌اجرای برنامه‌ها نیست.

همان‌طور که پیشتر مطرح شد، چنداجرایی امن تنها قادر به اعمال خط‌مشی عدم تداخل حساس به خاتمه و زمان است و از آن نمی‌توان برای اعمال سایر خط‌مشی‌های جریان اطلاعات استفاده کرد. برای رفع این مسئله، چارچوبی تحت عنوان نگاشت-کاهش<sup>۱۰۴</sup> توسط Ngo [۴۳] پیشنهاد شد که چارچوبی قابل برنامه‌ریزی برای اعمال خط‌مشی‌های امنیتی به شمار می‌شود که در توسعه آن از ایده چنداجرایی امن استفاده شده است. با نوشتن یک برنامه به عنوان نگاشت و برنامه دیگری به عنوان کاهش، نحوه کنترل ورودی‌ها و خروجی‌های رونوشت‌های برنامه اصلی مشخص می‌شوند. برای خط‌مشی‌های عدم تداخل حساس به خاتمه، عدم قابلیت استنتاج و حذف ورودی‌ها، مکانیزم‌هایی بر اساس همین چارچوب ارائه شده است که درست و دقیق هستند.

علاوه بر پیشرفت نظری روش چنداجرایی امن، پیاده‌سازی‌های مختلفی نیز توسط پژوهشگران مطرح شده است [۶]، [۳۴]، [۳۶]، [۳۹]، [۴۰] و [۴۴-۴۷]. Piessens و Devriese [۶] نمونه اولیه‌ای با استفاده از موتور جاوااسکریپت Spider-Monkey پیاده‌سازی کردند و آزمایش‌های خود را روی

<sup>104</sup> Map-Reduce Framework

مجموعه معیار<sup>۱۰۵</sup> نسخه هشت گوگل کروم انجام داده‌اند. Jaskelioff و Russo [۳۹] کتابخانه‌ای برای پیاده‌سازی چنداجرایی امن در زبان هسکل ارائه کرده‌اند. Bielova و همکاران [۴۵] نیز این روش را روی مدلی از مرورگرها به نام Featherfox به زبان Ocaml پیاده‌سازی کردند. همچنین، De Groef و همکاران [۳۴]، [۳۶]، [۴۶] چنداجرایی امن را در Firefox پیاده‌سازی و اولین مرورگر وب کاملاً تابعی<sup>۱۰۶</sup> (FlowFox) را معرفی کردند. تفاوت پیاده‌سازی‌های [۳۰] و [۴۵] با پیاده‌سازی FlowFox این است که آن‌ها برای تضمین امنیت، دو نسخه از مرورگر را اجرا می‌کردند، در حالی که در FlowFox دو نسخه از اسکرپت‌ها در یک مرورگر اجرا می‌شود. علاوه بر این، ایده وجه‌های چندگانه نیز در نسخه‌ای از Firefox پیاده‌سازی شده است [۴۰]. به طور کلی، گزارش نتایج پیاده‌سازی‌های انجام‌شده حکایت از آن دارد که اگرچه به طور ذاتی، هزینه کارایی و حافظه برای پیاده‌سازی این روش قابل توجه است اما با توجه به توانایی آن‌ها در اعمال خط‌مشی‌های حساس به خاتمه و زمان، این هزینه به اندازه‌ای نیست که باعث عدم استفاده از آن‌ها شود.

در جدول ۲، خلاصه‌ای از مکانیزم‌های اعمال مبتنی بر روش چنداجرایی امن، به ترتیب سال، آمده است [۴۳]. شش معیاری که در سرستون‌های جدول مشاهده می‌شود، بر اساس مشخصات مکانیزم‌های اعمال در نظر گرفته شده‌اند:

- (۱) مقایسه زمان‌بندها: مکانیزم‌هایی که تأثیر ترتیب اجرای رونوشت‌ها را در تضمین خط‌مشی‌های مختلف بررسی کرده‌اند؛
- (۲) حفظ ترتیب رویدادها: مکانیزم‌هایی که ترتیب رویدادهای خروجی را در بین کانال‌ها نسبت به یکدیگر حفظ می‌کنند؛
- (۳) پشتیبانی از کانال‌های پویا: مکانیزم‌هایی که در آن‌ها فرضی مبنی بر ثابت و مشخص بودن مجموعه کانال‌ها نشده است و سطوح امنیتی می‌توانند به طور پویا تغییر کنند؛
- (۴) برش نامتقارن: مکانیزم‌هایی که در آن‌ها لزومی ندارد ریشه‌های سطوح پایین دقیقاً رونوشت برنامه اصلی باشند؛

<sup>105</sup> Benchmark Suite

<sup>106</sup> Functional

(۵) پشتیبانی از حذف رده‌بندی: مکانیزم‌هایی که خط‌مشی‌های حذف رده‌بندی توسط آن‌ها قابل اعمال است؛

(۶) پیاده‌سازی: مکانیزم‌هایی که پیاده‌سازی شده‌اند.

جدول ۲ - مقایسه مکانیزم‌های مبتنی بر چنداجرایی امن [۴۳]

مکانیزم	سال ارائه	مقایسه زمان‌بندها	حفظ ترتیب رویدادها	پشتیبانی از کانال‌های پویا	برش نامتقارن	پشتیبانی از حذف رده‌بندی	پیاده‌سازی
Khatiwala و همکاران [۳۱]	۲۰۰۶						✓
Capizzi و همکاران [۳۰]	۲۰۰۸						✓
Piessens و Devriese [۶]	۲۰۱۰						✓
Russo و Jaskelioff [۳۹]	۲۰۱۱						✓
Kashyap و همکاران [۱۷]	۲۰۱۱	✓					
Bielova و همکاران [۴۵]	۲۰۱۱						✓
Flanagan و Austin [۴۰]	۲۰۱۲					✓	✓
Sabelfeld و Rafnsson [۳۲]	۲۰۱۳		✓			✓	
Zanarini و همکاران [۳۳]	۲۰۱۳		✓				
De Groef و همکاران [۳۴]	۲۰۱۴						✓
Vanhoef و همکاران [۳۶]	۲۰۱۴					✓	✓
Jaskelioff و Zanarini [۳۵]	۲۰۱۴			✓			
Garg و Bolosteanu [۳۸]	۲۰۱۶				✓	✓	

### ۳-۴ جمع‌بندی

ابتدا در این فصل به طور کلی به بیان تفاوت‌های مکانیزم‌های ایستا و پویا پرداخته شد و معیارهای مقایسه مکانیزم‌های هر یک از جمله درستی و شفافیت مورد بحث قرار گرفت. در ادامه با تمرکز روی مکانیزم‌های مبتنی بر روش چنداجرایی امن، پیشنهادهای مطرح‌شده به طور اجمالی بررسی و نقاط قوت و ضعف هر یک به طور اختصار بیان شد. در پایان نیز با ارائه جدول ۲، خلاصه‌ای از انواع مکانیزم‌ها که به نوعی با روش چنداجرایی امن مرتبط هستند و توانایی‌های هر یک از آنها ارائه شده است.

## فصل چهارم

### مکانیزم پیشنهادی



## مکانیزم پیشنهادی

در این فصل به بیان مسئله پژوهشی و شرح مکانیزم پیشنهادی می‌پردازیم. ابتدا با توجه به مرور انجام‌شده در فصل سوم درباره مکانیزم‌های اعمال، مزایا و معایب روش چنداجرایی امن مورد بحث قرار می‌گیرد. سپس با بیان مسائل باز موجود در این زمینه پژوهشی، به بیان مسئله این پایان‌نامه پرداخته و نحوه عملکرد مکانیزم پیشنهادی توضیح داده می‌شود.

### ۴-۱ چنداجرایی امن؛ مزایا و چالش‌ها

آن‌چنان که در فصل‌های قبلی مطرح شد، چنداجرایی امن به عنوان یک روش پویای جعبه‌سیاه برای اعمال خط‌مشی‌های جریان اطلاعات معرفی شده است. ایده کلی این روش، ایجاد چندین رونوشت از برنامه و اجرای آن‌ها به ازای هر سطح امنیتی است. منظور از جعبه‌سیاه بودن آن است که بدون در اختیار داشتن کد منبع برنامه و با لحاظ کردن شرایط خاصی برای ورودی‌ها و خروجی‌های تولیدشده رونوشت‌ها، خط‌مشی عدم تداخل اعمال می‌شود. شایان ذکر است که این مکانیزم نیازی به تحلیل ایستا یا نظارت زمان اجرا ندارد و با انتخاب استراتژی زمان‌بندی مناسب، برنامه را مطابق با خط‌مشی جریان اطلاعات اجرا می‌کند. برخلاف مکانیزم‌های نظارت اجرا، هدف از چنداجرایی امن جلوگیری از بروز نقض امنیت نیست، بلکه به دنبال اصلاح اجرای برنامه در زمان اجرا است. از این جهت می‌توان این روش را /امن به واسطه طراحی<sup>۱۰۷</sup> دانست. زیرا، امنیت به واسطه جداسازی محاسبات در سطوح مختلف امنیتی برآورده می‌شود.

با در نظر گرفتن یک شبکه امنیتی دوسطحی، نحوه عملکرد چنداجرایی امن این‌گونه است که دو نسخه از برنامه اصلی به عنوان رونوشت سطح بالا و پایین تهیه شده و در عمل، برنامه اصلی دو بار اجرا می‌شود. ورودی‌های سطح بالای برنامه تنها به رونوشت سطح بالا ارسال می‌شود، در حالی که ورودی‌های سطح پایین در هر دو رونوشت قابل استفاده هستند. به طور کلی، ورودی‌های رونوشت یک سطح امنیتی عبارتند از ورودی‌های همان سطح و سطوح پایین‌تر. همچنین، برای این‌که اجرای رونوشت سطح پایین به دلیل عدم وجود ورودی‌های سطح بالا دچار مشکل نشود، از مقادیر پیش‌فرض ثابت و جعلی برای تأمین

<sup>107</sup> Secure By Design

مقادیر ورودی سطح بالا استفاده می‌شود. درباره خروجی‌های تولیدشده توسط رونوشت‌ها هم فقط خروجی‌های تولیدشده توسط رونوشت همان سطح امنیتی در خروجی قرار می‌گیرد و خروجی‌های سطوح پایین‌تر یا بالاتر تولیدشده یک رونوشت مسدود می‌شوند. به بیان دیگر، هر رونوشت مسئول تولید خروجی‌های سطح امنیتی متناظر با سطح امنیتی خود است. پس به وضوح مشخص است که مقادیر محرمانه (سطح بالا) امکان تأثیرگذاری روی خروجی‌های سطح پایین را نخواهند داشت. زیرا این رونوشت سطح پایین است که وظیفه تولید خروجی‌های سطح پایین را برعهده دارد. همچنین به واسطه لحاظ کردن این‌گونه محدودیت‌ها روی ورودی‌های رونوشت‌ها، امکان وجود جریان‌های صریح و ضمنی نیز از بین می‌رود. در حالی که بررسی این دسته از جریان‌ها یکی از اصلی‌ترین مسائل در طراحی مکانیزم‌های مرسوم پویا به شمار می‌رود.

در ادامه مزایا و معایب موجود در روش چنداجزایی امن بیان می‌شود [۳۲] و مسائل باز مطرح در این حوزه را بیان می‌کنیم.

مزایای استفاده از این روش عبارتند از:

- امنیت به واسطه طراحی: اصلی‌ترین مزیت چنداجزایی امن این است که خط‌مشی عدم تداخل را با یک رویکرد کنترل دسترسی ساده اعمال می‌کند. به این معنا که محاسباتی که برای تولید خروجی یک متغیر سطح پایین نیاز است، هیچ‌گونه دسترسی به مقادیر سطح بالا یا سطوح غیرقابل مقایسه ندارد.
- مستقل از زبان برنامه‌نویسی: مزیت مهم دیگر این دسته از مکانیزم‌ها، جعبه‌سیاه بودن آن است. این روش تنها با اعمال محدودیت‌هایی روی ورودی‌ها و خروجی‌های برنامه عدم تداخل را تضمین می‌کند و برای این کار نیازی به در اختیار داشتن کد منبع برنامه ندارد. در نتیجه، فارغ از زبان برنامه‌نویسی و پیچیدگی‌های برنامه می‌تواند به درستی خط‌مشی را اعمال کند. پس برنامه می‌تواند به هر زبانی نوشته شده و حتی از ارزیابی‌های پویا<sup>۱۰۸</sup> - مانند آن‌چه در جاوااسکریپت وجود دارد - در آن استفاده شده باشد.

<sup>108</sup> Dynamic Evaluation

• شفافیت در هر کانال برای برنامه‌های امن: در روش چنداجرایی امن، در صورتی که برنامه اصلی امن باشد، یعنی مطابق خط‌مشی عدم تداخل باشد، معاشناخت اجرای برنامه تغییر داده نمی‌شود. پس ترتیب رویدادهای خروجی در هر کانال خروجی مشابه ترتیب برنامه اصلی خواهد بود. از طرفی، اجرای رونوشت مربوط به بالاترین سطح امنیتی دقیقاً معادل با همان اجرای برنامه اصلی است. زیرا، تمامی ورودی‌های سطوح مختلف در اختیار این رونوشت قرار گرفته و رفتار برنامه نیز تغییری نمی‌کند. اما نکته اینجاست که این شفافیت در بین کانال‌ها و نسبت به یکدیگر برقرار نیست. بنابراین، بر اساس استراتژی زمان‌بندی اتخاذ شده برای اجرای رونوشت‌ها ممکن است ترتیب و زمان قرارگیری رویدادهای خروجی در کانال‌های خروجی متفاوت از اجرای اصلی برنامه باشد و تضمینی برای حفظ ترتیب رویدادها در کانال‌های مختلف نسبت به یکدیگر وجود ندارد. پس برای مشاهده‌گر سطح بالا که توانایی مشاهده کانال‌های خروجی سطوح پایین‌تر از خود را دارد، نقض شفافیت قابل مشاهده است. اگرچه مکانیزم‌هایی برای رفع این مشکل برای خط‌مشی عدم تداخل حساس به خاتمه پیشنهاد شده است [۳۲]، [۳۳] اما کماکان این مسئله برای خط‌مشی حساس به زمان وجود دارد.

علاوه بر مزایای مطرح شده که این روش را از سایر روش‌های اعمال متمایز می‌کند، استفاده از چنداجرایی امن دارای چالش‌هایی نیز هست که در ادامه به آن‌ها پرداخته می‌شود:

• سطوح درشت‌دانه<sup>۱۰۹</sup> و ثابت: در این روش، سطوح امنیتی از ابتدا ثابت و مشخص در نظر گرفته می‌شوند و کانال‌های ورودی و خروجی برای بعضی از کاربردها ریزدانه‌گی مناسب را برآورده نمی‌کنند. برای کاربردهای آماری ممکن است لازم باشد که محتوای پیام محرمانه برای مشاهده‌گر سطح پایین افشا نشود اما وجود چنین پیامی قابل مشاهده باشد. Ragnsson و Sablefeld [۳۲] برای رفع این مشکل مکانیزمی پیشنهاد داده‌اند که یک سطح میانی به نام M نیز علاوه بر دو سطح بالا و پایین در نظر می‌گیرد. از طرف دیگر، برای قابلیت پشتیبانی از سطوح پویا و قابل تغییر در طول اجرا، Zanarini و Jaskelioff [۳۵] مکانیزمی را مبتنی بر ناظر چنداجرایی پیشنهاد کرده‌اند.

<sup>109</sup> Coarse-grained

- تام‌بودن ورودی‌ها<sup>۱۱۰</sup>: در مکانیزم اصلی پیشنهادشده [۶]، فرض بر تام‌بودن ورودی‌های برنامه است؛ یعنی ورودی‌ها همواره آماده برای استفاده در نظر گرفته می‌شوند و زمانی برای گرفتن ورودی‌ها از کاربر مصرف نمی‌شود. به تعبیر دیگر، ابتدا کاربر تمامی ورودی‌های لازم را در لیستی تهیه کرده و سپس برنامه اجرا می‌شود. در سامانه‌های تعاملی<sup>۱۱۱</sup> که همگام با اجرای برنامه ورودی‌های آن نیز مشخص می‌شوند، ممکن است صرف وجود ورودی‌های سطح بالا یا زمان تأمین آن‌ها نیز مثالی از نقض امنیت محسوب شود. مکانیزم‌های مطرح‌شده در [۳۲] و [۴۵] بدون در نظر گرفتن چنین فرضی عمل می‌کنند که البته قادر به اعمال شفاف خط‌مشی حساس به زمان نیستند.
- زمان‌بندی محدودکننده<sup>۱۱۲</sup>: بعضی از مکانیزم‌ها و پیاده‌سازی‌های انجام‌شده فقط با فرض بهره‌گیری از زمان‌بندی/اولویت با سطح پایین قابل استفاده هستند. دو اشکال اصلی این زمان‌بندی، فرض ترتیب کامل بودن<sup>۱۱۳</sup> شبکه امنیتی و امکان وقوع قحطی‌زدگی<sup>۱۱۴</sup> برای اجرای سایر رونوشت‌هاست. در صورتی که یکی از اجراهای سطوح پایین‌تر خاتمه نیابد، رونوشت‌های سطوح بالا هرگز امکان اجرا نخواهند یافت. از طرفی با در نظر گرفتن چنین فرضی، نمی‌توان امنیت را برای سطوح امنیتی غیرقابل مقایسه تضمین کرد. لازم به یادآوری است که Kashap و همکاران [۱۷] سایر زمان‌بندهای قابل استفاده در این روش را مورد بررسی قرار داده‌اند. همچنین، در پژوهش‌های بعدی نشان داده شده است که تنها شرط قطعی‌بودن و منصف‌بودن زمان‌بند برای اعمال خط‌مشی عدم تداخل توسط مکانیزم‌های مبتنی بر این روش کفایت می‌کند [۴]، [۳۲] و [۳۳].
- پشتیبانی از حذف رده‌بندی: با توجه به جداسازی کامل ورودی‌های سطح بالا از رونوشت سطح پایین، اعمال خط‌مشی‌های حذف رده‌بندی توسط مکانیزم اولیه پیشنهادشده پشتیبانی نمی‌شد، در حالی که چنین خط‌مشی‌هایی در دنیای واقعیت بسیار کاربردی

<sup>110</sup> Total Input<sup>111</sup> Interactive<sup>112</sup> Restrictive Scheduling<sup>113</sup> Total Order<sup>114</sup> Starvation

هستند و در شرایطی لازم است که اطلاعات سطح بالا به صورت کنترل شده و عامدانه به سطح پایین منتقل شوند. از همین رو، بسیاری از پژوهش‌های این حوزه برای رفع این نقص مکانیزم‌هایی را پیشنهاد کرده‌اند [۳۲]، [۳۶]، [۳۸] و [۴۴]. البته با توجه به ابعاد مختلف خط‌مشی‌های حذف رده‌بندی [۴۱]، تاکنون فقط محور اول؛ یعنی چه چیزی<sup>۱۱۵</sup>، توسط مکانیزم‌های پیشنهاد شده اعمال پذیر است.

- تغییر در ترتیب رویدادها: اگرچه اثبات می‌شود که چنداجرایی امن دقیق است [۶]، اما تنها ترتیب رویدادهای مربوط به یک کانال خاص حفظ می‌شود و رویدادها در کانال‌های مختلف نسبت به یکدیگر دچار تغییر می‌شوند. به این ترتیب و به خصوص در خط‌مشی حساس به زمان نیاز است که رفتار برنامه‌های امن در بین کانال‌های سطوح امنیتی مختلف نیز دست‌نخورده باقی بماند.

- عدم تشخیص نقض امنیت: با توجه به معاشناخت چنداجرایی امن، امکان تشخیص نقض امنیت وجود ندارد. زیرا، در طول زمان اجرا با تغییر مقادیر ورودی سطوح بالا به مقادیر از پیش تعیین شده و ثابت، به ازای مقادیر ثابت ورودی سطوح پایین، رفتار یکسانی از برنامه مشاهده می‌شود. به این ترتیب، مکانیزم مستقل از امن بودن یا نبودن برنامه عملکرد یکسانی خواهد داشت. از طرفی، تعیین مقادیر پیش فرض در تعیین مسیرهای ممکن اجرای رونوشت‌ها نیز تأثیرگذار است. با توجه به این که این روش در زمان اجرا به اصلاح رفتار برنامه می‌پردازد، کاربر نمی‌تواند تشخیص دهد که تغییر صورت گرفته در رفتار برنامه به دلیل ناامن بودن برنامه بوده است یا به واسطه معاشناخت چنداجرایی، رفتار برنامه تغییر کرده است. شفافیت کامل باعث ارائه تضمین به کاربر می‌شود که در تنها در صورتی رفتار برنامه اصلی دچار تغییر شده است که برنامه ورودی ناامن باشد. دستاورد دیگر Rafnsson و Sabelfeld [۳۲] ارائه مکانیزمی برای برقراری شفافیت کامل برای عدم تداخل حساس به خاتمه است. به این ترتیب مکانیزم قادر است که حملات احتمالی ناقض امنیت را نیز به کاربر گزارش دهد.

<sup>115</sup> What

- عدم قطعیت<sup>۱۱۶</sup>: با توجه به این که روش چنداجرایی امن مستقل از پیچیدگی‌های برنامه اصلی عمل می‌کند، می‌توان نتیجه گرفت که خط‌مشی عدم تداخل برای برنامه‌های غیرقطعی نیز توسط این روش قابل اعمال است. اما به دلیل عدم وجود صورت‌بندی<sup>۱۱۷</sup> مناسب در بیان خط‌مشی‌های امنیتی مربوط به عدم قطعیت، تنها به طور غیرصوری درباره توانایی‌های این روش صحبت می‌شود. اگرچه می‌توان نشان داد که خط‌مشی‌های سخت‌گیرانه‌ای مانند قطعیت مشاهده‌ای، که قابل بیان توسط عدم تداخل است، توسط روش چنداجرایی امن به درستی اعمال می‌شود.
- انتخاب مقادیر پیش‌فرض مناسب: اگرچه طبق تعریف خط‌مشی عدم تداخل، هر گونه ورودی دلخواهی به عنوان مقدار پیش‌فرض به جای ورودی‌های سطح بالا در رونوشت‌های سطح پایین قابل ارسال است اما انتخاب مناسب این مقادیر برحسب برنامه اصلی در رفتار اجرای برنامه تأثیرگذار بوده و ممکن است انتخاب نامناسب این مقادیر، منجر به بروز مشکلات در اجرای رونوشت‌ها شود. همچنین نشان داده شده است [۱۶] که انتخاب‌های مختلف این مقادیر در میزان شفافیت کاذب نسبی چنداجرایی امن تأثیر مستقیم دارد. اگرچه پیشنهاد مکانیزم چنداجرایی امن نامتقارن [۳۸] استفاده از برش‌های سطح پایین است اما نحوه تولید این برش‌ها و انتخاب مقادیر مناسب برای آن کماکان یکی از چالش‌ها به شمار می‌رود.
- کارایی پایین: می‌توان گفت که اصلی‌ترین اشکال ذاتی چنداجرایی امن، کارایی پایین آن است. اجرای چندباره یک برنامه قطعاً در افزایش زمان و حافظه مصرفی اجرا تأثیرگذار است. اگرچه با فرض در اختیار داشتن پردازنده‌های چند هسته‌ای و انجام بهینه‌سازی قبل از اجرا می‌توان این سربار را کاهش داد. با این حال، طبق نتایج حاصل از آزمایش‌های صورت گرفته بر روی مکانیزم‌های پیاده‌سازی شده نظیر [۶]، سربار روش چنداجرایی امن کمی کمتر از حاصل ضرب زمان اجرا در تعداد سطوح امنیتی است. در صورتی که از استراتژی زمان‌بندی موازی استفاده شود، تحلیل هزینه حافظه مصرفی جایگزین هزینه زمان اجرا می‌شود.

<sup>116</sup> Nondeterminism<sup>117</sup> Formalism

با توجه به نکات مطرح شده در بالا می‌توان دریافت که تضمین شفافیت کامل و حفظ ترتیب رویدادها در بین کانال‌های سطوح مختلف امنیتی برای خط‌مشی عدم تداخل حساس به زمان از مهم‌ترین سوالات پژوهشی این حوزه محسوب می‌شود. زیرا، عدم تغییر رفتار برنامه‌های امن از معیارهای اصلی مقایسه مکانیزم‌ها با یکدیگر است. همچنین، در صورتی که کاربر برای برنامه‌های امن نیز شاهد تغییر رفتار اجرا باشد، عملاً قادر به عیب‌یابی<sup>۱۱۸</sup> برنامه نخواهد بود. علاوه بر این، بررسی میزان تغییرات ناشی از چنداجرایی امن برای برنامه‌های ناامن به عنوان محور پژوهشی دیگری قابل بحث است.

## ۴-۲ شرح مکانیزم پیشنهادی

همان‌طور که در فصل سوم مطرح شد، تاکنون برای حل مسئله حفظ ترتیب رویدادهای خروجی در بین کانال‌های خروجی نسبت به یکدیگر دو مکانیزم [۳۲] و [۳۳] پیشنهاد شده است. در مکانیزم ناظر چنداجرایی امن [۳۳]، همراه با اجرای اصلی برنامه روش چنداجرایی امن با یک نوع استراتژی زمان‌بندی حفظ‌کننده ترتیب رویدادها نیز مورد استفاده قرار می‌گیرد و پس از هر بار تولید خروجی توسط یکی از این دو، نوبت به دیگری داده می‌شود تا در صورت امن‌بودن برنامه، همگام با یکدیگر پیش بروند. در صورتی که در هر مرحله از اجرا همگام‌بودن از بین برود، هشدار به کاربر مبنی بر نقض عدم تداخل داده شده و از ادامه اجرا جلوگیری می‌شود. البته به طور صریح ذکر شده است که برای سادگی و دقت مکانیزم اعمال پیشنهادی، کانال‌های نهان زمانی خارجی در این مکانیزم در نظر گرفته نشده است و استراتژی زمان‌بندی پیشنهادی نیز بر همین مبنا عمل می‌کند.

علاوه بر این، در فصل سوم با توضیح مختصر مکانیزم پیشنهادی Sabelfeld و Rafnsson [۳۲]، مشخص شد که این مکانیزم نیز برای اعمال خط‌مشی عدم تداخل حساس به زمان با شفافیت کامل مناسب نیست و به واسطه انتظار رونوشت سطح پایین برای رسیدن رونوشت سطح بالا به دستور خروجی مورد نظر و بررسی آن‌ها، کانال نهان زمانی ایجاد می‌شود. بنابراین، در ادامه مکانیزمی پیشنهاد می‌کنیم که خط‌مشی عدم تداخل حساس به زمان را درست و شفاف کامل اعمال می‌کند و قادر است تا در صورت بروز نقض امنیت هشدار به کاربر بدهد.

<sup>118</sup> Debugging

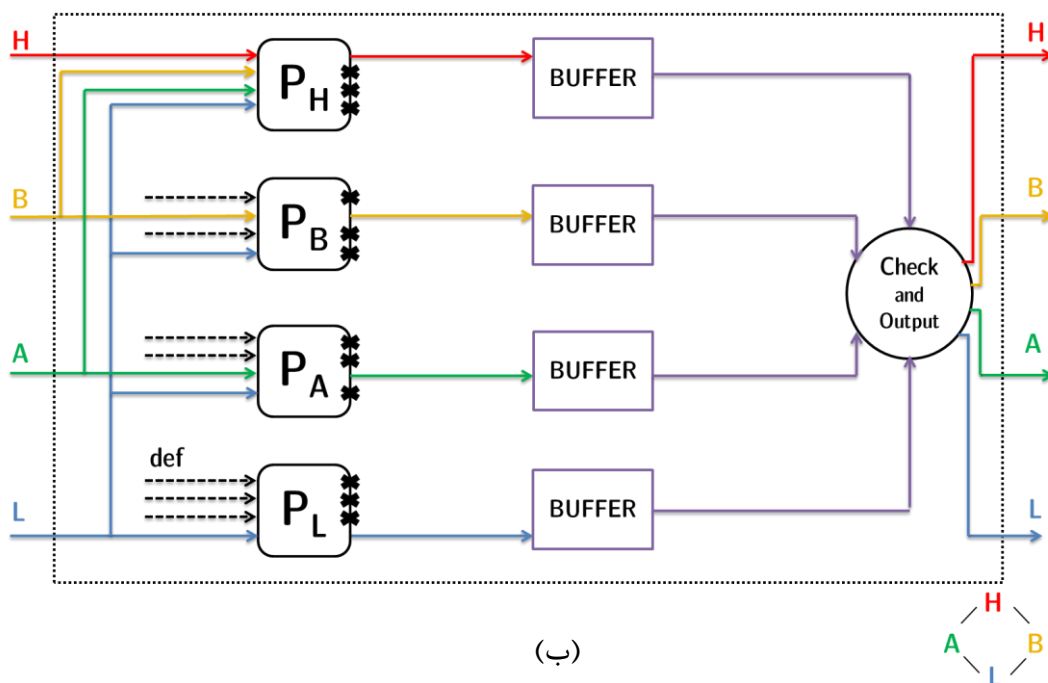
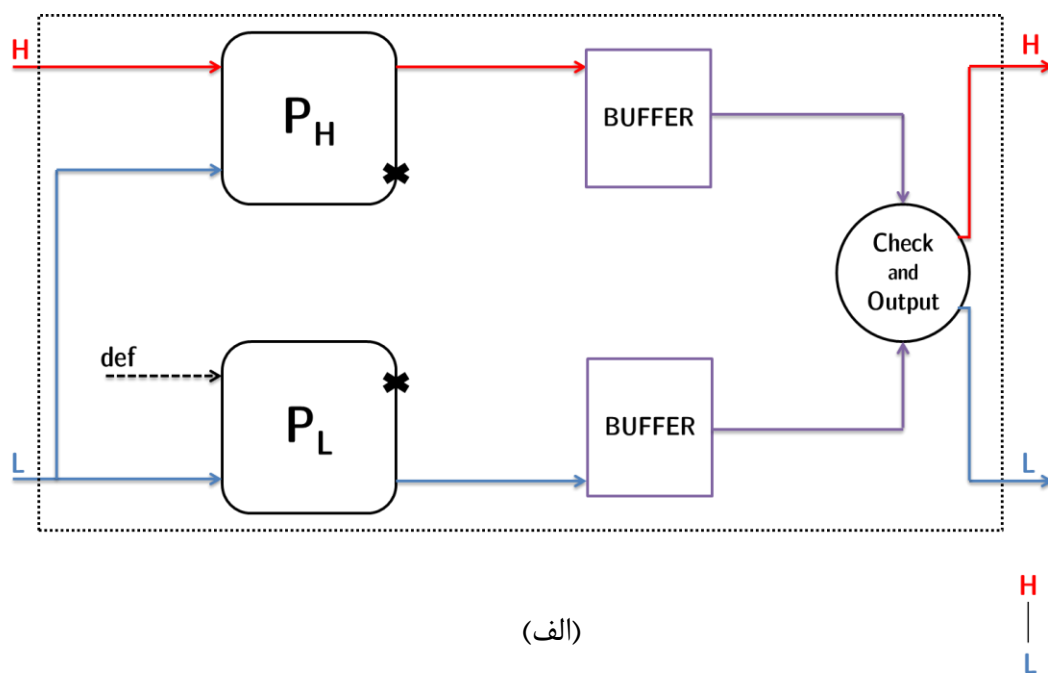
مکانیزم پیشنهادی را تحت عنوان چند/جاری/امن بافردار<sup>119</sup> (BSME) معرفی می‌کنیم. همان‌طور که از نام آن مشخص است، این مکانیزم از بافردی برای نگه‌داری موقت رویدادهای خروجی استفاده می‌کند. نحوه عملکرد مکانیزم به این صورت است که مطابق ایده اصلی روش چنداجرایی امن، به ازای هر سطح امنیتی نسخه‌ای از برنامه تهیه می‌شود. ورودی‌های هر رونوشت عبارتند از مقادیر موجود در کانال‌های ورودی همان سطح و سطوح پایین‌تر، و برای ورودی‌های سطوح دیگر از مقادیر پیش‌فرض ثابت استفاده می‌شود. همچنین، رویدادهای خروجی هم‌سطح هر رونوشت در بافر مربوط به آن رونوشت نگه‌داری می‌شود. در نتیجه، مشابه چنداجرایی امن، اجازه تأثیرگذاری به ورودی‌های مختلف سطوح بالاتر یا غیرقابل مقایسه به رونوشت داده نمی‌شود. تفاوت اصلی این مکانیزم با چنداجرایی امن مطرح‌شده در [۶] این است که رویدادهای خروجی، بلافاصله بعد از تولید توسط اجرای رونوشت به کانال‌های خروجی منتقل نمی‌شود. با در نظر گرفتن بافرهایی برای هر رونوشت، رویداد خروجی تولیدشده توسط اجرای آن رونوشت به بافر منتقل می‌شود و در ادامه پس از بررسی رویدادهای تولیدشده در رونوشت‌های مختلف، مقادیر خروجی به کانال‌های خروجی مرتبط انتقال می‌یابند. علاوه بر این، محل قرارگیری رویداد خروجی در بافر نیز اهمیت دارد و معادل است با تعداد گام‌های طی‌شده از اجرای رونوشت در نوبت جاری. در واقع، در هر نوبت از اجرای یک رونوشت، یک رویداد در خانه متناظر با زمان تولید آن رویداد بافر قرار می‌گیرد و به این ترتیب، زمان تولید رویداد نیز در اختیار مکانیزم است. در شکل ۸ الف، نمایی از عملکرد این مکانیزم برای شبکه دوسطحی  $\mathcal{L}_{L,H} = \{(L,L), (L,H), (H,H)\}$  و در شکل ۸ ب، برای شبکه چهار سطحی  $\mathcal{L}_{L,A,B,H} = \{(L,L), (L,A), (L,B), (A,A), (B,B), (A,H), (B,H), (H,H)\}$  قابل مشاهده است.

با توجه به این که به دنبال اعمال خط‌مشی عدم تداخل حساس به زمان هستیم، یک استراتژی زمان‌بندی برای اجرای رونوشت‌ها مشابه استراتژی زمان‌بندی تسهیم نوع ۲ [۱۷] پیشنهاد کرده‌ایم. از آنجایی که برای اجرای هر رونوشت به ورودی‌های سطوح پایین‌تر نیز نیاز است، پس ابتدا نوبت اجرا به رونوشت دارای پایین‌ترین سطح امنیتی داده می‌شود. سپس سطوح بالاتر از آن و به همین ترتیب تا در نهایت رونوشتی که دارای بالاترین سطح امنیتی است، اجرا می‌شوند. با توجه به این که وقتی نوبت به اجرای رونوشتی می‌رسد، حتماً پیش از آن رونوشت‌های سطوح پایین‌تر اجرا شده‌اند، نگرانی‌ای در خصوص

<sup>119</sup> Buffered Secure Multi-Execution



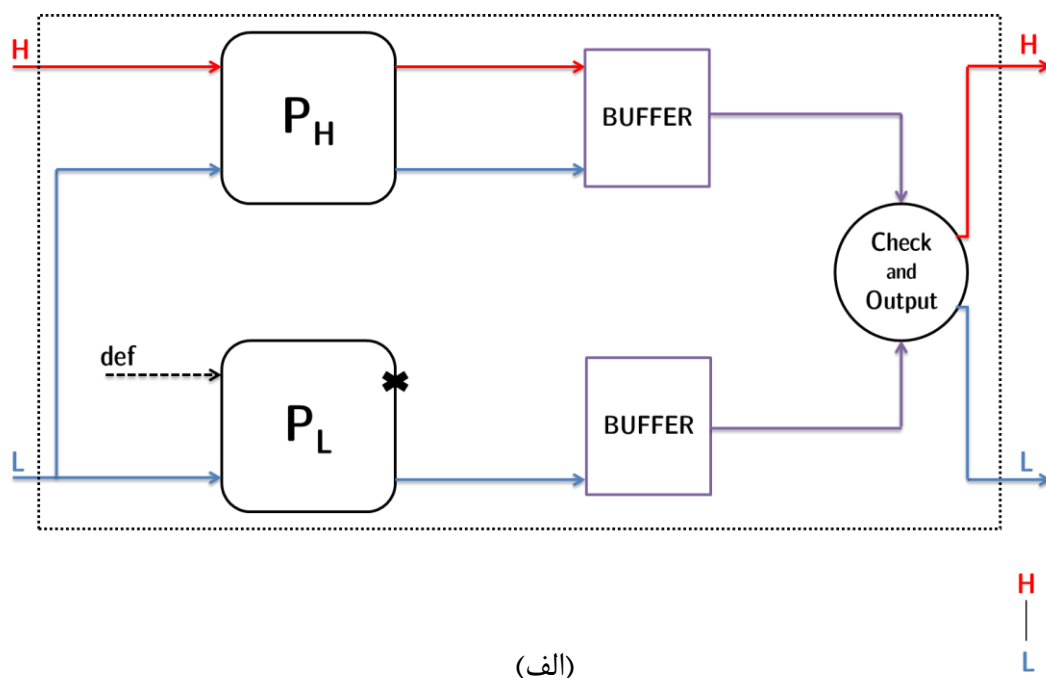
استفاده مجدد از ورودی‌های کانال‌های پایین‌تر وجود ندارد و ورودی‌های سطوح پایین‌تر آماده استفاده مجدد هستند.



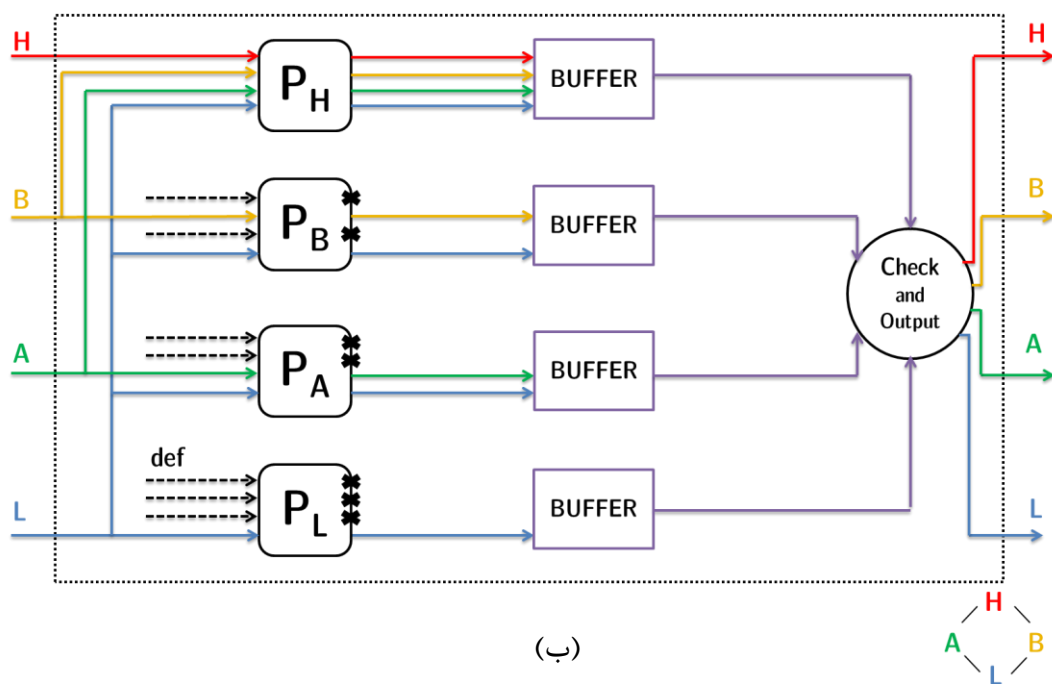
شکل ۸ - نمایی از عملکرد مکانیزم چنداجرایی امن بافردار (الف) برای شبکه دوسطحی، (ب) برای شبکه چهارسطحی

برای اعمال عدم تداخل حساس به زمان نیاز است که زمان رویدادهای خروجی مستقل از ورودی‌های سطح بالاتر باشد. به همین منظور، سهم زمانی مشخصی ( $t$ ) برای هر نوبت از اجرای رونوشت‌ها در نظر گرفته می‌شود. بنابراین، هر رونوشت به مدت تعیین‌شده در هر نوبت اجرا شده و سپس نوبت را به رونوشت بعدی می‌دهد. این کار تا زمانی که اجرا به رونوشت دارای بالاترین سطح امنیتی برسد، ادامه می‌یابد. پس از اجرای بالاترین رونوشت، به اندازه یک سهم زمانی برای بررسی رویدادهای خروجی موجود در بافرهای رونوشت‌ها اختصاص می‌یابد. در صورتی که برنامه اصلی امن بوده باشد؛ یعنی مطابق با خط‌مشی عدم تداخل حساس به زمان باشد، پس همه عنصرهای هم‌شماره از بافرها با یکدیگر سازگار هستند و در هر گام، تنها یک رویداد خروجی به کانال مربوط به سطح خودش منتقل می‌شود. اما در صورتی که برنامه ناقض خط‌مشی باشد، ممکن است این سازگاری وجود نداشته باشد. در این حالت نیز در هر گام از بررسی بافرها، رویدادهای خروجی تولیدشده توسط سطوح امنیتی به کانال‌های خروجی منتقل می‌شود که در این حالت ممکن است بیش از یک رویداد خروجی در هر گام به کانال‌ها ارسال شوند. تفاوت این مکانیزم با سایر مکانیزم‌ها در آن است که حتی در صورت نقض امنیت اجرا قطع نشده و خروجی‌ها به کانال‌های مرتبط منتقل می‌شوند. در ادامه مکانیزمی پیشنهاد می‌کنیم که به محض نقض امنیت، ناسازگاری به وجود آمده را تشخیص داده و از طریق هشدار به کاربر اطلاع می‌دهد.

برای حالتی که بخواهیم مکانیزم نقض امنیت را گزارش دهد، باید به جای ثبت و نگهداری تنها رویدادهای خروجی هر سطح در بافر، همه رویدادهای ورودی و خروجی همان سطح و سطوح پایین‌تر در بافر نگهداری شود. زیرا ممکن است ورودی محرمانه در برنامه ناامن بر روی زمان درخواست برنامه برای خواندن ورودی‌های سطح پایین تأثیر بگذارد که می‌تواند باعث نشت اطلاعات شود. همچنین، باید رویدادی خاص مانند  $\sim$  برای نشان دادن پیشروی اجرای برنامه لحاظ شود تا پیشروی برنامه برای انجام محاسبات از خاتمه متمایز شود. زیرا رفتار خاتمه اجرا نیز در مفهوم امنیت مورد استفاده اهمیت دارد. بنابراین، اجرای بالاترین رونوشت مشابه اجرای اصلی برنامه می‌شود و می‌توان رفتار رونوشت‌های سایر سطوح را از نظر زمان و مقدار تولیدشده رویدادها با آن مقایسه کرد. شکل ۹ بیانگر مکانیزم در حالت گزارش نقض امنیت برای دو حالت شبکه دوسطحی و چهارسطحی است.



(الف)



(ب)

شکل ۹ - نمایی از عملکرد مکانیزم چنداجرایی امن بافردار به همراه گزارش نقض امنیت  
(الف) برای شبکه دوسطحی، (ب) برای شبکه چهارسطحی

دقت شود در صورتی که زمان‌بندی دیگری مورد استفاده قرار بگیرد، ممکن است اجرای رونوشت سطح بالا به خاطر عدم فراهم‌بودن ورودی سطح پایین تا پایان سهم زمانی خود منتظر بماند و در نوبت بعدی اجرا ادامه پیدا کند. بنابراین، اگر برنامه اصلی امن بوده باشد، به واسطه زمان‌بندی نامناسب ترتیب رویدادها به هم ریخته و برنامه به اشتباه ناامن تلقی می‌شود. همچنین، در حالت بدون گزارش نقض امنیت نیز عدم استفاده از این زمان‌بندی باعث از دست رفتن شفافیت کامل مکانیزم می‌شود. از طرفی، با توجه به این که فاصله زمانی بین نوبت‌های اجرای یک رونوشت مقداری ثابت و مستقل از ورودی‌های سطوح بالاتر است، رونوشت سطح پایین نمی‌تواند هیچ اطلاعاتی از مقادیر محرمانه به دست آورد و مقادیر سطح بالا حتی روی زمان اجرای آن‌ها نیز تأثیرگذار نیستند. لازم به یادآوری است که انتخاب رونوشت بعدی برای اجرا از بین همه ریشه‌های آماده، مسدودشده و خاتمه‌یافته صورت می‌گیرد و زمانی اجرای برنامه تحت این مکانیزم خاتمه می‌یابد که همه ریشه‌ها در حالت خاتمه‌یافته باشند. بنابراین، به طور شهودی می‌توان دریافت که مکانیزم ارائه‌شده برای خط‌مشی عدم تداخل حساس به زمان هم درست و هم شفاف کامل است. علاوه بر این، در فصل پنجم با ارائه صورت‌بندی و قضایا این موضوع را اثبات می‌کنیم.

نکته حائز اهمیت آن است که طول بافر محدود و به اندازه مقدار سهم زمانی ( $t$ ) است. زیرا در صورتی که دستورات برنامه همگی دستورات خروجی باشند، اجرای یک رونوشت در مدت سهم زمانی تعیین‌شده نهایتاً قادر به تولید  $t$  رویداد خروجی خواهد بود و با توجه به این که در نوبت بعدی نیازی به نگهداری مقادیر بافر نیست، پس می‌تواند بافر مجدداً خالی شده و برای نگهداری مقادیر جدید باز استفاده شود. همچنین یکسان‌بودن اندازه هر بافر و مقدار سهم زمانی کمک می‌کند که بتوان در هر نوبت از اجرا دقیقاً تعیین کرد که یک رویداد در کدام عنصر بافر قرار بگیرد که این نکته با توجه به خط‌مشی حساس به زمان، نقشی تعیین‌کننده در قسمت بررسی بافرها و خروجی‌دادن نهایی ایفا می‌کند. باید توجه داشت که محدودیتی برای طول بافر وجود ندارد. اما کم‌بودن مقدار  $t$  باعث تعویض مکرر ریشه‌ها در هنگام اجرا و زیادبودن آن نیز باعث هدررفت زمان اجرا برای برنامه‌های کوچک می‌شود.

با توجه به این نوع از زمان‌بندی، مکانیزم قادر است تا عدم تداخل حساس به زمان را به درستی اعمال کند و با استفاده از بافر و نگهداشتن موقت رویدادها، برای برنامه‌های امن شفافیت و ترتیب رویدادهای خروجی را در تمامی کانال‌ها و بین کانال‌ها حفظ کند. دقت شود که در هر نوبت پس از اجرای بالاترین رونوشت، یک سهم زمانی صرف بررسی بافرها و انتقال خروجی‌ها به کانال‌های خروجی می‌شود.

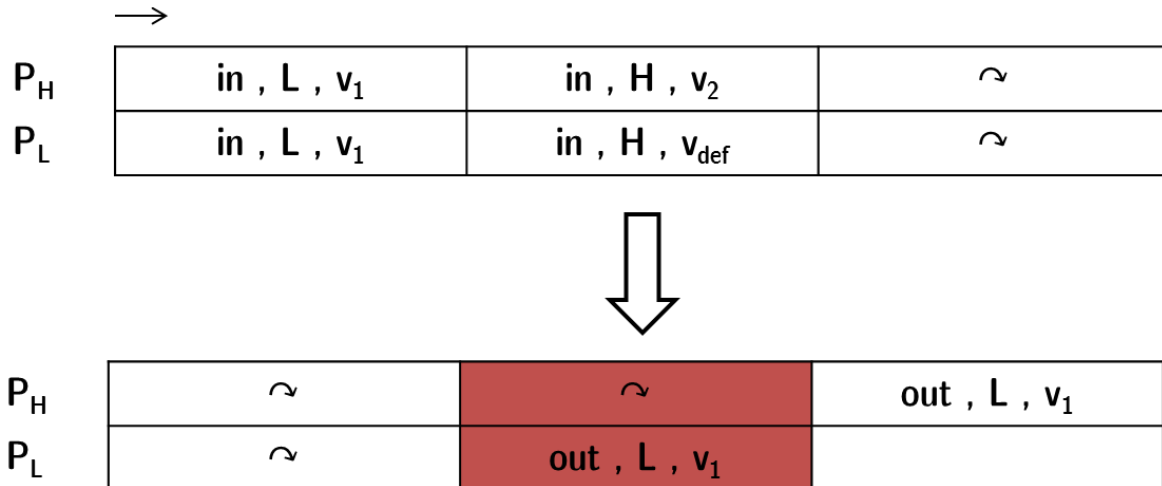
بنابراین، برای تعیین هزینه حافظه و زمان مصرفی باید گفت که مکانیزم پیشنهادی به تعداد حاصل ضرب تعداد سطوح امنیتی در اندازه سهم زمانی به حافظه بافر نیاز دارد. همچنین، در صورتی که اجرای برنامه اصلی  $T$  ثانیه به طول بکشد، در مکانیزم اولیه چنداجرایی امن این مقدار در تعداد سطوح امنیتی ضرب می‌شود و در مکانیزم پیشنهادی، این زمان برابر با حاصل ضرب مدت زمان اجرای برنامه اصلی در یکی بیشتر از تعداد سطوح امنیتی خواهد بود. اگرچه به نظر می‌رسد این مکانیزم هم از نظر حافظه و هم از نظر زمان کارایی پایین‌تری دارد، اما جدای از تضمین درستی و شفافیت کامل اعمال خط‌مشی حساس به جریان، مرحله بررسی و خروجی دادن نهایی می‌تواند همزمان با اولین اجرا از نوبت بعدی انجام بگیرد. شایان ذکر است که اندازه سهم زمانی حتی می‌تواند فقط یک باشد. بنابراین، حافظه مصرفی نیز در مقایسه با سایر مکانیزم‌های مبتنی بر چنداجرایی امن ناچیز خواهد بود.

در ادامه با ذکر مثال‌هایی از برنامه‌های ساده، نحوه عملکرد مکانیزم در حالت همراه با نقض گزارش امنیت تشریح می‌شود. در صورتی که تنها رویدادهای خروجی سطح امنیتی متناظر با سطح امنیتی رونوشت در بافر در نظر گرفته شود، رفتار مکانیزم در حالت اول نتیجه می‌شود. برای سادگی، شبکه دوسطحی، مقدار پیش‌فرض برای ورودی‌های سطح بالا صفر و مقدار اصلی برای این ورودی‌ها غیرصفر فرض شده است. همچنین، اندازه بافر و سهم زمانی برابر ۳ در نظر گرفته شده است. برای اختصار، تنها محتوای بافرها در انتهای هر نوبت نشان داده می‌شود. در هر عنصر از بافر، یا یک سه تایی ورودی/خروجی، سطح امنیتی رویداد و مقدار آن متغیر نگه‌داری می‌شود یا از نماد  $\sim$  برای بیان یک گام پیشروی محاسبه استفاده می‌شود. در صورتی که در عنصر نمادی وجود نداشته باشد، به معنای خاتمه اجراست. همچنین، ستون رنگ‌شده بیانگر محل نقض امنیت است.

```
inL y ;    // low input : y
inH x ;    // high input : x
if x then skip ; skip ;
else skip ;
outL y    // low output : y
```

شکل ۱۰ - برنامه دارای کانال زمانی

برنامه شکل ۱۰، برنامه‌ای است که طبق عدم تداخل حساس به زمان ناامن است. نحوه عملکرد مکانیزم پیشنهادی در قبال این برنامه در شکل ۱۱ مشخص شده است.



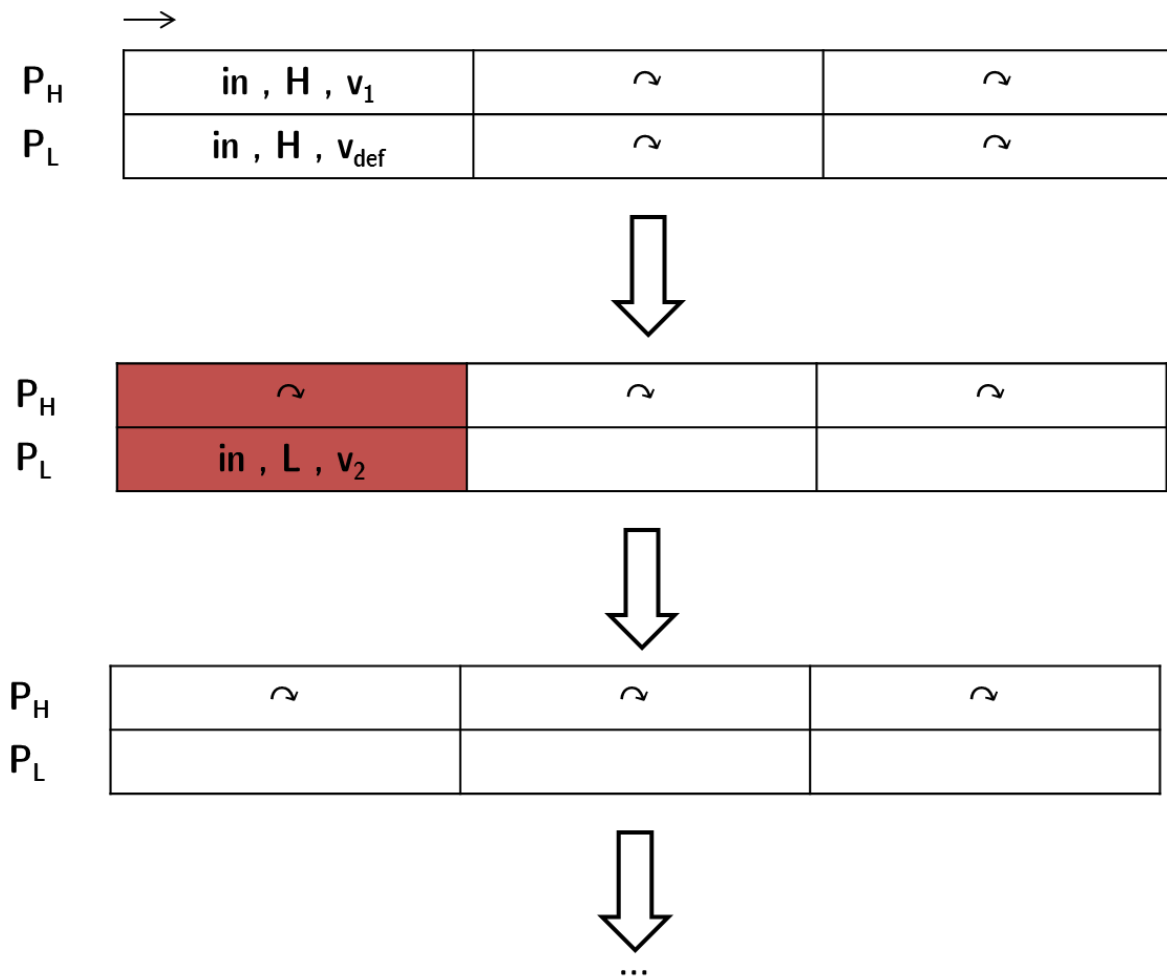
شکل ۱۱ - نمایی از محتوای بافرها در مکانیزم پیشنهادی در هنگام اجرای برنامه شکل ۱۰

در برنامه شکل ۱۰، در صورتی که مقدار پیش‌فرض ورودی سطح بالا با مقدار اصلی برابر باشد، آن‌گاه اجرای هر دو رونوشت مشابه یکدیگر خواهد بود و مکانیزم نیز بدون اعلام نقض امنیت مقدار خروجی را به کانال خروجی سطح پایین منتقل می‌کند. همچنین دقت شود که در آخرین عنصر بافر در شکل ۱۱، رویداد خروجی سطح پایین در رونوشت سطح بالا صرفاً برای بررسی امنیت قرار گرفته است و تأثیری در کانال‌های خروجی نخواهد داشت.

برنامه شکل ۱۲، نمونه‌ای از برنامه‌های دارای کانال خاتمه است و رفتار مکانیزم در قبال این برنامه نیز در شکل ۱۳ قابل مشاهده است. تفاوت این برنامه با برنامه قبلی در آن است که دستور خروجی برای سطح پایین وجود ندارد. اما باید دقت داشت که وجود دستور ورودی، خود باعث ایجاد کانال نهان می‌شود.

```
inH x ;    // high input : x
if x then while true do skip ;
else skip ;
inL y ;    // low input : y
```

شکل ۱۲ - برنامه دارای کانال خاتمه



شکل ۱۳ - نمایی از محتوای بافرها در مکانیزم پیشنهادی در هنگام اجرای برنامه شکل ۱۲

همان‌طور که در شکل ۱۳ مشاهده می‌شود، با لحاظ کردن رویداد ورودی سطح پایین در بافر، مکانیزم می‌تواند نقض امنیت را تشخیص دهد. همچنین با توجه به این که رونوشت سطح بالا در حالت واگرایی خاموش قرار گرفته است، اجرای این برنامه تحت مکانیزم نیز واگرا خواهد بود. با توجه به این که هیچ رویداد خروجی در کانال‌های خروجی ثبت نمی‌شود، اجرای برنامه تحت مکانیزم امن خواهد بود. نکته دیگر درباره این مثال آن است که در صورتی که مقدار پیش‌فرض ورودی سطح بالا غیرصفر در نظر گرفته شود و مقدار اصلی آن صفر باشد، آن‌گاه این بار اجرای رونوشت سطح بالا به دستور  $y \leftarrow inL$  می‌رسد، در حالی که اجرای رونوشت سطح پایین واگرا شده است. با توجه به این که مقداری دستورات ورودی فقط در اجرای رونوشت با همان سطح امنیتی انجام می‌شود، پس در این حالت، اجرای رونوشت سطح بالا باید به مقدار نامتناهی برای استفاده مجدد از ورودی خوانده‌شده توسط رونوشت سطح پایین منتظر بماند.

اگرچه این رفتار خللی به درستی مکانیزم وارد نمی‌کند، اما این نکته در ذات روش چنداجرایی امن وجود دارد.

برنامه شکل ۱۴ مثالی از برنامه ناامن است. این برنامه به دلیل وجود جریان ضمنی، طبق خط‌مشی عدم تداخل غیرحساس به خاتمه نیز ناامن شناخته می‌شود. همان‌طور که در شکل ۱۵ نشان داده می‌شود، اجرای این برنامه تحت مکانیزم امن خواهد شد.

```
inH x ;    // high input : x
if x then skip ;
else outL 0 ; // low input : 0
```

شکل ۱۴ - برنامه دارای کانال جریان ضمنی

→

$P_H$	in , H , $v_1$	$\leadsto$	$\leadsto$
$P_L$	in , H , $v_{\text{def}}$	$\leadsto$	out , L , 0

شکل ۱۵ - نمایی از محتوای بافرها در مکانیزم پیشنهادی در هنگام اجرای برنامه شکل ۱۴

مطابق با شکل ۱۵، مقدار صفر برای کانال خروجی سطح پایین ارسال می‌شود اما نقض امنیت نیز گزارش می‌شود. در صورتی که به جای دستور **outL 0**، دستوری مانند **inL y** قرار می‌گرفت، مجدداً نقض امنیت گزارش می‌شد. با این تفاوت که مقداری برای کانال خروجی سطح پایین داده نمی‌شد. در واقع باید توجه داشت که یک دستور ورودی، به نوعی شامل یک دستور خروجی و قابل مشاهده نیز می‌شود. زیرا پس از درخواست برنامه برای گرفتن ورودی، مقدار ورودی تأمین می‌شود.

برنامه شکل ۱۶، برنامه‌ای مطابق با عدم تداخل حساس به زمان است. رفتار مکانیزم در شکل ۱۷ بیانگر آن است که شفافیت کامل برای این برنامه به دست آمده است. در حالی که به طور مثال، در صورت استفاده از زمان‌بندی /اولویت با سطح پایین، ترتیب رویدادهای خروجی در کانال‌های مختلف حفظ نمی‌شود.

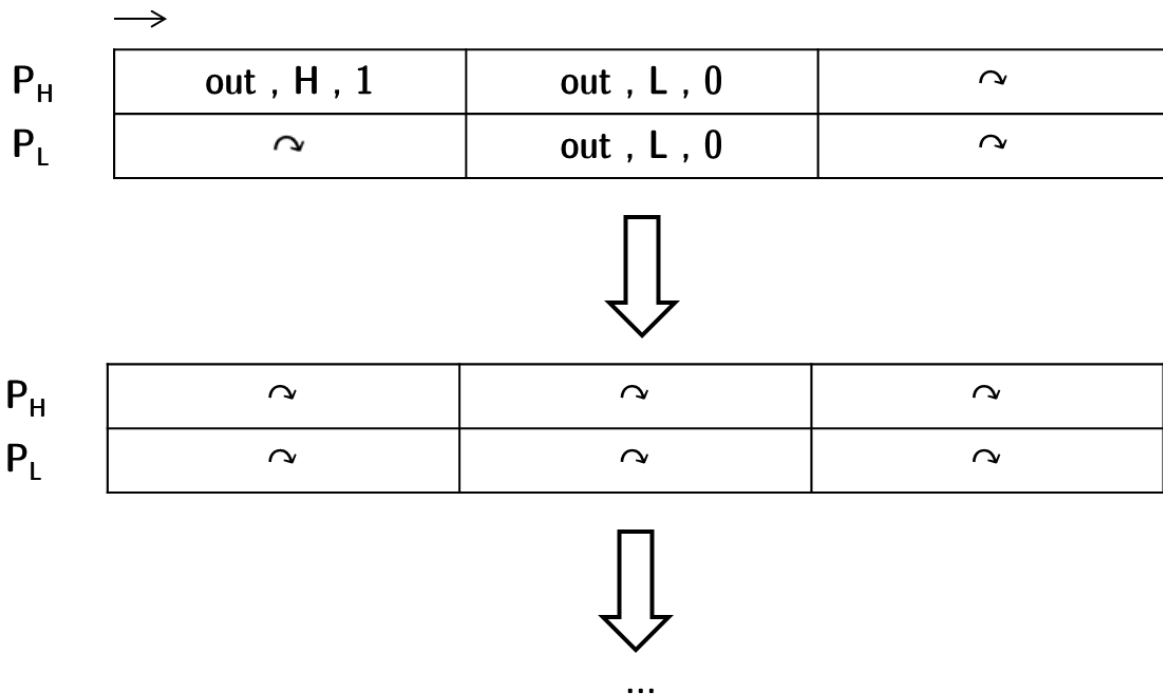


```

outH 1 ;    // high output : 1
outL 0 ;    // low output : 0
while true do skip ;

```

شکل ۱۶ - برنامه امن طبق عدم تداخل حساس به زمان



شکل ۱۷ - نمایی از محتوای بافرها در مکانیزم پیشنهادی در هنگام اجرای برنامه شکل ۱۶

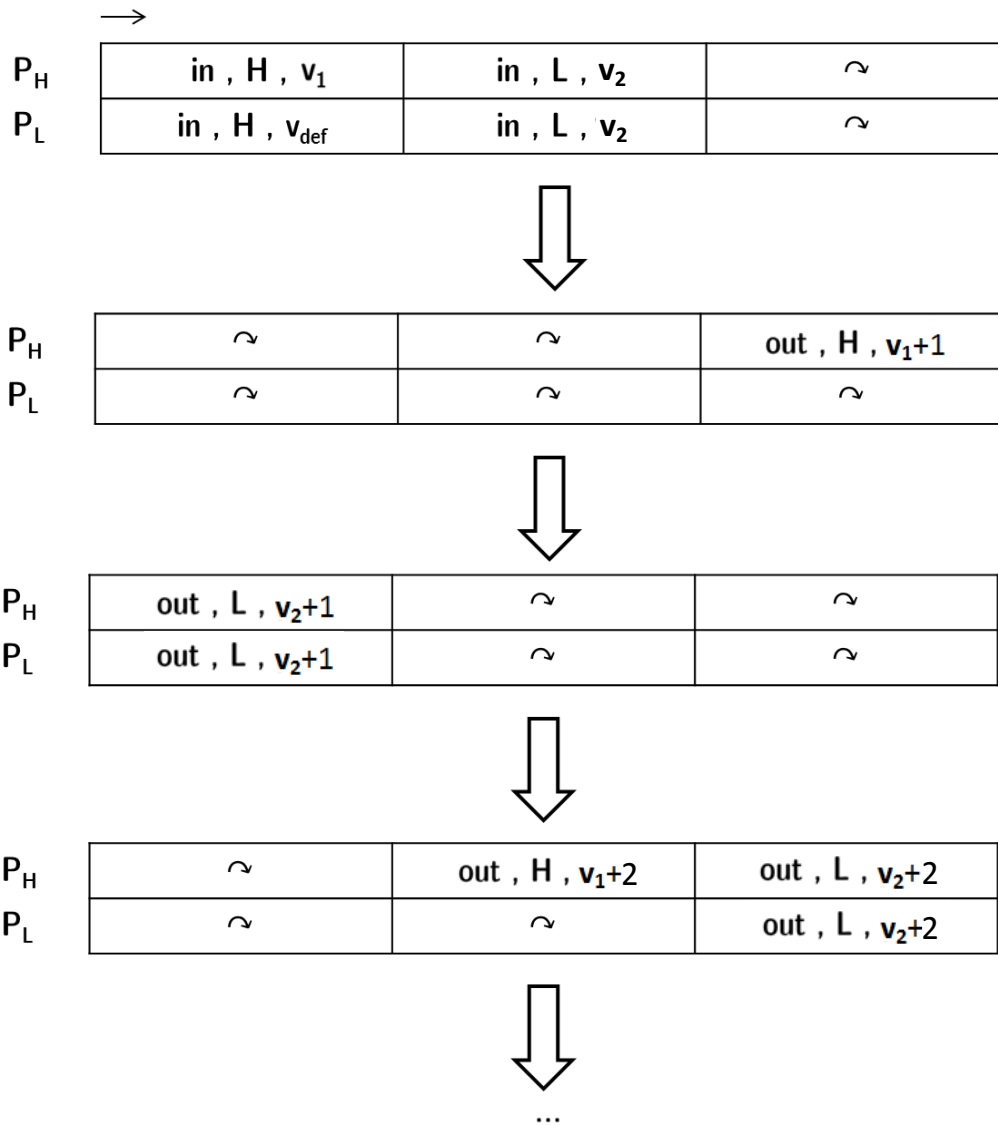
طبق شکل ۱۷، دقیقاً مطابق اجرای اصلی برنامه، ابتدا مقدار ۱ برای کانال خروجی سطح بالا ارسال می‌شود و سپس، مقدار صفر برای کانال خروجی سطح پایین. همچنین در ادامه نیز با توجه به واگرایی خاموش بودن هر دو اجرا، اجرای برنامه تحت مکانیزم نیز واگرا می‌ماند.

برنامه شکل ۱۸، نمونه‌ای از برنامه امن تحت عدم تداخل حساس به زمان است که واگرایی غیرخاموش دارد. پس در اجرای تحت مکانیزم نیز باید به تعداد نامتناهی خروجی در نظر گرفته شود. رفتار مکانیزم در شکل ۱۹ نشانگر برقراری شفافیت کامل برای این برنامه امن است.

```

inH x ;    // high input : x
inL y ;    // low input : y
while true do
    x++ ;
    y++ ;
    outH x ; // high output : x
    outL y ; // low output : y
    
```

شکل ۱۸ - برنامه امن طبق عدم تداخل حساس به زمان



شکل ۱۹ - نمایی از محتوای بافرها در مکانیزم پیشنهادی در هنگام اجرای برنامه شکل ۱۸

برنامه شکل ۲۰، نمونه دیگری از برنامه‌های امن است و مکانیزم پیشنهادی نیز شفافیت کامل را برای آن برآورده می‌کند.

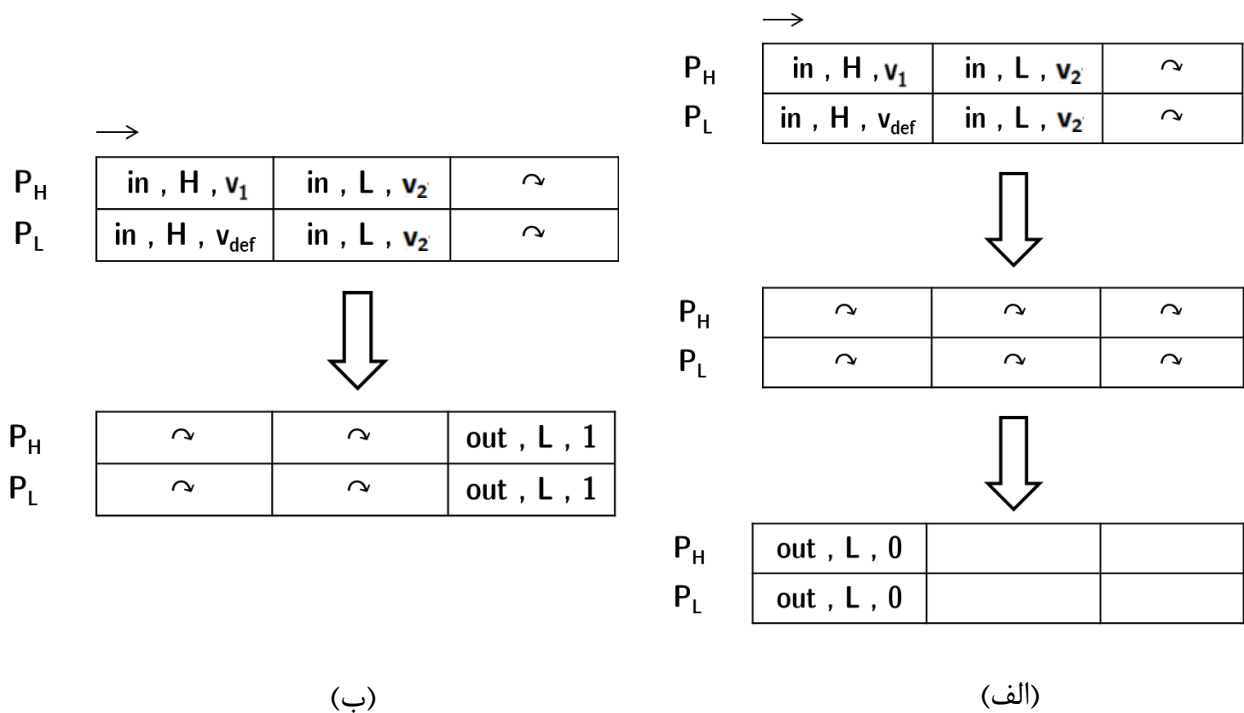
```

inH x ;    // high input : x
inL y ;    // low input : y
w = 0 ;
if y > 0 then
    if x > 0 then
        w = 0 ;
    else skip ;
else w = 1 ;
outL w ;   // low output : w

```

شکل ۲۰ - برنامه امن طبق عدم تداخل حساس به زمان

طبق شکل ۲۱ الف، در شرایطی که مقدار ورودی  $y$  بزرگتر از صفر باشد، مستقل از مقدار سطح بالای  $x$  رفتار برنامه یکسان خواهد بود. همین موضوع برای حالتی که  $y$  کوچکتر از صفر باشد نیز برقرار است که در شکل ۲۱ ب نشان داده شده است.



شکل ۲۱ - نمایی از محتوای بافرها در مکانیزم پیشنهادی در هنگام اجرای برنامه شکل ۲۰

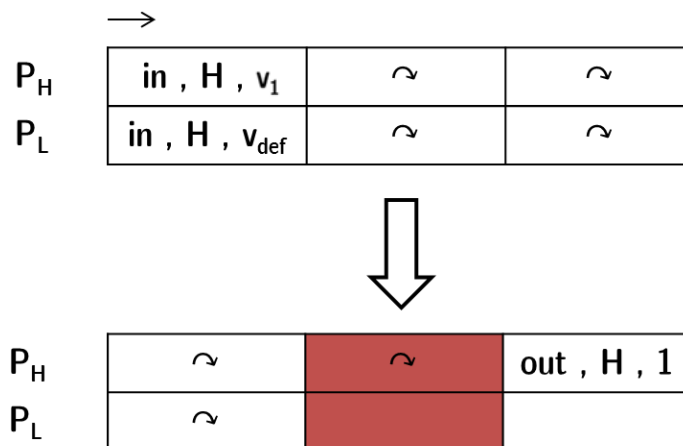
همانطور که مشاهده می‌شود، لحظه خاتمه هر دو اجرا نیز اهمیت دارد. زیرا زمان خاتمه اجرا نیز توسط مشاهده‌گر سطح پایین قابل مشاهده است. برنامه شکل ۲۲ مثالی است که بر همین موضوع تأکید دارد. اگرچه این برنامه هیچ دستور خروجی سطح پایینی ندارد، اما به واسطه تغییر در زمان خاتمه برنامه، ناامن محسوب می‌شود. شکل ۲۳ رفتار مکانیزم را با فرض مقدار واقعی ورودی سطح بالای ۱ نشان می‌دهد.

```

inH x ;    // high input : x
y = 0 ;
while y != x do
    y++ ;
outH x ;    // high output : x

```

شکل ۲۲ - برنامه ناامن طبق خط‌مشی عدم تداخل حساس به زمان



شکل ۲۳ - نمایی از محتوای بافرها در مکانیزم پیشنهادی در هنگام اجرای برنامه شکل ۲۲

اگرچه می‌توان مثال‌های دیگری از انواع برنامه‌ها آورد، اما به واسطه جعبه‌سیاه بودن این مکانیزم، فقط رویدادهای ورودی و خروجی و زمان وقوع آن‌ها برای اعمال امنیت کفایت می‌کند و جزئیات برنامه‌ها اهمیتی ندارد. همچنین، پیشتر مطرح شد که در صورتی که بخواهیم از گزارش نقض امنیت توسط مکانیزم صرف‌نظر کنیم، می‌توان فقط رویدادهای خروجی هر سطح در رونوشت سطح متناظر خودش نگهداری شود و مطابق مکانیزم عمل شود. بنابراین نیازی به ثبت ورودی‌ها یا گام‌های پیشروی اجرا نخواهد بود که باعث

سادگی مکانیزم و بافر می‌شود. بنابراین، عملاً نیازی به عملیات بررسی بافرها نخواهیم داشت و می‌توان مطابق رویه گفته‌شده، خروجی‌ها در هر گام به کانال‌های خروجی هم‌سطح ارسال شوند.

## ۳-۴ جمع‌بندی

در این فصل مکانیزم چنداجرایی امن بافردار به عنوان راه‌کاری برای حل شفافیت کامل معرفی و تشریح شد و با بیان مثال‌هایی، شهود درستی و شفافیت کامل مکانیزم برای اعمال خط‌مشی عدم تداخل حساس به زمان تبیین شد. فصل پنجم صوری‌سازی و اثبات ویژگی‌های این مکانیزم را دربردارد.

## فصل پنجم

### صوری سازی و اثبات

## صوری سازی و اثبات

فصل چهارم به بیان غیرصوری مکانیزم پیشنهادی اختصاص داشت. در این فصل، با صوری سازی سامانه و معناشناخت مکانیزم، به توضیح دقیق رفتار مکانیزم پیشنهادی پرداخته می شود و در ادامه، با تعریف معیارهای درستی و شفافیت کامل، اثبات می کنیم که مکانیزم چنداجرایی امن بافردار در اعمال خط مشی عدم تداخل حساس به زمان، درست و شفاف کامل است.

### ۵-۱ نحو و معناشناخت زبان مدل

برای مدل سازی مکانیزم پیشنهادی و اثبات ویژگی های مورد نظر از زبان برنامه نویسی مدل در [۶] استفاده می شود که دارای معناشناخت قطعی<sup>۱۲۰</sup> است. این زبان دارای دستورات انتساب، ساختار شرطی (if)، ساختار حلقه (while)، تأخیر یک گام اجرا (skip)، ورودی گرفتن از کانال (input) و خروجی دادن به کانال (output) است. مقادیر می توانند بولی یا عدد صحیح باشند. همچنین، فرض بر آن است که عبارات<sup>۱۲۱</sup> مورد استفاده اتمیک، قطعی و بدون اثرات جانبی هستند. یک برنامه به نام  $P$ ، دستوری است که قرار است توسط سامانه اجرا شود. شکل ۲۴، نحو دستورات این زبان مدل را نشان می دهد.

```

c ::= x := e
    | c ; c
    | if e then c else c
    | while e do c
    | skip
    | input x from i
    | output e to o

```

شکل ۲۴ – نحو دستورات موجود در زبان برنامه نویسی مدل

<sup>120</sup> Deterministic

<sup>121</sup> Expressions

برای بیان وضعیت سامانه،  $C_{in}$  مجموعه‌ای از کانال‌های ورودی و  $C_{out}$  مجموعه‌ای از کانال‌های خروجی در نظر گرفته می‌شود. منظور از ورودی برنامه  $I$  نیز نگاشتی از کانال‌های ورودی  $i \in C_{in}$  به صف‌های کانال ورودی  $q$  است. صف کانال ورودی نیز نگاشتی از اعداد صحیح نامنفی به مقادیر است. علاوه بر این، یک اشاره‌گر ورودی  $p$  را نگاشتی از کانال‌های ورودی  $i \in C_{in}$  به اعداد صحیح تعریف می‌کنیم و منظور از  $p_0$  اشاره‌گر ورودی اولیه سامانه است که هر کانال ورودی‌ای را به موقعیت صفر نگاشت می‌دهد. یک خروجی برنامه  $O$  به عنوان نگاشتی از کانال‌های خروجی  $o \in C_{out}$  به لیست‌های مقادیر تعریف می‌شود و نماد  $O_0$  برای نشان دادن خروجی اولیه برنامه در نظر گرفته شده است که هر کانال خروجی را به لیست خالی نگاشت می‌دهد.

به این ترتیب می‌توان عملیات خواندن ورودی و نوشتن خروجی را بر اساس ورودی برنامه  $I$  و خروجی برنامه  $O$  تعریف کرد:

$$\frac{I(i) = q \quad p(i) = n \quad q(n) = v}{read(I, i, p) = v}$$

$$\frac{O(o) = [v_1, \dots, v_n]}{write(O, o, v) = O[o \mapsto [v_1, \dots, v_n, v]]}$$

حال می‌توان معناساخت کوتاه‌گام<sup>۱۲۲</sup> این زبان مدل را، آن‌چنان که در شکل ۲۵ آمده است، تعریف کرد. پیکربندی اجرا<sup>۱۲۳</sup> به صورت  $\langle c, m, p, I, O \rangle$  مشخص می‌شود که در آن  $c$  یک دستور،  $m$  یک حافظه؛ یعنی نگاشتی از متغیرها به مقادیر،  $p$  یک اشاره‌گر کانال ورودی،  $I$  یک ورودی برنامه و  $O$  یک خروجی برنامه است. همچنین، از نماد  $m[x \mapsto v]$  برای تعریف یک حافظه جدید استفاده می‌شود که در آن متغیر  $x$  به مقدار  $v$  و بقیه متغیرها مانند  $y$ ، به مقدار  $m(y)$  نگاشت شده است. منظور از نماد  $m(e) = v$  نیز آن است که  $v$  برابر است با مقدار حاصل از ارزیابی عبارت  $e$ ، با توجه به مقادیر متغیرها در حافظه  $m$ . ضمناً از  $m_0$  برای نمایش حافظه اولیه سامانه استفاده می‌شود که هر متغیری را به مقدار صفر نگاشت می‌کند.

<sup>122</sup> Small-Step Semantics

<sup>123</sup> Execution Configuration



به منظور تمایز بین قواعد معناساخت استاندارد و معناساخت چنداجرایی امن بافردار، در قواعد معناساخت استاندارد از نماد  $\rightarrow$  و در قواعد معناساخت مکانیزم پیشنهادی از نماد  $\Rightarrow$  برای بیان گذارهای پیکربندی‌های اجرا استفاده می‌شود. می‌توان نماد  $\rightarrow^*$  را به عنوان بستار بازتابی و تریایی<sup>۱۲۴</sup> رابطه  $\rightarrow$  برای پیکربندی‌های اجرا تعریف کرد. نماد  $\rightarrow^n$  برای نمایش رابطه اجرای زمان‌دار در نظر گرفته می‌شود؛ به این صورت که اگر پیکربندی اجرای  $\langle c, m, p, I, O \rangle$  می‌تواند با  $n$  گام اجرا به  $\langle c', m', p', I, O' \rangle$  تبدیل شود، می‌توان چنین نوشت که  $\langle c, m, p, I, O \rangle \rightarrow^n \langle c', m', p', I, O' \rangle$ .

بنابراین، برنامه  $P$  می‌تواند با توجه به یک ورودی برنامه داده شده  $I$  با توجه به این معناساخت اجرا شود. اجرای این برنامه با پیکربندی اولیه  $\langle P, m_0, p_0, I, O_0 \rangle$  و به کار بستن قواعد معناساخت شکل ۲۵ انجام می‌گیرد.

برنامه  $P$  به ازای ورودی داده شده  $I$  زمانی خاتمه می‌یابد که وجود داشته باشد حافظه نهایی مانند  $m_f$ ، اشاره‌گر ورودی نهایی مانند  $p_f$  و خروجی برنامه نهایی مانند  $O_f$  به طوری که  $\langle \text{skip}, m_f, p_f, I, O_f \rangle \rightarrow^* \langle P, m_0, p_0, I, O_0 \rangle$ . در این حالت، می‌توان گفت که اجرای برنامه  $P$  به ازای ورودی برنامه  $I$  منجر به تولید اشاره‌گر نهایی  $p_f$  و خروجی برنامه  $O_f$  شده است، و می‌توان نوشت که  $(P, I) \rightarrow^* (p_f, O_f)$ . به طور مشابه، اگر  $\langle c', m', p', I, O' \rangle \rightarrow^n \langle P, m_0, p_0, I, O_0 \rangle$ ، آن‌گاه می‌توان  $(P, I) \rightarrow^n (p', O')$  را نوشت. باید دقت داشت که فرض می‌شود که حافظه نهایی  $m_f$  توسط مشاهده‌گر سطح پایین قابل مشاهده نیست. زیرا ممکن است برنامه طوری نوشته شده باشد که پس از آخرین دستور خروجی کانال سطح پایین، مقادیر محرمانه در متغیرهای سطح پایین نگهداری و محاسبه می‌شوند. طبق تعریف عدم تداخل مورد نظر، رویدادهای خروجی برنامه به عنوان موارد قابل مشاهده برای مشاهده‌گران سطح پایین محسوب می‌شوند.

<sup>124</sup> Reflexive and Transitive Closure

$$\frac{c = \mathbf{if} \ e \ \mathbf{then} \ c_{true} \ \mathbf{else} \ c_{false} \quad m(e) = b}{\langle c, m, p, I, O \rangle \rightarrow \langle c_b, m, p, I, O \rangle}$$

$$\frac{\langle c_1, m, p, I, O \rangle \rightarrow \langle c'_1, m', p', I, O' \rangle}{\langle c_1; c_2, m, p, I, O \rangle \rightarrow \langle c'_1; c_2, m', p', I, O' \rangle}$$

$$\overline{\langle \mathbf{skip}; c, m, p, I, O \rangle \rightarrow \langle c, m, p, I, O \rangle}$$

$$\frac{c = \mathbf{while} \ e \ \mathbf{do} \ c_{loop} \quad m(e) = true}{\langle c, m, p, I, O \rangle \rightarrow \langle c_{loop}; c, m, p, I, O \rangle}$$

$$\frac{c = \mathbf{while} \ e \ \mathbf{do} \ c_{loop} \quad m(e) = false}{\langle c, m, p, I, O \rangle \rightarrow \langle \mathbf{skip}, m, p, I, O \rangle}$$

$$\frac{m(e) = v \quad m' = m[x \mapsto v]}{\langle x := e, m, p, I, O \rangle \rightarrow \langle \mathbf{skip}, m', p, I, O \rangle}$$

$$\frac{c = \mathbf{output} \ e \ \mathbf{to} \ o \quad m(e) = v \quad O' = write(O, o, v)}{\langle c, m, p, I, O \rangle \rightarrow \langle \mathbf{skip}, m, p, I, O' \rangle}$$

$$\frac{c = \mathbf{input} \ x \ \mathbf{from} \ i \quad read(I, i, p) = v \quad p' = p[i \mapsto p(i) + 1] \quad m' = m[x \mapsto v]}{\langle c, m, p, I, O \rangle \rightarrow \langle \mathbf{skip}, m', p', I, O \rangle}$$

شکل ۲۵ - معناساخت کوتاه‌گام استاندارد زبان برنامه‌نویسی مدل [۶]

## ۲-۵ معناساخت مکانیزم چنداجرایی امن بافردار

پس از تعریف معناساخت اجرای استاندارد برنامه در زبان مدل، معناساخت رفتار مکانیزم پیشنهادی را معرفی می‌کنیم. زمان‌بندی اجرای رونوشت‌ها و همگام‌سازی آن‌ها در سطوح مختلف، از

چالش‌های اصلی در تعیین رفتار مکانیزم مبتنی بر چنداجرایی به شمار می‌رود. یکی از این چالش‌ها، همگام‌سازی رونوشت‌ها برای باز استفاده مقادیر ورودی خوانده شده توسط سطوح مختلف است. طبق توضیحات مطرح شده در فصل چهارم، فقط رونوشتی که سطح متناظر دستور ورودی را دارد قادر است از ورودی برنامه مقدار جدیدی را بخواند و بقیه رونوشت‌ها ملزم به انتظار برای باز استفاده از آن هستند. به همین دلیل، زمان‌بندی پیشنهادی و سایر بخش‌های مکانیزم با توجه به این موضوع و حفظ ترتیب رویدادها در بین کانال‌ها تعریف می‌شوند.

با توجه به این که هر رونوشت به طور مستقل از دیگری اجرا می‌شود، معناشناخت مکانیزم را به دو قسمت معناشناخت محلی<sup>۱۲۵</sup> و معناشناخت سراسری<sup>۱۲۶</sup> تقسیم‌بندی می‌کنیم. در معناشناخت محلی به مدل‌سازی گام‌های اجرای در یک رونوشت با سطح امنیتی مشخص پرداخته می‌شود و در معناشناخت سراسری، مدل‌سازی زمان‌بندی و همگام‌سازی اجراهای جداگانه محلی بیان می‌شود.

مشابه معناشناخت استاندارد، ابتدا نمادهای مورد استفاده معرفی می‌شوند. فرض می‌کنیم که یک شبکه سطوح امنیتی  $\mathcal{L}$  و توابع  $\sigma_{in}: \mathcal{C}_{in} \rightarrow \mathcal{L}$  و  $\sigma_{out}: \mathcal{C}_{out} \rightarrow \mathcal{L}$  وجود دارند. تابع  $\sigma_{in}$  نگاشتی از هر کانال ورودی  $i$  به یک سطح امنیتی، و تابع  $\sigma_{out}$  نگاشتی از هر کانال خروجی  $o$  به یک سطح امنیتی تعریف می‌شود. فرض بر این است که شبکه سطوح امنیتی  $\mathcal{L}$  متناهی است. می‌توان چنین در نظر گرفت که به ازای هر برنامه، فقط تمامی سطوح امنیتی مورد استفاده در آن برنامه در شبکه وجود دارند. پس این فرض، خللی در کلیت صوری سازی و اثبات ایجاد نخواهد کرد. همچنین برخلاف فرض موجود در [۶] نیازی به فرض درباره ترتیب کامل بودن شبکه نیست و شبکه در حالت کلی در نظر گرفته شده است.

<sup>125</sup> Local Semantics

<sup>126</sup> Global Semantics

## ۵-۲-۱ مکانیزم چنداجرایی امن بافردار بدون گزارش نقض امنیت

در ادامه، معاشناخت مکانیزم چنداجرایی امن بافردار برای حالت بدون گزارش نقض امنیت تعریف می شود. در پیوست، مکانیزم در حالتی که نقض امنیت گزارش شود نیز به شکل صوری بیان می شود.

### ۵-۲-۱-۱ معاشناخت محلی

مدل سازی نحوه اجرای هر رونوشت، مستقل از سایر رونوشت ها انجام می شود و هر یک از رونوشت ها در یک سطح امنیتی مشخص اجرا خواهند شد. در حالت کلی، این ریشه ها با یکدیگر تعاملی ندارند، به جز در شرایطی که ریشه ای بخواهد یکی از ورودی های سطح پایین تر از خود را بخواند و از آن در اجرای رونوشت خود استفاده کند. بنابراین، در صورتی که قبل از رسیدن اجرا به آن دستور خواندن ورودی، رونوشت سطح متناظر با دستور ورودی به آن دستور نرسیده باشد و مقدار ورودی را نخوانده باشد، ریشه فعلی باید منتظر بماند. در مکانیزم پیشنهادی، از آن جایی که رویدادهای ورودی نیز در بررسی امنیت برنامه ها تأثیرگذار هستند، در صورتی که برنامه ناامن بوده و رونوشت های مختلف همزمان به آن دستور نرسیده باشند، ممکن است اجراهای مختلف در زمان رسیدن به دستور خروجی از یکدیگر متفاوت شوند. از طرفی، برای رفع مشکل انتظار بی جا رونوشت های سطوح بالاتر، زمان بندی به نحوی صورت می گیرد که ابتدا رونوشت های با سطوح پایین تر، و سپس ریشه های سطوح بالاتر اجرا می شوند تا خللی در نحوه بررسی ترتیب رویدادهای خروجی ایجاد نشود.

همان طور که پیشتر گفته شد، رویدادهای خروجی ابتدا در بافر متناظر با رونوشت همان سطح امنیتی نگه داری می شوند. بنابراین، یک بافر برنامه  $B$  را به صورت آرایه ای از محل های خالی برای ثبت رویدادها در نظر می گیریم به طوری که  $\forall o \in C_{out}. |B(o)| = t$ ، که  $t$  همان مقدار سهم زمانی است. منظور از نماد  $B_0$  نمایش حالت اولیه بافر است که به ازای هر کانال خروجی، بافر در ابتدا کاملاً خالی است و می توان نوشت که  $B_0: \_ \mapsto [\emptyset_1, \dots, \emptyset_t]$ ؛ که در آن  $t$  نماد  $\emptyset$  به معنای خالی بودن بافر وجود دارد. در این حالت، تنها رویدادهای خروجی تولید شده توسط رونوشت همان سطح امنیتی می تواند در بافر قرار بگیرد. پس محتویات ممکن در بافر در هر عنصر از آن می تواند یک مقدار خروجی یا تهی باشد؛ که می توان به شکل  $d = v|\emptyset$  نشان داد.

با توجه به این که با استفاده از این مکانیزم دیگر خروجی تولید شده مستقیماً در خروجی ظاهر نمی شود، پس باید عملیات خواندن از ورودی و نوشتن در خروجی بازتعریف شود. شکل ۲۶ تعریف صوری این عملیات را نشان می دهد.

$$\begin{array}{l} I(i) = q \quad p(i) = n \quad q(n) = v \\ \hline read(I, i, p) = v \\ B(o) = [d_0, \dots, d_{t-1}] \quad 0 \leq index < t \\ \hline write(B, o, d, index) = B(o)[d_{index} \mapsto d] \end{array}$$

شکل ۲۶ - عملیات خواندن از ورودی و نوشتن خروجی در بافر

مشابه آن چه در معناشناخت استاندارد مطرح شد، به ازای هر سطح امنیتی، یک رونوشت اجرا می شود. به این ترتیب، یک پیکربندی/اجرای محلی برای هر سطح امنیتی و یک پیکربندی/اجرای سراسری برای توصیف وضعیت کل اجرای مکانیزم در نظر گرفته می شود. یک پیکربندی اجرای محلی به صورت  $\langle c, m, p, n \rangle_l$  تعریف می شود که در آن  $c$  یک دستور،  $m$  یک حافظه،  $p$  یک اشاره گر ورودی،  $n$  شمارنده تعداد گام های اجرا (با حالت اولیه  $n_0 = 0$ ) و  $l$  یک سطح امنیتی است. همچنین، پیکربندی اجرای سراسری به صورت  $\langle [lec_1, \dots, lec_j], r, I, O, B, s \rangle$  تعریف می شود که در آن  $[lec_1, \dots, lec_j]$  مجموعه ای از پیکربندی های اجرای محلی است و  $j = |\mathcal{L}|$ . دیگر عناصر تشکیل دهنده این پیکربندی عبارتند از یک اشاره گر ورودی سراسری  $r$ ، یک ورودی برنامه  $I$ ، یک خروجی برنامه  $O$ ، یک بافر برنامه  $B$  و یک شاخص<sup>۱۲۷</sup> زمان بند  $s$ .

حال می توان معناشناخت محلی را برای پیکربندی اجرای محلی، با توجه به اشاره گر ورودی سراسری  $r$ ، ورودی برنامه  $I$  و بافر برنامه  $B$ ، مطابق شکل ۲۷، تعریف کرد. معناشناخت محلی هر گام از اجرای یک رونوشت در سطح امنیتی  $l$  را مدل سازی می کند. دقت شود که منظور از  $n \bmod t$  باقی مانده مقدار  $n$  از تقسیم بر  $t$  است.

<sup>127</sup> Index

$$\frac{c = \text{if } e \text{ then } c_{true} \text{ else } c_{false} \quad m(e) = b \quad n' = n + 1}{\langle c, m, p, n \rangle_l, r, I, B \Rightarrow \langle c_b, m, p, n' \rangle_l, r, I, B}$$

$$\frac{\langle c_1, m, p, n \rangle_l, r, I, B \Rightarrow \langle c'_1, m', p', n' \rangle_l, r', I, B'}{\langle c_1; c_2, m, p, n \rangle_l, r, I, B \Rightarrow \langle c'_1; c_2, m', p', n' \rangle_l, r', I, B'}$$

$$\frac{n' = n + 1}{\langle \text{skip}; c, m, p, n \rangle_l, r, I, B \Rightarrow \langle c, m, p, n' \rangle_l, r, I, B}$$

$$\frac{c = \text{while } e \text{ do } c_{loop} \quad m(e) = true \quad n' = n + 1}{\langle c, m, p, n \rangle_l, r, I, B \Rightarrow \langle c_{loop}; c, m, p, n' \rangle_l, r, I, B}$$

$$\frac{c = \text{while } e \text{ do } c_{loop} \quad m(e) = false \quad n' = n + 1}{\langle c, m, p, n \rangle_l, r, I, B \Rightarrow \langle \text{skip}, m, p, n' \rangle_l, r, I, B}$$

$$\frac{m(e) = v \quad m' = m[x \mapsto v] \quad n' = n + 1}{\langle x := e, m, p, n \rangle_l, r, I, B \Rightarrow \langle \text{skip}, m', p, n' \rangle_l, r, I, B}$$

$$\frac{c = \text{output } e \text{ to } o \quad m(e) = v \quad \sigma_{out}(o) = l \quad B' = write(B, o, v, n \bmod t) \quad n' = n + 1}{\langle c, m, p, n \rangle_l, r, I, B \Rightarrow \langle \text{skip}, m, p, n' \rangle_l, r, I, B'}$$

$$\frac{c = \text{output } e \text{ to } o \quad \sigma_{out}(o) \neq l \quad n' = n + 1}{\langle c, m, p, n \rangle_l, r, I, B \Rightarrow \langle \text{skip}, m, p, n' \rangle_l, r, I, B}$$

$$\frac{c = \text{input } x \text{ from } i \quad \sigma_{in}(i) \not\leq l \quad m' = m[x \mapsto v_{default}] \quad n' = n + 1}{\langle c, m, p, n \rangle_l, r, I, B \Rightarrow \langle \text{skip}, m', p, n' \rangle_l, r, I, B}$$

$$\frac{c = \text{input } x \text{ from } i \quad \sigma_{in}(i) = l \quad v = read(I, i, p) \quad p' = p[i \mapsto p(i) + 1] \quad m' = m[x \mapsto v] \quad r' = r[i \mapsto p'(i)] \quad n' = n + 1}{\langle c, m, p, n \rangle_l, r, I, B \Rightarrow \langle \text{skip}, m', p', n' \rangle_l, r', I, B}$$

$$\frac{c = \text{input } x \text{ from } i \quad \sigma_{in}(i) < l \quad r(i) \leq p(i) \quad n' = n + 1}{\langle c, m, p, n \rangle_l, r, I, B \Rightarrow \langle c, m, p, n' \rangle_l, r, I, B}$$

$$\frac{c = \text{input } x \text{ from } i \quad \sigma_{in}(i) < l \quad r(i) > p(i) \quad v = read(I, i, p) \quad m' = m[x \mapsto v] \quad n' = n + 1}{\langle c, m, p, n \rangle_l, r, I, B \Rightarrow \langle \text{skip}, m', p, n' \rangle_l, r, I, B}$$

شکل ۲۷ - معناساخت محلی چنداجرای امن بافردار بدون گزارش نقض امنیت

## ۵-۲-۱-۲ معاشناخت سراسری

همان طور که پیشتر مطرح شد، زمان بندی نقشی تعیین کننده در حفظ ترتیب رویدادها و رفتار مکانیزم دارد. در مکانیزم پیشنهادی، نوعی از زمان بند تسهیم [۱۷] استفاده می شود که در آن به هر رونوشت در هر نوبت اجرا، به مقدار سهم زمانی مشخصی ( $t$ ) فرصت اجرا داده می شود که با طول بافر هم خوانی دارد. از آن جا که رونوشت های سطح بالا برای اجرا به مقادیر ورودی سطح پایین نیاز دارند، ابتدا سطوح پایین تر اجرا می شوند و در ادامه نوبت به اجرای رونوشت های سطح بالا می رسد. همچنین، با توجه به این که در صورتی که برنامه در حال اجرا امن باشد، باید زمان رسیدن به دستور ورودی برای همه رونوشت ها یکسان باشد. از طرفی، با توجه به این که به هر رونوشت و مرحله بررسی و خروجی دادن یک سهم زمانی در هر نوبت داده می شود، و انتخاب ریشه ها از بین ریشه های آماده، مسدود شده و یا خاتمه یافته است، پس هر رونوشت در هر نوبت هم زمان محدود و ثابتی برای اجرا دارد و هم پس از مدت محدود و معینی انتظار، مجدداً نوبت اجرا را در دست می گیرد. به این ترتیب، علاوه بر برآورده شدن درستی، عدم قحطی زدگی نیز تضمین خواهد شد. بنابراین، استفاده از این نوع زمان بندی، برای به دست آوردن درستی و شفافیت کامل مناسب خواهد بود.

```

scheduler ( $\mathcal{L}$ ) =  $S$  (input: security lattice, output: the priority array of threads)
function scheduler ( $\mathcal{L}$ ) {
    i=0;
    while ( $|\mathcal{L}| > 0$ ) do
         $x = \text{meet}(\mathcal{L});$  // return the minimum element(s) of the lattice
        for (element  $e$  in  $x$ ) do
             $S[i] = e;$ 
             $i++;$ 
         $\mathcal{L} = \mathcal{L} \setminus x;$ 
    return  $S;$ 
}

```

شکل ۲۸ - تابع تعیین اولویت زمان بندی سطوح امنیتی

با توجه به فرض متناهی بودن شبکه امنیتی، می توان قبل از شروع اجرای رونوشت ها، تابع شکل ۲۸ را فراخوانی کرد تا ترتیب اولویت ریشه های سطوح مختلف امنیتی مشخص شود. نحوه عملکرد زمان بند به این صورت است که سطوح قابل مقایسه با همان ترتیب حفظ می شوند و برای سطوح غیر قابل مقایسه، ترتیبی دلخواه در نظر گرفته می شود و پس از آن، همواره از آن ترتیب برای توزیع نوبت اجرا استفاده خواهد شد. خروجی تابع، آرایه زمان بندی خواهد بود.

در ادامه و با استفاده از زمان بند معرفی شده، معناشناخت سراسری مکانیزم را در حالتی که گزارش نقض امنیت ندارد، شرح داده می شود. شکل ۳۰ معناشناخت سراسری مکانیزم را به بیان صوری نشان می دهد. در هر گام، یکی از رونوشت ها مطابق زمان بند انتخاب و اجرا می شود و نتیجه در پیکربندی اجرای سراسری نگه داری می شود. پس از اجرای رونوشت دارای بالاترین سطح، نوبت به قسمت بررسی و خروجی دادن می رسد و این قسمت نیز به اندازه یک سهم زمانی، رویدادهای خروجی های تولید شده را به کانال های خروجی منتقل می کند. همچنین، در صورتی که همه رونوشت ها در حالت خاتمه یافته باشند، اجرا خاتمه خواهد یافت.

پیکربندی اجرای سراسری به صورت  $\langle [lec_1, \dots, lec_j], r, I, O, B, s \rangle$ ، که در آن  $j$  تعداد سطوح امنیتی است، معرفی شد. حالت اولیه متغیرهای پیکربندی عبارتند از

$$L_0 = L_{P,0} = \forall lec_s \in L. lec_s = \langle P, m_0, p_0, n_0 \rangle_I, \quad r_0 = \_ \mapsto [], \quad O_0 = \_ \mapsto [], \\ B_0: \_ \mapsto [\emptyset_1, \dots, \emptyset_t], \quad s_0 = 1$$

به این معنا که همه پیکربندی های اجرای محلی رونوشت ها در حالت اولیه خود قرار می گیرند، اشاره گر سراسری ورودی و خروجی های برنامه لیستی خالی هستند و بافر دارای  $t$  حافظه خالی است. همچنین، زمان بند روی اجرای رونوشت دارای پایین ترین سطح امنیتی تنظیم است. به این ترتیب، اجرای برنامه توسط مکانیزم چند اجرایی امن بافردار زمانی خاتمه می یابد که  $\langle L_0, r_0, I, O_0, B_0, s_0 \rangle \Rightarrow^n \langle L_f, r_f, I, O_f, B_f, s_f \rangle$  در حالی که برای مقادیری از  $r_f$  و  $O_f$ ، لیست ریشه ها خالی باشد و نهایتاً بافر و شاخص زمان بندی مجدداً در حالت اولیه قرار گرفته باشند. به بیان دیگر،  $L_f = []$ ،  $B_f = B_0$  و  $s_f = s_0$  لازم به یادآوری است که شاخص زمان بندی می تواند عددی بین یک تا  $j$  باشد و بیانگر ریشه بعدی برای اجرا است.



تنها قسمت باقی مانده از مکانیزم، مرحله بررسی و خروجی دادن به کانال های خروجی است. نحوه عملکرد این قسمت به این شرح است که در هر نوبت، پس از اجرای آخرین رونوشت، با شروع از اولین عنصر بافرهای همه سطوح، رویدادهای خروجی تولیدشده در هر سطح را به خروجی متناظر ارسال می کند. در حالت عدم گزارش نقض امنیت، نیاز به بررسی رویدادها وجود ندارد. زیرا در صورتی که برنامه امن باشد، حتماً ترتیبها رعایت خواهد شد و در هر گام، تنها یکی از بافرها در شماره عنصر مورد بررسی، رویداد خروجی خواهد داشت. زیرا اگر چنین نباشد، یعنی برنامه اصلی در یک دستور دو خروجی را تولید کرده است، که چنین چیزی امکان پذیر نیست. اما در شرایطی که برنامه ناامن باشد، ممکن است بیش از یک خروجی در یک گام از بررسی بافر به خروجی منتقل شود. زیرا دو رونوشت به انشعاب های مختلفی از برنامه رفته اند و رفتار متفاوتی از یکدیگر خواهند داشت. در هر دو حالت، نیازی به مقایسه رویدادهای تولیدشده در یک عنصر از یک بافر با سایر بافرها در همان شماره عنصر وجود ندارد. دقت شود که در حالتی که مایل باشیم تا نقض امنیت توسط مکانیزم گزارش شود، علاوه بر ثبت رویدادهای ورودی، خروجی و پیشروی برنامه، باید در هر گام، رویداد موجود در یک عنصر از یک بافر با رویدادهای تولیدشده توسط رونوشت های سطوحی که پایین تر نیستند، سازگار باشد.

در شکل ۲۹، معاشناخت قسمت بررسی و خروجی دادن مکانیزم مشاهده می شود. در هر گام در این قسمت، وضعیت بافر، خروجی برنامه و شماره عنصری از بافر که در حال بررسی است، نگهداری و به روز می شود؛ یعنی گذاری در قالب  $B, O, k \Rightarrow B', O', k'$  برای هر گام از بررسی و خروجی دادن قابل تعریف است. واضح است که مقدار اولیه  $k_0 = 0$  در نظر گرفته می شود. علاوه بر این، با توجه به این که ممکن است حالتی وجود داشته باشد که بیش از یک رویداد خروجی در یک گام به کانال های خروجی منتقل شود، حافظه موقتی به نام *Temp* در نظر گرفته می شود و مقدار و سطح خروجی ها در آن نگهداری می شود و سپس این حافظه موقت به کانال های خروجی اصلی فرستاده می شوند. طبیعتاً حداکثر طول این حافظه بیشتر از  $j$  نخواهد بود. دو تابع  $Temp.v$  و  $Temp.o$  نیز کانال های خروجی و مقادیر تولیدشده خروجی را مشخص می کنند.

با توجه به این که ممکن است در یک گام بیش از یک رویداد خروجی به کانال های خروجی ارسال شود، لازم است تا تغییری در تابع قراردادن خروجی ها در کانال های مرتبط لحاظ شود. تابع *writeOut* در شکل ۲۹ به همین منظور تعریف شده است.

The definition of *writeOut* function:

$$\frac{O(o_1) = [v_{11}, \dots, v_{1n}] \quad O(o_2) = [v_{21}, \dots, v_{2n'}] \quad \dots \quad O(o_s) = [v_{s1}, \dots, v_{sn''}]}{\text{writeOut}(O, (o_1, o_2, \dots, o_s), (v_1, v_2, \dots, v_s)) = O[o_1 \mapsto [v_{11}, \dots, v_{1n}, v_1], o_2 \mapsto [v_{21}, \dots, v_{2n'}, v_2], \dots, o_s \mapsto [v_{s1}, \dots, v_{sn''}, v_s]]}$$

$$\text{Temp}.o = (o_1', \dots, o_{|\text{Temp}|}); \quad 0 \leq |\text{Temp}| \leq j$$

$$\text{Temp}.v = (v_1', \dots, v_{|\text{Temp}|})$$

The following rules apply for exact  $t$  times, each for one index of the buffer  $B$ . So it takes  $t$  times.

$$\frac{k' = k + 1 \quad k \leq t - 1 \quad \forall o_s. (B(o_s)[k] = v \leftrightarrow (o_s, B(o_s)[k]) \in \text{Temp})}{O' = \text{writeOut}(O, \text{Temp}.o, \text{Temp}.v) \quad B' = B[\forall o_s. B(o_s)[k] \mapsto \emptyset]} \quad \frac{}{B, O, k \models B', O', k'}$$

$$\frac{k' = k + 1 \quad k \leq t - 1 \quad \forall o. B(o)[k] = \emptyset}{B, O, k \models B, O, k'}$$

شکل ۲۹ - معناساخت قسمت بررسی و خروجی دادن در حالت عدم گزارش نقض امنیت

$$\frac{s \neq j \quad \text{lec} = \text{lec}_s \quad s' = s + 1 \quad \text{lec}, r, I, B \models^t \text{lec}', r', I, B' \quad L' = L[\text{lec} \mapsto \text{lec}']}{\langle L, r, I, O, B, s \rangle \models^t \langle L', r', I, O, B', s' \rangle}$$

$$\frac{s = j \quad \text{lec} = \text{lec}_s \quad s' = 1 \quad \text{lec}, r, I, B \models^t \text{lec}', r', I, B' \quad L' = L[\text{lec} \mapsto \text{lec}'] \quad B', O, 0 \models^t B_0, O', t}{\langle L, r, I, O, B, s \rangle \models^{2t} \langle L', r', I, O', B_0, s' \rangle}$$

$$\frac{s = 1 \quad \forall \text{lec}_{\text{index}} \in L. \text{lec}_{\text{index}} = \langle \text{skip}, m, p, n \rangle_l}{L = []}$$

شکل ۳۰ - معناساخت سراسری برای چنداجزایی امن بافردار در حالت عدم گزارش نقض امنیت

معناساخت مکانیزم چنداجزایی امن بافردار به همراه گزارش نقض امنیت در پیوست آمده است.

### ۵-۳ اثبات درستی و شفافیت کامل مکانیزم

در این بخش به بیان قضایای مربوط به درستی و شفافیت کامل مکانیزم پیشنهادی پرداخته می شود و اثبات می شود که مکانیزم چنداجرایی امن بافردار قادر است خطمشی عدم تداخل حساس به زمان را درست و شفاف اعمال کند. اثبات های مطرح شده برای حالت بدون گزارش نقض امنیت است. به طور مشابه می توان قضیه های عنوان شده را برای مکانیزم با گزارش نقض امنیت اثبات کرد. در روند بیان قضایا و اثبات ها از صورت بندی مکانیزم چنداجرایی امن [۶] استفاده شده است.

لم ۱ (نامتغیرهای حالت اجرای سراسری) - فرض کنید

$$\langle L_0, r_0, I, O_0, B_0, s_0 \rangle \Rightarrow^n \langle L_f, r_f, I, O_f, B_f, s_f \rangle$$

پس خواهیم داشت،

- برای هر  $\langle c, m, p, n \rangle_l \in L$ ، به ازای تمام  $i \in C_{in}$  که  $\sigma_{in}(i) = l$ ،  $r(i) = p(i)$  است.

- برای هر سطح امنیتی  $l$ ، فقط یک اجرا  $\langle c, m, p, n \rangle_l$  در سطح امنیتی  $l$  در  $L$  وجود دارد.

**اثبات** - به سادگی قابل می توان دید که گزاره های بالا به ازای پیکربندی اولیه سراسری برقرار است و همچنین، طبق قواعد معناشناخت سراسری نیز برقرار می ماند. ■

در ادامه باید نشان دهیم که مکانیزم پیشنهادی عدم تداخل حساس به زمان را برای هر برنامه ای مانند  $P$  برآورده می سازد و برای برنامه های امن نیز شفافیت کامل را تضمین می کند. به همین دلیل، باید ابتدا خطمشی در این مدل سازی تعریف شود.

برای یک سطح امنیتی داده شده  $l \in \mathcal{L}$ ، دو ورودی برنامه  $I$  و  $I'$  را معادل از دید سطح  $l$  تعریف می کنیم (و می نویسیم  $I =_l I'$ ) اگر و فقط اگر برای هر  $i \in C_{in}$  که  $\sigma_{in}(i) \leq l$  با  $I(i)$  برابر  $I'(i)$  باشد. به طور مشابه، دو خروجی برنامه  $O$  و  $O'$  را از دید سطح  $l$  معادل می نامیم (و می نویسیم  $O =_l O'$ ) اگر و فقط اگر به ازای هر  $o \in \sigma_{out}$  که  $\sigma_{out}(o) \leq l$  با  $O(o)$  برابر  $O'(o)$  برابر باشد. به طور مشابه می توان برای بافرهای  $B$  و  $B'$  نیز همین تعریف را داشت. همچنین، دو اشاره گر ورودی  $p$  و  $p'$  از دید مشاهده گر سطح  $l$  یکسان خواهد بود (و می نویسیم  $p =_l p'$ ) اگر و فقط اگر برای هر  $i \in C_{in}$  که  $\sigma_{in}(i) \leq l$ ،  $p(i) = p'(i)$  باشد. به طور مشابه برای دو اشاره گر ورودی

سراسری  $r$  و  $r'$  نیز قابل تعریف است. در نهایت، دو شاخص زمان بندی  $S$  و  $S'$  را از دید سطح  $l$  معادل می دانیم (و می نویسیم  $S =_l S'$ ) اگر و فقط اگر  $S = S'$  باشد.

خط مشی عدم تداخل حساس به زمان به هر دو کانال نهان خاتمه و زمانی توجه دارد. تعریف این خط مشی را بر اساس رابطه ترایی، انتزاعی و زمان دار اجرای  $\hookrightarrow^n$  تعریف می کنیم. هر دوی روابط اجرای استاندارد  $\rightarrow^n$  و اجرای مکانیزم چنداجرایی امن بافردار  $\Rightarrow^n$  را می توان با  $\hookrightarrow^n$  جایگزین کرد.

**تعریف ۱** (تعریف خط مشی عدم تداخل حساس به زمان) [۶] - برنامه  $P$  عدم تداخل حساس به زمان را طبق رابطه معناشناخت داده شده  $\hookrightarrow^*$  برآورده می کند اگر برای هر سطح امنیتی  $l \in \mathcal{L}$ ، برای هر  $n \geq 0$ ، برای هر ورودی برنامه  $I$  و  $I'$  که  $I =_l I'$  برای آن ها برقرار است، اگر  $(p, O) \hookrightarrow^n (P, I)$ ، آنگاه  $(p', O') \hookrightarrow^n (P, I')$  و  $p =_l p'$  و  $O =_l O'$ .

**قضیه ۱** (درستی مکانیزم چنداجرایی امن بافردار) - هر برنامه  $P$  تحت مکانیزم چنداجرایی امن بافردار عدم تداخل حساس به زمان را برآورده می کند.

**لم ۲** (صیانت<sup>۱۲۸</sup> درستی برای معناشناخت محلی، بخش اول) - فرض کنید  $l_s$  یک سطح امنیتی و  $l \leq l_s$  باشد.  $\langle c, m, p, n \rangle_{l, r_1, I_1, B_1} \Rightarrow \langle c', m', p', n' \rangle_{l, r'_1, I'_1, B'_1}$  را در نظر بگیرید و همچنین، فرض کنید  $r_1 =_{l_s} r_2$  و  $I_2 =_{l_s} I_1$  و  $B_2 =_{l_s} B_1$  برقرار باشد. آنگاه

$$\langle c, m, p, n \rangle_{l, r_2, I_2, B_2} \Rightarrow \langle c', m', p', n' \rangle_{l, r'_2, I'_2, B'_2}$$

که در آن  $r'_1 =_{l_s} r'_2$  و  $B'_1 =_{l_s} B'_2$  است.

**اثبات** - گام های اجرا باید مطابق با یکی از قواعد موجود در معناشناخت محلی باشد. با بررسی هر یک از این قواعد می توان دریافت که گزاره مطرح شده برقرار خواهد بود. ■

**لم ۳** (صیانت درستی برای معناشناخت محلی، بخش دوم) - فرض کنید  $l_s$  یک سطح امنیتی و  $l \not\leq l_s$  باشد.  $\langle c, m, p, n \rangle_{l, r, I, B} \Rightarrow \langle c', m', p', n' \rangle_{l, r', I', B'}$  را در نظر بگیرید. بنابراین باید  $r' =_{l_s} r$  و  $B' =_{l_s} B$  باشد.

<sup>128</sup> Preservation

**اثبات** - گام‌های اجرا باید مطابق با یکی از قواعد موجود در معناشناخت محلی باشد. با بررسی هر یک از این قواعد می‌توان دریافت که گزاره مطرح شده برقرار خواهد بود. ■

لم ۴ (صیانت درستی برای معناشناخت سراسری) - فرض کنید  $l_s$  یک سطح امنیتی باشد. در نظر بگیرید که

$$\langle L_1, r_1, I_1, O_1, B_1, s_1 \rangle \Rightarrow \langle L'_1, r'_1, I_1, O'_1, B'_1, s'_1 \rangle$$

9

$$\langle L_2, r_2, I_2, O_2, B_2, s_2 \rangle \Rightarrow \langle L'_2, r'_2, I_2, O'_2, B'_2, s'_2 \rangle$$

که  $L_1 =_{l_s} L_2$ ,  $r_1 =_{l_s} r_2$ ,  $I_1 =_{l_s} I_2$ ,  $O_1 =_{l_s} O_2$ ,  $B_1 =_{l_s} B_2$  و  $s_1 =_{l_s} s_2$  باشد. آنگاه به خاطر استفاده از زمان‌بند پیشنهاد شده خواهیم داشت  $L'_1 =_{l_s} L'_2$ ,  $r'_1 =_{l_s} r'_2$ ,  $O'_1 =_{l_s} O'_2$ ,  $B'_1 =_{l_s} B'_2$  و  $s'_1 =_{l_s} s'_2$ .

**اثبات** - تعریف می‌کنیم  $lec_1 = \langle P_1, m_1, p_1, n_1 \rangle_{l_1} = L_1[s_1]$  و می‌نویسیم  $lec_2 = \langle P_2, m_2, p_2, n_2 \rangle_{l_2} = L_2[s_2]$  ابتدا حالتی را در نظر می‌گیریم که  $l_1 \leq l_s$  باشد. از آنجایی که  $L_1 =_{l_s} L_2$  و  $s_1 =_{l_s} s_2$  داریم  $lec_1 = lec_2$ . پس با بررسی قواعد معناشناخت سراسری و در نظر گرفتن لم ۲ می‌توان به سادگی نتیجه گرفت.

در حالتی که  $l_1 \not\leq l_s$  باشد، از لم ۳ و قواعد مربوط به قسمت بررسی و خروجی‌دادن مکانیزم داریم که  $B'_1 =_{l_s} B_1 =_{l_s} B_2 =_{l_s} B'_2$  و  $O'_1 =_{l_s} O_1 =_{l_s} O_2 =_{l_s} O'_2$ ,  $r'_1 =_{l_s} r_1 =_{l_s} r_2 =_{l_s} r'_2$  علاوه بر این، با بررسی قواعد معناشناخت سراسری می‌توان دید که  $L'_1 =_{l_s} L'_2$  و  $s'_1 =_{l_s} s'_2$  خواهد بود. ■

با بیان لم‌های بالا، اکنون آماده اثبات قضیه ۱ هستیم.

**اثبات قضیه ۱** (درستی مکانیزم) - یک برنامه  $P$ ، یک سطح امنیتی  $l \in \mathcal{L}$  و دو وضعیت کانال ورودی  $I$  و  $I'$  به طوری که  $I =_l I'$  باشد را در نظر بگیرید. فرض کنید  $(P, I) \Rightarrow^n (r, O)$  یا به عبارتی،  $\langle L, r, I, O, B, s \rangle \Rightarrow^n \langle L_{P,0}, r_0, I, O_0, B_0, s_0 \rangle$  و  $(P, I') \Rightarrow^n (r', O')$ ، یا به عبارت دیگر،  $\langle L, r', I', O', B', s' \rangle \Rightarrow^n \langle L_{P,0}, r_0, I', O_0, B_0, s_0 \rangle$  برقرار باشد. با استقرار روی تعداد گام‌های اجرا  $(n)$  و استفاده از لم ۴ می‌توان نشان داد که  $L =_l L'$ ,  $r =_l r'$ ,  $O =_l O'$ ,  $B =_l B'$  و  $s =_l s'$  خواهد بود. ■

همان گونه که در فصل های قبل عنوان شد، مکانیزمی شفاف نامیده می شود که رفتار برنامه ای که مطابق با خط مشی تعیین شده است، دست نخورده باقی بماند. با توجه به تعریف عدم تداخل، خروجی اجرای برنامه امن تحت یک مکانیزم شفاف باید مشابه اجرای اصلی آن برنامه باشد. برای عدم تداخل حساس به زمان، علاوه بر یکسان بودن مقادیر خروجی، ترتیب رویدادهای خروجی نیز اهمیت دارد. منظور از شفافیت کامل برای عدم تداخل حساس به زمان این است که ترتیب رویدادهای خروجی اجرای تحت مکانیزم دقیقاً با ترتیب رویدادهای خروجی اجرای اصلی برنامه امن باشد. حفظ ترتیب نباید صرفاً با در نظر گرفتن یک کانال خروجی خاص باشد و باید در بین همه کانال ها، ترتیب مشابه بماند.

**قضیه ۲** (شفافیت کامل مکانیزم چنداجرایی امن بافردار) - اگر برنامه  $P$  عدم تداخل حساس به زمان را برآورده می کند، آنگاه برای هر ورودی برنامه  $I$ ، برای هر  $n \geq 0$  وجود دارد  $g'$  و  $g$  به طوری که

$$(P, I) \rightarrow^n (r_1, O_1) \Rightarrow (P, I) \Rightarrow^g (r_1, O_1)$$

9

$$(P, I) \rightarrow^{n+1} (r_2, O_2) \Rightarrow (P, I) \Rightarrow^{g'} (r_2, O_2)$$

که  $g' > g \geq n$ .

ابتدا لازم است تا ورودی های قابل مشاهده از ورودی برنامه  $I$  برای یک سطح امنیتی  $l \in \mathcal{L}$  را تعریف کنیم.

$$I_l(i) = \begin{cases} I(i) & \text{if } \sigma_{in}(i) \leq l \\ \_ \mapsto v_{default} & \text{otherwise} \end{cases}$$

طبق تعریف، می توان دریافت که  $I_l(i) = I$  است.

**لم ۵** (تناظر بین اجرای استاندارد و چنداجرایی امن بافردار) - فرض کنید  $l \in \mathcal{L}$  و  $P$  یک برنامه باشد. در نظر بگیرید که  $\langle L, r, I, O, B, s \rangle \Rightarrow^g \langle L_{P,0}, r_0, I, O_0, B_0, s_0 \rangle$  که در آن  $\langle c, m, p, n \rangle_l \in L$  تعریف می کنیم که  $I_l = I_l(i)$  است. آنگاه  $\langle c, m, p, I_l, O' \rangle \rightarrow^n \langle P, m_0, p_0, I_l, O_0 \rangle$  خواهد بود که برای هر کانال خروجی  $o$  که  $\sigma_{out}(o) = l$  باشد،  $O'(o) = O(o)$  برقرار است. علاوه بر این، فرض کنید  $j$  تعداد سطوح امنیتی موجود در شبکه و  $t$  مقدار سهم زمانی باشد. می دانیم که اندازه بافر هر

سطح نیز برابر با  $t$  عنصر است. پس رابطه بین تعداد گام‌های محلی و سراسری اجرا تحت مکانیزم چنداجرایی امن بافردار و تعداد گام‌های اجرای استاندارد یک برنامه عبارت است از

$$g = (n/t). (j + 1). t + j. t + n \bmod t$$

**اثبات -** لم را با استقرار روی تعداد گام‌های اجرای سراسری که برای اشتقاق گزاره  $\langle L, r, I, O, B, S \rangle \Rightarrow^g \langle L_{P,0}, r_0, I, O_0, B_0, S_0 \rangle$  اثبات می‌کنیم. با استقرا روی قواعد موجود در معناساخت سراسری می‌توان نشان داد که لم بالا برقرار است. اگر  $l$  یکی از سطوح امنیتی موجود در شبکه به غیر از بالاترین سطح باشد، طبق قاعده اول، در صورتی که زمان‌بند نوبت را به آن داده باشد، به مدت  $t$  و براساس معناساخت محلی رونوشت سطح  $l$  را اجرا می‌کند. پس تنها در زمانی که نوبت به آن داده شود اجرا را پیگیری می‌کند و در غیر این صورت، به مدت زمان مشخص و ثابتی منتظر می‌ماند.

از طرفی می‌توان مشاهده کرد که قواعد موجود در معناساخت محلی مشابه معناساخت اجرای استاندارد است، با این تفاوت که در صورت مشاهده دستور output مربوط به سطح امنیتی  $l$ ، مقدار به بافر منتقل می‌شود. قاعده دوم معناساخت سراسری زمانی مورد استفاده قرار می‌گیرد که نوبت اجرا به بالاترین سطح امنیتی موجود در شبکه رسیده باشد. در این حالت، به اندازه  $t$  گام به اجرای رونوشت بالاترین سطح تخصیص داده می‌شود. سپس بررسی بافرها و انتقال مقادیر آن‌ها به کانال‌های خروجی خواهد بود. با توجه به این که هر بافر دارای دقیقاً  $t$  جای حافظه است، پس بررسی هر عنصر از بافر، مقایسه آن با عنصر متناظر در بافرهای سطوح دیگر و انتقال خروجی به کانال مربوط نیز  $t$  گام طول می‌کشد. به این ترتیب، در این مرحله از اجرا،  $2t$  زمان مصرف می‌شود. قاعده سوم نیز برای خاتمه در نظر گرفته شده است. پس همانطور که مشاهده شد، از نظر رفتار برنامه هیچ‌گونه تفاوتی ایجاد نخواهد شد و تنها زمان قرارگیری رویداد خروجی در کانال‌های خروجی نسبت به اجرای استاندارد متفاوت شده است که البته این اختلاف، مقدار مشخص و ثابتی است. طبق توضیحات مطرح‌شده، اگر رویداد خروجی‌ای در گام  $m$  در اجرای استاندارد تولید شود و به کانال خروجی فرستاده شود، در اجرای مبتنی بر مکانیزم معرفی‌شده، به اندازه تعداد دورهای قبلی تسهیم زمانی بین رونوشت‌ها باید صبر کرد تا نوبت اجرا به سهم زمانی مورد نظر برسد که در آن، آن رویداد خروجی تولید شود. در این قسمت نیز هر رونوشت باید در نوبت اجرای خود بافر را پر کند. پس از آن و در مرحله بررسی و خروجی‌دادن، در کانال خروجی مقدار فرستاده خواهد شد. به این ترتیب، برای رسیدن به دور فعلی اجرای رونوشت‌ها،

$(j+1).t$  زمان طی شده است. پس از آن، همه  $n$  رونوشت اجرای خود را انجام داده و محتوای خروجی تولیدشده را به بافر منتقل می کنند که این مدت نیز معادل است با  $j.t$ . سپس در مرحله بررسی و خروجی دادن بافر در دور فعلی، به اندازه تعداد گام های طی شده در آن سهم زمانی، خروجی تولیدشده به کانال های خروجی منتقل می شوند که این کار نیز  $n \bmod t$  گام زمان لازم دارد. پس در کل، مجموع همه این زمان ها برای انتقال رویداد تولیدشده به کانال خروجی مربوط مصرف شده است. به سادگی می توان مشاهده کرد که هر گام از اجرای استاندارد برنامه امن، به  $(n/t).(j+1).t + j.t + n \bmod t$  گام در اجرا توسط مکانیزم نیازمند است. ■

**اثبات قضیه ۲ (شفافیت کامل مکانیزم) – می دانیم که**

$$\langle P, m_0, p_0, I, O_0 \rangle \rightarrow^* \langle \text{skip}, m_f, p, I, O \rangle$$

باید حالات مختلف برای اثبات قضیه بررسی شوند:

(۱) خاتمه: اگر فرض شود برای برنامه امن  $P$ ،  $(P, I) \rightarrow^* (p, O)$  برقرار است، باید نشان داده شود که اجرای همان برنامه تحت مکانیزم نیز خاتمه خواهد یافت.  $\langle L_{P,0}, r_0, I, O_0, B_0, S_0 \rangle$  را به عنوان پیکربندی اولیه اجرای سراسری برای برنامه  $P$  بگیرید. باید نشان داد که اجرای  $\langle L_{P,0}, r_0, I, O_0, B_0, S_0 \rangle$  خاتمه می یابد. اثبات بر اساس برهان خلف انجام می شود. فرض کنید خاتمه نمی یابد. پس باید لیست نامتناهی از حالت های اجرای سراسری به شکل زیر باشد.

$$\langle L_{P,0}, r_0, I, O_0, B_0, S_0 \rangle \Rightarrow \langle L_1, r_1, I, O_1, B_1, S_1 \rangle \Rightarrow \langle L_2, r_2, I, O_2, B_2, S_2 \rangle \Rightarrow \dots$$

با توجه به متناهی بودن مجموعه سطوح امنیتی که برای هر یک از آن ها یک حالت اجرای محلی در  $L_i$  وجود دارد و نیز زمان بندی سهم های زمانی مشخص و ثابت برای هر دور از اجرای آن ها و همچنین این موضوع که برای هر گام حالت اجرای سراسری روی یک گام حالت محلی برای یکی از عناصر  $L_i$  اعمال می شود، پس باید حداقل یک سطح امنیتی مانند  $l$  وجود داشته باشد که تعداد نامتناهی از گام های اجرای سراسری برای یک گام اجرای محلی به یک حالت اجرای محلی در سطح امنیتی  $l$  اعمال شود. پس با توجه به لم بالا، باید مجموعه ای نامتناهی از حالت های اجرای استاندارد برای  $\langle c_i, m_i, p_i, I_l, O_i \rangle$  وجود داشته باشد به طوری که



$$\langle P, m_0, p_0, I_l, O_0 \rangle \rightarrow^1 \langle c_1, m_1, p_1, I_l, O_1 \rangle$$

$$\langle P, m_0, p_0, I_l, O_0 \rangle \rightarrow^2 \langle c_2, m_2, p_2, I_l, O_2 \rangle$$

...

که در آن  $I_l = I_{|l}$  است. با توجه به این که برنامه عدم تداخل را برآورده می کند و فرض  $(P, I) \rightarrow^* (p, O)$ ، پس می توان داشت که  $(P, I_l) \rightarrow^* (p', O')$  که  $p' =_l p$  و  $O' =_l O$ . همچنین، بر اساس قطعی بودن معناساخت اجرای استاندارد، برنامه  $P$  با ورودی  $I_l$  قطعاً خاتمه خواهد یافت که در تناقض با فرض خلف است. شایان ذکر است که اگر اجرای استاندارد برنامه نامتناهی باشد، متعاقباً اجرای برنامه تحت مکانیزم نیز نامتناهی خواهد بود. زیرا در صورتی که برنامه ناامن باشد، حداقل رونوشت با سطح امنیتی بالاترین سطح، همان ورودی ها را گرفته است و نحوه اجرای برنامه دقیقاً مشابه اجرای اصلی خواهد بود. با فرض امن بودن برنامه، به سادگی و متشابهاً می توان نتیجه گرفت که همه رونوشت ها نامتناهی خواهند بود و در هر گام، معادل اجرای استاندارد خواهد بود.

حال در صورتی که  $\langle L_f, r_f, I, O_f, B_f, S_f \rangle$  را حالت اجرای سراسری در نظر بگیریم به طوری که

$$\langle L_{P,0}, r_0, I, O_0, B_0, S_0 \rangle \Rightarrow^* \langle L_f, r_f, I, O_f, B_f, S_f \rangle$$

و  $\langle L_f, r_f, I, O_f, B_f, S_f \rangle$  خاتمه یافته است. به سادگی می توان مشاهده کرد که  $L_f$  تهی است و دیگر نمی توان قاعده ای از معناساخت سراسری به آن اعمال کرد. این موضوع با قاعده شماره ۳ معناساخت سراسری نیز هم خوانی دارد.

۲) عدم انتظار نامحدود برای اجرای رونوشت ها: با توجه به نحوه زمان بندی و شروع اجرا از رونوشت های با سطح کمتر، به این ترتیب در صورت امن بودن برنامه داده شده، ورودی های سطح پایین برنامه در سهم های زمانی خوانده می شوند و هیچ یک از رونوشت ها نیازی به انتظار برای خوانده شدن ورودی سطح پایین تر توسط رونوشت همان سطح وجود ندارد. از طرف دیگر، هر یک از رونوشت ها تنها به اندازه  $t$  گام فرصت اجرا دارند و بلافاصله نوبت اجرای رونوشت بعدی خواهد بود. پس معناساخت به طوری طراحی شده است که هیچ یک از رونوشت ها در وضعیت انتظار نامحدود برای پیشروی اجرا نخواهند داشت. علاوه بر این، حتی اگر برنامه ناامن باشد و اجرای رونوشتی نامتناهی باشد، بعد از زمان ثابت و مشخصی مجدداً سهم زمانی به رونوشت ها خواهد رسید و اجرا در وضعیت قحطی زدگی قرار نمی گیرد.

۳ ورودی/خروجی صحیح: تنها بخش باقی مانده برای اثبات شفافیت کامل مکانیزم، نشان دادن  $r_f = p$  و  $O_f = O$  است. با توجه به خالی بودن  $L_f$  و طبق قاعده ۳ معناساخت سراسری و ۱۳ معناساخت محلی، برای هر  $l \in \mathcal{L}$  اجرا در حالت  $\langle \text{skip}, m_l, p_l, n_l \rangle_l$  خواهد بود. طبق لم ۵، داریم

$$\langle P, m_0, p_0, I_l, O_0 \rangle \rightarrow^* \langle \text{skip}, m_l, p_l, I_l, O_l \rangle$$

که در آن  $I_l = I|_l$  و  $O_l(o) = O_f(o)$  برای هر  $o$  به طوری که  $\sigma_{out}(o) = l$  با توجه به این که برنامه  $P$  عدم تداخل قوی را برآورده می کند و  $I_l = I$  در نتیجه  $O_l = O$  و  $p_l = p$  طبق لم ۱، می توان نتیجه گرفت که برای هر  $i$  که  $\sigma_{in}(i) = l$ ،  $r_f(i) = p_l(i) = p(i)$  متشابهاً می توان برای هر گام از اجرای برنامه امن  $P$  همین نتیجه را گرفت. با توجه به این که گزاره های بالا برای هر سطح امنیتی  $l$  برقرار است، پس قضیه اثبات شده است. ■

## ۴-۵ جمع بندی

در این فصل، با صوری سازی مکانیزم پیشنهادی به بیان دقیق معناساخت رفتار مکانیزم پرداخته شد. همچنین، پس از تعریف درستی و شفافیت کامل برای عدم تداخل حساس به زمان، اثبات شد که مکانیزم چنداجرایی امن بافردار برای این خط مشی درست و شفاف کامل است.

## فصل ششم

### جمع‌بندی و کارهای آینده

## جمع‌بندی و کارهای آینده

این فصل به نتیجه‌گیری و ارائه پیشنهادهایی برای پژوهش‌های آتی اختصاص دارد که در آن با بیان مجدد مسئله پژوهشی و توضیح مختصر مکانیزم پیشنهادشده، تعدادی از مسائل باز این حوزه نیز معرفی و ایده‌های کلی برای حل آن‌ها عنوان می‌شود.

### ۶-۱ جمع‌بندی و نتیجه‌گیری

برنامه‌های غیرقابل اعتماد هر روزه توسط کاربران نرم‌افزارهای تحت وب و سیار در حال استفاده است. از اصلی‌ترین محورهای امنیت باید به حفظ محرمانگی و صحت داده‌ها اشاره کرد. خط‌مشی‌های جریان اطلاعات برای بیان خواسته‌های امنیتی در این حوزه مطرح می‌شوند. به این ترتیب که برای حفظ محرمانگی، نباید جریانی از داده‌های محرمانه به سمت داده‌های قابل مشاهده برای عموم وجود داشته باشد. از آنجایی که مهاجم تنها به داده‌های عمومی دسترسی دارد، وجود چنین جریانی از اطلاعات نوعی نقض امنیت محسوب می‌شود. لازم به ذکر است که خط‌مشی‌های صحت را می‌توان دوگانی از خط‌مشی‌های محرمانگی در نظر گرفت.

اغلب خط‌مشی‌های جریان اطلاعات را می‌توان در قالب خط‌مشی‌های عدم تداخل بیان کرد. منظور از خط‌مشی عدم تداخل آن است که مشاهدات کاربران عمومی فقط به داده‌های همان سطح وابستگی داشته باشد. بنابراین، در یک برنامه ورودی‌های محرمانه نباید هیچ تأثیری روی خروجی‌های عمومی قابل مشاهده برای کاربران سطح پایین داشته باشد. انواع مختلفی از خط‌مشی عدم تداخل وجود دارد که هر یک به یکی از کانال‌های نهان مرتبط می‌شود.

با توجه به ضرورت حفظ امنیت و محرمانگی کاربران، مکانیزم‌هایی برای برقراری امنیت در سامانه‌ها مطرح شده است. به طور کلی می‌توان مکانیزم‌ها را به سه دسته کلی ایستا، پویا و بازنویسی برنامه تقسیم‌بندی کرد. مکانیزم‌های ایستا به دلیل رویکرد محافظه‌کارانه، معمولاً تعدادی از برنامه‌های امن را نیز مردود اعلام می‌کنند؛ در حالی که مکانیزم‌های پویا، به دلیل دسترسی به داده‌های زمان‌اجرا می‌توانند آسان‌گیرتر باشند. یکی از مکانیزم‌هایی که اخیراً توجه بسیاری از پژوهشگران و مهندسين امنیت را به خود معطوف کرده است، روش چنداجرای امن (SME) است. این مکانیزم پویا و جعبه‌سیاه

که بدون نیاز به در اختیار داشتن کد منبع برنامه قادر است تا نوعی از خط‌مشی‌های امنیتی جریان اطلاعات را اعمال کند، زمینه جدیدی را در پژوهش‌های این حوزه ایجاد کرده است.

این روش قادر است تا با اجرای چندباره برنامه برای سطوح مختلف امنیتی و لحاظ کردن قواعد خاصی برای ورودی‌ها و خروجی‌های آن‌ها، خط‌مشی‌های عدم تداخل حساس به خاتمه و حساس به زمان را در زمان اجرا اعمال کند. منظور از خط‌مشی‌های عدم تداخل حساس به خاتمه و زمان این است که رفتار برنامه باید به ازای ورودی‌های محرمانه مختلف و ورودی‌های عمومی یکسان، چه از نظر خاتمه یا واگرایی و چه از نظر زمان مورد نیاز برای تولید هر خروجی برنامه، یکسان باقی بماند. به این ترتیب، مهاجم قادر نخواهد بود تا با اجرای مختلف برنامه، چیزی از اطلاعات سطح بالا به دست بیاورد.

نحوه عملکرد چنداجرایی امن به این صورت است که به تعداد سطوح امنیتی، از برنامه رونوشت تهیه می‌کند و به ازای هر سطح، یکبار برنامه را اجرا می‌کند. برای اجرای هر یک از این رونوشت‌ها قواعد خاصی برای ورودی‌ها و خروجی‌های تولیدشده در نظر می‌گیرد. ورودی‌های هر رونوشت عبارتند از ورودی‌های همان سطح و سطوح پایین‌تر، و برای ورودی‌های سطح بالا، از مقادیر پیش‌فرض جعلی استفاده می‌کند تا اجرای برنامه به مشکل نخورد. به این ترتیب رونوشت سطح پایین، هیچ‌گونه دسترسی‌ای به ورودی‌های واقعی سطح بالا نخواهند داشت و طبیعتاً خروجی‌های تولیدشده توسط آن رونوشت، مستقل از مقادیر ورودی محرمانه خواهد بود. همچنین هر رونوشت فقط مسئول تولیدکردن رویدادهای خروجی مرتبط با سطح امنیتی خودش است و خروجی‌های تولیدشده مستقیماً به کانال‌های خروجی منتقل می‌شوند. همان‌طور که گفته شد، این روش به نوعی برنامه را مطابق با خط‌مشی عدم تداخل اجرا می‌کند. پس درستی این روش مشخص است. اما درباره شفافیت مکانیزم در قبال برنامه‌های امن، که نباید رفتار آن‌ها تحت اجرا توسط مکانیزم دچار تغییر شود، کاملاً به زمان‌بندی و همگام‌سازی اجرای رونوشت‌ها وابسته است.

در این پایان‌نامه، به دنبال اعمال درست و شفاف خط‌مشی عدم تداخل حساس به زمان توسط مکانیزمی مبتنی چنداجرایی امن بودیم و مکانیزمی به نام چنداجرایی امن بافردار مطرح شد که با بهره‌گیری از بافرهای محدود، می‌تواند این خواسته را برآورده کند. نحوه عملکرد مکانیزم به این صورت است که با بهره‌گیری از یک زمان‌بندی مشابه تسهیم یا گردش به نوبت، رونوشت‌ها با اولویت سطوح پایین‌تر، در هر نوبت به مقدار محدودی اجرا می‌شوند و پس از پایان کار اجرای بالاترین سطح، به بررسی

مقادیر موجود در بافرها، که رویدادهای تولیدشده توسط رونوشت‌ها هستند، پرداخته می‌شود. در هر گام، خروجی‌های تولیدشده توسط رونوشت‌های مختلف به کانال خروجی مرتبط و هم‌سطح رویداد، منتقل می‌شود. در صورتی که مقادیر موجود در عناصر با شماره اندیس یکسان بافرها با یکدیگر سازگار نباشد و بیش از یک رویداد خروجی در شماره اندیس یکسان بافرها وجود داشته باشد، می‌توان نتیجه گرفت که برنامه ورودی امن نبوده است.

با توجه به قواعد اعمال‌شده برای ورودی‌ها و خروجی‌های رونوشت‌ها و این‌که هر رونوشت در هر نوبت فرصت محدود و ثابتی برای اجرا دارد و مدت انتظار برای ادامه اجرا نیز مشخص و ثابت است، پس از اجراهای سایر سطوح و مقادیر غیر از سطوح پایین‌تر، نقشی در نحوه اجرای آن‌ها نخواهد داشت. همچنین در صورتی که برنامه امن بوده باشد، به دلیل نگهداری رویدادهای تولیدشده در شماره اندیس مرتبط با زمان اجرا در بافر، به نحوی رویداد و تعداد گام لازم برای تولید آن نیز ذخیره شده است. پس با بررسی بافر و انتقال خروجی‌ها به کانال‌های مرتبط با سطوح امنیتی، ترتیب رویدادهای خروجی مطابق با ترتیب رویدادها در اجرای برنامه اصلی خواهد بود. دقت شود که مکانیزم‌های پیشنهادشده قبلی، تنها حفظ ترتیب رویدادها را با در نظر نگرفتن سایر کانال‌ها برآورده می‌کنند اما چنداجرایی امن بافردار این ترتیب را با توجه به همه کانال‌ها و اجرای اصلی حفظ می‌کند.

از نظر کارایی می‌توان در دو بُعد حافظه مصرفی و هزینه زمانی بحث کرد. با توجه به این‌که بافرها به تعداد سطوح امنیتی است و هر بافر دارای تعداد محدودی (به اندازه سهم زمانی در نظر گرفته‌شده) حافظه است، از این منظر تفاوت چندانی با مکانیزم‌های دیگر ارائه‌شده نمی‌کند. حتی می‌توان سهم زمانی را یک در نظر گرفت و تنها به تعداد سطوح امنیتی موجود در برنامه، حافظه نیاز خواهد بود و مکانیزم بدون مشکل عمل خواهد کرد. از منظر هزینه زمانی، با توجه به این‌که قسمت بررسی بافرها و خروجی‌دادن نیز برای اجرا به یک سهم زمانی نیاز دارد، زمانی به اندازه حاصل ضرب زمان اجرای استاندارد در یکی بیشتر از تعداد سطوح امنیتی مصرف خواهد شد. باید اشاره کرد که این هزینه زمانی برای استفاده از روش چنداجرایی امن به اندازه حاصل ضرب زمان اجرای استاندارد در تعداد سطوح امنیتی است. پس می‌توان گفت که به نسبت مکانیزم‌های مبتنی بر روش چنداجرایی امن، هزینه زمانی قابل صرف‌نظر است. همچنین، برای بهبود این هزینه می‌توان قسمت بررسی بافرها و خروجی‌دادن

همزمان با اجرای آخرین رونوشت صورت بگیرد. در این صورت، زمانی معادل با زمان مصرف‌شده توسط چنداجرائی امن صرف می‌شود.

برای اثبات درستی و شفافیت کامل مکانیزم پیشنهادشده از زبان مدل استفاده شد و با بهره‌گیری از معناشناخت عملیاتی<sup>۱۲۹</sup>، رفتار مکانیزم صوری‌سازی شد. سپس با بیان صوری قضایای درستی و شفافیت کامل، اثبات شد که مکانیزم پیشنهادی این ویژگی‌ها را داراست.

## ۶-۲ کارهای آینده

پژوهش روی بهبود و تعمیم مکانیزم چنداجرائی امن بافردار در زمینه‌های مختلفی می‌تواند ادامه پیدا کند. یکی از چالش‌های مهم و مورد نیاز در کاربردهای واقعی، خط‌مشی‌های حذف رده‌بندی است. همان‌طور که در کارهای گذشته توجه ویژه‌ای به این زمینه شده است، می‌توان رویکردهای مشابهی را برای افزودن قابلیت پشتیبانی از خط‌مشی‌های حذف رده‌بندی برای این مکانیزم پیشنهاد کرد. با توجه به این که اغلب راه کارهای ارائه‌شده در این زمینه تغییراتی در رویدادهای ورودی و خروجی اعمال می‌کنند، می‌توان به طور مشابه برای مکانیزم چنداجرائی امن بافردار نیز از آن‌ها استفاده کرد.

از دیگر چالش‌های موجود برای روش چنداجرائی امن، می‌توان به نحوه انتخاب مقدار پیش‌فرض مناسب برای جایگزینی ورودی‌های سطح بالا در رونوشت‌های سطح پایین اشاره کرد. انتخاب مقدار پیش‌فرض نامناسب ممکن است به خرابی برنامه منجر شود. از طرفی، اگرچه انتخاب مقدار پیش‌فرض خلی به درستی مکانیزم وارد نمی‌کند اما تأثیر مستقیمی روی میزان شفافیت کاذب مکانیزم دارد. همین مشکل در مکانیزم چنداجرائی امن بافردار نیز مطرح است. راهکار پیشنهادی استفاده از تلفیق تحلیل ایستا، مشابه تحلیل روی گراف‌های وابستگی برنامه، و مکانیزم برای به دست آوردن مقادیر بهتر برای استفاده در اجرای رونوشت‌های سطح پایین است.

همان‌طور که پیشتر عنوان شد، هزینه زمانی بالای ذاتی روش چنداجرائی امن در مکانیزم پیشنهادی ما نیز وجود دارد. به این منظور می‌توان از ایده‌هایی نظیر استفاده از تحلیل ایستا پیش از اجرا توسط مکانیزم استفاده کرد. به این ترتیب، به شرطی که کد منبع برنامه در اختیار باشد، می‌توان پیش از

<sup>129</sup> Operational Semantics

اجرا از امن بودن برنامه مطمئن شد. اما برای کاهش سربار زمان اجرا می‌توان از روش‌های بهینه‌سازی مانند برش خودکار برنامه استفاده کرد. باید دقت داشت که نحوه استفاده از این روش‌ها نباید به صورتی باشد که به درستی مکانیزم لطمه بزند.

از دیگر چالش‌های مهم این حوزه می‌توان از پشتیبانی از قابلیت همروندی و عدم قطعیت برنامه‌ها نام برد. اگرچه به طور کلی می‌توان ادعا کرد که مکانیزم پیشنهادی مستقل از پیچیدگی‌های زبان برنامه‌نویسی قادر است تا خط‌مشی عدم تداخل را اعمال کند، اما نکته این‌جاست که در فضای سامانه‌های همروند، تعریف مناسب از امنیت جریان اطلاعات کار آسانی نیست. اگرچه تلاش‌هایی، مانند قطعیت مشاهده‌ای و عدم تداخل احتمالاتی، برای تعریف دقیق و صوری خط‌مشی‌ها با در نظر گرفتن همروندی و عدم قطعیت برنامه‌ها انجام شده است اما کماکان پژوهشگران این حوزه آن‌ها را سخت‌گیرانه می‌دانند. می‌توان نشان داد که خط‌مشی قطعیت مشاهده‌ای نیز توسط روش چنداجرایی امن و مکانیزم پیشنهادی این پایان‌نامه قابل اعمال است.

همچنین، پیاده‌سازی چنداجرایی امن بافردار یکی دیگر از کارهایی است که می‌تواند به عنوان کارهای آتی این پایان‌نامه مطرح شود. همان‌طور که چنداجرایی امن در زبان‌ها و بسترهای مختلف پیاده‌سازی شده است، می‌توان این مکانیزم را نیز برای زبان‌های گوناگون نظیر جاوااسکریپت پیاده‌سازی کرد. در آینده به دنبال پیاده‌سازی این تکنیک برای مرورگرهای وب هستیم. علاوه بر این، پیاده‌سازی مکانیزم برای محیط تلفن همراه نیز قابل بررسی است. تاکنون هیچ پیاده‌سازی‌ای از روش چنداجرایی امن برای محیط‌های تلفن همراه انجام نشده است. البته پیچیدگی و درهم‌تنیدگی بخش‌های مختلف سیستم‌عامل‌های تلفن همراه نظیر اندروید و نیز شناور بودن برچسب‌های سطوح امنیتی در آن محیط‌ها باعث افزایش دشواری‌های انجام آن می‌شود، اما به نظر می‌رسد که پیاده‌سازی این مکانیزم در بستر تلفن‌های همراه می‌تواند بخش عمده‌ای از نگرانی‌ها در نقض محرمانگی کاربران را برطرف سازد.



## منابع و مراجع

- [۱] D. Volpano, C. Irvine, and G. Smith, "A sound type system for secure flow analysis," *Journal of computer security*, vol. 4, no. 2-3, 1996, pp. 167–187.
- [۲] G. Le Guernic, "Confidentiality enforcement using dynamic information flow analyses," Ph.D thesis, Kansas State University, 2007.
- [۳] K. W. Hamlen, F. B. Schneider, and G. Morrisett, "Computability Classes for Enforcement Mechanisms," *ACM Transactions on Programming Languages and Systems*, vol. 28, no. 1, 2006, pp. 175–205.
- [۴] A. Lamei, "Formal Characterization of Security Policy Enforcement through Program Rewriting," Ph.D. thesis, Amirkabir University of Technology, 2016.
- [۵] F. B. Schneider, "Enforceable security policies," *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 1, 2000, pp. 30–50.
- [۶] D. Devriese and F. Piessens, "Noninterference through secure multi-execution," in *Proceedings - IEEE Symposium on Security and Privacy*, 2010, pp. 109–124.
- [۷] S. A. Zdancewic, "Programming languages for information security," Ph.D. thesis, Cornell University, 2002.
- [۸] M. R. Clarkson and F. B. Schneider, "Hyperproperties," *Journal of Computer Security*, vol. 18, no. 6, 2010, pp. 1157–1210.
- [۹] M. A. Bishop, *The Art and Science of Computer Security*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.
- [۱۰] G. Barthe, J. M. Crespo, D. Devriese, F. Piessens, and E. Rivas, "Secure multi-execution through static program transformation," in *Formal Techniques for Distributed Systems '12*, 2012, pp. 186–202.
- [۱۱] D. E. Denning, "A lattice model of secure information flow," *Communications of the ACM*, vol. 19, no. 5, 1976, pp. 236–243.
- [۱۲] E. Cohen, "Information Transmission in Computational Systems," *Proceedings of Sixth ACM Symposium on Operating Systems Principles, SOSP '77*, 1977, pp. 133–139.
- [۱۳] J. A. Goguen and J. Meseguer, "Security Policies and Security Models," *IEEE Symposium on Security and Privacy*, Oakland, CA, USA, 1982, pp. 11-20.

- [١٤] D. Sutherland, "A model of information," in *Proceedings - 9th National Computer Security Conference*, 1986, pp. 175–183.
- [١٥] D. Hedin and A. Sabelfeld, "A perspective on information-flow control," in *NATO Science for Peace and Security Series - D: Information and Communication Security*, vol. 33: Software Safety and Security, no. 10, 2012, pp. 319–347.
- [١٦] N. Bielova and T. Rezk, "A taxonomy of information flow monitors," in *Proceedings of the 5th International Conference on Principles of Security and Trust*, vol. 9635, 2016, pp. 46–67.
- [١٧] V. Kashyap, B. Wiedermann, and B. Hardekopf, "Timing- and Termination-Sensitive Secure Information Flow: Exploring a New Approach," in *IEEE Symposium on Security and Privacy (S&P 2011)*, 2011, pp. 413–428.
- [١٨] J. Agat, "Transforming out Timing Leaks," in *Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2000, pp. 40–53.
- [١٩] D. McCullough, "Noninterference and the composability of security properties," in *Proceedings. IEEE Symposium on Security and Privacy (S&P)*, 1988, pp. 177–186.
- [٢٠] D. Volpano and G. Smith, "Probabilistic noninterference in a concurrent language," in *Proceedings of 11th IEEE Computer Security Foundations Workshop*, Rockport, MA, 1998, pp. 34–43.
- [٢١] S. Zdancewic and A. C. Myers, "Observational determinism for concurrent program security," in *16th IEEE Computer Security Foundations Workshop*, vol. 2003–January, 2003, pp. 29–43.
- [٢٢] G. Barthe, P. R. D'Argenio, and T. Rezk, "Secure information flow by self-composition," in *Proceedings of 17th IEEE Computer Security Foundations Workshop*, 2004, pp. 100–114.
- [٢٣] A. Sabelfeld and A. C. Myers, "Language-based information-flow security," in *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 1, 2003, pp. 5–19.
- [٢٤] J. Ligatti and S. Reddy, "A theory of runtime enforcement, with results," in *European Symposium on Research in Computer Security*, 2010, pp. 87–100.
- [٢٥] J. Ligatti, L. Bauer, and D. Walker, "Enforcing non-safety security policies with program monitors," in *Proceedings of the 10th European Conference on Research in Computer Security (ESORICS'05)*, 2005, pp. 355–373.
- [٢٦] Ú. Erlingsson, "The inlined reference monitor approach to security policy enforcement," Ph.D. thesis, Cornell University, 2004.
- [٢٧] G. Gheorghe and B. Crispo, "A survey of runtime policy enforcement techniques and implementations," University of Trento, Technical Report #DISI-11-477, 2011.

- [۲۸] A. Sabelfeld and A. Russo, "From dynamic to static and back: riding the roller coaster of Information-flow control research," in *Proceedings of the 7th International Andrei Ershov Memorial Conference on Perspectives of Systems Informatics*, 2010, pp. 352–365.
- [۲۹] F. Imanimehr and M. S. Fallah, "How Powerful Are Run-Time Monitors with Static Information?," *The Computer Journal*, vol. 59, no. 11, 2016, pp. 1623–1636.
- [۳۰] R. Capizzi, A. Longo, V. N. Venkatakrishnan and A. P. Sistla, "Preventing Information Leaks through Shadow Executions," *Annual Computer Security Applications Conference (ACSAC)*, Anaheim, CA, 2008, pp. 322-331.
- [۳۱] T. Khatiwala, R. Swaminathan, and V. N. Venkatakrishnan, "Data sandboxing: A technique for enforcing confidentiality policies," in *22nd Annual Computer Security Applications Conference (ACSAC'06)*, Miami Beach, FL, USA, 2006, pp. 223-234.
- [۳۲] W. Rafnsson and A. Sabelfeld, "Secure Multi-execution: Fine-Grained, Declassification-Aware, and Transparent," in *2013 IEEE 26th Computer Security Foundations Symposium (CSF '13)*, 2013, pp. 33–48.
- [۳۳] D. Zanarini, M. Jaskelioff, and A. Russo, "Precise enforcement of confidentiality for reactive systems," in *Proceedings of the 2013 IEEE 26th Computer Security Foundations Symposium (CSF '13)*, 2013, pp. 18–32.
- [۳۴] W. De Groef, D. Devriese, N. Nikiiforakis, and F. Piessens, "Secure multi-execution of web scripts: Theory and practice," *Journal of Computer Security - Web Application Security*, vol. 22, no. 4, 2014, pp. 469–509.
- [۳۵] D. Zanarini and M. Jaskelioff, "Monitoring Reactive Systems with Dynamic Channels," in *Proceedings of the Ninth Workshop on Programming Languages and Analysis for Security*, 2014, pp. 66–78.
- [۳۶] M. Vanhoef, W. De Groef, D. Devriese, F. Piessens, and T. Rezk, "Stateful Declassification Policies for Event-Driven Programs," in *2014 IEEE 27th Computer Security Foundations Symposium (CSF '14)*, 2014, pp. 293–307.
- [۳۷] M. Ngo, F. Massacci, D. Milushev, and F. Piessens, "Runtime Enforcement of Security Policies on Black Box Reactive Programs," *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, vol. 3, no. 1, 2015, pp. 43–54.
- [۳۸] I. Bolosteanu and D. Garg, "Asymmetric Secure Multi-execution with Declassification," in *Proceedings of the 5th International Conference on Principles of Security and Trust*, vol. 9635, Springer-Verlag New York, Inc., 2016, pp. 24–45.
- [۳۹] M. Jaskelioff and A. Russo, "Secure multi-execution in haskell," in *Proceedings of the 8th International Conference on Perspectives of System Informatics*, 2012, pp. 170–178.
- [۴۰] T. H. Austin and C. Flanagan, "Multiple facets for dynamic information flow,"

- 
- ACM SIGPLAN-SIGACT Symposium on Principles of programming languages - POPL '12*, vol. 47, no. 1, 2012, p. 165-178.
- [٤١] A. Sabelfeld and D. Sands, "Declassification: Dimensions and principles," *Journal of Computer Security*, vol. 17, no. 5, 2009, pp. 517–548.
  - [٤٢] N. Bielova and T. Rezk, "Spot the Difference: Secure Multi-execution and Multiple Facets," in *Computer Security -- ESORICS 2016: 21st European Symposium on Research in Computer Security*, Springer International Publishing, 2016, pp. 501–519.
  - [٤٣] N. N. M. Ngo, "A Programmable Enforcement Framework for Security Policies," Ph.D. thesis, University of Trento, 2016.
  - [٤٤] T. H. Austin, T. Schmitz, and C. Flanagan, "Multiple Facets for Dynamic Information Flow with Exceptions," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 39, no. 3, 2017, pp. 1–56.
  - [٤٥] N. Bielova, D. Devriese, F. Massacci, and F. Piessens, "Reactive non-interference for a browser model," in *2011 5th International Conference on Network and System Security*, 2011, pp. 97–104.
  - [٤٦] W. De Groef, D. Devriese, N. Nikiforakis, and F. Piessens, "FlowFox: a web browser with flexible and precise information flow control," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 748–759.
  - [٤٧] W. Rafnsson, L. Jia, and L. Bauer, "Timing-Sensitive Noninterference through Composition," in *Proceedings of the 6th International Conference on Principles of Security and Trust*, vol. 10204, Springer-Verlag New York, Inc., 2017, pp. 3–25.

## پیوست

### الف - صوری سازی معاشناخت مکانیزم چنداجرایی امن بافردار همراه با گزارش نقض امنیت

معاشناخت مکانیزم در حالت همراه با گزارش نقض امنیت مشابه صوری سازی انجام شده در فصل چهارم است. در این قسمت، بخش‌هایی از معاشناخت مکانیزم متفاوت خواهد بود که در ادامه آمده است. تنها فرضی که به مفروضات اولیه اضافه می‌شود این است که تابع  $\sigma_{out}$  باید یک به یک باشد. البته با توجه این که در صورتی که برای یک سطح، بیش از یک کانال خروجی در نظر گرفته شده باشد، می‌توان کانال‌ها را با یکدیگر ادغام شده در نظر گرفت. پس چنین فرضی به کلیت صوری سازی را خدشه‌ای وارد نمی‌کند. همچنین، یادآوری می‌شود که هر رونوشت در یک سطح امنیتی، تمامی رویدادهای خروجی و ورودی مربوط به سطوح پایین‌تر از خود را در بافر ثبت می‌کنند. بنابراین، اجرای رونوشت دارای بالاترین سطح امنیتی دقیقاً از منظر تولید رویدادهای ورودی و خروجی مشابه اجرای اصلی خواهد بود. به این ترتیب می‌توان در هر گام، رفتار برنامه تحت مکانیزم را با رفتار اصلی برنامه مقایسه و بررسی کرد. در صورتی که تنها رونوشت دارای بالاترین سطح امنیتی همه رویدادها را در بافر خودش ثبت کند، ممکن است حالتی وجود داشته باشد که به واسطه مقادیر واقعی داده شده در آن رونوشت، نقض امنیت تشخیص داده نشود. لازم است که هر رونوشت همه رویدادهای مرتبط با سطوح پایین‌تر خود را در بافر ذخیره و نگهداری کند.

محتویاتی که در بافر می‌تواند ذخیره شود، در قالب زیر خواهد بود.  $\sim$  نمادی به معنای پیشروی یک گام از اجراست. دلیل استفاده از این نماد، تمایز بین اجرا و خاتمه برنامه است که برای بررسی و گزارش نقض امنیت نیاز است. همچنین، رویدادهای ورودی نیز در بافر ثبت می‌شوند. زیرا ممکن است

زمانی که برنامه می‌خواهد ورودی را بخواند در دو اجرا متفاوت باشد که در این حالت نیز این تفاوت برای کاربر سطح پایین قابل مشاهده خواهد بود.

$$d: \langle in/out; l; v \rangle \mid \sim \mid \emptyset$$

به این ترتیب عملیات خواندن از ورودی و نوشتن رویدادها در بافر به شرح زیر خواهد بود:

$$\frac{I(i) = q \quad p(i) = n \quad q(n) = v}{read(I, i, p) = v}$$

$$\frac{B(o) = [d_0, \dots, d_{t-1}] \quad 0 \leq index < t}{write(B, o, d, index) = B(o)[d_{index} \mapsto d]}$$

به طور مشابه، دو معناشناخت محلی و سراسری برای صوری‌سازی رفتار مکانیزم تعریف می‌شود. یک پیکربندی اجرای محلی به شکل  $\langle c, m, p, n \rangle_l$  تعریف می‌شود که در آن  $c$  یک دستور،  $m$  یک حافظه،  $p$  اشاره‌گر ورودی،  $n$  تعداد گام‌های اجرا و  $l$  سطح امنیتی است. همچنین یک پیکربندی اجرای سراسری به شکل  $\langle [lec_1, \dots, lec_j], r, I, O, B, s, T \rangle$  مشخص می‌شود که تنها نماد  $T$  به منظور نگه‌داری وضعیت نقض یا عدم نقض امنیت اضافه شده است. به این ترتیب که  $T$  متغیر بولی است که در به طور پیش‌فرض  $TRUE$  است و به محض مشاهده نقض امنیت، به مقدار  $FALSE$  تغییر می‌کند و تا پایان اجرا به همان مقدار می‌ماند. پس اگر در پایان اجرا، مقدار این متغیر  $TRUE$  باشد، به این معناست که اجرای این برنامه تحت مکانیزم چنداجرای امن بافردار بدون نقض امنیت بوده است. از طرفی، در گامی از اجرا که مقدار به  $FALSE$  تبدیل شود، نقض امنیت رخ داده است و گزارش می‌شود. پس می‌توان حمله احتمالی را و متعاقباً مسیر اجرای برنامه را تشخیص داد.

## الف - ۱ قواعد معناشناخت محلی

$$c = \text{if } e \text{ then } c_{true} \text{ else } c_{false} \quad m(e) = b$$

$$\frac{B' = write(B, \sigma_{out}^{-1}(l), \sim, n \bmod t) \quad n' = n + 1}{\langle c, m, p, n \rangle_l, r, I, B \Rightarrow \langle c_b, m, p, n' \rangle_l, r, I, B'}$$

$$\frac{\langle c_1, m, p, n \rangle_l, r, I, B \Rightarrow \langle c'_1, m', p', n' \rangle_l, r', I, B'}{\langle c_1; c_2, m, p, n \rangle_l, r, I, B \Rightarrow \langle c'_1; c_2, m', p', n' \rangle_l, r', I, B'}$$

$$\frac{B' = \text{write}(B, \sigma_{out}^{-1}(l), \simeq, n \bmod t) \quad n' = n + 1}{\langle \text{skip}; c, m, p, n \rangle_l, r, I, B \Rightarrow \langle c, m, p, n' \rangle_l, r, I, B'}$$

$$\frac{c = \text{while } e \text{ do } c_{loop} \quad m(e) = \text{true} \quad B' = \text{write}(B, \sigma_{out}^{-1}(l), \simeq, n \bmod t) \quad n' = n + 1}{\langle c, m, p, n \rangle_l, r, I, B \Rightarrow \langle c_{loop}; c, m, p, n' \rangle_l, r, I, B'}$$

$$\frac{c = \text{while } e \text{ do } c_{loop} \quad m(e) = \text{false} \quad B' = \text{write}(B, \sigma_{out}^{-1}(l), \simeq, n \bmod t) \quad n' = n + 1}{\langle c, m, p, n \rangle_l, r, I, B \Rightarrow \langle \text{skip}, m, p, n' \rangle_l, r, I, B'}$$

$$\frac{m(e) = v \quad m' = m[x \mapsto v] \quad B' = \text{write}(B, \sigma_{out}^{-1}(l), \simeq, n \bmod t) \quad n' = n + 1}{\langle x := e, m, p, n \rangle_l, r, I, B \Rightarrow \langle \text{skip}, m', p, n' \rangle_l, r, I, B'}$$

$$\frac{c = \text{output } e \text{ to } o \quad m(e) = v \quad \sigma_{out}(o) \leq l \quad B' = \text{write}(B, \sigma_{out}^{-1}(l), \langle \text{out}; \sigma_{out}(o); v \rangle, n \bmod t) \quad n' = n + 1}{\langle c, m, p, n \rangle_l, r, I, B \Rightarrow \langle \text{skip}, m, p, n' \rangle_l, r, I, B'}$$

$$\frac{c = \text{output } e \text{ to } o \quad m(e) = v \quad \sigma_{out}(o) \not\leq l \quad B' = \text{write}(B, \sigma_{out}^{-1}(l), \simeq, n \bmod t) \quad n' = n + 1}{\langle c, m, p, n \rangle_l, r, I, B \Rightarrow \langle \text{skip}, m, p, n' \rangle_l, r, I, B'}$$

$$\frac{c = \text{input } x \text{ from } i \quad \sigma_{in}(i) \not\leq l \quad m' = m[x \mapsto v_{default}] \quad B' = \text{write}(B, \sigma_{out}^{-1}(l), \langle \text{in}; \sigma_{in}(i); v_{default} \rangle, n \bmod t) \quad n' = n + 1}{\langle c, m, p, n \rangle_l, r, I, B \Rightarrow \langle \text{skip}, m', p, n' \rangle_l, r, I, B'}$$

$$\frac{c = \text{input } x \text{ from } i \quad \sigma_{in}(i) = l \quad v = \text{read}(I, i, p) \quad p' = p[i \mapsto p(i) + 1] \quad m' = m[x \mapsto v] \quad r' = r[i \mapsto p'(i)] \quad B' = \text{write}(B, \sigma_{out}^{-1}(l), \langle \text{in}; \sigma_{in}(i); v \rangle, n \bmod t) \quad n' = n + 1}{\langle c, m, p, n \rangle_l, r, I, B \Rightarrow \langle \text{skip}, m', p', n' \rangle_l, r', I, B'}$$

$$\frac{c = \text{input } x \text{ from } i \quad \sigma_{in}(i) < l \quad r(i) \leq p(i) \quad B' = \text{write}(B, \sigma_{out}^{-1}(l), \simeq, n \bmod t) \quad n' = n + 1}{\langle c, m, p, n \rangle_l, r, I, B \Rightarrow \langle c, m, p, n' \rangle_l, r, I, B'}$$

$$\frac{c = \text{input } x \text{ from } i \quad \sigma_{in}(i) < l \quad r(i) > p(i) \quad v = \text{read}(I, i, p) \quad m' = m[x \mapsto v] \quad B' = \text{write}(B, \sigma_{out}^{-1}(l), \langle \mathbf{in}; \sigma_{in}(i); v \rangle, n \bmod t) \quad n' = n + 1}{\langle c, m, p, n \rangle_l, r, I, B \Rightarrow \langle \mathbf{skip}, m', p, n' \rangle_l, r, I, B'}$$

$$\frac{B' = \text{write}(B, \sigma_{out}^{-1}(l), \emptyset, n \bmod t) \quad n' = n + 1}{\langle \mathbf{skip}, m, p, n \rangle_l, r, I, B \Rightarrow \langle \mathbf{skip}, m, p, n' \rangle_l, r, I, B'}$$

## الف - ۲ قواعد معناشناخت سراسری

قواعد مورد استفاده در این قسمت مشابه حالت قبل است، با این تفاوت که متغیر  $T$  در هر وضعیت پیکربندی به روز می شود.

$$\frac{s \neq j \quad lec = lec_s \quad s' = s + 1 \quad lec, r, I, B \Rightarrow^t lec', r', I, B' \quad L' = L[lec \mapsto lec']}{\langle L, r, I, O, B, s, T \rangle \Rightarrow^t \langle L', r', I, O, B', s', T \rangle}$$

$$\frac{s = j \quad lec = lec_s \quad s' = 1 \quad lec, r, I, B \Rightarrow^t lec', r', I, B' \quad L' = L[lec \mapsto lec'] \quad B', O, 0, T \Rightarrow^t B_0, O', t, T'}{\langle L, r, I, O, B, s, T \rangle \Rightarrow^{2t} \langle L', r', I, O', B_0, s', T' \rangle}$$

$$\frac{s = 1 \quad \forall lec_{index} \in L. lec_{index} = \langle \mathbf{skip}, m, p, n \rangle_l}{L = [ ]}$$

اما بخش متفاوت در این حالت از مکانیزم، قسمت بررسی بافر و خروجی دادن است. در این قسمت، در هر گام از بررسی بافر، مقدار حافظه های متناظر با شماره اندیسی که در حال بررسی است، با یکدیگر مقایسه می شوند. در صورتی که رویدادهای موجود در آن ها با یکدیگر سازگار نباشند، مقدار متغیر  $T$  به  $FALSE$  تغییر می کند. با فرض مقدار اولیه  $TRUE$  برای متغیر  $T$ ، در صورتی که در هر گام از بررسی



یکی از حالت‌های زیر باشد، مقدار  $T$  بدون تغییر خواهد ماند و تا پایان اجرای فعلی برنامه، نقض امنیت گزارش نخواهد شد:

- رویداد خروجی تولیدشده توسط یکی از سطوح، در همان شماره اندیس بافر در سطوح بالاتر از خودش وجود داشته باشد و برای سطوح پایین‌تر یا غیرقابل مقایسه، باید مقدار  $\surd$  باشد.
- رویداد ورودی ثبت‌شده توسط یکی از سطوح، در همان شماره اندیس بافر در سطوح بالاتر عیناً وجود داشته باشد و برای سطوح و برای سطوح پایین‌تر یا غیرقابل مقایسه، باید مقدار  $\surd$  باشد.
- اگر برای یکی از رونوشت‌ها مقدار برابر  $\emptyset$  (به معنای خاتمه) بود، برای همه رونوشت‌های دیگر نیز باید همین مقدار در همان شماره اندیس بافر وجود داشته باشد.

در ادامه پیکربندی مربوط به قسمت بررسی بافر و خروجی‌دادن آمده است:

$$B, O, k, T \mapsto B', O', k', T'$$

$$k_0 = 0$$

$$T_0 = TRUE$$

$$d: \langle in/out; l; v \rangle$$

توابع مورد نیاز:

$$d.type = in/out \quad (\text{برگرداندن مولفه اول})$$

$$d.level = l \quad (\text{برگرداندن مولفه دوم})$$

$$d.value = v \quad (\text{برگرداندن مولفه سوم})$$

$$Temp.o = (o_1', \dots, o_{|Temp|}); \quad 0 \leq |Temp| \leq j$$

$$Temp.v = (v_1', \dots, v_{|Temp|})$$

تعریف تابع خروجی‌دادن در کانال‌های خروجی:

$$\frac{O(o_1) = [v_{11}, \dots, v_{1n}] \quad O(o_2) = [v_{21}, \dots, v_{2n'}] \quad \dots \quad O(o_s) = [v_{s1}, \dots, v_{sn''}]}{writeOut(O, (o_1, o_2, \dots, o_s), (v_1, v_2, \dots, v_s)) = O[o_1 \mapsto [v_{11}, \dots, v_{1n}, v_1], o_2 \mapsto [v_{21}, \dots, v_{2n'}, v_2], \dots, o_s \mapsto [v_{s1}, \dots, v_{sn''}, v_s]]}$$

قواعد معناشناخت قسمت بررسی بافر و خروجی‌دادن در کانال‌های خروجی به شرح زیر است.

$$\begin{array}{c}
k' = k + 1 \quad k \leq t - 1 \quad B(o)[k].type = \mathbf{out} \quad l = B(o)[k].level \quad \sigma_{out}(o) = l \\
\forall l' > l. B(\sigma_{out}^{-1}(l'))[k] = B(o)[k] \quad \forall l'' \not\geq l. B(\sigma_{out}^{-1}(l''))[k] = \sim \quad T = TRUE \\
O' = writeOut(O, o, B(o)[k].value) \\
B' = B[\forall o_s. B(o_s)[k] \mapsto \emptyset] \\
\hline
B, O, k, T \models B', O', k', T
\end{array}$$

$$\begin{array}{c}
k' = k + 1 \quad k \leq t - 1 \quad B(o)[k].type = \mathbf{in} \quad l = B(o)[k].level \quad \sigma_{out}(o) = l \\
\forall l' > l. B(\sigma_{out}^{-1}(l'))[k] = B(o)[k] \quad \forall l'' \not\geq l. B(\sigma_{out}^{-1}(l''))[k] = \langle \mathbf{in}; l; v_{default} \rangle \quad T = TRUE \\
B' = B[\forall o_s. B(o_s)[k] \mapsto \emptyset] \\
\hline
B, O, k, T \models B', O, k', T
\end{array}$$

$$\begin{array}{c}
k' = k + 1 \quad k \leq t - 1 \quad \forall o. B(o)[k] = \sim \quad T = TRUE \quad B' = B[\forall o_s. B(o_s)[k] \mapsto \emptyset] \\
\hline
B, O, k, T \models B', O, k', T
\end{array}$$

$$\begin{array}{c}
k' = k + 1 \quad k \leq t - 1 \quad \forall o. B(o)[k] = \emptyset \quad T = TRUE \\
\hline
B, O, k, T \models B, O, k', T
\end{array}$$

$$\begin{array}{c}
k' = k + 1 \quad k \leq t - 1 \quad T = FALSE \quad Temp_0 = () \\
\forall o_s. ((B(o_s)[k].type = \mathbf{out} \wedge B(o_s)[k].level = \sigma_{out}(o_s)) \Rightarrow (Temp = concat(Temp, (o_s, B(o_s)[k].value)))) \\
O' = writeOut(O, Temp.o, Temp.v) \\
B' = B[\forall o_s. B(o_s)[k] \mapsto \emptyset] \\
\hline
B, O, k, T \models B', O', k', T
\end{array}$$

$$\begin{array}{c}
k' = k + 1 \quad k \leq t - 1 \quad T = TRUE \quad violation = TRUE \quad Temp_0 = () \\
\forall o_s. ((B(o_s)[k].type = \mathbf{out} \wedge B(o_s)[k].level = \sigma_{out}(o_s)) \Rightarrow (Temp = concat(Temp, (o_s, B(o_s)[k].value)))) \\
O' = writeOut(O, Temp.o, Temp.v) \quad T' = FALSE \quad B' = B[\forall o_s. B(o_s)[k] \mapsto \emptyset] \\
\hline
B, O, k, T \models B', O', k', T'
\end{array}$$

منظور از متغیر *violation* در قاعده آخر این گزاره منطقی است:

*violation* =

$$\begin{aligned}
& \neg(\forall o. B(o)[k] = \emptyset) \wedge \neg(\forall o. B(o)[k] = \sim) \wedge ( \\
& ((\exists o'. B(o')[k] = \mathbf{out}) \wedge \neg(\exists o. (B(o)[k] = \mathbf{out} \wedge B(o)[k].level = l \wedge \sigma_{out}(o) = l \wedge \\
& \forall l' > l. B(\sigma_{out}^{-1}(l'))[k] = B(o)[k] \wedge \forall l'' \not\geq l. B(\sigma_{out}^{-1}(l''))[k] = \sim))) \vee \\
& ((\exists o'. B(o')[k] = \mathbf{in}) \wedge \neg(\exists o. (B(o)[k] = \mathbf{in} \wedge B(o)[k].level = l \wedge \sigma_{out}(o) = l \wedge \\
& \forall l' > l. B(\sigma_{out}^{-1}(l'))[k] = B(o)[k] \wedge \forall l'' \not\geq l. B(\sigma_{out}^{-1}(l''))[k] = \langle \mathbf{in}; l; v_{default} \rangle))) \\
& )
\end{aligned}$$

## Abstract

Information flow security means guaranteeing confidentiality and integrity by preventing illegal flows of information. This concept has led to the emergence of the theory of noninterference that acts as the semantics of information flow policies. Among various static and dynamic enforcement mechanisms already proposed for information flow policies, secure multi-execution (SME) has attracted enormous attention. SME is a dynamic black-box mechanism that enforces timing- and termination-sensitive noninterference by executing multiple copies of a given program, one copy for each security level, and restricting I/O in these copies. The soundness and transparency of the mechanism highly depends on appropriate scheduling and synchronizing of the copies. Soundness means that the resulting system must be secure and transparency stipulates that the mechanism must preserve the executions of any secure program. Although an acceptable level of transparency at enforcing termination-sensitive noninterference can be achieved by using mechanisms based on SME, it does not hold for timing-sensitive noninterference.

In this thesis, we propose a novel mechanism based on SME to enforce timing-sensitive noninterference which is proven to be sound and attains a level of transparency that whenever the original program is secure, the order of output events are preserved between different output channels. We call the proposed mechanism buffered secure multi-execution (BSME) wherein the events raised by different runs of the program are buffered. Moreover, a round-robin-like scheduler determines how to run the copies of the target program.

**Key Words:** Buffering, Secure Multi-Execution, Soundness, Timing-Sensitive Noninterference, Transparency.



**Amirkabir University of Technology  
(Tehran Polytechnic)**

**Department of Computer Engineering and Information Technology**

**M.Sc. Thesis**

**Improving Multi-Execution-based Mechanisms for  
Enforcing Information Flow Policies**

**By  
Seyed Mohammad Mehdi Ahmadpanah**

**Supervisor  
Dr. Mehran S. Fallah**

**October 2017**