دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر و فناوری اطلاعات

گزارش کتبی درس سمینار

عنوان

اِعمال پویای فوق‌خاصیت‌های امنیتی

نگارش

سید محمدمهدی احمدپناه

استاد راهنما

دکتر مهران سلیمان‌فلاح

استاد درس

دکتر بابک صادقیان

مهر ۱۳۹۵

چکیده

خط‌مشی‌های امنیتی را می‌توان به دو دسته خاصیت و فوق‌خاصیت تقسیم‌بندی کرد. جریان اطلاعات، و عدم تداخل به عنوان معروف‌ترین آن‌ها، به عنوان یکی از مهم‌ترین خط‌مشی‌های محرمانگی و صحت مطرح هستند. این خط‌مشی‌ها از آن جهت که با گزاره‌ای روی بیش از یک اجرا قابل بیان هستند، فوق‌خاصیت به شمار می‌روند. همین تفاوت در نحوه بیان خط‌مشی‌ها باعث پدیدآمدن رویکردهای مختلف در مکانیزم‌های اعمال خط‌مشی‌ها شده است.

انواع مکانیزم‌های امنیتی را می‌توان به دو دسته کلی ایستا و پویا دسته‌بندی کرد که ویژگی اصلی مکانیزم‌های ایستا، تحلیل ایستای کد منبع برنامه قبل از اجرا و محافظه‌کارانه‌بودن آن‌ها است. از طرف دیگر، مکانیزم‌های نظارت بر اجرا دسته اصلی مکانیزم‌های پویا محسوب می‌شود. در گذشته با تصور این‌که مکانیزم‌های اعمال پویا در اعمال خط‌مشی‌های جریان اطلاعات کارایی ندارند، بیشتر پژوهش‌های امنیت زبان‌مبنا حول تحلیل ایستا و روش‌های نوع‌مبنا بوده است. اما در سال‌های اخیر، به دلیل محدودیت‌های تحلیل ایستا و مزایای روش‌های پویا در اعمال خط‌مشی‌های جریان اطلاعات، مانند آسان‌گیربودن این مکانیزم‌ها در مقایسه با روش‌های تحلیل ایستا، توجه ویژه‌ای به این گونه مکانیزم‌ها شده است.

به دلیل ایجاد کانال نهان در شرایطی که مکانیزم اعمال پویا فقط از اطلاعات زمان اجرا استفاده کند، این نوع مکانیزم‌ها نیازمند تحلیل کد منبع برنامه‌ها نیز هستند. با استفاده از اطلاعات تحلیل کد برنامه، می‌توان اطلاعاتی از سایر اجراها به دست آورد و در اعمال خط‌مشی مورد نظر بهره برد.

همچنین با درنظر گرفتن دو معیار درستی و کامل‌بودن، می‌توان مکانیزم‌های مختلف اعمال خط‌مشی‌های امنیتی را مقایسه و دسته‌بندی کرد. مقایسه توانایی مکانیزم‌های مختلف پویا و ناظرهای زمان‌اجرا از دستاوردهای این گزارش به شمار می‌رود.

واژه‌های کلیدی:

امنیت جریان اطلاعات؛ مکانیزم‌های اعمال؛ فوق‌خاصیت؛ مکانیزم‌های پویا؛ نظارت بر اجرا

فهرست عنوان‌ها

صفحه

فهرست شکل‌ها

صفحه

فهرست جدول‌ها

صفحه

فصل اول

مقدمه

مقدمه

امنیت اطلاعات کامپیوتری در سه محور محرمانگی[۱]، صحت[۲] و دسترس‌پذیری[۳] قابل بحث است. کنترل جریان اطلاعات[۴] در دو بُعد محرمانگی و صحت اهمیت دارد. مکانیزم‌های کنترل دسترسی برای حفاظت از محرمانه‌بودن اطلاعات استفاده می‌شوند. مکانیزم‌های کنترل دسترسی را می‌توان به دو نوع کنترل دسترسی اجباری[۵] و کنترل دسترسی تفویضی[۶] دسته‌بندی کرد. یکی از انواع مکانیزم‌های کنترل دسترسی، رمزنگاری[۷] است. تغییر اطلاعات محرمانه به‌طوری‌که برای مهاجم[۸] قابل فهم نباشد، از اهداف روش‌های گوناگون رمزنگاری است. در بُعد محرمانگی، هدف از کنترل جریان اطلاعات کسب اطمینان از عدم وجود جریان‌های اطلاعاتی ناامن از اطلاعات محرمانه به اطلاعات دیگر است. به تعبیر دیگر، وجود جریان اطلاعات از بالا به پایین مجاز محسوب نمی‌شود.

از جهت صحت؛ یعنی اطمینان از صحیح‌بودن اطلاعات و عدم دستکاری غیرمجاز آن‌ها، هدف از کنترل جریان اطلاعات کسب اطمینان از عدم وجود جریان اطلاعات ناامن از اطلاعات غیرقابل اعتماد به اطلاعات قابل اعتماد است. به بیان دیگر، نباید جریان اطلاعات از پایین به بالا باشد.

تعریف جریان اطلاعات امن در یک سامانه اطلاعاتی توسط خط‌مشی جریان اطلاعات صورت می‌پذیرد. در اکثر این‌گونه خط‌مشی‌ها، از قالب خط‌مشی عدم‌تداخل[۹] و گزاره‌هایی روی اجراهای برنامه برای بیان خط‌مشی استفاده می‌شود. سپس با بهره‌گیری از مکانیزم‌های اِعمال خط‌مشی‌های امنیتی، می‌توان از چگونگی برقراری خط‌مشی‌های جریان اطلاعات مطمئن شد. مکانیزم‌های اعمال خط‌مشی‌های

---

[1] Confidentiality

[2] Integrity

[3] Availability

[4] Information Flow Control

[5] Mandatory Access Control (MAC)

[6] Discretionary Access Control (DAC)

[7] Cryptography

[8] Attacker

[9] Noninterference

امنیتی به دو دسته کلی ایستا[10] و پویا[11] تقسیم می‌شوند. در مکانیزم‌های ایستا، قبل از اجرا کد منبع[12] برنامه تحلیل و در خصوص برقراری یا عدم‌برقراری خط‌مشی مورد نظر تصمیم‌گیری می‌شود. این‌گونه 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. حال با داشتن اجراهای مختلف برنامه می‌توان خط‌مشی جریان اطلاعات را اعمال کرد. بدیهی است که به دلیل عدم دسترسی به اطلاعات زمان اجرای برنامه‌ها، مکانیزم‌ها به طور محافظه‌کارانه عمل می‌کنند. به این معنا که این‌گونه مکانیزم‌ها تنها به برنامه‌هایی که امن تشخیص بدهد، اجازه اجرا می‌دهد و ممکن است بعضی از برنامه‌های امن نیز توسط این مکانیزم‌ها، به درستی تشخیص داده نشده و به مرحله اجرا نرسند. از این رو، میزان کامل‌بودن[13] مکانیزم‌های ایستا در کم‌ترین مقدار ممکن است. بیشتر مکانیزم‌های تحلیل ایستا نوع‌مبنا[14] هستند. در این مکانیزم‌ها، با طراحی یک نوع‌سامانه[15] به دنبال تضمین برقراری خط‌مشی جریان اطلاعات انجام می‌شود. نوع دیگری از مکانیزم‌های تحلیل ایستا بر پایه گراف‌های وابستگی کنترل[16] هستند.

دسته دیگر مکانیزم‌های اعمال، مکانیزم‌های پویا هستند. برخلاف مکانیزم‌های ایستا، این مکانیزم‌ها در زمان اجرای برنامه، با نظارت بر اجرای آن و با مشاهده رویداد[17]های امنیتی برنامه، خط‌مشی امنیتی مورد نظر را اعمال می‌کنند. از آن‌جایی که ناظرهای زمان اجرا تنها به یک اجرا از برنامه دسترسی دارند، به نظر می‌رسد که این مکانیزم‌ها برای اعمال خط‌مشی‌های جریان اطلاعات مناسب نباشند. زیرا خط‌مشی‌های جریان اطلاعات به صورت گزاره‌هایی بر روی بیش از یک اجرا تعریف می‌شوند [۱]. 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 تا

[10] Static

[11] Dynamic

[12] Source Code

[13] Completeness

[14] Type-based

[15] Type System

[16] Control Dependency Graph

[17] Event

حد مکانیزم‌های تحلیل ایستا برای خط‌مشی‌های جریان اطلاعات فراهم می‌شود. مزیت دیگر این مکانیزم‌ها، پذیرفتن برنامه‌هایی است که گرچه 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 کامل‌بودن 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.

فصل دوم

خط‌مشی امنیتی؛ خاصیت و فوق‌خاصیت

خط‌مشی امنیتی؛ خاصیت و فوق‌خاصیت

در این فصل به تعریف مفاهیم اولیه مورد استفاده در فصل‌های بعدی پرداخته می‌شود.

۲-۱ تعریف خاصیت و فوق‌خاصیت امنیتی

خط‌مشی امنیتی تعریفی از امن‌بودن یک سامانه یا برنامه را ارائه می‌دهد که رفتارهای مجاز و غیرمجاز در آن مشخص می‌شود. خط مشی امنیتی، قیودِ روی توابع و جریان‌های بین آن‌ها را مشخص می‌کند؛ مثل قیود دسترسی بر روی برنامه‌ها و سطوح دسترسیِ داده‌های بین کاربران که مانع از بروز مشکلات امنیتی از طریق سامانه‌های خارجی و مهاجمان شود.

یک سامانه شامل مجموعه‌ای از اجراها است. به بیان صوری، خط‌مشی امنیتی گزاره‌ای روی مجموعه‌های اجراها است. به عبارت دیگر، مجموعه اجراهای $\Sigma \subseteq X^{\infty}$ خط‌مشی امنیتی P را برآورده می‌کند اگر و تنها اگر $P(\Sigma)$. طبق تعریف ارائه‌شده در [۲]، یک خط‌مشی امنیتی P را یک خاصیت[۱۸] امنیتی می‌نامیم اگر برای آن یک مسند مشخصه $\hat{P}$ روی X^{∞} وجود داشته باشد، به طوری که برای همه $\Sigma \subseteq X^{\infty}$، عبارت زیر برقرار باشد:

$$P(\Sigma) \Leftrightarrow \forall X \in \Sigma . \hat{P}(\Sigma)$$

طبق تعریف بالا، یک خاصیت امنیتی روی یک اجرا تعریف می‌شود، این در حالی است که خط‌مشی امنیتی می‌تواند گزاره‌ای روی اجراهای مختلف یک برنامه تعریف شود. به عنوان مثال، خط‌مشی عدم‌خاتمه[۱۹] یا خط‌مشی‌های کنترل دسترسی خاصیت هستند. زیرا می‌توان آن‌ها را در قالب تعریف بالا بیان کرد. اما خط‌مشی عدم تداخل، که از خط‌مشی‌های محرمانگی مهم برای امنیت جریان اطلاعات است، خاصیت نیست. 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

[18] Property

[19] Non-Termination

متفاوتی نداشته باشند. این خط‌مشی را نمی‌توان در قالب تعریف مطرح‌شده بیان کرد؛ یعنی با گزاره‌ای روی یک اجرا از برنامه قابل بیان نیست و روی دو اجرا از برنامه تعریف می‌شود.

پس می‌توان یک خط‌مشی امنیتی را به عنوان زیرمجموعه‌ای از مجموعه توانی همه اجراها تعریف کرد که هر اجرا دنباله‌ای دلخواه از حالت[۲۰]ها است. همچنین، می‌توان آن را به عنوان مجموعه برنامه‌هایی در نظر گرفت که آن خط‌مشی را برآورده می‌کنند. از این حیث می‌توان خط‌مشی‌های امنیتی را به دو دسته خاصیت و فوق‌خاصیت[۲۱] تقسیم‌بندی کرد. بعضی از خط مشی‌های امنیتی از آن‌جا که قابل دسته‌بندی و تشخیص توسط مجموعه اجراهای جداگانه هستند، خاصیت نامیده می‌شوند؛ یعنی می‌توان آن‌ها را به صورت مجموعه‌ای از اجراها بیان کرد. اما فوق‌خاصیت‌ها را باید با مجموعه توانیِ مجموعه اجراها بیان کرد. به همین دلیل، روش اعمال خاصیت‌ها با نحوه اعمال فوق‌خاصیت‌ها متفاوت است. لازم به ذکر است که بسیاری از خواسته‌های امنیتی مهم، مانند عدم تداخل، خاصیت نیستند. پس توجه به نحوه اعمال آن‌ها دارای اهمیت است.

۲-۲ تعریف مکانیزم اعمال

مکانیزم اِعمال یا مکانیزم امنیتی عبارتست از روش، ابزار یا رویه‌ای برای اعمال خط‌مشی امنیتی [۳]. مکانیزم روش‌های نرم‌افزاری و یا سخت‌افزاری تعریف می‌کند که کنترل‌هایی که توسط خط‌مشی تحمیل شده و به قالب صوری در مدل بیان می‌گردد را پیاده‌سازی کند.

مکانیزم‌های اعمال به دنبال دست‌یابی به این اهداف برای خط‌مشی‌های جریان اطلاعات هستند [۴]: ۱) درستی[۲۲]: اجازه وقوع جریان غیرمجاز اطلاعات در طول اجرا داده نشود. ۲) دقت[۲۳]: از اجرای امن برنامه‌ها جلوگیری نشود. ۳) عملی‌بودن[۲۴]: هزینه اعمال مکانیزم قابل قبول باشد. هزینه‌ها ممکن است در

[20] State

[21] Hyperproperty

[22] Soundness

[23] Precision

[24] Practicality

زمان توسعه[۲۵]، استقرار[۲۶] یا اجرای برنامه باشد. گرچه تلاش‌های بسیاری در دهه‌های اخیر برای پاسخ به این مسئله شده است، اما کماکان مکانیزم‌های اعمالی که به طور همزمان به همه این اهداف دست‌یابند، مطرح نشده است.

۳-۲ کنترل جریان اطلاعات

خط‌مشی‌های جریان اطلاعات، خط‌مشی‌های محرمانگی و صحت هستند که انتشار داده‌ها را در برنامه کنترل می‌کنند. هدف از کنترل جریان اطلاعات کسب اطمینان از عدم وجود جریان‌های اطلاعاتی ناامن از اطلاعات محرمانه به اطلاعات دیگر است. اولین تعریف صوری از جریان اطلاعات امن که برای بیشتر مکانیزم‌های اعمال خط‌مشی‌های جریان اطلاعات مورد استفاده قرار می‌گیرد، تعریف ارائه‌شده در [۵] است. در این تعریف، مفهوم وابستگی قوی[۲۷] بین ورودی i و خروجی o چنین بیان می‌شود که یک جریان اطلاعات از ورودی i به خروجی o در پردازه[۲۸] p وجود دارد هرگاه با اجرای p، تغییر در i به o منقل شود. بیان دیگر تعریف وابستگی قوی در سامانه‌های قطعی[۲۹] به این شکل است که بین ورودی i و خروجی o وابستگی قوی وجود دارد اگر و تنها اگر دو اجرا از p وجود داشته باشد که همه ورودی‌های آن‌ها به جز i با یکدیگر مشابه هستند ولی خروجی‌های آن‌ها در o با یکدیگر متفاوت است [۶].

تعریف جریان اطلاعات در قالب عدم تداخل در [۷] توجه بسیاری از پژوهشگران این حوزه را معطوف کرد، به طوری که در بیشتر کارهای مطرح‌شده در این زمینه، ارجاعی به این مقاله وجود دارد. این تعریف این‌گونه بیان می‌شود که:

یک گروه از کاربران با استفاده از مجموعه دستورات مشخص، با گروه دیگری از کاربران عدم تداخل دارند، اگر آن‌چه که آن کاربران با آن دستورات می‌توانند انجام دهند، هیچ اثری روی آن‌چه

[25] Development

[26] Deplyment

[27] Strong Dependency

[28] Process

[29] Deterministic

کاربران گروه دیگر می‌بینند، نداشته باشد.

و به بیان صوری، دستورات موجود در مجموعه A که از طرف کاربران گروه G صادر می‌شوند، با کاربران در گروه G' تداخل دارند اگر هر توالی از دستوراتی که به سامانه وارد می‌شود، روی کاربران G' اثری مشابه با همان توالی از دستورات داشته باشند که در آن، دستورات ارسال‌شده از کاربران G حذف[۳۰] شده باشد.

بیان دیگر عدم تداخل به زبان دنباله اجرا[۳۱] نیز می‌توان چنین داشت:

اگر در یک دنباله اجرای قابل قبول – دنباله‌ای که تولید آن توسط سامانه امکان‌پذیر باشد- که به صورت دنباله‌ای از ورودی‌ها و خروجی‌های سطح بالا و سطح پایین است، با حذف دستورهای ورودی سطح بالا، مجدداً دنباله اجرایی قابل قبول حاصل شود، عدم تداخل بین کاربران سطح بالا و سطح پایین برقرار خواهد بود.

به سادگی قابل مشاهده است که چنین تعبیری از امنیت، بسیار سخت‌گیرانه است [۸]. زیرا باید در یک دنباله اجرای قابل قبول، درج چند ورودی سطح بالا به هر ترتیبی، قابل قبول باشد. طبق این تعبیر، رمزنگاری ایده‌آل نیز عدم تداخل را برآورده نمی‌کند. در [۹] بیان آسان‌گیرتری از عدم تداخل ارائه شد که با نام عدم قابلیت استنتاج[۳۲] شناخته می‌شود.

برای هر دو دنباله اجرای T و S، یک دنباله اجرای قابل قبول R وجود دارد که شامل رویدادهای سطح پایین از T، با همان ترتیب، و رویدادهای ورودی سطح بالا از S، با همان ترتیب، و رویدادهای دیگر که نه از رویدادهای سطح پایینِ از T و نه از رویدادهای ورودی سطح بالا از S هستند.

در [۱۰]، عدم تداخل برای برنامه‌ها با ذکر عنوان *امنیت مبتنی بر معناشناخت*[۳۳] مطرح شده است. در این تعریف آمده است که عدم تداخل در برنامه‌ها لزوماً به این معناست که تغییر در ورودی‌های محرمانه سطح بالا باعث تغییر در خروجی‌های عمومی سطح پایین نشود.

---

[30] Purge

[31] Trace

[32] Nondeducibility

[33] Semantic-based Security

از طرفی، می‌توان جریان‌های اطلاعات را به دو نوع جریان‌های صریح[۳۴] و جریان‌های ضمنی[۳۵] دسته‌بندی کرد. به جریان حاصل از انتساب[۳۶] متغیری به متغیر دیگر، جریان صریح گفته می‌شود. جریان ضمنی، جریانی است که اطلاعات را از طریق ساختار کنترلی برنامه منتقل می‌کند. به عنوان مثال، در صورتی‌که شرط یک عبارت شرطی مانند *if* در زبان‌های رایج برنامه‌نویسی، به یک متغیر سطح بالای برنامه وابستگی داشته باشد، پس اجرا یا عدم اجرای آن شرط به یک مقدار سطح بالا وابسته خواهد بود. در واقع، طبق تعریف موجود در [۱۱]، که در آن برای اولین بار از اصطلاح *امنیت جریان اطلاعات* استفاده شده است، یک جریان ضمنی به متغیر b به عنوان نتیجه اجراشدن یا عدم اجرای یک گزاره است که در آن گزاره یک جریان صریح به b وجود دارد و آن گزاره به یک عبارت مشروط شده است. به عنوان نمونه، برنامه *if a:=0 then b:=c;* دارای یک جریان ضمنی از a به b است.

لازم به ذکر است که در کنترل جریان اطلاعات این‌طور فرض می‌شود که مهاجم به کد منبع برنامه دسترسی دارد و می‌تواند ورودی‌های عمومی برنامه را مدیریت کرده و خروجی‌های عمومی برنامه را نیز مشاهده کند [۱۲]. گرچه ممکن است در بعضی از کارهای مطرح‌شده، توانایی مهاجم بیشتر از موارد ذکرشده نیز باشد، اما به طور پیش‌فرض می‌توان چنین توانایی‌هایی را در مدل مهاجم تصور کرد.

نحوه بیان خط‌مشی‌های امنیتی جریان اطلاعات نیز می‌تواند متفاوت باشد. خط‌مشی‌های جریان اطلاعات معمولاً در قالب عدم تداخل بیان می‌شوند. جریان اطلاعات امن در برنامه‌ها با استفاده از مدل مشبکه[۳۷] مدل می‌شود که آن را *مشبکه سطوح امنیتی* می‌نامند [۱۱]. به طور معمول از مشبکه $(\mathcal{L}.\sqsubseteq)$ استفاده می‌شود که در آن $\mathcal{L} = \{low.high\}$ مجموعه سطوح امنیتی و $\sqsubseteq$ رابطه ترتیب جزئی[۳۸] روی سطوح امنیتی موجود در $\mathcal{L}$ است. عنصر کمین در مشبکه low و عنصر بیشین $high$ است که رابطه $low \sqsubseteq high$ بین آن‌ها برقرار است.

---

[34] Explicit

[35] Implicit

[36] Assignment

[37] Lattice Model

[38] Patial Order

از نکات حائز اهمیت در خط‌مشی امنیتی، توانایی مهاجم در تشخیص عدم خاتمه برنامه از طولانی‌شدن محاسبات برنامه است. به این ترتیب که آیا مهاجم می‌تواند واگراشدن یک برنامه را از زمان‌بربودن محاسبات لازم برای نمایش نتیجه تشخیص دهد. این توانایی می‌تواند باعث ایجاد کانال نهان[۳۹] باشد. به همین شکل می‌توان خط‌مشی‌های امنیتی را به دو گونه حساس به خاتمه[۴۰] و غیر حساس به خاتمه[۴۱] دسته‌بندی کرد.

همچنین، دو نوع تحلیل در این حوزه قابل بحث است. تحلیل غیر حساس به جریان[۴۲] به تحلیلی گفته می‌شود که در آن سطح امنیتی متغیرها بدون توجه به جریان برنامه مشخص می‌شود؛ یعنی سطح امنیتی متغیرها تا پایان اجرای برنامه ثابت می‌ماند. اما در تحلیل حساس به جریان[۴۳]، سطح امنیتی متغیرها با توجه به جریان برنامه تعیین می‌شود. در این نوع تحلیل، متغیرها می‌توانند سطوح امنیتی متفاوتی داشته باشند و با توجه به جریان برنامه، ارتقا[۴۴] و یا تنزل[۴۵] یابند. در [۱۳] بیان شده است که تحلیل ایستای حساس به جریان، تعمیمی از تحلیل ایستای غیر حساس به جریان است.

[39] Covert Channel

[40] Termination-Sensitive

[41] Termination-Insensitive

[42] Flow Insenitive Analysis

[43] Flow Sensitive Analysis

[44] Upgrade

[45] Downgrade

فصل سوم

روش‌های اِعمال خط‌مشی امنیتی

روش‌های اعمال خط‌مشی امنیتی

یکی از معیارهای دسته‌بندی انواع مختلف روش‌های اعمال خط‌مشی‌های امنیتی، زمانِ اعمال‌شدن خط‌مشی است و این‌که یک مکانیزم اعمال از چه نوع اطلاعاتی برای اعمال می‌تواند استفاده کند. زمان اعمال را می‌توان به دو بخش قبل از اجرای برنامه یا زمان کامپایل[۴۶] و در طول اجرای برنامه یا زمان اجرا[۴۷] تقسیم‌بندی کرد. البته می‌تواند روشی وجود داشته باشد که در هر دو زمان مکانیزم مطرح می‌شود. در زمان کامپایل یا قبل از اجرای برنامه، نمی‌توان به اطلاعات زمان اجرای برنامه دسترسی داشت. پس مکانیزم تنها می‌تواند با استفاده از اطلاعات حاصل از تحلیل کد منبع برنامه اقدام کند. در حالی‌که در طول اجرای برنامه، علاوه بر اطلاعات زمان اجرا ممکن است دسترسی به همه یا بخشی از کد منبع نیز وجود داشته باشد.

۳-۱ دسته‌بندی روش‌های اعمال

در [۱۴]، با استفاده از مدل‌سازی برنامه به صورت ماشین برنامه[۴۸] (PM) به دسته‌بندی مکانیزم‌های اعمال خط‌مشی‌های امنیتی پرداخته می‌شود. حال می‌توان روش‌های اعمال را به دسته‌های تحلیل ایستا[۴۹]، تحلیل پویا[۵۰]، بازنویسی برنامه[۵۱] و تحلیل ترکیبی[۵۲] تقسیم کرد. ابتدا به طور خلاصه و براساس مدل مطرح‌شده، به هر یک پرداخته می‌شود و در ادامه این گزارش، به طور مفصل درباره آن‌ها بحث خواهد شد.

---

[46] Compile Time

[47] Run Time

[48] Program Machine

[49] Static Analysis

[50] Dynamic Analysis

[51] Program Rewriting

[52] Hybrid Analysis

۳-۱-۱ مکانیزم‌های ایستا

مکانیزم‌های اعمالی که برنامه‌های غیرمطمئن را پیش از اجرای آن‌ها رد یا قبول می‌کنند، تحلیل ایستا نامیده می‌شوند. در این دسته از روش‌ها، مکانیزم اعمال باید در زمان متناهی برقراری خط‌مشی مورد نظر توسط برنامه را تشخیص دهد. به این ترتیب، برنامه‌های پذیرفته‌شده می‌توانند اجرا شوند ولی برنامه‌هایی که مکانیزم آن‌ها را امن تشخیص نداده است، اجازه اجرا پیدا نمی‌کنند. به بیان صوری، در مدل ارائه‌شده در [۱۴]، خط‌مشی امنیتی P قابل اعمال توسط مکانیزم تحلیل ایستا است اگر و تنها اگر ماشین تورینگ M_p وجود داشته باشد که ماشین برنامه PM، M را به عنوان ورودی دریافت کند و در صورت برآورده‌شدن $P(M)$، برنامه را در زمان متناهی بپذیرد، و در غیر این صورت آن را در زمان متناهی رد کند. این مکانیزم‌ها می‌توانند تمامی خاصیت‌های تصمیم‌پذیر[۵۳] 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. پس طبق مطالب بالا، با مکانیزم‌های تحلیل ایستا قابل اعمال نخواهد بود. اما می‌توان تقریبی از این خط‌مشی به صورت «برنامه حداکثر پس از هزار عمل محاسباتی خاتمه خواهد یافت»، که یک خاصیت تصمیم‌پذیر است، را در نظر گرفت. حال این خاصیت تصمیم‌پذیر را می‌توان با بهره‌گیری از روش‌های تحلیل ایستا اعمال کرد.

از جمله روش‌های تحلیل ایستا می‌توان به مکانیزم‌های نوع‌مبنا [۱۵] و مکانیزم‌های مبتنی بر راستی‌آزمایی[۵۴] [۱۶] اشاره کرد. این‌گونه مکانیزم‌ها درست هستند و هزینه‌ای در زمان اجرا یا استقرار تحمیل نمی‌کنند. با این حال مکانیزم‌های نوع‌مبنا دقیق نیستند و ممکن است برنامه‌های امن زیادی توسط آن‌ها پذیرفته نشود و محافظه‌کار هستند. البته این دسته از روش‌های اعمال هزینه زمان توسعه زیادی خواهند داشت. انواع دیگر این روش‌ها عبارتند از تحلیل جریان داده، وارسی مدل و تفسیر انتزاعی که شرح آن‌ها در خارج از حوزه این گزارش است.

---

[53] Decidable

[54] Verification-based

۳-۱-۲ مکانیزم‌های پویا

قسمت مهمی از روش‌های این دسته، روش‌های نظارت بر اجرا[۵۵] [۲]، [۱۷] است. 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 آن است که با اجرای هر مرحله از برنامه، یک انتزاعی از اجرای آن به صورت یک رویداد برای ناظر ارسال می‌شود و ناظر با بررسی این رویداد در صورت عدم مغایرت آن با خط‌مشی مورد نظر، آن را می‌پذیرد و برای اجرا ارسال می‌کند. گرچه روش‌های دیگری مانند اجرای چنداجرایی امن [۱۸] نیز اخیراً مطرح شده است.

مکانیزم‌های نظارت بر اجرا را می‌توان در انواع مختلفی از جمله انتقال‌دهنده اجرا[۵۶] و تشخیص‌دهنده اجرا[۵۷] دسته‌بندی کرد [۱۹]. 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، به اجرای برنامه خاتمه می‌دهند. 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 نظارت بر اجرای از نوع انتقال‌دهنده اجرا، اجرای برنامه را به عنوان ورودی می‌گیرند، روی آن نظارت می‌کنند و در صورت نیاز و طبق خط‌مشی، تغییری به صورت جلوگیری از انتقال یک رویداد یا درج و حذف یک رویداد در اجرای فعلی ایجاد می‌کنند تا خط‌مشی مورد نظر اعمال شود.

مدل ماشین‌برنامه (PM) قادر است تا کلیه مکانیزم‌های نظارت بر اجرا، شامل هسته‌های امنیتی[۵۸]، ناظر مرجع[۵۹] (RM) [۲۰]، ناظر مرجع برخط[۶۰] [۲۱] و سایر مکانیزم‌های اعمال خط‌مشی

---

[55] Execution Monitoring

[56] Execution Transformer

[57] Execution Recognizer

[58] Security Kernel

[59] Reference Monitor

[60] Inlined Reference Monitor

امنیتی مبتنی بر سیستم‌عامل را مدل‌سازی کند.

مکانیزم‌های تحلیل پویا می‌توانند بر اساس اطلاعات قابل دسترس آن‌ها برای تحلیل برنامه مورد مطالعه قرار گیرند. در تعدادی از این مکانیزم‌ها، دسترسی به کد منبع برنامه وجود ندارد، حال آن‌که در برخی دیگر، دسترسی به طور کامل وجود دارد. حتی ممکن است ناظر صرفاً در شرایطی به کد منبع برنامه دسترسی داشته باشد. پس از این حیث نیز می‌توان دسته‌بندی دیگری برای مکانیزم‌های تحلیل پویا در نظر داشت. از منظر اطلاعات در دسترس و زمان دسترسی، روش‌های پویا را می‌توان به سه گونه روش‌های *تحلیل پویای محض*[۶۱]، *تحلیل پویای ترکیبی به صورت تحلیل کد در ابتدا* و *تحلیل پویای ترکیبی به صورت تحلیل کد در طول اجرا* دسته‌بندی کرد. در دسته اول؛ یعنی 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 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 اجرا می‌تواند به بخش‌های مختلف کد منبع برنامه دسترسی پیدا کند. از جهت شباهت دو دسته آخر، می‌توان به طور کلی به آن‌ها روش‌های *تحلیل پویای ترکیبی* نیز گفت.

همان‌طور که قبلاً گفته شد، در روش تحلیل پویای محض، مکانیزم اعمال هیچ‌گونه اطلاعاتی از کد منبع برنامه در اختیار ندارد. پس در رویدادها اطلاعاتی از کد منبع وجود ندارد. ناظرهای مرجع و ناظرهای مرجع برخط در این دسته از مکانیزم‌های تحلیل پویا قرار می‌گیرند. زیرا در آن‌ها مکانیزم اعمال فقط ناظر به رویدادهایی است که از برنامه دریافت می‌کند که در آن‌ها نیازی به تحلیل کد منبع برنامه برای تحلیل و پیش‌بینی رفتار آینده اجرای برنامه نیست.

در دسته دوم، مکانیزم‌ها پیش از اجرای برنامه کد منبع برنامه را در دسترس دارند. پس می‌توانند تحلیل کد منبع را در همان مرحله انجام دهند. به این ترتیب، می‌توان اطلاعات مورد نیاز برای تصمیم‌گیری در زمان اجرا را در این گام از تحلیل کد منبع استخراج کرد و در طول اجرای برنامه، در هنگام نظارت بر اجرای برنامه، از آن‌ها استفاده کرد. پس طبیعتاً تحلیل و اعمال دقیق‌تری نسبت به روش‌های دسته اول خواهیم داشت.

---

[61] Pure Dynamic Analysis

در دسته سوم روش‌ها، اطلاعات همزمان با تولید رویدادها در اختیار ناظر قرار می‌گیرد. پس لازم است که بعضی از رویدادهای موجود در اجرا، از رویدادهای مربوط به کد منبع برنامه باشد تا ناظر بتواند بر اساس مدل ذکرشده، اعمال را انجام دهد.

۳-۱-۳بازنویسی برنامه

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 غیرمطمئن قبل از اجرا توسط مکانیزم اعمال، با هدف اعمال خط‌مشی امنیتی مورد نظر، دچار تغییر خواهد شد و کد منبع برنامه، با توجه به الگوریتم بازنویسی، اصلاح می‌شود. الگوریتم بازنویسی برنامه باید این تضمین را ارائه کند که برنامه بازنویسی‌شده، از نظر معناشناخت[۶۲]، هم‌ارز برنامه ورودی اصلی است. پس 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 که:

(۱) $P(R(M))$ برقرار باشد.

(۲) $P(M) \Rightarrow M \approx R(M)$

از دیدگاهی می‌توان بازنویسی برنامه را به صورت تعمیمی از روش‌های نظارت بر اجرا دانست. در صورتی‌که مداخله ناظر زمان اجرا به صورت تغییر 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 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 [۲۱] را می‌توان یک بازنویس برنامه دانست. زیرا در ناظر مرجع برخط فرض شده است که از یک تغییردهنده برنامه استفاده می‌شود. این تغییردهنده برنامه، کدهایی را به منظور اطمینان از برقراری

[62] Semantics

خط‌مشی و اعمال آن در لابه‌لای دستورات برنامه اصلی درج می‌کند. پس مداخله این ناظر اجرا، نوعی بازنویسی برنامه محسوب می‌شود.

۳-۱-۴ تحلیل ترکیبی

در واقع، این روش‌ها ترکیبی از روش‌های تحلیل ایستا و پویا هستند. به این شکل که 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 اجراشدن آن داده نشود. در بعضی موارد احتمال دارد که با اعمال تغییراتی در برنامه، برنامه را برای تحلیل پویا به مرحله بعدی بفرستد.

(۲) مرحله تحلیل پویا: در صورتی که برنامه به این مرحله برسد؛ یعنی توسط تحلیل ایستا جلوی اجرا آن گرفته نشود، به اجرای برنامه نظارت می‌شود و خط‌مشی مورد نظر اعمال خواهد شد. در این مرحله می‌تواند روش‌های مختلف تحلیل پویای مطرح‌شده در قبل وجود داشته باشد.

۳-۲ خط‌مشی‌های قابل اعمال توسط مکانیزم‌های نظارت بر اجرا

مکانیزم نظارت اجرای مورد نظر در [۱۴]، از جمله ناظرهای اجرای سنتی[۶۳] محسوب می‌شود. زیرا فرض بر این است که در هنگام نظارت، در صورت مشاهده رویداد ناقض خط‌مشی، به اجرای برنامه خاتمه می‌دهد [۲]. گرچه در تعریف کلی این‌گونه روش‌ها، لفظ *مداخله* مطرح شده است که می‌تواند با توجه به توانایی‌های ناظر اجرا، برخوردهای متفاوتی در صورت مشاهده نقض خط‌مشی داشته باشد. در [۲]، ناظر را به صورت یک ماشین[۶۴] قطعی مدل کرده است. سپس مطرح می‌شود که به ازای هر

[63] Traditional Execution Monitor

[64] Automaton

خط‌مشی امنیتی P قابل اعمال توسط ناظر، گزاره‌ای به نام $\hat{P}$ وجود دارد که سه شرط زیر برای آن برقرار است:

$(EM1) \quad P(M) \equiv \forall X.\, \hat{P}(X)$

$(EM2) \quad \hat{P}(X) \Rightarrow (\forall i.\, 1 \leq i \leq |X|.\, \hat{P}(X[..i]))$

$(EM3) \quad \neg\hat{P}(X) \Rightarrow (\exists i.\, 1 \leq i\, .\, \neg\, \hat{P}(X[..i]))$

منظور از شرط اول آن است که هر گاه خط‌مشی برقرار است، آنگاه به ازای هر اجرای برنامه، مسند متناظر آن خط‌مشی برای یک اجرا یا همان $\hat{P}$ برقرار باشد و برعکس. همان‌طور که قبلاً مطرح شد، به این چنین خط‌مشی‌ها، خاصیت گفته می‌شود.

شرط دوم بیانگر آن است که هرگاه $\hat{P}$ برای یک اجرا از برنامه برقرار بود، آنگاه $\hat{P}$ برای هر پیشوند از آن دنباله اجرا نیز برقرار باشد. پس خط‌مشی باید از حیث پیشوند اجراها، بسته[۶۵] باشد. به همین دلیل ناظر نمی‌تواند درباره خط‌مشی‌های درمان‌پذیر[۶۶] قضاوت کند.

در شرط سوم این چنین بیان می‌شود که اگر $\hat{P}$ برای یک اجرا برقرار نبود، یک پیشوند اجرای متناهی از آن وجود دارد که $\hat{P}$ در آن اجرا نقض می‌شود. پس مطمئن خواهیم بود که در زمان متناهی درباره نقض خط‌مشی تصمیم گرفته خواهد شد.

اگر خط‌مشی‌ای هر سه شرط بالا را دارا باشد، آن را *خاصیت ایمنی*[۶۷] نیز می‌نامند.

البته در [۲۲] اشاره شده که علاوه بر شرط‌های سه‌گانه فوق، برای اعمال خط‌مشی توسط ناظر، مسند $\hat{P}$ باید تصمیم‌پذیر باشد. به همین دلیل شرط دیگری برای این‌گونه خط‌مشی‌ها اضافه شد:

$(EM4) \quad \hat{P}\ is\ recursively\ decidable\ whenever\ X\ is\ finite$

و اثبات شد که کلاس خط‌مشی‌های تعیین‌شده توسط شرط‌های EM بالا هم‌ارز است با کلاس مکمل خاصیت‌های شمارش‌پذیر بازگشتی[۶۸] ($coRE$). می‌توان خط‌مشی امنیتی P را $coRE$ به حساب آورد اگر

---

[65] Prefix-closed

[66] Remediable

[67] Safety Property

[68] Co-Recursively Enumerable

یک ماشین تورینگ M_p وجود داشته باشد که برنامه M را به عنوان ورودی دریافت و در صورت برقراری $\neg P(M)$، برنامه را در زمان متناهی رد کند. در غیر این صورت، همواره در حلقه بماند. برای دقیق‌ترشدن تعاریف فوق، در [۱۹] برقراری شرط $\hat{P}(.)$ نیز عنوان شد. به این معنا که $\hat{P}$ باید برای اجرای تهی نیز برقرار باشد. در صورتی که خط‌مشی شرط اخیر را نیز دارا باشد، به آن *خاصیت ایمنی معقول*[۶۹] گفته می‌شود.

قابل توجه است که ناظرها سرباری به زمان اجرا می‌افزایند که باعث کاهش کارایی آن‌ها می‌شود. طبق [۲۳]، این روش‌ها بین ۵۰٪ تا ۱۰۰٪ بار به زمان اجرا اضافه می‌کنند.

۳-۳ مقایسه روش‌های اعمال خط‌مشی امنیتی

پیش‌تر تصور بر این بود که مکانیزم‌های پویا از آن جهت که می‌توانند فقط یک اجرا را تحلیل و بررسی نمایند و خط‌مشی‌های جریان اطلاعات به صورت گزاره‌ای روی بیش از یک اجرا تعریف می‌شود، نمی‌توانند به درستی این خط‌مشی‌ها را اعمال کنند 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 مزایای خاص خود را 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 با دقت بیشتری در مقایسه با تحلیل‌های ایستا می‌توانند خط‌مشی‌ها را اعمال کنند. دقت اعمال از دو منظر قابل بحث است. اول آن‌که قدرت مکانیزم‌های پویا در تشخیص برنامه‌هایی که گرچه خط‌مشی در آن‌ها برقرار است، اما تحلیل ایستا جلوی اجرا آن‌ها را می‌گیرد. به عنوان نمونه، ممکن است برنامه‌ای همه اجراهای آن از نظر خط‌مشی امن باشد اما به واسطه وجود یک عبارت شرطی که هیچ‌گاه اجرا نمی‌شود و وجود یک جریان ضمنی ناقض خط‌مشی، و البته محافظه‌کارانه‌بودن تحلیل ایستا، برنامه امن تشخیص داده نشود. حال

[69] Reasonable Safety Property

آن‌که همین برنامه در تحلیل پویا بدون مشکل امنیتی اجرا خواهد شد. در واقع، نقطه قوت روش‌های تحلیل پویا، تحلیل اجراها و رفتار برنامه یا به بیان دیگر، معناشناخت برنامه است، و نه صرف نحو[۷۰] و کد منبع آن. از منظر دوم، قدرت مکانیزم‌های پویا در امکان اجرای برنامه‌ها تا زمانی است که نشت اطلاعات ندارند. ممکن است در برنامه‌ای خط‌مشی جریان اطلاعات مورد نظر برقرار نباشد، اما اجراهای درستی داشته باشد که در آن هیچ‌گونه نشت اطلاعاتی وجود ندارد. پس مکانیزم پویا می‌تواند در این برنامه، اجرای درست را از اجرای نادرست تشخیص داده و تنها زمانی در اجرا مداخله کند که آن اجرا به نشت اطلاعات منجر می‌شود. اما در تحلیل ایستا در صورتی که برنامه تنها دارای یک اجرای نادرست باشد، کل برنامه امن شناخته نمی‌شود. بنابراین با بهره‌گیری از تحلیل پویا می‌توان تا زمانی که برنامه نشت اطلاعاتی ندارد، از آن استفاده کرد. از این رو، مکانیزم‌های پویا را آسان‌گیرتر از تحلیل ایستا می‌دانند [۲۴].

- 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 نیاز به برچسب‌گذاری پویا[۷۱] احساس می‌شود. در مکانیزم‌های تحلیل ایستای محض[۷۲] امکان سطح‌دهی پویا وجود ندارد [۲۵]. گرچه در مکانیزم‌های تحلیل ایستای دیگر می‌توان با بررسی‌های بیشتر در زمان اجرا، این قابلیت را فراهم کرد [۲۶]. همچنین، تعریف مشبکه سطوح امنیتی در مکانیزم‌های تحلیل ایستا باید حتما قبل از کامپایل‌شدن برنامه صورت گیرد. ولی در مکانیزم‌های تحلیل پویا می‌توان پیش از هر اجرای برنامه، مشبکه سطوح امنیتی را تعریف کرد. علاوه بر این، امکان سطح‌دهی پویا برای تعیین سطح 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 ایستای آن‌ها ممکن

[70] Syntax

[71] Dyn

[72] Pure Static Analysis

نیست. پس در این گونه زبان‌ها، تنها مکانیزم‌های تحلیل پویا پاسخ‌گو خواهند بود [۲۷]. نوع‌دهی پویا[۷۳] در بعضی از زبان‌های پویا وجود دارد. قواعد نوع‌دهی[۷۴] به صورتی طراحی می‌شوند که برنامه‌های خوش‌نوع[۷۵] ، خط‌مشی را برآورده می‌کنند. طبق دسته‌بندی مطرح‌شده، می‌توان روش نوع‌دهی پویا را در دسته مکانیزم‌های تحلیل پویا قرار داد.

علی‌رغم دارابودن مزایای فوق، مکانیزم‌های تحلیل پویا با توجه به این‌که همزمان با هر اجرای برنامه اعمال می‌شوند، برخلاف تحلیل ایستا، سرباری به اجرای برنامه اضافه می‌کنند که از کارایی آن‌ها کاسته می‌شود.

همچنین می‌توان مکانیزم‌های اعمال خط‌مشی امنیتی را بر اساس دو معیار *درستی* و *کامل‌بودن* مقایسه کرد. درستی مکانیزم اعمال به این مفهوم است که 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 طوری‌که در صورتی که مکانیزمی وجود داشته باشد که درستی را تضمین نکند، عملاً بی‌استفاده است. همان‌طور که پیش‌تر ذکر شد، مکانیزم‌های تحلیل پویا فقط زمانی می‌توانند خط‌مشی جریان اطلاعات را درست اعمال کنند که متن برنامه نیز در دسترس آن‌ها باشد. به این معنا که علاوه بر اطلاعات حاصل از اجرای برنامه، اطلاعاتی از تحلیل برنامه برای این کار نیاز است. در غیر این صورت، این مکانیزم‌ها قادر خواهند بود تا فقط از بروز جریان صریح جلوگیری کنند. برای تشخیص و جلوگیری از جریان ضمنی، حتما نیاز به دسترسی به کد منبع برنامه وجود دارد. حال آن‌که همین مورد؛ یعنی 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 امن تشخیص داده شود. تفاوت بین درستی و کامل‌بودن با مثالی به خوبی قابل

---

[73] Dynamic Typing

[74] Typing Rules

[75] Well-Typed

درک خواهد بود. فرض کنید که مکانیزم اعمال مورد استفاده همه برنامه‌ها را به 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 معناشناخت برنامه به طور کلی تغییر کرده است و هر برنامه امنی، توسط این مکانیزم ناامن شناخته شده و بازنویسی می‌شود. همین معیار است که باعث مقایسه برتری روش‌ها در برابر یکدیگر می‌شود. گرچه هنوز روشی ارائه نشده است که کامل‌بودن در آن اثبات شده باشد. زیرا در مکانیزم‌های تحلیل ایستا، محافظه‌کاربودن این گونه روش‌ها، محدودیت جدی‌ای به شمار می‌رود. پس می‌توان گفت هیچ‌گاه تحلیل ایستا نمی‌تواند کامل‌بودن را برای ما فراهم آورد. این مشکل در مکانیزم‌های پویا نیز وجود دارد [۶]. گرچه نسبت به تحلیل ایستا آسان‌گیرتر و دقیق‌تر است، اما کامل‌بودن به طور صددرصدی برای آن‌ها برقرار نیست. به عبارت دیگر، در اعمال خط‌مشی‌های جریان اطلاعات، مکانیزم‌های اعمال به جای اعمال این خط‌مشی‌ها، در عمل تقریبی از آن‌ها را اعمال می‌کنند که همین باعث کامل‌نبودن این مکانیزم‌ها برای خط‌مشی‌های جریان اطلاعات شده است.

فصل چهارم

مروری بر مکانیزم‌های ایستا

مروری بر مکانیزم‌های ایستا

یک خط‌مشی محرمانگی انتهابه‌انتها به دنبال این است که داده ورودی محرمانه توسط یک مهاجم از طریق مشاهداتش از خروجی سیستم قابل استنباط نباشد. به این گونه خط‌مشی‌ها، جریان اطلاعات گفته می‌شود[۱۰].

مکانیزم‌های امنیتی رایج مانند کنترل دسترسی و رمزنگاری به طور مستقیم به اعمال خط‌مشی‌های جریان اطلاعات نمی‌پردازند. روش استاندارد برای حفاظت از داده‌های محرمانه کنترل دسترسی است. اما در این مکانیزم تنها محدودیت‌هایی روی نحوه افشای اطلاعات گذاشته می‌شود، نه انتشار آن‌ها. به همین دلیل رویکردی برای این موضوع مطرح شده است که در آن از تکنیک‌های زبان‌های برنامه‌سازی برای توصیف و اعمال خط‌مشی‌های جریان اطلاعات استفاده می‌شود. در مقاله [۱۰] به مروری درباره تحلیل ایستای برنامه‌ها پرداخته می‌شود.

۴-۱ نوع‌سامانه‌های امنیتی

یکی از رویکردهای مورد استفاده برای اعمال خط‌مشی‌های جریان اطلاعات، استفاده از نوع‌سامانه‌ها برای جریان اطلاعات است. در یک زبان دارای نوع امنیتی، نوع‌های متغیرها و عبارات برنامه با نوع‌آرایی[۷۶]هایی مشخص می‌شوند که خط‌مشی‌های مورد استفاده در نوع داده را تعیین می‌کنند. این‌گونه خط‌مشی‌های امنیتی توسط وارسی نوع در زمان کامپایل اعمال می‌شوند و سرباری در زمان‌اجرا ندارند. مشابه وارسی‌های نوع عادی، وارسی نوع امنیتی نیز ذاتاً ترکیبی[۷۷] است. به این معنا که زیرسامانه‌های امن با یکدیگر ترکیب شده و تشکیل یک سامانه بزرگ امن را می‌دهند.

در تحلیل ایستا، کانال‌های نهان از اهمیت ویژه‌ای برخوردار هستند. انواع کانال‌های نهان عبارتند از جریان‌های ضمنی، کانال خاتمه، کانال زمانی، کانال احتمالاتی، کانال استفاده کامل منابع، کانال توان.

[76] Annotation

[77] Compositional

جدا از سربار محاسباتی و حافظه‌ای، ضعف دیگر مکانیزم‌های اعمال زمان‌اجرا در تشخیص جریان‌های اطلاعات ضمنی است. جریان‌های ضمنی ناشی از ساختار کنترلی برنامه است. اما جریان‌های صریح به دلیل انتساب مستقیم داده محرمانه به یک متغیر عمومی است.

اولین بار در [۲۸] مشاهده شد که می‌توان از تحلیل ایستای برنامه برای کنترل جریان اطلاعات استفاده کرد که دقت بیشتر و سربار زمان‌اجرای کمتری را به همراه دارد.

در رویکرد وارسی نوع، هر عبارت برنامه دارای یک نوع امنیتی شامل دو بخش است: یک نوع عادی مانند int، و یک برچسب که بیانگر این است که از این مقدار چگونه استفاده خواهد شد. این برچسب‌ها ایستا هستند و در زمان اجرای برنامه محاسبه نمی‌شوند. امنیت توسط وارسی نوع اعمال می‌شود. با استفاده از *برچسب شمارنده* برنامه، که وابستگی‌های شمارنده برنامه را دنبال می‌کند، می‌توان جریان‌های ضمنی را به درستی کنترل کرد.

محرمانگی یک خاصیت برای یک مسیر اجرای منفرد نیست، بلکه یک خاصیت از مجموعه‌ای از همه مسیرهای اجرا است. از معروف‌ترین خط‌مشی‌های امنیتی در حوزه محرمانگی، عدم تداخل است.

تعریف عدم تداخل - تغییری در یک ورودی محرمانه (سطح بالا) باعث ایجاد تغییر در خروجی عمومی (سطح پایین) نشود. به این ترتیب، برنامه C امن است، اگر و فقط اگر:

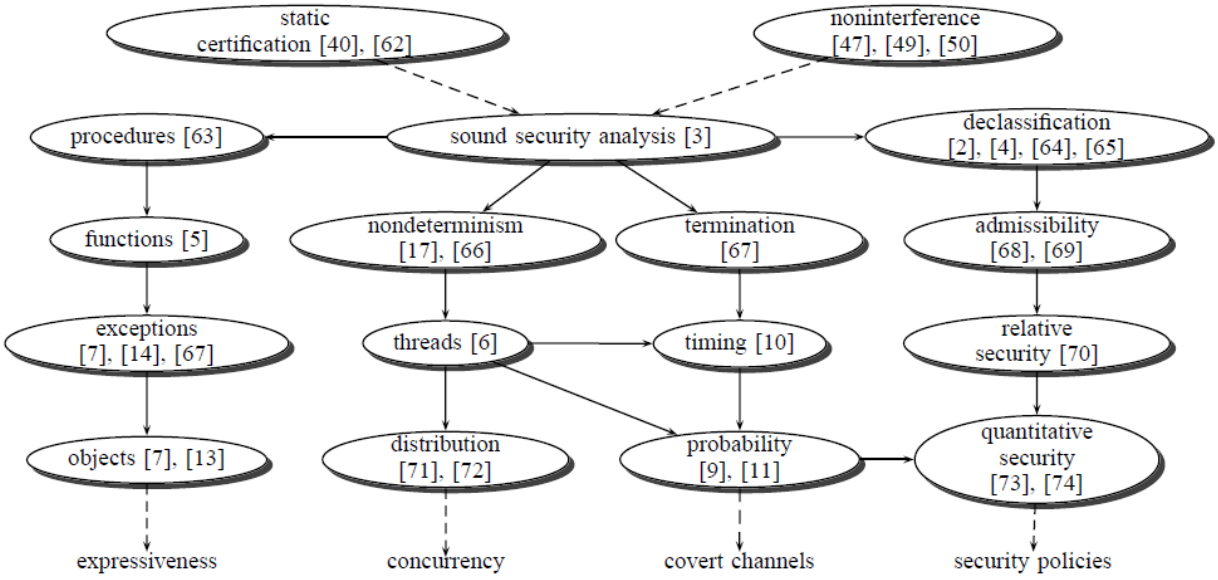
$$\forall s_1, s_2 \in S.\, s_1 =_L s_2 \Longrightarrow [\![C]\!]s_1 \approx_L [\![C]\!]s_2$$

یک نوع‌سامانه امنیتی مجموعه‌ای از قاعده‌های نوع‌دهی است که چه نوع امنیتی‌ای به یک برنامه منتسب شود.

از چهار جهت می‌توان پژوهش در امنیت زبان‌مبنا را در تلاقی عدم تداخل و تصدیق ایستا مورد بررسی قرار داد: (۱) غنی‌سازی بیانگری[۷۸] زبان برنامه‌سازی (۲) بررسی تأثیر هم‌روندی بر امنیت (۳) تحلیل کانال‌های نهان (۴) پالایش خط‌مشی‌های امنیتی.

[78] Expressiveness

عدم تداخل طبق تعریف اصلی انجام‌شده در گوگن-مسگر یک خاصیت در محاسبات قطعی است.



شکل ۱ – تکامل جریان اطلاعات زبان‌مبنا [۱۰]

یکی از تعابیر تعمیم‌یافته از عدم تداخل برای عدم قطعیت، در نظر گرفتن رفتار قابل مشاهده یک برنامه به عنوان مجموعه‌ای از نتایج ممکن است. به این معنا که ورودی‌های سطح بالا تأثیری روی مجموعه خروجی‌های سطح پایین ممکن نداشته باشند. این تعبیر به نام عدم تداخل امکان‌گرایانه[۷۹] معروف است.

یکی از پیچیدگی‌های مدل هم‌روندی این است که قسمت سطح بالا از حالت‌های برنامه باید در همه زمان محاسبات حفاظت شود.

[79] Possibilistic

۴-۲ تحلیل ایستا در برنامه‌های هم‌روند

Smith و Volpano [۲۹] عدم تداخل را برای یک زبان با قابلیت چندریسگی اثبات کرده‌اند. آن‌ها نشان دادند که علاوه بر شرط‌های لازم برای زبان‌های ترتیبی[۸۰]، دو خواسته دیگر کافی است تا عدم تداخل تحت یک زمان‌بند[۸۱] کاملاً غیرقطعی برآورده شود. این دو خواسته عبارتند از هیچ حلقه whileای نباید دارای عبارت شرطی سطح بالا باشد و هیچ ساختار شرطی سطح بالایی حاوی یک حلقه while در شاخه‌هایش نباشد.

گرچه در کارهای بعدی، Volpano و Smith [۳۰] امنیت را در حضور زمان‌بند یکنواخت بررسی کردند و امنیت حساس به احتمال را مورد تحقیق قرار دادند. ضمناً Sabelfeld و Sands [۳۱] درباره امنیت مستقل از زمان‌بند بحث کردند و تعبیری از عدم تداخل را برای یک نوع‌سامانه امنیتی اثبات کردند.

Zdancewic [۳۲] رویکرد دیگری را برای وارسی امنیت در سامانه‌های هم‌روند توسعه داد که بر مبنای ایده قطعیت دید پایین[۸۲] است. یک برنامه امن در نظر گرفته می‌شود اگر نتایج در رابطه از دید پایین $\approx_L$ قطعی باشد، هر چند ورودی‌های سطح بالا به طور غیرقطعی انتخاب شده باشند.

تکنیک‌های زبان‌مبنا در مدل‌سازی و تحلیل جریان اطلاعات در پروتکل‌های امنیتی نیز کاربرد دارند [۱۰].

گرچه عدم تداخل به عنوان یک تعبیر مصداقی[۸۳] مناسب است، اما محدودیت‌هایی ایجاد می‌کند که نمی‌توان همیشه در عمل از آن استفاده کرد. به طور مشخص، عدم تداخل اجازه تنزل[۸۴] از سطح امنیتی بالا به پایین را نمی‌دهد. در حالی‌که این‌گونه علنی‌سازی[۸۵] برای این‌که یک مقدار محرمانه

[80] Sequential
[81] Scheduler
[82] Low-view Determinisim
[83] Extensional
[84] Downgrading
[85] Declassification

رمزشده از یک وسیله قابل مشاهده برای عموم بگذرد، الزامی است. مثال دیگر یک برنامه وارسی گذرواژه است. واضح است که نتیجه این برنامه به گذرواژه محرمانه وابسته است.

نظارت زمان‌اجرا روی فراخوانی‌های سیستمی سیستم‌عامل کاربرد محدودی دارد. زیرا خط‌مشی‌های جریان‌اطلاعات خاصیتی از یک اجرای تکی نیست. به طور کلی لازم است روی همه مسیرهای ممکن اجرا نظارت انجام شود.

فصل پنجم

خاصیت‌های قابل‌اعمال توسط ناظرهای زمان‌اجرا

خاصیت‌های قابل‌اعمال توسط ناظرهای زمان‌اجرا

یکی از راه‌کارهای مورد استفاده برای امن‌سازی سامانه‌ها، نظارت زمان‌اجرا است. نظارت زمان‌اجرا رویکردی برای برآورده‌سازی ایمنی کد است که اجازه می‌دهد تا کد غیرمورد اعتماد اجرا شود در حالی‌که ناظر با مشاهده اجرا، در صورت بروز احتمالی نقض یک خط‌مشی، واکنش مناسبی برای جلوگیری از آن نشان می‌دهد. پژوهش‌هایی درباره ناظرهای زمان‌اجرا انجام شده است که سوال اصلی آن‌ها این است که دقیقاً کدام مجموعه از خاصیت‌ها قابل نظارت[۸۶] هستند. از این بابت که در شرایط مختلف و مجموعه محدویت‌های گوناگون، چه مجموعه‌ای از خاصیت‌ها را می‌توان اعمال کرد.

شایان ذکر است که مجموعه خاصیت‌های قابل اعمال توسط ناظرها به عوامل زیادی بستگی دارد. از جمله این عوامل می‌توان به اطلاعات در دسترس از رفتارهای ممکن برنامه هدف، ابزاری که ناظر به کمک آن به نقض احتمالی واکنش نشان می‌دهد، محدودیت‌های حافظه و محاسباتی و تعریف اعمال مورد استفاده‌شده اشاره کرد.

۵-۱ نمادگذاری و تعاریف اولیه

اولین گام برای تحلیل یک چارچوب نظری، بیان صریح و روشن مفاهیم مورد استفاده در آن است. در این حوزه، باید مفاهیمی مانند اجرا، خط‌مشی و ناظر تعریف شوند. برای بیان صوری این تعابیر، از مقاله [۲] استفاده شده است.

۵-۱-۱ نمادگذاری

یک اجرا σ از یک برنامه یا یک دنباله اجرا[۸۷] به عنوان دنباله‌ای متناهی یا نامتناهی از کنش‌های تجزیه‌نشدنی[۸۸] مدل می‌شود.

---

[86] Monitorable

[87] Trace

[88] Atomic

$\sigma = a_0, a_1, a_2, \ldots$

می‌توان مجموعه‌ای متناهی یا نامتناهی شمارا از کنش‌های تجزیه‌نشدنی به نام Σ در نظر گرفت (تکرار تعریف انواع سیگمای بزرگ).

در این بخش به نمادگذاری، بحث درباره مفاهیم اولیه و تعاریف اصلی پرداخته می‌شود. یک اجرای برنامه، یا σ، دنباله‌ای متناهی یا نامتناهی از کنش‌ها[۸۹] است که یک برنامه در زمان اجرا انجام می‌دهد. مجموعه کنش‌ها، مجموعه شمارای Σ است. مجموعه همه اجراهای متناهی را با Σ^*، و مجموعه همه اجراهای نامتناهی را با Σ^ω نمایش می‌دهند. همچنین تعریف می‌شود $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. برای هر اجرای $\sigma \epsilon \Sigma^\infty$ و هر عدد طبیعی $i \epsilon \mathbb{N}$، منظور از $\sigma[i]$، i+۱امین کنش در σ است. همچنین، نماد $\sigma[\cdot\cdot i]$ بیانگر دنباله i+۱ کنش اول σ است. زیردنباله‌ای از σ با شروع از $\sigma[i]$ را با $\sigma[i \cdot\cdot]$ نمایش داده می‌شود.

طول یک دنباله $\tau \epsilon \Sigma^*$ با $|\tau|$ نشان داده می‌شود و منظور از ε یک دنباله تهی است. برای $\sigma \epsilon \Sigma^\infty$ و τ، نماد $\tau; \sigma$ به معنای دنباله حاصل از اضافه‌شدن σ به انتهای τ است. همچنین، به τ پیشوند σ گفته می شود، با نماد $\tau \preceq \sigma$ ، اگر $\sigma' \epsilon \Sigma^\infty$ای وجود داشته باشد که $\sigma = \tau; \sigma'$. دنباله $\sigma \epsilon \Sigma^\infty$، دنباله $\tau \epsilon \Sigma^*$ را گسترش می‌دهد اگر τ پیشوندی از σ باشد و به شکل $\sigma \succeq \tau$ نمایش داده می‌شود. ضمناً منظور از $S \preceq S'$ برای $S \subseteq \Sigma^*$ و $S' \subseteq \Sigma^\infty$ این است که به ازای هر $\tau \epsilon S$، دنباله $\sigma \epsilon S'$ وجود داشته باشد که $\tau \preceq \sigma$. علاوه بر این، به زیرمجموعه متناهی از Σ^*، یک مشاهده[۹۰] گفته میشود. و در نهایت، برای $S \subseteq \Sigma^\infty$، منظور از $pref(S)$ مجموعه همه پیشوندهای دنباله‌های موجود در S است. به عبارت دیگر، $pref(S) = \{\tau | \exists \sigma \in S: \tau \preceq \sigma\}$. لازم به ذکر است که از همین نمادگذاری در فصل‌های بعدی این گزارش نیز استفاده خواهد شد.

[89] Action

[90] Observation

۵-۱-۲ تعاریف اولیه

یک خط‌مشی امنیتی (Σ^{∞}) $\mathrm{P} \subseteq \mathcal{P}$ مجموعه‌ای از مجموعه اجراهای مجاز است. خط‌مشی P یک خاصیت است اگر و فقط اگر بتوان آن را با مجموعه‌ای از دنباله‌ها سرشت‌نمایی کرد که یک مسند تصمیم‌پذیر $\hat{P}$ روی اجراهایی از Σ^{∞} وجود داشته باشد؛ یعنی $\hat{P}(\sigma)$ برقرار است اگر و فقط اگر σ در خط‌مشی وجود داشته باشد [۲]. به بیان دیگر، یک خاصیت خط‌مشی‌ای است که عضویت هر دنباله در آن خط‌مشی، فقط با ارزیابی همان دنباله قابل تعیین باشد. از آن‌جایی که همه خط‌مشی‌های قابل اعمال توسط ناظرها، خاصیت هستند، P و $\hat{P}$ را می‌توان به جای یکدیگر به کار برد.

نمونه‌ای از خط‌مشی‌هایی که خاصیت نیستند، خط‌مشی‌های جریان اطلاعات است که اجراشدن دنباله اجرای خاصی که دنباله اجرای دیگری را تحت تأثیر قرار می‌دهد را محدود می‌کند. به طور کلی، خط‌مشی‌هایی که جلوی وقوع دو دنباله اجرای یکسان را می‌گیرد یا لازم است که یک اجرای مشخص اول انجام شود تا دیگری مجاز باشد، خاصیت‌های امنیتی نیستند. زیرا ممکن نیست که با ارزیابی یک اجرای منفرد بتوان درباره برآورده‌شدن خط‌مشی امنیتی در آن برنامه تصمیم گرفت.

۵-۱-۲-۱ خاصیت ایمنی

کلاس‌هایی از خاصیت‌ها در ادبیات حوزه مطرح شده‌اند که در مطالعه ناظرها اهمیت بسزایی دارند. دسته اول، خاصیت‌های ایمنی[۹۱] هستند. این خاصیت‌ها وقوع یک چیز بد مشخص را در طول اجرا نهی می‌کنند. اگر Σ مجموعه‌ای از کنش‌ها و $\hat{P}$ یک خاصیت باشد، آن‌گاه $\hat{P}$ یک خاصیت ایمنی خواهد بود اگر و فقط اگر

$$\forall \sigma \in \Sigma^{\infty} : \neg \hat{P}(\sigma) \Rightarrow \exists \sigma' \preceq \sigma : \forall \tau \succeq \sigma' : \neg \hat{P}(\tau)$$

به بیان غیرصوری این خط‌مشی بیان می‌کند که هر اجرایی که این خاصیت امنیتی را برآورده نمی‌کند، پیشوندی از آن دنباله وجود دارد که هر گسترشی از آن نیز خط‌مشی را برآورده نخواهد کرد؛

[91] Safety Property

یعنی نقض یک خاصیت ایمنی، غیرقابل اصلاح است. هرگاه نقضی رخ دهد، نمی‌توان برای تصحیح آن کاری کرد.

۵-۱-۲-۲ خاصیت مانایی

دسته دیگر از خاصیت‌ها، خاصیت‌های مانایی[۹۲] است. یک خاصیت مانایی، خاصیتی است که وقوع یک چیز خوب مشخص را در هر اجرای معتبر تعیین می‌کند. به بیان صوری، برای مجموعه کنش‌های Σ و خاصیت $\hat{P}$، این خاصیت یک خاصیت مانایی خواهد بود اگر و فقط اگر

$$\forall \sigma \in \Sigma^* : \exists \tau \in \Sigma^\infty : \tau \succeq \sigma \wedge \hat{P}(\tau).$$

به بیان شهودی، یک خاصیت مانایی است اگر هر دنباله متناهی به یک دنباله معتبر تبدیل قابل گسترش باشد.

۵-۱-۲-۳ خاصیت معقول

دسته دیگری از خاصیت‌ها، آن‌هایی هستند که دنباله تهی ε جزو دنباله‌های معتبر محسوب شود. به این دسته، خاصیت‌های معقول[۹۳] [۳۳] گفته می‌شود. به بیان صوری:

$$\forall \hat{P} \subseteq \Sigma^\infty : \hat{P}(\epsilon) \Leftrightarrow \hat{P} \quad \text{is reasonable}.$$

۵-۱-۲-۴ خودکاره بوکی

یکی از نتایج پژوهش‌های انجام‌شده در حوزه خاصیت‌های امنیتی، بیان آن‌ها به کمک خودکاره‌ها است. یکی از انواع خودکاره‌های مورد استفاده، خودکاره بوکی است. خودکاره بوکی، خودکاره‌ی حالت‌متناهیِ قطعی یا غیرقطعی است که دنباله‌های با طول نامحدود را می‌پذیرد.

تعریف – یک خودکاره بوکی غیرقطعی، پنج‌تایی $\langle \Sigma. Q. Q_0. \delta. F \rangle$ به شرح زیر است:

- Σ مجموعه‌ای متناهی یا نامتناهی شمارا از نمادها است.

[92] Liveness Property

[93] Reasonable Property

- Q مجموعه‌ای متناهی یا نامتناهی شمارا از حالت‌ها است.
- $Q_0 \subseteq Q$، زیرمجموعه‌ای از حالت‌های اولیه است.
- $\delta: Q \times \Sigma \to 2^Q$ یک رابطه گذار است.
- $F \subseteq Q$ مجموعه حالت‌های نهایی است.

دنباله نامتناهی ρ تشکیل‌شده از نمادهای Σ معتبر است اگر و فقط اگر حداقل یک حالت از مجموعه حالت‌های نهایی به تعداد بی‌شمار دفعه دیده شود.

پس از این، نیاز است تا تعریفی از آن‌چه که *اعمال* یک خاصیت امنیتی نامیده می‌شود، ارائه شود. تعاریف گوناگونی ارائه شده است که همه آن‌ها درباره دو محور مطرح‌شده در [۲] تأکید دارند:

۱) درستی: همه رفتارهای قابل مشاهده برنامه هدف، مطابق با خاصیت امنیتی باشد. به بیان دیگر، هر دنباله خروجی در مجموعه اجراهای تعریف‌شده توسط $\hat{P}$ باشد.
۲) شفافیت: معناشناخت اجراهای معتبر حفظ شود؛ یعنی هر اجرای امن از برنامه، در اجراهای پس از اعمال آن برنامه نیز وجود داشته باشد. در این محور است که می‌توان تفاوت بین اعمال دقیق و اعمال هم‌ارز[۹۴] را درک کرد. در اعمال دقیق، اجازه هیچ‌گونه تبدیلی در دنباله ورودی داده نمی‌شود. در حالی‌که در اعمال هم‌ارز، می‌توان یک دنباله معتبر را به دنباله هم‌ارز دیگری تبدیل کرد. رابطه هم‌ارزی دنباله‌ها باید از قبل تعریف شده باشد.

۵-۲ خاصیت‌های قابل‌اعمال

سوال اساسی این حوزه آن است که *کدام‌یک از خط‌مشی‌های امنیتی توسط ناظرها قابل اعمال هستند؟* اولین کار در این حوزه توسط Schneider [۲] صورت گرفته است. در آن کار، تنها ناظرهایی که فقط اجرا را مشاهده می‌کنند که تنها توانایی قطع اجرای برنامه را دارند و هیچ دانش قبلی‌ای از برنامه هدف ندارند، مورد بررسی قرار گرفته است. در واقع با توجه به محدودیت‌ها، مجموعه مورد مطالعه کران پایینی از خط‌مشی‌های قابل اعمال توسط ناظرها به شمار می‌رود. سه محدودیت اصلی در این مقاله در

[94] Equivalent Enforcement

نظر گرفته شده است. اول آن‌که این مکانیزم اعمال تنها می‌تواند یک اجرا را بپذیرد یا رد کند و برای این کار نیز به سایر اجراهای همان برنامه دسترسی ندارد. به بیان صوری:

$$\exists \hat{P} : \forall \sigma \in \Sigma : \sigma \in P \Leftrightarrow \hat{P}(\sigma)$$

پس می‌توان نتیجه گرفت که تنها خاصیت‌های امنیتی توسط این‌گونه مکانیزم‌ها قابل اعمال هستند.

دوم آن‌که با توجه به این‌که ناظر هیچ‌گونه دسترسی به رفتار ممکن آینده اجرا ندارد، نمی‌تواند به اجرایی که پیشوندی از آن نامعتبر است و قرار است در ادامه اصلاح شود، اجازه اجرا بدهد. بنابراین، این ناظر تنها قادر به اعمال خاصیت‌هایی خواهد بود که نقض خاصیت، غیرقابل اصلاح باشد؛ یعنی

$$\forall \tau \in \Sigma^* : \neg\widehat{P}(\tau) \Rightarrow (\forall \sigma \in \Sigma^\infty : \neg\widehat{P}(\tau;\sigma))$$

و سوم آن‌که قبل از هر کنشِ اجرای داده‌شده، ناظر باید در خصوص پذیرفتن یا ردکردن اجرا تصمیم بگیرد. به این ترتیب، برای هر اجرای ردشده، مدت زمان متناهی از اجرا سپری شده است.

$$\forall \sigma \in \Sigma^\infty : \neg\widehat{P}(\sigma) \Rightarrow (\exists i \in \mathbb{N} : \neg\widehat{P}(\sigma[0..i]))$$

همان‌طور که پیشتر عنوان شد، خط‌مشی امنیتی‌ای که سه ویژگی بالا را داشته باشد، یک خاصیت ایمنی است. اما نمی‌توان گفت که همه ناظرها می‌توانند خاصیت‌های ایمنی را اعمال کنند. زیرا ناظرهای مورد بحث در [۲]، دارای محدویت‌های زیادی هستند و هیچ اطلاعی از رفتارهای ممکن برنامه‌های هدف ندارند. پس با در اختیار داشتن اطلاعات و رفع محدودیت‌های دیگر، کلاس بزرگتری از خط‌مشی‌ها را اعمال کنند. اما در طرف دیگر باید توجه داشت که محدودیت‌های دیگری مانند محدودیت‌های محاسباتی و حافظه‌ای، که در عمل وجود دارند، روی نحوه عملکرد ناظرها تأثیرگذار خواهند بود. پس بهترین بیان این است که کلاس خاصیت‌های ایمنی، کران بالایی از خط‌مشی‌های قابل اعمال توسط ساده‌ترین و محدودترین ناظر است.

در ادامه مقاله [۲]، Schneider نشان می‌دهد که می‌توان خاصیت‌های ایمنی را با زیرکلاس خاصی از خودکاره بوکی مدل کرد، که به آن اصطلاحاً *خودکاره امنیتی*[۹۵] اطلاق می‌شود.

تعریف – یک خودکاره امنیتی، خودکاره‌ای قطعی به شکل $\langle \Sigma. Q. q_0. \delta \rangle$ به شرح زیر است:

- Σ مجموعه‌ای متناهی یا نامتناهی شمارا از نمادها است.
- Q مجموعه‌ای متناهی یا نامتناهی شمارا از حالت‌ها است.
- $q_0 \subseteq Q$، زیرمجموعه‌ای از حالت‌های اولیه است.
- $\delta: Q \times \Sigma \to Q$ یک تابع گذار است.

حال در این خودکاره، به جای تعریف حالت‌های نهایی، در صورتی که اجرا منجر به انجام یک گذار شود که در تابع δ تعریف شده نیست، اجرا قطع می‌شود. به این ترتیب، همه حالت‌های این خودکاره بوکی خاص، حالت نهایی است و در صورت وقوع رویدادی که گذار آن تعریف‌نشده باشد، خودکاره پایان می‌یابد و اجرای متناظر آن، قطع می‌شود.

۵-۲-۱ محورهای تأثیرگذار در توانایی ناظرها

همان‌طور که پیشتر گفته شد، می‌توان با افزایش توانایی ناظر برای مواجهه با رویدادهای اجرا و در نظر نگرفتن محدودیت‌های ذکرشده، مطالعه در خصوص مجموعه خط‌مشی‌های قابل اعمال توسط ناظرها را ادامه داد. از این رو، Ligatti و همکارانش [۳۴]، [۳۵] تعریف ارائه‌شده توسط Schneider را در سه محور زیر تغییر دادند:

۱) توانایی در اختیار ناظر برای واکنش در برابر نقض احتمالی خط‌مشی امنیتی. از این جهت ناظرها می‌توانند:
   - اجرای برنامه را قطع کنند.
   - یک کنش غیرمجاز را توقیف کنند و اجرا ادامه یابد.
   - کنش یا کنش‌هایی را درج کنند.
   - کنش‌ها را درج و یا توقیف کنند یا به بیان دیگر، اجرا را ویرایش کنند. (ترکیبی از دو حالت قبلی)

[95] Security Automaton

۲) اطلاعات در دسترس ناظر درباره اجراهای ممکن برنامه. از این جنبه می‌توان ناظرها را در دو زمینه یکنواخت[۹۶] و غیریکنواخت[۹۷] دسته‌بندی کرد. زمینه یکنواخت به این معنی که ناظر هیچ دانشی در خصوص رفتار ممکن برنامه هدف ندارد. در طرف دیگر، زمینه غیریکنواخت بیانگر آن است که ناظر می‌داند که برنامه هدف رفتارهایی را از خود بروز نمی‌دهد.
اگر δ را مجموعه دنباله‌هایی که ناظر آن‌ها را به عنوان اجراهای ممکن برنامه می‌داند، در نظر بگیریم، ناظر در زمینه یکنواخت عمل می‌کند اگر $\delta=\Sigma^*$ باشد، و اگر $\delta\subseteq\Sigma^*$ یعنی ناظر در زمینه غیریکنواخت کار می‌کند.

۳) میزان آزادی عمل ناظر در تبدیل اجرای داده‌شده یا به بیان دیگر، پارادایم اعمال. در این محور نیز می‌توان بین اعمال دقیق[۹۸]، اعمال مؤثر[۹۹] و اعمال اصلاحی[۱۰۰] تفاوت قائل شد. در اعمال دقیق، در یک اجرای معتبر هر کنش انجام‌شده توسط برنامه هدف باید نگه داشته شود. در اعمال مؤثر اجازه داده می‌شود تا یک اجرای معتبر به اجرای دیگری که از نظر معناشناخت معادل است، تبدیل شود که لازم است یک رابطه هم‌ارزی از پیش تعریف شده باشد. در اعمال اصلاحی، علاوه بر اعمال مؤثر، اجراهای ناسالم هم باید حداقل تغییرات ممکن را داشته باشند.

۵-۲-۲ مدل‌سازی پارادایم‌های اعمال به کمک خودکاره‌ها

برای مدل‌سازی پارادایم‌های مختلف مطرح‌شده، Ligatti خودکاره‌های جدیدی را معرفی کرد که البته شباهت‌هایی به خودکاره امنیتی مطرح‌شده توسط Schneider نیز دارند. اما تفاوت اصلی بین این دو خودکاره در آن است که خودکاره امنیتی برای تشخیص[۱۰۱] معتبر بودن یا نبودن دنباله ورودی ساخته شده است. در حالی‌که خودکاره‌هایی که در ادامه تعریف خواهند شد، دنباله ورودی را تغییر می‌دهند تا خروجی جدید سازگار با خط‌مشی امنیتی تولید کنند. باید توجه داشت که اجرای برنامه، که ورودی ناظر

[96] Uniform Context
[97] Nonuniform Context
[98] Precise Enforcement
[99] Effective Enforcement
[100] Corrective Enforcement
[101] Recognize

است، توسط مشاهده‌گرهای بیرونی قابل مشاهده نیست. بلکه مشاهده‌گرهای بیرونی می‌توانند خروجی تولیدشده توسط ناظر را ببینند.

اجرای یک خودکاره را می‌توان با حکم[۱۰۲]های تک‌گامی[۱۰۳] به شکل $(q, \sigma) \xrightarrow{\tau}_{A} (q', \sigma')$ نمایش داد که در آن q حالت فعلی است، σ دنباله‌ای است که برنامه هدف می‌خواهد آن را اجرا کند، τ دنباله‌ای است که ناظر در این گام تولید کرده است، q' بیانگر حالت بعدی خودکاره و σ' دنباله ورودی گام بعدی اجرا است. این حکم‌های تک‌گامی را می‌توان به حکم‌های چندگامی[۱۰۴] به شکل $(q, \sigma) \xRightarrow{\tau} (q', \sigma')$ تعمیم داد.

تعریف – یک حکم چندگامی $(q, \sigma) \xRightarrow{\tau} (q', \sigma')$ از حکم‌های تک‌گامی با شرط‌های زیر ساخته شده است:

۱) $(q, \sigma) \xRightarrow{\epsilon} (q', \sigma')$

۲) $(q, \sigma) \xrightarrow{\tau} (q'', \sigma'') \wedge (q'', \sigma'') \xrightarrow{\tau'}_{A} (q', \sigma') \Longrightarrow (q, \sigma) \xRightarrow{\tau;\tau'} (q', \sigma')$

تفاوت خودکاره‌های جدید در رابطه‌های گذار δ مختلف است. ساده‌ترین مدل، خودکاره قطع‌کننده [۳۵] است که رفتاری مشابه خودکاره امنیتی Schneider را شبیه‌سازی می‌کند. این خودکاره یا کنش ورودی را می‌پذیرد یا اجرا را قطع می‌کند.

۵-۲-۲-۱ خودکاره قطع‌کننده

تعریف – خودکاره قطع‌کننده A، یک چهارتایی به شکل $\langle Q. \Sigma. q_0. \delta \rangle$ است که در آن:

- Q مجموعه‌ای متناهی یا نامتناهی شمارا از حالت‌ها است.
- Σ مجموعه‌ای متناهی یا نامتناهی شمارا از کنش‌های تجزیه‌نشدنی است.
- q_0 حالت اولیه است.

[102] Judgment

[103] Single-step

[104] Multi-step

- $\delta: Q \times \Sigma \to Q$ یک تابع گذار قطعی است.

این خودکاره کنش ورودی گرفته‌شده را خروجی می‌دهد اگر برای آن کنش در حالت فعلی، تابع گذار تعریف شده باشد. در غیر این صورت، کار خودکاره تمام می‌شود. رفتار این خودکاره را می‌توان با عبارات زیر بیان کرد:

$(q, \sigma) \overset{a}{\Rightarrow} (q', \sigma')$ if $\sigma = a$; σ' and $\delta(a, q) = q'$

$(q, \sigma) \overset{\epsilon}{\Rightarrow} (q', \epsilon)$ otherwise

۵-۲-۲-۲ خودکاره توقیف

خودکاره توقیف [۳۵] توانایی این را دارد تا یک کنش ورودی را به دنباله خروجی ارسال نکند و خودکاره کار خودش را ادامه دهد. توصیف این کار در تابع گذار آن آمده است.

تعریف – خودکاره توقیف A، یک پنج‌تایی به شکل $\langle Q. \Sigma. q_0. \delta. \omega \rangle$ است که چهار عنصر اول مطابق تعریف قبلی است. همچنین، $\omega: Q \times \Sigma \to \{+. -\}$ یک تابع قطعی با دامنه مشابه تابع گذار است و نشان می‌دهد که یک کنش ورودی باید در خروجی باشد (+) یا توقیف شود (-). رفتار خودکاره توقیف به شرح زیر است:

$(q, \sigma) \overset{a}{\Rightarrow} (q', \sigma')$ if $\sigma = a$; σ' , $\delta(a, q) = q'$ and $\omega(a,q) = +$

$(q, \sigma) \overset{\epsilon}{\Rightarrow} (q', \sigma')$ if $\sigma = a$; σ' , $\delta(a, q) = q'$ and $\omega(a,q) = -$

$(q, \sigma) \overset{\epsilon}{\Rightarrow} (q', \epsilon)$ otherwise

۵-۲-۲-۳ خودکاره درج

خودکاره درج [۳۵] ناظری را مدل می‌کند که می‌تواند به جریان کنترل کنش‌هایی را اضافه کند. بنابراین، تابع گذار این خودکاره تعیین می‌کند که کنش ورودی باید به تنهایی یا در کنار دنباله‌ای متناهی از کنش‌های دیگر در خروجی ظاهر شود. حالت دیگر آن است که اجرا در همان نقطه خاتمه یابد.

تعریف - خودکاره درج A، یک پنج‌تایی به شکل $\langle Q. \Sigma. q_0. \delta. \gamma \rangle$ است که چهار عنصر اول مشابه تعاریف قبلی هستند. تابع $\gamma: Q \times \Sigma \to Q \times \Sigma^*$، تابعی قطعی است با دامنه‌ای غیرمشترک با تابع δ؛ به این

معنا که امکان استفاده از هر دو تابع δ و γ به طور همزمان وجود ندارد. این تابع دنباله متناهی خروجی متناظر با دنباله ورودی داده‌شده را تعیین می‌کند.

$$(q, \sigma) \stackrel{a}{\Rightarrow} (q', \sigma') \qquad \text{if } \sigma = a;\ \sigma' \text{ and } \delta(a, q) = q'$$

$$(q, \sigma) \stackrel{\tau}{\Rightarrow} (q', \sigma') \qquad \text{if } \sigma = a;\ \sigma',\ \gamma(a, q) = (\tau, q')$$

$$(q, \sigma) \stackrel{\epsilon}{\Rightarrow} (q', \epsilon) \qquad \text{otherwise}$$

۵-۲-۲-۴ خودکاره ویرایش

آخرین و قوی‌ترین مدل، خودکاره ویرایش [۳۶] است که توانایی هر دو خودکاره توقیف و درج را دارد.

تعریف – خودکاره ویرایش A، یک چهارتایی به شکل $\langle Q.\Sigma.q_0.\delta \rangle$ است که $\delta: Q \times \Sigma \to Q \times \Sigma^{\infty}$، تابع قطعی گذار تعریف می‌شود. این تابع تعیین می‌کند که خروجی خودکاره به ازای کنش ورودی در حالت فعلی چه دنباله‌ای باشد.

اگر هر خودکاره تعریف‌شده در بالا با A نشان داده شود، منظور از $A(\sigma)$، خروجی خودکاره A برای دنباله ورودی σ است. همچنین، الگوریتم‌هایی برای ساختن ناظر از روی یک خودکاره بوکی مطرح شده است.

۵-۲-۳ پارادایم‌های اعمال

همان‌طور که پیشتر ذکر شد، دو معیار درستی و شفافیت برای مکانیزم‌های اعمال خط‌مشی‌های امنیتی همواره مطرح هستند. گرچه تعبیر درستی مشخص است، اما درباره شفافیت، تعابیر متفاوتی بیان می‌شود. تعابیری مانند اعمال دقیق، اعمال مؤثر و اعمال اصلاحی از جمله آن‌ها است.

۵-۲-۳-۱ پارادایم اعمال دقیق

یک ناظر خاصیتی را به طور *دقیق* اعمال می‌کند اگر در هر گام از اجرا، کنش ورودی را به عنوان بخشی از یک دنباله معتبر بپذیرد.

تعریف – اگر Σ مجموعه‌ای از کنش‌های تجزیه‌نشدنی و $\mathcal{S} \subseteq \Sigma^{\infty}$ زیرمجموعه‌ای از دنباله‌ها باشد، خودکاره $A = \langle Q. \Sigma. q_0. \delta \rangle$ خاصیت $\hat{P}$ را به طور *دقیق* اعمال می‌کند اگر و فقط اگر $\forall \sigma \in \mathcal{S}$

۱) $(q_0, \sigma) \overset{\sigma'}{\Rightarrow}_A (q', \epsilon)$
۲) $\hat{P}(\sigma')$
۳) $\hat{P}(\sigma) \Rightarrow \forall i \in \mathbb{N} : i \leq |\sigma| : \exists q'' \in Q : (\sigma, q_0) \overset{\sigma'[0..i]}{\Longrightarrow}_{\mathcal{A}} (\sigma[i+1..], q'')$

واضح است که این تعبیر بسیار سخت‌گیرانه است و از تساوی نحوی دو اجرای ورودی و خروجی نیز فراتر می‌رود. ناظری یک خاصیت را به طور *دقیق* اعمال می‌کند که به ازای هر کنش ورودی در یک اجرای معتبر، بلافاصله پس از مشاهده کنش ورودی، همان را در خروجی تکرار کند. این شرط را می‌توان به نحوی تغییر داد که دو دنباله ورودی و خروجی تحت رابطه‌ای هم‌ارز باشند.

۵-۲-۳-۲پارادایم مؤثر

تعریف - اگر Σ مجموعه‌ای از کنش‌های تجزیه‌نشدنی و $\mathcal{S} \subseteq \Sigma^{\infty}$ زیرمجموعه‌ای از دنباله‌ها باشد، خودکاره A به طور *مؤثر* تحت رابطه $\cong$ خاصیت $\hat{P}$ را اعمال می‌کند اگر و فقط اگر $\forall \sigma \in \mathcal{S}$

۱) $(q_0, \sigma) \overset{\sigma'}{\Rightarrow}_A (q', \epsilon)$
۲) $\hat{P}(\sigma')$
۳) $A(\sigma) \cong \sigma'$.

رابطه هم‌ارزی $\cong$ می‌تواند هر رابطه‌ای بین معناشناخت دنباله‌ها باشد. گرچه Ligatti و همکارانش [۳۵] تأکید دارند که رابطه باید خواسته غیرقابل‌تمایزبودن[۱۰۵] را برآورده کند.

$$\forall \hat{P} : \ \forall \sigma. \sigma' \in \Sigma^{\infty} : \sigma \cong \sigma' \Rightarrow (\hat{P}(\sigma) \Leftrightarrow \hat{P}(\sigma'))$$

نشان داده شده است که تنها رابطه هم‌ارزی که تعریف بالا را برآورده می‌کند، تساوی نحوی است [۳۷].

در ادامه به بررسی خط‌مشی‌های امنیتی قابل اعمال در مدل‌های مختلف اعمال پرداخته می‌شود [۳۷]. هر یک از محورهای سه‌گانه مورد مطالعه قرار گرفته است.

[105] Indistinguishability

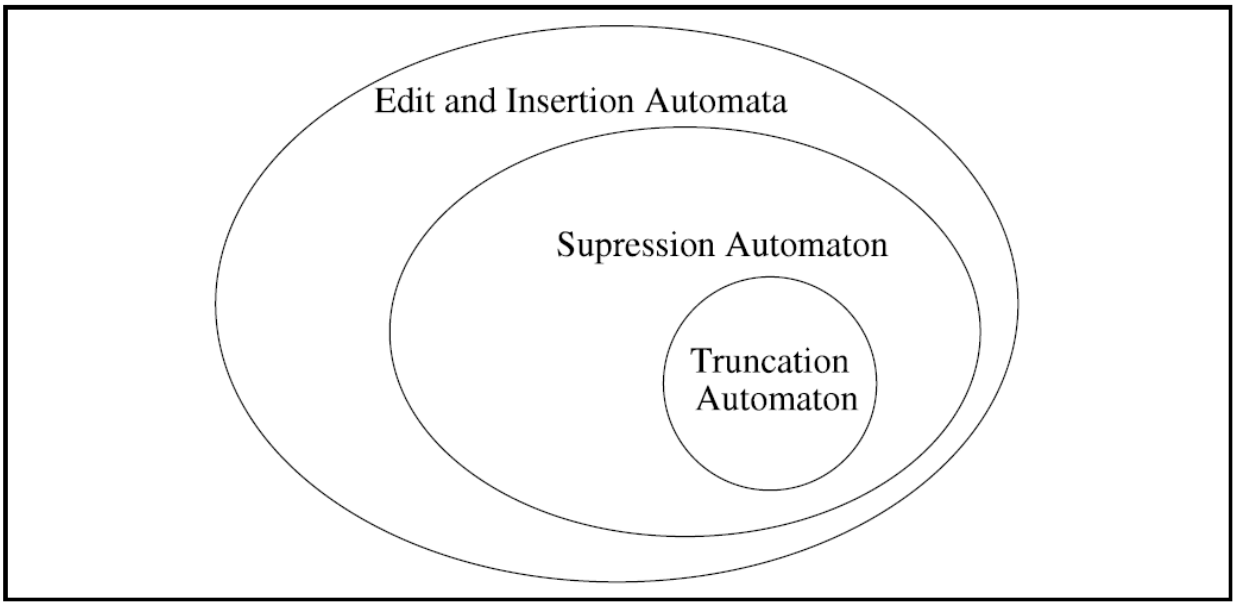
۵-۲-۴ بررسی توانایی ناظرها در محورهای سه‌گانه

ساده‌ترین مدل، استفاده از یک خودکاره قطع‌کننده است که یک خاصیت را در محیطی یکنواخت به طور دقیق اعمال می‌کند. این مدل، معادل با مدل بحث‌شده درباره خودکاره امنیتی می‌شود که نشان داده شده است دقیقاً مجموعه خاصیت‌های ایمنی را می‌توانند اعمال کنند. البته در محیط غیریکنواخت، می‌توان بعضی از خاصیت‌های مانایی را نیز اعمال کرد. همچنین، مجموعه خط‌مشی‌های قابل اعمال توسط خودکاره قطع‌کننده و اعمال مؤثر بزرگتر از همین مکانیزم در اعمال دقیق است.

مجموعه خاصیت‌های قابل اعمال توسط خودکاره قطع‌کننده در سه حالت دیگر نیز قابل بررسی است. در یک محیط یکنواخت، مجموعه خط‌مشی‌های قابل اعمال در پارادایم دقیق توسط خودکاره‌های توقیف، درج و ویرایش، مشابه با مجموعه خط‌مشی‌های قابل اعمال در همین شرایط توسط خودکاره قطع‌کننده است. ممکن است در نگاه اول عدم افزایش قدرت و توانایی ناظر، با توجه به افزایش توانایی خودکاره متناظر آن تعجب‌برانگیز باشد. اما تعریف اعمال دقیق باعث این نتیجه می‌شود. طبق تعریف، باید اجراهایی که خط‌مشی را برآورده می‌کنند بدون تغییر پذیرفته شوند و هر کنش بلافاصله در خروجی ناظر منعکس شود. از آن‌جایی که در زمینه یکنواخت، تبعیت از این خواسته و تغییر یک اجرا به طور همزمان شدنی نیست، توانایی‌های اضافه‌شده در خودکاره‌های درج، توقیف و ویرایش نمی‌تواند تأثیری در گستره خط‌مشی‌های امنیتی قابل اعمال در یک محیط یکنواخت شود.

باید توجه داشت که برای اعمال دقیق در زمینه غیریکنواخت بازه خط‌مشی‌های قابل اعمال توسط خودکاره‌های مختلف، متفاوت خواهد بود. در همین راستا، می‌توان به افزایش خاصیت‌های قابل اعمال توسط خودکاره توقیف نسبت به خودکاره قطع‌کننده اشاره داشت. ضمناً مکانیزمی با این شرایط توسط خودکاره درج، اکیداً قوی‌تر از خودکاره توقیف است. همچنین، در محیط غیریکنواخت، خودکاره درج می‌تواند هر خاصیتی که توسط خودکاره توقیف به طور دقیق قابل اعمال است را به طور دقیق اعمال کند. زیرا خودکاره درج می‌تواند به نحوی رفتار خودکاره توقیف را شبیه‌سازی کند. خودکاره توقیف با توقیف یک کنش، طبق تحلیل ایستایی که از برنامه هدف در اختیار دارد، می‌داند که ادامه اجرا می‌تواند حالت‌های مختلفی داشته باشد. خودکاره درج کافی است تا یکی از این پسوندهای اجرا را که مطابق خط‌مشی امنیتی است، در نظر بگیرد. گرچه خاصیت‌هایی وجود دارد که به طور دقیق توسط خودکاره درج اعمال می‌شوند اما خودکاره توقیف قادر به اعمال آن‌ها نیست. به این دلیل که ممکن است

خودکاره درج با استفاده از توانایی‌های خود، کنش‌هایی را برای اصلاح یک دنباله نامعتبر به جریان کنترل اضافه کند. با توجه به مطالب مطرح‌شده، می‌توان گفت که قدرت یک خودکاره ویرایش در محیط غیریکنواخت دقیقاً با قدرت خودکاره درج برابر است. زیرا همان‌طور که پیشتر گفته شد، خودکاره ویرایش، ترکیبی از خودکاره درج و توقیف است، و از آن‌جایی که خودکاره درج فوق‌مجموعه‌ای از

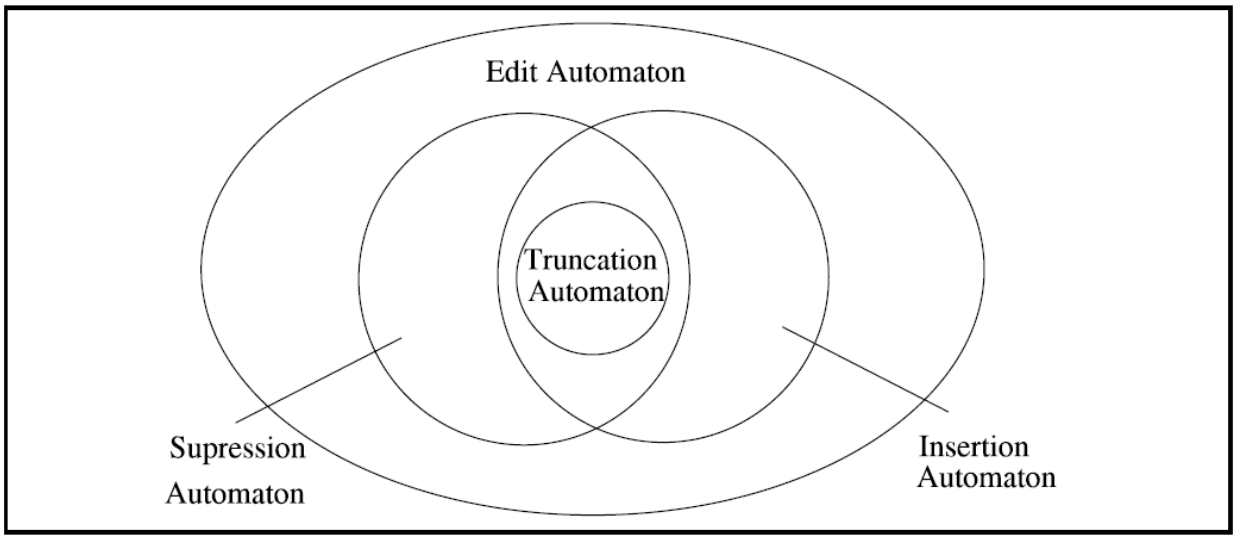


شکل ۲ – نمای گرافیکی قدرت خودکاره‌های مختلف در اعمال دقیق در سامانه‌های غیریکنواخت [۳۴]

خط‌مشی‌های قابل اعمال توسط خودکاره توقیف را اعمال می‌کند، پس نتیجه درستی خواهد بود.

نکته این‌جاست که تمامی نتایج بالا برای پارادایم دقیق بودند و نمی‌توان به سادگی به اعمال مؤثر نیز تعمیم داد. به طور مثال، در این پارادایم، توانایی دو خودکاره درج و توقیف قابل مقایسه نیست. زیرا در حالت‌های خاصی، نمی‌توان کنشی را درج کرد و معناشناخت هم دست‌نخورده باقی بماند، در حالی‌که اگر یک یا چند کنش توقیف شوند، می‌توان به این خواسته رسید. دیگر آن‌که مجموعه خط‌مشی‌هایی که خودکاره ویرایش می‌تواند آن‌ها را به طور مؤثر اعمال کند، فوق‌مجموعه‌ای از مجموعه خط‌مشی‌های قابل اعمال توسط هر پارادایم دیگری است که پیشتر بحث شد. این قدرت ناظر ویرایش از آن‌جا است که می‌تواند اگر یک دنباله ورودی نامعتبر باشد، آن را توقیف کند و اگر در ادامه معتبر شد، آن را مجدداً درج کند. با توجه به این‌که چنین تحلیل‌هایی برای دنباله‌های متناهی قابل طرح است و

دنباله تهی همواره معتبر در نظر گرفته می‌شود، بنابراین خودکاره ویرایش می‌تواند هر خاصیتی را به طور مؤثر اعمال کند.



شکل ۳ – مقایسه توانایی خودکاره‌های مختلف در اعمال مؤثر تحت رابطه هم‌ارزی [۳۴]

همان‌طور که در شکل‌های بالا نیز مشاهده می‌شود، می‌توان دو مورد را خاطرنشان کرد. اول آن‌که گستره خط‌مشی‌های قابل اعمال توسط بیشتر پارادایم‌های نظارت در زمینه غیریکنواخت، با ثابت نگه‌داشتن بقیه عوامل، وسیع‌تر شده است. در حالی‌که چنین چیزی برای زمینه یکنواخت صحت ندارد. دوم آن‌که ناظری که امکانات بیشتری در قبال نقض خط‌مشی در اختیار دارد، می‌تواند بازه بزرگتری از خط‌مشی‌ها را اعمال کند. البته در شرایطی که آزادی عمل کافی برای تغییر دنباله ورودی معتبر تحت یک رابطه هم‌ارزی را نیز داشته باشد. زیرا اگر رابطه هم‌ارزی بیش از حد سخت‌گیرانه باشد و ناظر نتواند اجراهای معتبر را تغییر دهد، قدرت ناظر برای اعمال خط‌مشی‌ها افزایش پیدا نخواهد کرد.

تا اینجا تمرکز اصلی مقایسه قدرت ناظرها در پارادایم‌ها و شرایط مختلف روی دنباله‌های متناهی بوده است [۳۷]. در حالی‌که رفتار بیشتر سامانه‌ها خاتمه‌پذیر نیست. به همین خاطر، Ligatti و همکارانش در [۱۹]، [۳۳] به تعمیم تحلیل‌های بالا به منظور ارائه یک چارچوب مناسب برای دنباله‌های متناهی و نامتناهی پرداختند. مطالعات آن‌ها روی اعمال مؤثر تحت رابطه $\cong$ ، و به طور دقیق‌تر رابطه تساوی نحوی تساوی، توسط خودکاره ویرایش متمرکز بوده است. گرچه ممکن است روابط هم‌ارزی

دیگری نیز قابل تصور باشد. به همین دلیل، مجموعه خط‌مشی‌های قابل اعمال به طور مؤثر تحت رابطه = توسط خودکاره ویرایش، کران پایینی از اعمال مؤثر خودکاره ویرایش به شمار می‌آید.

به بیان غیرصوری، اعمال مؤثر تحت رابطه تساوی توسط خودکاره ویرایش به این ترتیب است که در صورتی که کنش ورودی با خط‌مشی سازگار باشد، فوراً در خروجی ظاهر می‌شود. اگر خودکاره کنش ورودی را بخشی از یک دنباله بدخواهانه تشخیص دهد، خودکاره ویرایش کنش را توقیف می‌کند و چیزی به خروجی نمی‌دهد. سپس کنش‌ها را نگه می‌دارد تا منتظر شود که آیا در ادامه اجرا معتبر خواهد شد یا خیر. در صورتی که در ادامه اجرا، دنباله معتبر باشد، کنش‌های توقیف‌شده را در کنار بقیه کنش‌های دنباله به خروجی اضافه می‌کند. اگر دنباله ورودی نامعتبر باشد و در ادامه نیز معتبر نشود، خروجی ناظر تنها پیشوند معتبری از دنباله ورودی خواهد بود. در واقع می‌توان گفت که ناظر اجرا را تا قسمتی از آن‌که به معتبربودن آن اطمینان دارد، شبیه‌سازی می‌کند؛ یعنی طولانی‌ترین پیشوند از دنباله ورودی که معتبر است، به عنوان خروجی این ناظر اعلام می‌شود.

برای توصیف دسته خط‌مشی‌های قابل اعمال توسط خودکاره ویرایش در این شرایط، مجموعه خاصیت‌های تجدید نامتناهی[۱۰۶] (یا به طور اختصار، تجدید) تعریف می‌شود. یک خاصیت در این دسته قرار می‌گیرد اگر هر دنباله نامتناهی معتبر آن، بی‌شمار پیشوند معتبر داشته باشد و هر دنباله نامتناهی نامعتبر، دارای تعداد متناهی پیشوند معتبر باشد. به بیان صوری:

$$\forall \sigma \in \Sigma^{\omega} : \hat{\mathrm{P}}(\sigma) \Leftrightarrow \{\sigma' \preceq \sigma \mid \hat{\mathrm{P}}(\sigma')\} \text{ is an infinite set}$$

یا

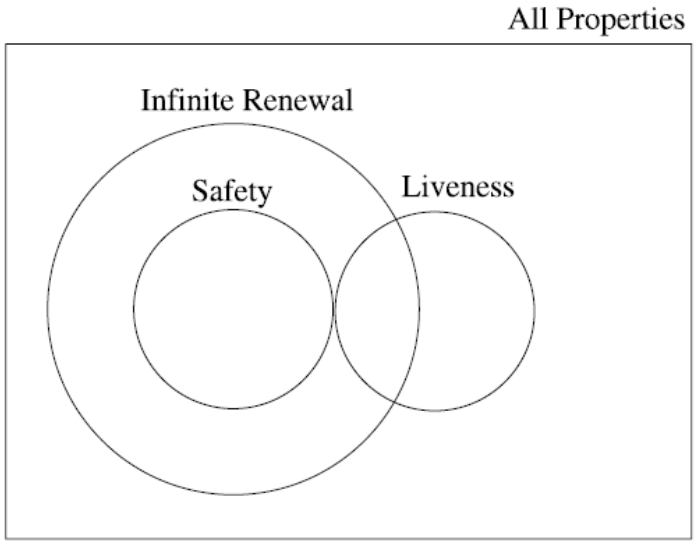
$$\forall \sigma \in \Sigma^{\omega} : \hat{\mathrm{P}}(\sigma) \Leftrightarrow (\forall \sigma' \preceq \sigma : \exists \tau \preceq \sigma : \sigma' \preceq \tau \wedge \hat{\mathrm{P}}(\tau))$$

با کمی دقت مشخص می‌شود که تعریف خاصیت تجدید نامتناهی هیچ محدودیتی روی دنباله‌های متناهی موجود در $\hat{P}$ نمی‌گذارد. بنابراین خللی در نتایج ذکرشده قبلی وارد نمی‌شود و می‌توان گفت که در سامانه‌های حاوی فقط دنباله‌های متناهی، همه خاصیت‌ها توسط خودکاره ویرایش قابل

[106] Infinite Renewal Properties

اعمال هستند [۳۴]. برای دنباله‌های نامتناهی، مجموعه خاصیت‌های تجدید قابل اعمال است. این مجموعه شامل همه خاصیت‌های ایمنی، تعدادی از خاصیت‌های مانایی و بعضی از خاصیت‌هایی که نه ایمنی و نه مانایی هستند، می‌شود. مثالی از این‌گونه خاصیت‌ها این است که یک کنش خاص بالاخره در یک اجرای ناتهی رخ دهد. برای هر اجرای معتبر با طول نامتناهی در این خاصیت، بی‌شمار پیشوند معتبر وجود دارد؛ پیشوندهایی که در آن‌ها کنش مورد نظر به وقوع پیوسته باشد. در حالی‌که هر اجرای نامعتبر با طول نامتناهی، تعداد متناهی پیشوند معتبر دارند؛ هیچ پیشوندی برای خاصیت ذکرشده در این حالت وجود ندارد.

بازنمایی مجموعه خاصیت‌های تجدید نامتناهی و مقایسه آن‌ها با خاصیت‌های ایمنی و مانایی در شکل زیر آمده است [۱۹].



شکل ۴ – تعیین محدوده خاصیت‌های تجدید نامتناهی [۱۹]

مجموعه خاصیت‌های معقول تجدید نامتناهی، کران پایینی برای مجموعه خاصیت‌های قابل اعمال خودکاره ویرایش به طور مؤثر تحت رابطه تساوی است. این مکانیزم می‌تواند خاصیت‌های غیرتجدید را نیز اعمال کند. در صورتی که در نقطه مشخصی از اجرا، خودکاره تشخیص دهد که پیشوند فعلی اجرا قابل گسترش به فقط یک دنباله برآورده‌کننده خط‌مشی است، می‌تواند آن دنباله را در خروجی ثبت کند و اجرا را فوراً به پایان برساند. البته که این یک حالت خاص است.

همچنین، اعمال مؤثر تحت رابطه تساوی کران پایینی از مجموعه خاصیت‌هایی است که خودکاره ویرایش می‌تواند به طور مؤثر اعمال کند. زیرا در اعمال تحت رابطه تساوی، اجازه تغییر اجرا

برای تبدیل به یک اجرای مطابق با خط‌مشی امنیتی داده نمی‌شود. در حالی‌که در اعمال مؤثر در حالت کلی، این خودکاره می‌تواند کنش‌ها را حذف یا درج کند و تغییراتی را در دنباله برای تبدیل آن به یک دنباله معتبر انجام دهد. این توانایی‌ها باعث افزایش قدرت ناظر می‌شود. علاوه بر این، مجموعه تجدید نیز کران پایینی از مجموعه خاصیت‌های قابل اعمال در محیط غیریکنواخت است. در زمینه غیریکنواخت، این مکانیزم می‌تواند خاصیت‌های غیرتجدید را نیز اعمال کند.

گرچه تمرکز اصلی ناظرهای مبتنی بر خودکاره، به مدل پیشنهادشده در [۳۴]، [۳۶] است اما خودکاره‌های جریان و خودکاره‌های نتایج اجباری[۱۰۷] [۱۷] از انواع دیگر مطرح‌شده برای مدل‌سازی محسوب می‌شوند. تفاوت این دو نوع خودکاره و خودکاره ویرایش در آن است که در آن‌ها بین مجموعه کنش‌های برنامه هدف و کنش‌های تعامل با سامانه تمایز قائل می‌شود [۳۷]. بنابراین، بررسی و مدل‌سازی تعامل بین برنامه هدف، ناظر و سامانه در این‌گونه خودکاره‌ها بهتر مشخص می‌شود. منظور از مدل ناظر اجرا توسط خودکاره نتایج اجباری آن است که ناظر اجرا الزاماً پیش از دریافت کنش ورودی بعدی، باید نتیجه کنش قبلی را برای برنامه تحت نظارت بفرستد. در واقع، این خودکاره علاوه بر واسطه‌گری کنش‌های اجراشده برنامه تحت نظارت، می‌تواند در نتیجه اجرای این کنش‌ها در محیط مقصد نیز دخالت کند.

قضایای زیر جمع‌بندی و مقایسه‌ای از قدرت اعمال مکانیزم‌های مختلف مطرح‌شده را مشخص می‌کنند. پیش از آن، نمادگذاری مورد استفاده در آن‌ها توضیح داده می‌شود. مجموعه {T, D, I, E} حالت‌های مختلف خودکاره‌ها را نشان می‌دهد. T برای خودکاره قطع‌کننده، D برای توقیف، I برای درج، E برای ویرایش و A به معنای هر ناظری از این مجموعه است. منظور از $\delta \subseteq \Sigma^{\infty}$ زیرمجموعه‌ای از دنباله‌های اجرای ممکن است. مقصود از نوشتن $A^{\delta}_{\cong}$ - *eff*، مجموعه خاصیت‌هایی است که یک ناظر کلاس A می‌تواند به طور مؤثر تحت رابطه هم‌ارزی $\cong$ اعمال کند، در شرایطی که مجموعه دنباله‌های ممکن δ باشد. در ادامه رابطه هم‌ارزی، رابطه تساوی در نظر گرفته شده است. همچنین، A^{δ} - *prcs* یعنی مجموعه خاصیت‌هایی که توسط یک ناظر کلاس A و با فرض مجموعه دنباله‌های ممکن δ، می‌تواند در پارادایم دقیق اعمال کند.

[107] Mandatory Results Automata

قضیه [۳۵] –

1) $T^{\Sigma^*}\text{-prcs} = I^{\Sigma^*}\text{-prcs} = D^{\Sigma^*}\text{-prcs} = E^{\Sigma^*}\text{-prcs} = \text{Safety}$
2) $\exists S \subset \Sigma^* : A^{\Sigma^*}\text{-prcs} \subset T^{S}\text{-prcs}$
3) $\forall S \subset \Sigma^* : \text{Safety} \subset T^{S}\text{-prcs} \subset D^{S}\text{-prcs} \subset I^{S}\text{-prcs} = E^{S}\text{-prcs}$
4) $(\text{Safety} \subset T_{=}^{\Sigma^*}\text{-eff} \subset I_{=}^{\Sigma^*}\text{-eff}) \wedge (T_{=}^{\Sigma^*}\text{-eff} \subset D_{=}^{\Sigma^*}\text{-eff})$
5) $(I_{=}^{\Sigma^*}\text{-eff} \subset E_{=}^{\Sigma^*}\text{-eff}) \wedge (D_{=}^{\Sigma^*}\text{-eff} \subset E_{=}^{\Sigma^*}\text{-eff})$
6) $\forall \hat{P} \subseteq \Sigma^* : \hat{P} \in E_{=}^{\Sigma^*}\text{-eff}$

۵-۳ بررسی قدرت ناظرها با در نظرگرفتن محدودیت‌های حافظه‌ای و محاسباتی

در بررسی توانایی ناظرها برای اعمال خط‌مشی‌های امنیتی مختلف، محدودیت‌های دیگری مانند محدودیت‌های حافظه و محاسباتی تأثیرگذار است که در سرشت‌نمایی مطرح‌شده تا اینجا، این محدودیت‌ها لحاظ نشده‌اند. در این بخش به بررسی تأثیر این‌گونه محدودیت‌ها روی مجموعه خاصیت‌های قابل اعمال توسط ناظرها پرداخته می‌شود. البته پیش‌بینی می‌شود که با لحاظ‌کردن چنین محدویت‌هایی، بعضی از خاصیت‌هایی که قبلاً توسط مکانیزم اعمال می‌شدند، قابل اعمال نباشند. به عنوان مثال، ممکن است یک خاصیت ایمنی دیگر توسط ناظر قطع‌کننده قابل اعمال نباشد. زیرا ناظر قادر به تشخیص نقض امنیتی در هنگام روی‌دادن آن نباشد. به همین دلیل، بحث‌های انجام‌شده در بخش‌های قبلی، کران بالایی از مجموعه خاصیت‌هایی است که ناظر می‌توانند در عمل اعمال کنند.

۵-۳-۱ محدویت محاسباتی

اولین کار در این حوزه توسط Kim و همکارانش [۳۸] انجام شده است. طبق این بررسی و قضیه زیر، یک خودکاره امنیتی زمانی می‌تواند یک خاصیت را اعمال کند که قادر باشد تا هر دنباله غیرمعتبری را با وارسی پیشوندی متناهی تشخیص دهد. پس ناظرهای امنیتی تنها می‌توانند چنین خاصیت‌هایی را اعمال کنند.

پیش از بیان قضیه زیر لازم است مجموعه شمارش‌پذیر بازگشتی تعریف شود. یک مجموعه را شمارش‌پذیر بازگشتی می‌نامند اگر الگوریتمی وجود داشته باشد به طوری که مجموعه اعداد ورودی که

الگوریتم پایان می‌یابد، دقیقاً برابر آن مجموعه باشد. یا به بیان دیگر، الگوریتمی وجود دارد که می‌تواند اعضای موجود در چنین مجموعه‌ای را بشمارد.

قضیه – خاصیت $\hat{P}$ توسط یک خودکاره امنیتی قابل اعمال است اگر و فقط اگر $\hat{P}$ یک خاصیت ایمنی باشد و نیز مجموعه $\Sigma^* \backslash pref(\hat{P})$ شمارش‌پذیر بازگشتی[۱۰۸] باشد.

طبق قضیه بالا، تنها خاصیت‌های ایمنی‌ای را می‌توان توسط یک خودکاره امنیتی اعمال کرد که مجموعه اجراهای ناقض آن خط‌مشی، شمارش‌پذیر بازگشتی باشند. به عبارت دیگر، مجموعه خاصیت‌های قابل اعمال، کلاس خاصیت‌های مکمل شمارش‌پذیر بازگشتی[۱۰۹] است [۳۸].

Hamlen و همکارانش با معرفی ماشین‌های برنامه[۱۱۰] (PM)، به مقایسه خط‌مشی‌های قابل اعمال توسط سه مکانیزم تحلیل ایستا، بازنویسی کد و ناظرها پرداخته‌اند [۱۴]. در این مقاله نشان داده می‌شود که یک خاصیت $\hat{P}$ می‌تواند به طور ایستا اعمال شود اگر و فقط اگر یک ماشین تورینگ $M_{\hat{P}}$ای وجود داشته باشد که یک ماشین برنامه M به عنوان ورودی می‌گیرد و با توجه به این‌که خاصیت روی M برقرار است یا خیر، در زمان متناهی تصمیم به پذیرش یا رد آن برنامه می‌گیرد. این تعریف، دقیقاً کلاس خاصیت‌های تصمیم‌پذیر بازگشتی را تداعی می‌کند. از آن‌جایی که هر خط‌مشی با این خصوصیات، مکمل شمارش‌پذیر بازگشتی نیز هست، پس می‌توان نتیجه گرفت که خاصیت‌های قابل اعمال با تحلیل ایستا، زیرمجموعه‌ای از خاصیت‌های قابل نظارت هستند.

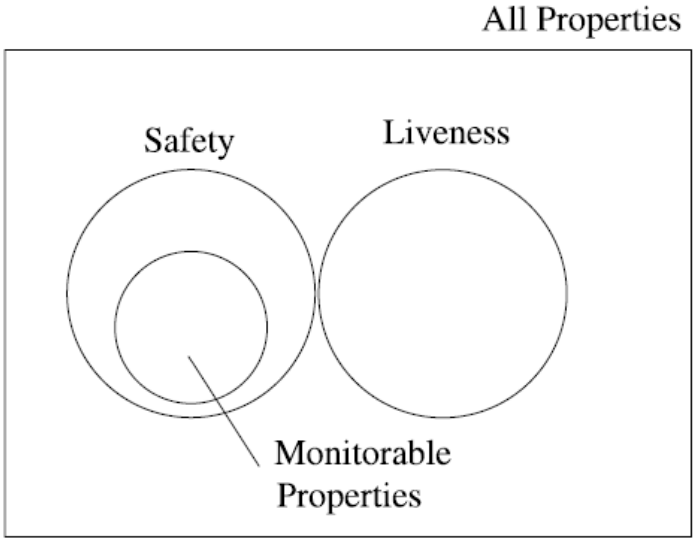
همچنین در [۱۴]، در خصوص کلاس خاصیت‌های قابل اعمال توسط مکانیزم بازنویسی برنامه مطالعاتی انجام شده است. البته در [۳۹]، سرشت‌نمایی در خصوص این روش اعمال صورت گرفته است که نتایج بیشتری حاصل شده است. برای یادآوری، روش بازنویسی برنامه مکانیزمی برای تغییر یک برنامه غیرمورد اعتماد برای سازگارکردن برنامه بازنویسی‌شده با خط‌مشی امنیتی است. در این روش نیز قدرت بازنویس وابسته به رابطه هم‌ارزی تعریف‌شده برای تبدیل برنامه‌ها است. با در نظر گرفتن M_1 و

[108] Recursively Enumerable

[109] Co-recursively Enumerable

[110] Program Machines

M_2 به عنوان دو ماشین برنامه، M_1 با M_2 هم‌ارز خواهد بود ($M_1 \approx M_2$) اگر و فقط اگر : $\forall \sigma \in \Sigma^{\infty}$ $M_1(\sigma) \cong M_2(\sigma)$.



شکل ۵ -خاصیت‌های قابل‌اعمال توسط ناظرهای [۳۸]

در صورتی که یک رابطه هم‌ارزی مشخص $\approx$ در نظر گرفته شود، خاصیت $\hat{P}$ توسط بازنویسی قابل اعمال خواهد بود اگر و فقط اگر یک تابع بازنویسی محاسبه‌پذیر $R \subseteq PM \times PM$ وجود داشته باشد که بتواند هر ماشین برنامه PMای را با توجه به خاصیت مورد نظر، به ماشین برنامه معتبری تبدیل کند. البته باید معناشناخت همه ماشین‌های برنامه تحت تابع بازنویسی حفظ شود. به بیان صوری:

۱) $\hat{P}(R(M))$ و

۲) $\hat{P}(M) \Longrightarrow M \approx R(M)$

با بررسی تعریف پارادایم‌های ارائه‌شده برای ناظرها، می‌توان مشاهده کرد که تعریف بالا معادل اعمال مؤثر تحت رابطه $\cong$ است. همچنین به طور شهودی قابل مشاهده است که هر خاصیت قابل اعمال با تحلیل ایستا را می‌توان با بازنویسی تحت رابطه هم‌ارزی $\approx$ (که مجموعه خط‌مشی‌های قابل اعمال آن به اختصار $RW_{\approx}$ نامیده می‌شود) نیز اعمال کرد. علاوه بر این، Hamlen و همکاران [۱۴] مطرح کرده‌اند که کلاس مکمل شمارش‌پذیر بازگشتی، کران بالایی برای مجموعه خاصیت‌های قابل اعمال توسط مکانیزم نظارت محسوب می‌شود. زمانی که یک ناظر نقضی را در خاصیت امنیتی تشخیص می‌دهد، باید برای جلوگیری از وقوع آن واکنش نشان دهد. این دخالت در اجرای برنامه با اضافه‌کردن تعدادی کنش

به دنباله ورودی انجام می‌شود. اگر I مجموعه همه دخالت‌های ممکن توسط ناظر در نظر گرفته شود، خاصیت $\hat{P}_I$ که اجازه انجام چنین دخالت‌هایی را نمی‌دهد، توسط ناظر قابل اعمال نخواهد بود. این در حالی است که اگر I یک مجموعه محاسبه‌پذیر باشد، این خاصیت مکمل شمارش‌پذیر بازگشتی است. ضمناً قابل اعمال‌بودن یا نبودن یک خاصیت توسط ناظر بستگی دارد که مسند $\hat{P}$ خاصیت مورد نظر را چگونه سرشت‌نمایی می‌کند.

سرشت‌نمایی بهتری از مجموعه خاصیت‌های قابل اعمال توسط مکانیزم نظارت اجرا، اشتراک کلاس مکمل شمارش‌پذیر بازگشتی و مجموعه $\mathrm{RW}_{\approx}$ است. این دسته از خاصیت‌ها دارای رفتار خاصی به نام *خیرخواهی*[111] هستند [۳۷]. یک خاصیت را خیرخواه می‌نامند اگر یک رویه تصمیم‌گیری $M_{\hat{P}}$ وجود دارد که برای همه ماشین‌های برنامه M، هر پیشوند نامعتبر از یک اجرای نامعتبر را رد می‌کنند، اما هر پیشوندی از یک اجرای معتبر را می‌پذیرد. همین ویژگی باعث می‌شود تا اگر ناظر تشخیص دهد که پیشوندهای معتبر به اجراهای نامعتبر تبدیل می‌شوند، آن‌ها را نپذیرد. به بیان صوری، یک خاصیت $\hat{P}$ خیرخواه است اگر رویه تصمیمی به نام $M_{\hat{P}}$ وجود داشته باشد به طوری که برای همه ماشین‌های برنامه M شرایط زیر برقرار باشد:

$$\neg(\forall \sigma \in X_M : \hat{P}(\sigma)) \Rightarrow (\forall \sigma \in pref(X_M) : (\neg\hat{P}(\sigma) \Rightarrow M_{\hat{P}}(\sigma) \text{ rejects}))$$

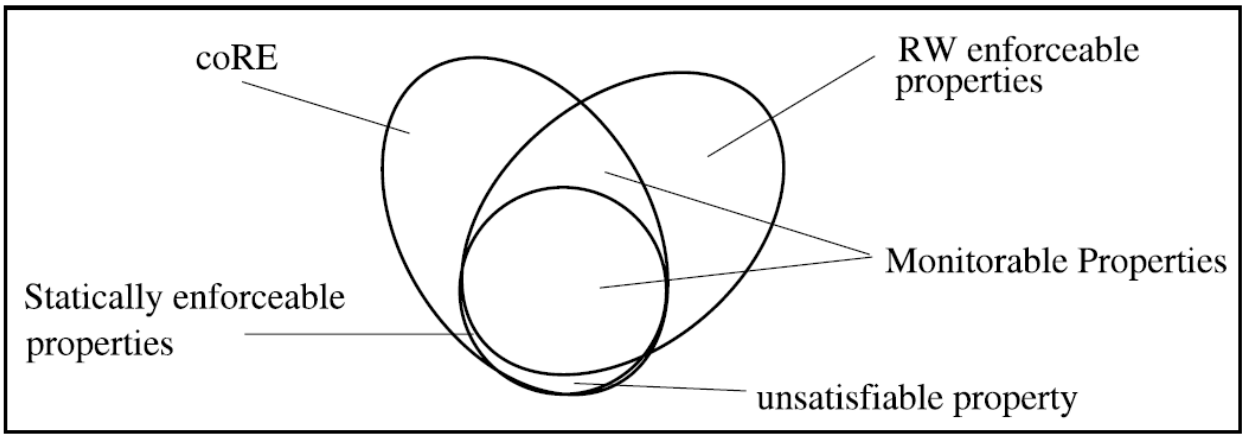
$$(\forall \sigma \in X_M : \hat{P}(\sigma)) \Rightarrow (\forall \sigma \in pref(X_M) : (M_{\hat{P}}(\sigma) \text{ accepts})).$$

طبق این تعابیر می‌توان مشاهده کرد که می‌توان ناظرها را به کد برنامه برد تا از اجراهای برنامه محافظت کند. به این ترتیب، می توان گفت که هر خاصیت قابل اعمال توسط ناظر، به کمک بازنویسی برنامه نیز قابل اعمال است.

شکل زیر نمایشی از دستاوردهای مقاله [۱۴] است. همان‌طور که پیشتر عنوان شد، هر خاصیت قابل اعمال با تحلیل ایستا، توسط روش نظارت و بازنویسی برنامه نیز قابل اعمال است. البته با این فرض که یک ناظر می‌تواند قبل از شروع اجرا، به تحلیل ایستای برنامه هدف دسترسی داشته باشد. مجموعه خاصیت‌های قابل اعمال توسط ناظرها عبارتند از اشتراک بین خاصیت‌های مکمل شمارش‌پذیر بازگشتی

[111] Benevolence

و $RW_{\approx}$ که این خاصیت‌ها دارای رفتار خیرخواهی هستند. همین ویژگی باعث می‌شود تا ناظر بتواند همه رفتارهای نامعتبر برنامه را قبل از وقوع آن‌ها، تشخیص و رد کند. با بیان این نتایج مشاهده می‌شود که کلاس خاصیت‌های قابل نظارت کوچکتر از مجموعه‌ای است که در بخش قبلی بیان شد. زیرا در بحث‌های پیشین، فرض بر آن بوده است که حتی اگر ناظر پس از وقوع نقض خط‌مشی از آن اطلاع یابد، می‌تواند آن را اعمال کند.



شکل ۶ -خاصیت‌های قابل‌اعمال توسط مکانیزم‌های مختلف معرفی‌شده در [۱۴]

در پایان مشخص شد که به دلیل توانایی بیشتر بازنویس‌ها برای تبدیل برنامه، مجموعه خاصیت‌های قابل اعمال با استفاده از بازنویسی برنامه، فوق‌مجموعه‌ای از خاصیت‌های قابل اعمال در مکانیزم‌های نظارت است. البته که این نتیجه با بحث مطرح‌شده در [۳۴] هم‌خوانی دارد. زیرا خودکاره ویرایش که توانایی تغییر معناشناخت یک برنامه را مشابه بازنویس‌ها دارد، از خودکاره‌های قطع‌کننده، توقیف و درج قدرت بیشتری داشت.

۵-۳-۲ محدودیت‌های حافظه‌ای

گرچه بحث‌های انجام‌شده درباره توانایی ناظرها بدون در نظر گرفتن محدودیت‌های حافظه‌ای بوده است، اما در دنیای واقعی، حافظه محدودی در اختیار ناظر قرار دارد تا بتواند برای اعمال خط‌مشی‌های امنیتی از آن استفاده کند. پس باید بررسی‌های بیشتری در این شرایط و با لحاظ‌کردن فرض‌های جدید صورت بگیرد. پژوهش‌گران از سه روش برای بیان این محدودیت‌ها استفاده کرده‌اند

[۳۷]: (۱) ناظرهایی با حافظه‌ای محدود با مقدار متناهی k، (۲) ناظرهایی با حافظه متناهی ولی نامحدود و (۳) ناظرهای که فقط مجموعه نامرتب کنش‌های قبلی برنامه را ثبت می‌کنند.

اولین مطالعات در این حوزه با معرفی خودکاره تاریخچه کم‌عمق[۱۱۲] (به اختصار SHA) توسط Fong [۴۰] صورت گرفته است. این خودکاره صرفاً مجموعه نامرتب رویدادهای انجام‌شده توسط برنامه هدف را ثبت می‌کند.

تعریف – یک خودکاره تاریخچه کم‌عمق، چهارتایی $\langle \Sigma. F(\Sigma). H_0. \delta \rangle$ به شرح زیر است:

- Σ مجموعه‌ای متناهی یا نامتناهی شمارا از رویدادها است.
- $F(\Sigma)$ مجموعه همه تاریخچه‌های کم‌عمق ممکن است.
- $H_0 \in F(\Sigma)$ تاریخچه دسترسی اولیه است که معمولاً تهی در نظر گرفته می‌شود.
- $\delta: F(\Sigma) \times \Sigma \to F(\Sigma)$ یک تابع گذار است که به شکل $\delta(H, a) = H \cup \{a\}$ تعریف می‌شود.

مجموعه خاصیت‌های قابل اعمال توسط این خودکاره (EM_{SHA}) زیرمجموعه محضی[۱۱۳] از مجموعه خاصیت‌های قابل اعمال توسط خودکاره امنیتی Schneider است. به بیان صوری، $EM_{SHA} \subset T^{\Sigma^*}$ - prcs.

دسته دیگری از کارهای انجام‌شده در زمینه خودکاره‌های از نظر حافظه‌ای محدود، توسط Talhi و همکارانش [۳۶] صورت پذیرفته است که منجر به طراحی خودکاره تاریخچه محدود[۱۱۴] (به اختصار BHA) شد. این خودکاره فضای محدودی برای ذخیره تاریخچه اجرای برنامه دارد. همچنین می‌توان ناظرهای دارای حافظه محدود را با خودکاره‌های امنیتی محدود[۱۱۵] (به اختصار BSA) [۲] و خودکاره‌های ویرایش محدود[۱۱۶] (به اختصار BEA) [۳۴] مدل‌سازی کرد.

---

[112] Shallow History Automaton

[113] Strict Subset

[114] Bounded History Automaton

[115] Bounded Security Automata

[116] Bounded Edit Automata

خودکاره تاریخچه محدود کلاسی از خودکاره‌های مورد استفاده برای مدل‌سازی خاصیت‌های قابل اعمال توسط ناظرها است که تنها می‌توانند از حافظه‌ای محدود برای ذخیره‌سازی تاریخچه اجراهای برنامه هدف بهره ببرند. هر حالتی از این خودکاره بیانگر انتزاعی با اندازه متناهی از دنباله ورودی است که تاکنون دریافت شده است. برای BHA دو زیرکلاس خودکاره‌های امنیتی محدود و خودکاره‌های ویرایش محدود قابل تصور است. اولی مشابه یک خودکاره قطع‌کننده است و دومی شبیه یک خودکاره ویرایش است که در هر دو تاریخچه‌ای محدود از کنش‌ها و رویدادها قابل نگه‌داری است. تاریخچه قابل نگه‌داری برای یک خودکاره ویرایش محدود شامل دو نوع دنباله است. دنباله اول، دنباله خروجی ناظر است و دنباله دوم توسط ناظر توقیف می‌شود تا یک پیشوند معتبر تشخیص داده شود.

بدیهی است که مجموعه خاصیت‌های قابل اعمال توسط یک خودکاره امنیتی محدود با مقدار k (به اختصار EM_{kSA}) زیرمجموعه‌ای از خاصیت‌های ایمنی است که توسط خودکاره‌های امنیتی نامحدود اعمال می‌شوند. از طرف دیگر، مجموعه خاصیت‌های قابل اعمال توسط خودکاره ویرایش محدود با مقدار k (به اختصار EM_{kEA}) زیرمجموعه‌ای از خاصیت‌های معقول تجدید نامتناهی است که خودکاره‌های ویرایش نامحدود توانایی اعمال آن‌ها را دارند. همچنین، هر چه حافظه بیشتری در اختیار ناظر قرار بگیرد، طبیعتاً می‌تواند مجموعه بزرگ‌تری از خاصیت‌های اشاره‌شده را اعمال کند.

قضیه [۳۶]– برای هر دو عدد صحیح k و k' به طوری که k<k' باشد، $EM_{kSA} \subset EM_{k'SA}$ و $EM_{kEA} \subset EM_{k'EA}$.

لازم به ذکر است که هر خودکاره تاریخچه کم‌عمق را می‌توان به یک خودکاره ویرایش محدود تبدیل کرد [۳۶]. همچنین، ارتباط نزدیکی بین مجموعه خاصیت‌های قابل اعمال توسط خودکاره تاریخچه کم‌عمق و خاصیت‌های آزمون‌پذیر محلی[۱۱۷] [۳۷] وجود دارد. این‌گونه خاصیت‌ها توسط دسته‌ای از خودکاره‌ها به نام پویشگرها[۱۱۸] قابل تشخیص هستند. پویشگرها خودکاره‌هایی هستند که حافظه‌ای متناهی و پنجره لغزانی به طول k دارند. در این خودکاره‌ها، تنها کنش‌های موجود در پنجره لغزان برای پویشگر قابل مشاهده است. به این ترتیب، خاصیت‌هایی توسط پویشگرها قابل اعمال هستند

[117] Locallay Testable Properties

[118] Scanner

که بتوان آن‌ها را فقط با مشاهده پیشوندها و پسوندهایی به طول کمتر از k تشخیص داد. چنین خاصیت‌هایی را می‌توان با روش نظارت نیز اعمال کرد. در صورتی که قرار باشد یک ناظر چندین خاصیت آزمون‌پذیر محلی را اعمال کند، می‌تواند سربار ناشی از ثبت کنش‌ها را به شدت کاهش دهد. انواع مختلفی از این خاصیت به نام‌های خاصیت‌های آزمون‌پذیر پیشوند[۱۱۹]، آزمون‌پذیر پسوند[۱۲۰]، آزمون‌پذیر پیشوند-پسوند[۱۲۱] و قویاً آزمون‌پذیر محلی[۱۲۲] مطرح شده است.

اثبات شده است [۳۶] که خاصیت‌های آزمون‌پذیر محلی که تحت پیشوند بسته باشند، توسط خودکاره‌های امنیتی محدود قابل اعمال هستند. در حالی‌که برای خاصیت‌های آزمون‌پذیر پسوند و آزمون‌پذیر پیشوند-پسوند نمی‌توان به طور کلی چنین مطلبی را بیان کرد. از طرفی، خودکاره ویرایش محدود که نسبت به خودکاره امنیتی محدود توانایی بیشتری دارد، می‌تواند خاصیت‌های بیشتری را اعمال کند. این خودکاره می‌تواند هر خاصیت آزمون‌پذیری را اعمال کند و شرط بسته‌بودن تحت پیشوند وجود ندارد. اما کماکان در حالت کلی، خاصیت‌های آزمون‌پذیر پسوند و آزمون‌پذیر پیشوند-پسوند با خودکاره‌های ویرایش محدود نیز قابل اعمال نیستند.

نتایج بالا نشان می‌دهد که می‌توان بین خاصیت‌های قابل اعمال توسط خودکاره‌های با محدودیت حافظه و کلاس‌هایی از زبان‌های صوری تناظرهایی برقرار کرد.

رویکرد دیگر برای بررسی توانایی ناظرهای دارای محدودیت حافظه‌ای، مدل‌سازی آن‌ها با خودکاره‌های متناهی است. در [۴۱] به بررسی مجموعه خاصیت‌های قابل اعمال توسط یک خودکاره ویرایش با قید حافظه پرداخته شده است. این خودکاره ویرایش دارای مجموعه‌ای متناهی، ولی نامحدود، از حالت‌ها است. البته تمرکز اصلی این مقاله به اعمال مؤثر روی محیط یکنواخت است و هر دو نوع دنباله متناهی و نامتناهی را تحت رابطه تساوی بررسی می‌کند. ارائه کلاس جدیدی از خاصیت‌ها به نام

[119] Prefix Testable

[120] Suffix Testable

[121] Prefix-Suffix Testable

[122] Strongly Locally Testable

خاصیت‌های حافظه‌محدود[۱۲۳] از دیگر دستاوردهای این کار به شمار می‌رود که مشابه مجموعه خاصیت‌های قابل اعمال در پارادایم مؤثر تحت رابطه تساوی توسط یک خودکاره ویرایش متناهی است.

از نماد $F_{=}^{\Sigma^{\infty}}$ -eff برای بیان مجموعه خاصیت‌های قابل اعمال در پارادایم مؤثر تحت رابطه تساوی توسط یک خودکاره ویرایش متناهی در یک زمینه یکنواخت استفاده می‌شود. به این ترتیب، طبق قضیه‌ای در [۴۱]، خاصیت‌های موجود در $F_{=}^{\Sigma^{\infty}}$ -eff همان خاصیت‌های حافظه‌محدود هستند و برعکس.

برای مقایسه این نتیجه با کار Ligatti [۳۳] و Tahli [۳۶] می‌توان گفت که $F_{=}^{\Sigma^{\infty}}$ -eff زیرمجموعه‌ای از خاصیت‌های تجدید است که خودکاره ویرایش بدون محدودیت‌های حافظه‌ای می‌تواند آن‌ها را اعمال کند. از طرف دیگر، فوق‌مجموعه‌ای برای مجموعه خاصیت‌های قابل اعمال توسط خودکاره ویرایش محدود با محدودیت‌های بیشتر است [۳۷]. به طور کلی، می‌توان مشاهده کرد که ناظرها با در نظر گرفتن محدودیت‌های حافظه‌ای هنوز می‌توانند گستره مناسبی از خاصیت‌های امنیتی را اعمال کنند. پس می‌توان ناظرها را در عمل نیز به کار بست.

۵-۴ تعابیر دیگری از مفهوم اعمال

از دیگر پژوهش‌های انجام‌شده بررسی تعابیر مطرح‌شده برای پارادایم‌های اعمال و ارائه پارایم‌های جدید است. به عنوان نمونه، تعبیر اعمال مؤثر، تعبیر کاملی نیست. زیرا شرط کافی برای ناظر وجود ندارد تا رفتار مناسبی در مواجهه با نقض خط‌مشی از خود نشان بدهد. این‌که هر بخش از اجرای ورودی که معتبر است، در خروجی ظاهر شود در این پارادایم رعایت نمی‌شود و ممکن است قسمت‌های امن یک اجرای نامعتبر توسط ناظر نیز رد شود. همان‌طور که در [۳۷] نیز عنوان شده است، چیزی که یک مکانیزم اعمال را از دیگری متمایز می‌کند، رفتار آن در برابر دنباله‌های اجرای معتبر نیست؛ بلکه چگونگی تبدیل یک اجرای بد به اجرای خوب اهمیت دارد. گرچه معیار صحت، لازمه یک مکانیزم اعمال است اما ارائه مکانیزمی که به دنباله‌های نامعتبر نیز توجه داشته باشد، در عمل کاراتر خواهد بود. به

[123] Memory Bounded Properties

همین خاطر، زیرکلاس‌هایی از خودکاره ویرایش در [۴۲] مطرح می‌شود که بتوان دقیق‌تر درباره توانایی اعمال ناظرها اظهارنظر کرد. در ادامه تعریف تعدادی از این خودکاره‌ها آمده است.

تعریف – خودکاره معوق[۱۲۴] A ، یک خودکاره ویرایش است که شرط زیر را نیز داشته باشد:

$$\forall \sigma \in \Sigma^* : A(\sigma) \preceq \sigma$$

دلیل نام‌گذاری این خودکاره آن است که ظاهرشدن کنش‌های ورودی را تا تشکیل‌شدن یک پیشوند معتبر به تعویق می‌اندازد.

در بیان خودکاره‌های پیشین، این محدودیت وجود داشت که در هر گام، ناظر باید یا همه کنش‌های توقیف‌شده را به خروجی بدهد یا کنش فعلی را توقیف کند و خروجی ندهد؛ یعنی ناظر نمی‌تواند تنها بخشی از کنش‌های توقیف‌شده را به خروجی بدهد و یا این‌که در هنگام توقیف کنش فعلی، کنش توقیف‌شده دیگری را در خروجی ظاهر کند.

تعریف – خودکاره همه-یا-هیچ[۱۲۵] A ، یک خودکاره ویرایش است که شرط‌های زیر را نیز دارا باشد:

$$\forall \sigma \in \Sigma^* : A(\sigma) \preceq \sigma \quad \wedge \quad (\forall \sigma \in \Sigma^* : \forall a \in \Sigma : A(\sigma; a) = \sigma; a \vee A(\sigma; a) = A(\sigma))$$

دسته دیگری از خودکاره‌های ویرایش شرط دیگری را نیز اضافه دارد و همواره طولانی‌ترین پیشوند از دنباله ورودی را به عنوان خروجی می‌دهد.

تعریف – خودکاره لیگاتی برای خاصیت $\hat{P}$ [۱۲۶]، یک خودکاره ویرایش است که شرط‌های زیر باید در آن برقرار باشد:

$$\forall \sigma \in \Sigma^* : A(\sigma) \preceq \sigma \quad \wedge$$
$$\left(\forall \sigma \in \Sigma^* : \forall a \in \Sigma : A(\sigma; a) = \sigma; a \vee A(\sigma; a) = A(\sigma)\right) \quad \wedge$$
$$\forall \sigma \in \Sigma^\infty : \hat{P}(\sigma) \Rightarrow A(\sigma) = \sigma.$$

دسته دیگر، خودکاره ویرایشی است که پیشوندی معتبر از دنباله اجرای ورودی معتبر به عنوان خروجی می‌دهد و در حالت دیگر، ناظر محدودیتی ندارد.

---

[124] Delayed Automaton

[125] All-Or-Nothing Automaton

[126] Liggati's Automaton for Property $\hat{P}$

تعریف – خودکاره معوق برای خاصیت $\hat{P}$ [۱۲۷]، یک خودکاره ویرایش است که شرط زیر را نیز دارد:

$$\forall \sigma \in \Sigma^* : A(\sigma) \preceq \sigma \quad \vee \quad \hat{P}(\mathrm{A}(\sigma))$$

از آن‌جایی که تعبیر اعمال مؤثر به نظر کافی نمی‌رسید، مفهوم جدیدی از اعمال به نام اعمال دقیق معوق[۱۲۸] [۴۲] مطرح شد. در این پارادایم اعمال، ناظر باید همواره دنباله‌ای معتبر را به خروجی بدهد و همیشه پیشوندی از دنباله ورودی باشد. در صورت معتبربودن دنباله ورودی، خروجی باید از نظر نحوی معادل باشد.

تعریف – خودکاره $\mathrm{A}=\langle Q.\Sigma.q_0.\delta.\omega\rangle$ به طور دقیقِ معوق خاصیت $\hat{P}$ را اعمال می‌کند اگر و فقط اگر $\forall \sigma \in \Sigma^{\infty}$:

۱) $(\mathrm{q}_0, \sigma) \overset{\sigma'}{\Rightarrow}_{\mathrm{A}} (\mathrm{q}', \epsilon)$

۲) $\hat{P}(\sigma')$

۳) $\hat{P}(\sigma') \Rightarrow \sigma = \sigma' \wedge \forall \mathrm{i} \in \mathbb{N}: \exists \mathrm{j} \in \mathbb{N}: \mathrm{j} \leq \mathrm{i}: \exists \mathrm{q}* \in \mathrm{Q}: (\mathrm{q}_0, \sigma) \overset{\sigma[0..j]}{\Longrightarrow}_{\mathrm{A}} (\mathrm{q}^*, \sigma[\mathrm{i}+1..])$

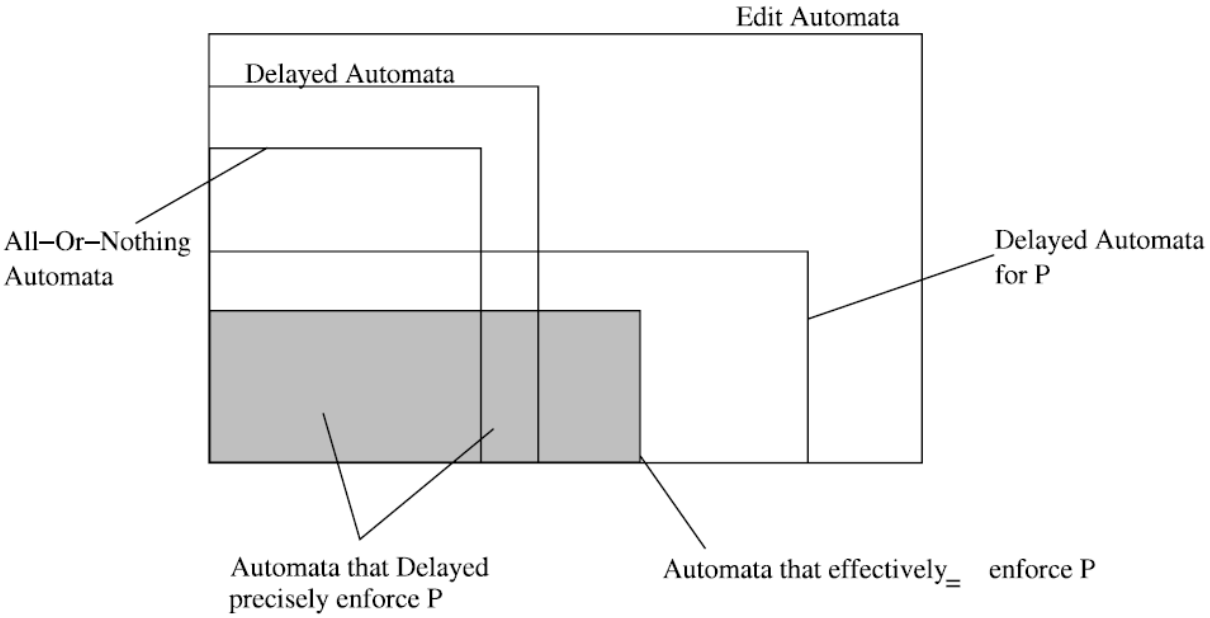
در واقع چنین تعبیری بین اعمال دقیق و اعمال مؤثر تحت رابطه تساوی است. هر خودکاره‌ای که می‌تواند به طور دقیق معوق خاصیتی را اعمال کند، می‌تواند آن خاصیت را به طور مؤثر تحت رابطه تساوی نیز اعمال کند.

در بررسی قدرت خودکاره‌های معرفی‌شده می‌توان گفت [۳۷] که خودکاره معوق و خودکاره معوق برای خاصیت $\hat{P}$ با یکدیگر اشتراک دارند اما یکی زیرمجموعه دیگری نیست. همچنین، یک خودکاره همه-یا-هیچ، یک خودکاره معوق است اما لزوماً یک خودکاره معوق برای خاصیت $\hat{P}$ نیست. به طور مشابه، یک خودکاره معوق حتی اگر به طور دقیق معوق خاصیتی را اعمال کند، حتماً یک خودکاره لیگاتی برای خاصیت $\hat{P}$ نخواهد بود. ضمناً کلاس خودکاره‌های ویرایشی که به طور مؤثر تحت رابطه تساوی اعمال می‌کنند، زیرکلاسی از خودکاره معوق برای خاصیت $\hat{P}$ به شمار می‌روند. علاوه بر این، کلاس خودکاره‌های ویرایشی که به طور دقیق معوق خاصیتی را اعمال می‌کنند، برابر اشتراک کلاس خودکاره‌های معوق، کلاس خودکاره‌های معوق برای خاصیت $\hat{P}$ و کلاس خودکاره‌های ویرایشی است که می‌توانند خاصیت $\hat{P}$ را به طور مؤثر تحت رابطه تساوی اعمال کنند. به علاوه، هر خودکاره همه-یا-هیچ

[127] Delayed Automaton for Property $\hat{P}$

[128] Delayed Precise Enforcement

برای اعمال خاصیت $\hat{P}$ معادل خودکاره لیگاتی برای خاصیت $\hat{P}$ است. در شکل زیر، خلاصه نتایج مطرح‌شده آمده است.



شکل ۷ –مقایسه زیرکلاس‌های مختلف خودکاره ویرایش [۴۲]

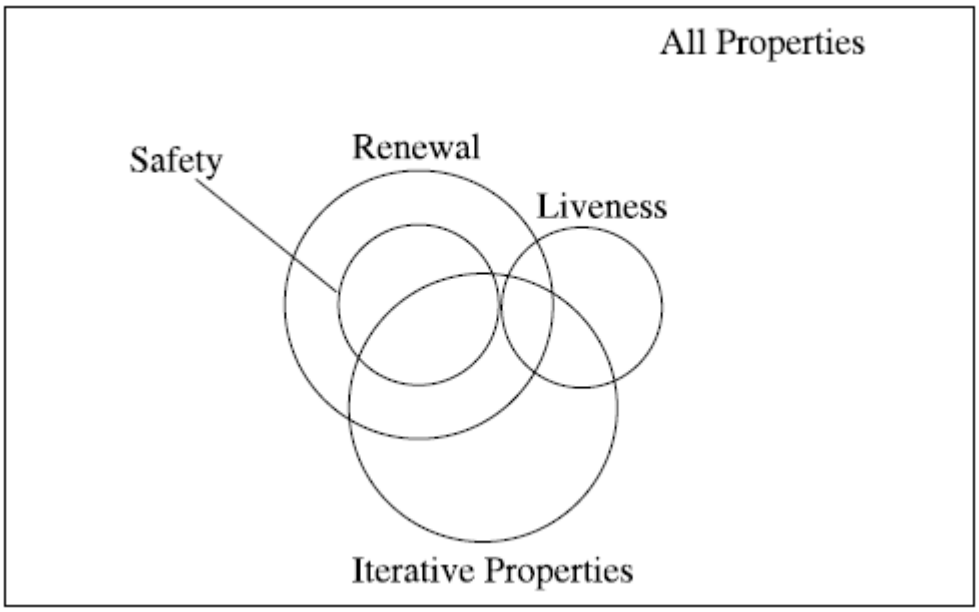
رویکرد دیگر در تولید خروجی‌های معتبر این است که پیشوندی از ورودی نباشند. در این رابطه، کلاس دیگری از خاصیت‌ها به نام خاصیت‌های تکرارشونده[۱۲۹] [۳۷] معرفی شده‌اند. این خاصیت‌ها اجراهای مکرر تراکنش‌ها را مدل می‌کنند.

تعریف - خاصیت $\hat{P}$ یک خاصیت تکرارشونده است اگر و فقط اگر

$$\forall \sigma . \sigma' \in \Sigma^* : \hat{P}(\sigma) \wedge \hat{P}(\sigma') \Rightarrow \hat{P}(\sigma . \sigma')$$

خاصیت‌های تکرارشونده برای مدل‌سازی رفتارهای سامانه‌هایی ارائه شده است که قرار است به دفعات تعدادی تراکنش متناهی انجام دهند. کلاس خاصیت‌های تکرارشونده شامل بخشی از خاصیت‌های ایمنی، مانایی و تجدید می‌شود.

[129] Iterative Properties



شکل ۸ -خاصیت‌های تکرارشونده [۳۷]

تعبیر دیگری از اعمال مبتنی بر همین نوع از خاصیت‌ها مطرح شده است. یک ناظر به طور تکرارشونده با استفاده از توقیف یک خاصیت تکرارشونده $\hat{P}$ را اعمال می‌کند اگر و فقط اگر هر تراکنش معتبر در خروجی ظاهر شود و هر تراکنش نامعتبر توقیف شود. همچنین نحوه ساخت یک خودکاره ویرایش که بتواند به طور تکرارشونده یک خاصیت تراکنش را اعمال کند، وجود دارد. [۳۷] همان‌طور که مشاهده می‌شود، تعبیر اعمال می‌تواند برای ناظر متفاوت باشد و همین موضوع در توانایی و قدرت ناظر در اعمال خط‌مشی‌های امنیتی تأثیرگذار است.

۵-۴-۱ اعمال اصلاحی

در ادامه کارهای انجام‌شده برای بهبود تعبیر اعمال خط‌مشی‌های امنیتی، اعمال اصلاحی[۱۳۰] [۴۳] پیشنهاد شد. همان‌طور که قبلاً نیز اشاره شد، در اعمال مؤثر هیچ محدودیتی روی رفتار ناظر در برخورد با یک دنباله نامعتبر وجود ندارد. به عنوان مثال، اگر یک ناظر در پارادایم اعمال مؤثر به جای یک دنباله نامعتبر، دنباله تهی را خروجی دهد، معیار صحت برای این ناظر برقرار است اما واضح است که این تغییر

[130] Corrective Enforcement

در اجرا، مورد نظر سامانه نبوده است؛ بلکه در عمل انتظار می‌رود تا با کم‌ترین تغییرات ممکن نسبت به دنباله ورودی، یک دنباله خروجی معتبر تولید شود. به این ترتیب، باید رفتارهای معتبر موجود در دنباله ورودی نامعتبر تا حد امکان حفظ شوند. پس باید محدودیت‌هایی به رفتار ناظر افزوده شود تا این قابلیت را نیز داشته باشد.

تعریف – یک خودکاره ویرایش A به طور اصلاحی تحت رابطه هم‌ارزی $\cong$ خاصیت $\hat{P}$ را اعمال می‌کند اگر و فقط اگر $\forall \sigma \in \Sigma^{\infty}$:

۱) $\hat{P}(A(\sigma))$

۲) $A(\sigma) \cong \sigma$

طبق تعریف بالا، ناظری می‌تواند یک خاصیت را به طور اصلاحی تحت رابطه هم‌ارزی اعمال کند اگر و فقط اگر برای هر دنباله ممکن ورودی، دنباله معتبر معادلی وجود داشته باشد که تعداد متناهی یا نامتناهی پیشوند معتبر داشته باشد و تبدیل از دنباله ورودی به دنباله معتبر خروجی محاسبه‌پذیر باشد.

باید دقت داشت که بین رابطه هم‌ارزی در اعمال مؤثر و اصلاحی تفاوت وجود دارد [۳۷]. اگر یک رابطه هم‌ارزی شرط غیرقابل‌تمایزبودن پیشنهادشده توسط Ligatti را داشته باشد، نمی‌توان از آن رابطه برای اعمال اصلاحی استفاده کرد. زیرا یک پیشوند نامعتبر را نمی‌توان با در نظر گرفتن چنین رابطه هم‌ارزی به پیشوندی معتبر تبدیل کرد. چون باید پیشوند نامعتبر ورودی *اصلاح* شود و سپس به عنوان خروجی داده شود.

اگر $F: \Sigma^{*} \rightarrow 1$ تابع انتزاعی در نظر گرفته شود که به ازای هر دنباله ورودی به ناظر، رفتاری از دنباله را برمی‌گرداند که باید توسط ناظر حفظ شود، شرطی که رابطه هم‌ارزی برای اعمال اصلاحی باید داشته باشد به شکل زیر خواهد بود:

$$F(\sigma) = F(\sigma') \Rightarrow \hat{P}(\sigma) \Leftrightarrow \hat{P}(\sigma')$$

طبق بیان صوری بالا، رابطه هم‌ارزی مورد استفاده برای پارادایم اعمال اصلاحی باید علاوه بر سازگاری با خط‌مشی امنیتی، با انتزاع مورد نظر از رفتار برنامه نیز منطبق باشد.

در چنین شرایطی می‌توان مشاهده کرد که اعمال تکرارشونده با استفاده از توقیف و اعمال مؤثر تحت رابطه تساوی، حالت‌های خاصی از پارادایم اعمال اصلاحی تحت رابطه هم‌ارزی محسوب می‌شوند. برای تبدیل اعمال اصلاحی به اعمال مؤثر تحت رابطه تساوی کافی است تا رابطه هم‌ارزی $\preccurlyeq\cong$ در نظر

گرفته شود. این رابطه چنین تعریف می‌شود که $\forall \sigma. \sigma' \in \Sigma^* : \sigma \cong_{\preccurlyeq} \sigma' \Leftrightarrow pref(\sigma) \cap \hat{P} = pref(\sigma') \cap \hat{P}$. طبق این تعریف دو دنباله تحت خاصیت $\hat{P}$ معادل یکدیگر هستند اگر و فقط اگر مجموعه پیشوندهای معتبر مشابه داشته باشند.

قضیه [۴۴] – خاصیت $\hat{P}$ قابل اعمال در پارادایم مؤثر تحت رابطه تساوی است اگر و فقط اگر به طور اصلاحی تحت رابطه هم‌ارزی $\cong_{\preccurlyeq}$ قابل اعمال باشد.

همچنین لازم به ذکر است که مجموعه خاصیت‌های قابل اعمال در پارادایم اصلاحی با اختیار داشتن اطلاعات ایستا و تعریف دقیق‌تر رابطه هم‌ارزی قابل گسترش است.

با لحاظ‌کردن توانایی تبدیل دنباله‌های نامعتبر، پارادایم اصلاحی می‌تواند در اعمال خاصیت‌های امنیتی، نزدیک‌تر به شهود رفتار کند. گرچه این تعبیر نیز اشکالاتی دارد. یکی از اشکال‌های این نحوه تعبیر آن است که چندین دنباله معتبر مختلف که می‌توانند تبدیل‌هایی از یک دنباله نامعتبر باشند، باید معادل یکدیگر باشند. به بیان دیگر، اگر یک دنباله معتبر اصلاح‌شده چند دنباله نامعتبر مختلف باشد، آن دنباله‌ها هم‌ارز تلقی می‌شوند.

به همین خاطر در [۴۳] تعبیر دیگری از اعمال اصلاحی به نام اعمال اصلاحی تحت رابطه ترتیب جزئی مطرح شد. در اعمال اصلاحی، یک تابع انتزاع معرفی شد که خروجی آن بیانگر جنبه‌هایی از دنباله اجرا بود که باید توسط ناظر حفظ می‌شد. حال در تعبیر اخیر، به جای رابطه هم‌ارزی از ترتیب جزئی استفاده می‌شود و دنباله‌ها با توجه به ترتیب جزئی $\sqsubseteq$ مرتب می‌شوند. به این ترتیب، ناظر می‌تواند دنباله ورودی σ را به دنباله σ' تبدیل کند اگر و فقط اگر $\sigma \sqsubseteq \sigma'$.

تعریف – با فرض این‌که A یک خودکاره ویرایش و $\sqsubseteq$ بیانگر یک ترتیب جزئی روی دنباله‌های Σ^∞ باشد، خودکاره A خاصیت $\hat{P}$ را به طور اصلاحی تحت ترتیب جزئی اعمال می‌کند اگر و فقط اگر $\forall \sigma \in \Sigma^\infty$:

۱) $\hat{P}(A(\sigma))$
۲) $\sigma \sqsubseteq A(\sigma)$

بهره‌گیری از ترتیب جزئی به جای رابطه هم‌ارزی باعث برطرف‌شدن نقاط ضعف مطرح‌شده برای اعمال اصلاحی می‌شود. از طرف دیگر، در اعمال مؤثر نیز اشاره شد که رابطه هم‌ارزی بیان خط‌مشی‌های امنیتی را محدود می‌سازد اما بسیاری از خط‌مشی‌های امنیتی با کمک ترتیب جزئی بیان می‌شوند. همچنین، می‌توان اعمال اصلاحی تحت رابطه هم‌ارزی را حالت خاصی از تعبیر جدید دانست؛ گرچه

می‌توان یکی از معایب این پارادایم را چنین گفت که کاربر باید به ازای هر خاصیت امنیتی، محدودیت‌های مورد نظر برای اعمال روی دنباله‌های نامعتبر را تعیین کند که کار ساده‌ای نخواهد بود و بستگی به نوع خاصیت دارد.

پس از ارائه این تعبیر، مفهوم دیگری به نام پیش‌بینی‌پذیری[۱۳۱] معرفی شد [۳۷]. در این تعبیر، رفتار ناظر در مواجهه با دنباله‌های نامعتبر از محدودیت‌های نحوی آن‌ها پیروی می‌کند؛ به این معنا که یک مکانیزم اعمال پیش‌بینی‌پذیر است اگر هر دنباله ورودی نامعتبر که نزدیک[۱۳۲] یک دنباله معتبر است، به دنباله‌ای که نزدیک همان دنباله معتبر است تبدیل شود. معیار و حد آستانه نزدیکی باید از پیش تعریف شود. بیان صوری این تعبیر در ادامه آمده است.

تعریف – اگر A یک خودکاره و d و d' اندازه فاصله بین دو دنباله باشد، یک مکانیزم اعمال را با مقدار فاصله ε پیش‌بینی‌پذیر گویند اگر و فقط اگر به ازای هر دنباله اجرا $\sigma \in \hat{P}$ رابطه زیر برقرار باشد:

$$\forall v \in \mathbb{R}: v \geq \mathcal{E} : \exists \delta \in \mathbb{R}: \delta > 0: \forall \sigma' \in \Sigma^*: \big(d(\sigma.\sigma') \leq \delta \Rightarrow d'\big(A(\sigma).A(\sigma')\big) \leq v\big)$$

به بیان غیرصوری، برای هر دنباله معتبر، شعاعی به نام δ وجود دارد به طوری که همه دنباله‌های اجرا در این شعاع به دنباله‌های اجرایی تبدیل می‌شوند که در شعاع ε از همان دنباله قرار دارند. مزیت این تعبیر نسبت به تعبیر قبلی این است که نیازی به تعریف صریح محدودیت رفتار برای ناظر به ازای هر خاصیت نیست.

[131] Predictability

[132] Close

فصل ششم

سرشت‌نمایی ناظرهای زمان‌اجرای با اطلاعات ایستا

سرشت‌نمایی ناظرهای زمان‌اجرای با اطلاعات ایستا

واژه سرشت‌نمایی به معنای نشان‌داده طبیعت و نهاد یک شیء است. منظور از سرشت‌نمایی صوری ارائه تعریف، ابهام‌زدایی، توصیف و اثبات خصوصیات یک موضوع با استفاده از روش‌های صوری است [۳۹]. سرشت‌نمایی خط‌مشی‌های امنیتی قابل اعمال توسط ناظرهای زمان‌اجرا در سال‌های اخیر مورد توجه قرار گرفته است. پیشتر پژوهش‌های سرشت‌نمایی خط‌مشی‌های امنیتی قابل اعمال با ناظرها محدود بود به ناظرهایی که هیچ دانش قبلی از رفتارهای ممکن برنامه نداشتند و کلاس خاصی از خط‌مشی‌ها به نام خاصیت‌ها را در بر می‌گرفت. حال با در نظر گرفتن اطلاعات ایستا، که تقریبی نادقیق از اجراهای ممکن برنامه است، به بررسی قدرت ناظرها پرداخته می‌شود. ضمناً به انواع مختلف خط‌مشی‌ها توجه می‌شود و نه صرف خاصیت‌ها. به این ترتیب، یک ناظر زمان‌اجرا، یک تقریب از اجراهای برنامه و یک خودکاره می‌گیرد و اجراهای منفرد را به اجراهای امن تبدیل می‌کند. به کمک بازتعریف پارادایم‌ها، کلاس‌های جدیدی از اعمال مؤثر[۱۳۳] و اعمال دقیق[۱۳۴] ارائه می‌شود تا بتوان برای همه خط‌مشی‌ها از آن‌ها استفاده کرد.

ناظرهای زمان‌اجرا از طریق بررسی و تغییر اجراهای برنامه‌ها، خط‌مشی‌های امنیتی را اعمال می‌کنند. سرشت‌نمایی خط‌مشی‌های قابل اعمال در کارهای [۱۴]، [۲]، [۳۳] مورد توجه بوده است. اما تمرکز اصلی روی ناظرهایی بوده است که هیچ دانش قبلی‌ای از رفتارهای ممکن برنامه‌ها نداشته‌اند. همین موضوع باعث می‌شود تا تنها کلاس خاصی از خط‌مشی‌ها، به نام خاصیت‌ها، مورد تحقیق قرار بگیرد. منظور از خاصیت‌ها، خط‌مشی‌هایی هستند که مجموعه‌ای از اجراهای معتبر را تعریف می‌کنند. خاصیت‌های قابل اعمال طبق توانایی ناظر در تبدیل برنامه‌ها [۳۴]، [۳۵] و نیز تعبیر اعمال [۳۶–۳۸] دسته‌بندی شده‌اند. حال سوال این‌جاست که آیا با اضافه‌شدن اطلاعات ایستا به دانش ناظر، قدرت آن در اعمال خط‌مشی‌ها بیشتر خواهد شد؟

---

[133] Effective Enforcement

[134] Precise Enforcement

۶-۱ تعاریف و مقایسه با کارهای قبلی

اساساً می‌توان خط‌مشی‌های امنیتی را به دو دسته خاصیت‌ها و ناخاصیت‌ها[۱۳۵] تقسیم‌بندی کرد. ناخاصیت‌ها خط‌مشی‌هایی هستند که نمی‌توان آن‌ها را با مجموعه‌ای از اجراهای معتبر سرشت‌نمایی کرد؛ بلکه باید توسط خانواده‌ای از مجموعه اجراهای معتبر مشخص کرد.

پیشتر نشان داده شده است که با داشتن اطلاعات ایستا، ناظرها قادر به اعمال خاصیت‌های بیشتری هستند. با دریافت پیشوندی از یک اجرا، ناظر می‌تواند از این دانش برای پیش‌بینی دستورات بعدی استفاده کند. به این ترتیب ناظر می‌تواند تصمیم‌های دقیق‌تری اتخاذ کند و خاصیت‌های بیشتری اعمال شود [۳۵]، [۴۵].

در مقاله [۴۶]، تقریبی از مجموعه اجراهای ممکن برنامه هدف به عنوان ورودی به ناظر داده می‌شود. این تقریب، از تحلیل ایستای برنامه به دست می‌آید. از آن‌جایی که در حالت کلی، مجموعه دقیق اجراهای ممکن برنامه محاسبه‌پذیر نیست، فرض بر این است که به طور محافظه‌کارانه تقریبی از مجموعه اجراهای برنامه به دست می‌آید. پس مجموعه واقعی همه اجراهای ممکن برنامه بخشی از تقریب داده‌شده خواهد بود.

منظور از پارادایم اعمال، قواعدی است که روش مجاز تبدیل مجموعه اجراهای برنامه توسط ناظر را کنترل می‌کند. پارادایم‌های اعمال مطرح‌شده قبلی روی نحوه تبدیل اجراهای منفرد، و نه نحوه تبدیل مجموعه اجراها، محدودیت می‌گذارند. این در حالی است که ناخاصیت‌هایی مانند خط‌مشی جریان اطلاعات به دنبال برقراری رابطه‌ای روی مجموعه اجراهای برنامه هستند و این سوال که آیا یک اجرا به تنهایی امن است، سوال بی‌معنی‌ای خواهد بود. پس برای اعمال یک ناخاصیت لازم است ناظر به‌گونه‌ای عمل کند که مجموعه اجراها تبدیل شوند.

یک ناظر باید درست باشد؛ به این معنا که مجموعه اجراهایی که در پاسخ به یک مجموعه اجراهای ورودی تولید می‌کند، باید خط‌مشی را برآورده کند. همچنین، ناظر باید شفاف باشد؛ از این جهت که معناشناخت برنامه با توجه به خط‌مشی باقی نگه‌داشته شود. با رابطه‌ای بین مجموعه اجراهای

---

[135] Nonproperties

تولیدشده توسط ناظر و مجموعه اجراهای ورودی به آن، می‌توان شفافیت را صوری کرد. تصمیم‌گیری در خصوص همین رابطه بین مجموعه ورودی-خروجی برای بیان شفافیت نظارت، مفهوم اولیه‌ای از اعمال را مطرح می‌کند. با توجه به سوال پژوهشی مقاله، تعابیر اعمال مؤثر و دقیق ارائه‌شده در [۳۵]، بازتعریف می‌شود تا بتوان برای خط‌مشی‌ها، چه خاصیت و چه ناخاصیت، به کار بست.

تعبیر جدید اعمال مؤثر به این شکل است که اگر مجموعه ورودی خط‌مشی را برآورده می‌کند، مجموعه اجراهای خروجی از نظر نحوی معادل با مجموعه اجراهای ورودی باشند. همچنین تعبیر جدید اعمال دقیق به این معنا است که برای برنامه‌های امن، علاوه بر تعبیر ارائه‌شده در اعمال مؤثر، ناظر باید بلافاصله پس از هر دستور[۱۳۶] برنامه ورودی، خروجی مورد نظر را فراهم کند. میزان دقیق‌بودن[۱۳۷] اطلاعات ایستا را می‌توان به عنوان پارامتری از پارادایم اعمال در نظر گرفت.

نشان داده می‌شود که ناخاصیت‌ها در این پارادایم‌ها، قابل اعمال نیستند. ضمناً ناظرهای زمان‌اجرا با اطلاعات ایستای نادقیق، همان مجموعه از خاصیت‌ها را اعمال می‌کند که با ناظرهای بدون اطلاعات ایستای سنتی قابل اعمال هستند. به بیان دیگر، باید اطلاعات ایستا دقیق باشد تا توانایی اعمال ناظر بیشتر شود. البته نشان داده می‌شود که خاصیت‌هایی که توسط ناظرهای سنتی قابل اعمال نبودند و بعضی از ناخاصیت‌ها مانند فوق‌خاصیت‌های k-ایمنی [۴۷]، توسط این ناظرها قابل اعمال است. گرچه کماکان خط‌مشی‌هایی هستند که در این پارادایم‌ها قابل اعمال نیستند، حتی اگر مجموعه دقیق اجراهای ممکن برنامه نیز در اختیار ناظر باشد.

علاوه بر معیارهای درستی، شفافیت و دانش قبلی از رفتارهای ممکن برنامه، مجموعه خط‌مشی‌هایی که یک ناظر اجرایی می‌تواند اعمال کند به عوامل دیگری نیز بستگی دارد. ابزاری که ناظر با آن اجرا را تغییر می‌دهد و محدودیت‌های حافظه‌ای و محاسباتی از جمله این عوامل هستند. در مقاله [۴۶]، فرض بر آن است که هیچ محدودیتی روی ابزار ناظر برای تغییر اجرا و حافظه ناظر وجود ندارد. البته روی محاسبه‌پذیری محدودیت‌هایی گذاشته شده است. یک ناظر را می‌توان یک تابع محاسبه‌پذیر دانست اگر با مجموعه قابل محاسبه‌ای از اجراها شامل اجراهای ممکن برنامه هدف تهیه شده باشد.

---

[136] In lockstep

[137] Accuracy

۶-۱-۱ مقایسه با کارهای قبلی

سرشت‌نمایی خاصیت‌های قابل اعمال توسط کلاس‌های مختلفی از ناظرها در بعضی از پارادایم‌های اعمال خط‌مشی امنیتی در [۳۷] مطرح شده است. پژوهش در این حوزه از کار Schneider [۲] آغاز شده است که ناظر را بدون هیچ دانشی نسبت به آینده رفتار برنامه تصور می‌کند و تنها قابلیت قطع اجرا را توسط خودکاره قطع‌کننده می‌دهد. البته فرض بر قواعد موجود در اعمال دقیق بوده است. همچنین نشان داده می‌شود که خط‌مشی‌هایی که توسط خودکاره قطع‌کننده قابل اعمال در پارادایم دقیق هستند، همان خاصیت‌های ایمنی‌اند. در [۳۴]، [۳۵] عنوان می‌شود که ناظری با قابلیت ویرایش یک اجرا می‌تواند کلاس بزرگ‌تری از خاصیت‌ها را اعمال می‌کند.

یکی از پارادایم‌های معرفی‌شده در کارهای گذشته، اعمال $\precsim$effectively بوده است. در این اعمال، ناظرها باید اجراهای معتبر را به اجراهای معادلی تبدیل کنند که رابطه هم‌ارزی $\simeq$ بین آن‌ها برقرار است. تاکنون فقط رابطه هم‌ارزی نحوی تساوی مورد بررسی قرار گرفته است [۱۹]، [۳۳]. اثبات می‌شود که یک کران پایین برای مجموعه خط‌مشی‌های خودکاره‌های ویرایش که می‌توانند $\precsim$effectively را اعمال کنند، مجموعه خاصیت‌های معقول تجدید نامتناهی[۱۳۸] است.

اعمال دقیق و مؤثر محدودیتی روی ناظر برای چگونگی تبدیل برنامه از اجراهای نامعتبر به معتبر ندارد. البته منطقی است که تغییرات در اجراهای نامعتبر حداقل باشد [۴۲]. البته بعضی از پارادایم‌های اعمال ارائه‌شده می‌خواهند حتما رابطه مشخصی بین اجرای ورودی به ناظر و اجرای خروجی متناظر وجود داشته باشد، فارغ از این‌که اجرای ورودی معتبر بوده است یا نه [۴۴]. پارادایم دیگر، اعمال اصلاحی[۱۳۹] [۴۳]، [۴۸] است. این پارادایم می‌خواهد که اجرای تبدیل‌شده رابطه پیش‌ترتیبیِ[۱۴۰] خاصی برقرار باشد. به این ترتیب مشخص می‌شود که لازم است کدام رفتارهای اجرای ورودی باید حفظ شوند.

[138] Reasonable Infinite Renewal

[139] Corrective Enforcement

[140] Preorder Relation

محور دیگر مقایسه ناظرها، میزان دانش ناظرها درباره رفتارهای ممکن برنامه‌های هدف است. یک ناظر ممکن است در زمینه غیریکنواخت[۱۴۱] عمل کند؛ به این معنا که ناظر می‌داند برنامه‌ها هیچ‌گاه یک رفتار مشخص را انجام نمی‌دهند. یک زمینه غیریکنواخت به صورت زیرمجموعه‌ای از اجراهای جهانی[۱۴۲] نمایش داده می‌شود و برای همه برنامه‌های هدف، یکسان در نظر گرفته می‌شود. پس ناظرها در زمینه غیریکنواخت می‌توانند مجموعه بزرگتری از خط‌مشی‌ها را اعمال کنند [۳۵]، [۴۵].

محدودیت‌های حافظه و محاسباتی نیز از عوامل تعیین‌کننده در قدرت یک ناظر به شمار می‌رود. به همین دلیل نتایج کلی مطرح‌شده برای مقایسه توانایی‌های ناظرها، در واقع کران بالایی است از آن‌چه که واقعاً می‌توانند اعمال کنند. پژوهش‌های موجود در [۱۴]، [۳۶]، [۴۰]، [۴۱] به خوبی به این محدودیت‌ها پرداخته‌اند.

در [۴۹]، به توانایی ناظرها برای اعمال خط‌مشی‌های کلی پرداخته شده است. برای این کار، ناظری مبتنی بر اجرای چندباره امن [۴] ارائه کرده‌اند که نسخه‌های ایزوله‌ای از برنامه هدف را اجرا کنند و رفتار برنامه را به ازای مقادیر مختلف ورودی بیازمایند. برنامه هدف را مانند یک جعبه سیاه مدل می‌شود که در زمان متناهی، عمل خروجی متناسب با ورودی را تولید می‌کند. به این ترتیب نشان داده می‌شود که فوق‌خاصیت‌های ایمنی آزمون‌پذیر[۱۴۳] و نیمه‌آزمون‌پذیر[۱۴۴]، کران پایین و کران بالای خط‌مشی‌هایی هستند که این ناظرها می‌توانند اعمال کنند. در واقع، به کمک اجراهای مختلف از برنامه با ورودی‌های متناظر با اجرای مشاهده‌شده قبلی، ناظر زیرمجموعه‌ای از اجراهای ممکن برنامه هدف که برای تصمیم‌گیری درباره دستور بعدی نیاز دارد را به دست می‌آورد. ناظرهای پیشنهادشده در این مقاله نیز در صورتی که اطلاعات ایستای دقیق درباره اجراهای ممکن برنامه هدف به آن‌ها داده شود، می‌توانند قدرتی مشابه ناظرهای [۴۹] داشته باشند. گرچه در بعضی پارادایم‌ها، ناظرهای پیشنهادشده کلاس بزرگتری از خط‌مشی‌های امنیتی را به نسبت ناظرهای کار دیگر مطرح‌شده اعمال می‌کنند.

---

[141] Non-Uniform Context

[142] Universal

[143] Testable

[144] Semi-Testable

۶-۱-۲ تعاریف

یک خط‌مشی امنیتی مجموعه‌ای از برنامه‌ها است که هر برنامه زیرمجموعه‌ای از Σ^{∞} است. به بیان دیگر، هر خط‌مشی یک فوق‌خاصیت [۴۷] است که زیرمجموعه‌ای از مجموعه توانی Σ^{∞} محسوب می‌شود. یک برنامه C خط‌مشی P را برآورده می‌کند اگر $C \in P$. به بیان غیرصوری، منظور از یک خط‌مشی امنیتی همان رفتار مورد انتظار از یک برنامه است.

یک خط‌مشی P را خاصیت[۱۴۵] می‌نامیم اگر مجموعه توانی از یک مجموعه اجراها به نام $\psi \subseteq \Sigma^{\infty}$ باشد. در واقع، خاصیت $P=\mathcal{P}(\psi)$ توسط مجموعه ψ از اجراها سرشت‌نمایی می‌شود. از طرف دیگر، برای هر خاصیت سرشت‌نمایی‌شده توسط ψ، یک مسند مشخصه[۱۴۶] $\hat{P}$ روی Σ^{∞} وجود دارد که $\forall \sigma \in \Sigma^{\infty}: \hat{P}(\sigma) \Leftrightarrow \sigma \in \psi$. برنامه C خاصیت $\psi \subseteq \Sigma^{\infty}$ (منظور خاصیتی است که با ψ سرشت‌نمایی می‌شود) را برآورده می‌کند، اگر و فقط اگر $C \subseteq \psi$.

در ادامه به بیان کلاس‌های مهمی از خاصیت‌های امنیتی شناخته‌شده پرداخته می‌شود. خاصیت‌های ایمنی[۱۴۷] تعیین می‌کنند که هیچ‌گاه چیز بدی اتفاق نمی‌افتد. به عبارت دیگر، هر اجرای نامعتبر، پیشوندی نامعتبر دارد که همه گسترش‌های آن نیز نامعتبر هستند؛ یعنی با اجرای آن پیشوند از اجرا، دیگر نمی‌توان اجرایی معتبر برای آن پیشوند تصور کرد. بنابراین، یک خاصیت $\psi \subseteq \Sigma^{\infty}$ ایمنی خواهد بود اگر و فقط اگر

$$\forall \sigma \in \Sigma^{\infty}: \sigma \notin \psi \Leftrightarrow \exists \tau \preceq \sigma: \forall \sigma' \succeq \tau: \sigma' \notin \psi$$

دسته مهم دیگر از خاصیت‌ها، خاصیت‌های مانایی[۱۴۸] هستند. در یک خاصیت مانایی مطرح می‌شود که هر اجرای متناهی را می‌توان به یک اجرای معتبر گسترش داد. به بیان صوری، یک خاصیت $\psi \subseteq \Sigma^{\infty}$ مانایی است اگر و فقط اگر

---

[145] Property

[146] Characteristic Predict

[147] Safety Property

[148] Liveness Property

$$\forall \tau \in \Sigma^{*}: \exists \sigma \geq \tau: \sigma \in \psi$$

همچنین، به یک خاصیت، خاصیت تجدید نامتناهی[۱۴۹] گفته می‌شود اگر هر اجرای نامتناهی معتبر، دارای بی‌شمار پیشوند معتبر باشد و هر اجرای نامتناهی نامعتبر، تعداد متناهی پیشوند معتبر داشته باشد. به بیان صوری، $\psi \subseteq \Sigma^{\infty}$ یک خاصیت تجدید نامتناهی است اگر و فقط اگر

$$\forall \sigma \in \Sigma^{\omega}: \sigma \in \psi \Leftrightarrow (\forall \tau \preceq \sigma: \exists \tau' \preceq \sigma: \tau \preceq \tau' \wedge \tau' \in \psi)$$

طبق تعاریف ارائه‌شده بالا، به خط‌مشی‌ای که خاصیت نباشد، اصطلاحاً ناخاصیت[۱۵۰] گفته می‌شود. درواقع، ناخاصیت‌ها خط‌مشی‌هایی هستند که توسط مجموعه توانی زیرمجموعه‌ای از Σ^{∞} قابل بیان نیستند. برای $S.S' \subseteq \Sigma^{\infty}$، گوییم S'، S را پالایش[۱۵۱] می‌کند اگر $S' \subseteq S$. به این ترتیب، خط‌مشی P تحت پالایش بسته[۱۵۲] است اگر

$$\forall S.S' \subseteq \Sigma^{\infty}: (S \in P \ \wedge \ S' \subseteq S) \Longrightarrow S' \in P$$

به وضوح مشخص است که همه خاصیت‌ها، تحت پالایش بسته هستند. البته درباره ناخاصیت‌ها نمی‌توان چنین گفت.

۶-۱-۳ مقایسه ناظرهای زمان‌اجرا در اعمال خط‌مشی‌ها

نظارت زمان‌اجرا مکانیزمی برای اعمال خط‌مشی‌های امنیتی است. یک ناظر زمان‌اجرا، که به آن ناظر اجرا هم گفته می‌شود، اجراهای منفرد برنامه هدف را مشاهده می‌کند و در صورت تخطی از خط‌مشی، آن اجرا را تغییر می‌دهد [۴۶]. به طور کلی می‌توان ناظر زمان‌اجرا را با خودکاره ویرایش [۳۳] مدل‌سازی کرد. خودکاره ویرایش عبارتند از یک مجموعه متناهی یا شمارای نامتناهی از حالت‌ها به نام Q، یک حالت اولیه q_0 و یک تابع گذار محاسبه‌پذیر کامل[۱۵۳] $\delta: Q \times \Sigma \ \rightarrow Q \times (\Sigma \cup \{\cdot\})$. تابع

[149] Infinite Renewal Property

[150] Non-Property

[151] Refine

[152] Refinement-closed

[153] Total

گذار δ هر زوج شامل یک حالت و یک کنش ورودی را به یک حالت جدید و حداکثر یک کنش خروجی نگاشت می‌دهد. منظور از نماد نقطه (.) آن است که کنش خروجی تابع گذار تهی است. هر گذار نیز در هنگام واردشدن یک ورودی، تابع گذار را به کار می‌اندازد.

اجرای یک خودکاره ویرایش به نام A با یک حکم[۱۵۴] تک‌گامی[۱۵۵] $(q, \sigma) \xrightarrow{\tau}_A (q', \sigma')$ مشخص می‌شود که در آن q حالت فعلی است و برنامه هدف می‌خواهد دنباله‌ای از کنش‌های σ را اجرا کند. طبق حکم ارائه‌شده، خودکاره به حالت q' می‌رود و دنباله τ را به عنوان خروجی می‌دهد که حداکثر یک کنش دارد. پس از آن، دنباله σ' به عنوان بقیه دنباله اجرا مطرح می‌شود. یک خودکاره ویرایش در دو حالت کار می‌کند: (۱) توقیف[۱۵۶] و (۲) درج[۱۵۷]. اگر ناظر یک کنش ورودی را توقیف کند، ورودی مصرف می‌شود، و اگر در حالت درج باشد، بدون آن‌که کنش ورودی مصرف شود، کنشی درج می‌شود. در زیر دو قاعده برای بیان صوری این دو حالت عنوان شده است.

$$\frac{\sigma = a;\, \sigma' \quad \delta(q, a) = (q', a') \quad a' \neq \cdot}{(q, \sigma) \xrightarrow{\langle a' \rangle}_{\mathcal{A}} (q', \sigma)}$$

در این قاعده، ناظر بدون استفاده از کنش ورودی، حالت خودکاره را تغییر می‌دهد؛ اما تغییری در دنباله باقی‌مانده برای اجرا داده نمی‌شود. پس کنش خروجی a' درج شده است. در حالی‌که در قاعده بعدی، کنش ورودی مصرف می‌شود ولی کنشی برای خروجی داده نمی‌شود. به این ترتیب، می‌توان چنین گفت که کنش a توقیف شده است.

$$\frac{\sigma = a;\, \sigma' \quad \delta(q, a) = (q', \cdot)}{(q, \sigma) \xrightarrow{\varepsilon}_{\mathcal{A}} (q', \sigma')}$$

[154] Judgment

[155] Single-Step

[156] Suppression

[157] Insertion

البته می‌توان با استفاده از قواعد بازتابی[۱۵۸] و تراگذری[۱۵۹] زیر، حکم تک‌گامی مطرح‌شده را به حکم چندگامی $(q, \sigma) \overset{\tau}{\Rightarrow}_A (q', \sigma')$ تبدیل کرد.

$$(q, \sigma) \overset{\varepsilon}{\Longrightarrow}_A (q, \sigma) \quad \text{(REF)}$$

$$\frac{(q, \sigma) \overset{\tau_1}{\longrightarrow}_A (q', \sigma') \quad (q', \sigma') \overset{\tau_2}{\Longrightarrow}_A (q'', \sigma'')}{(q, \sigma) \overset{\tau_1;\tau_2}{\Longrightarrow}_A (q'', \sigma'')} \quad \text{(TRS)}$$

تبدیل یک اجرای ورودی احتمالاً نامتناهی σ توسط خودکاره ویرایش A به یک اجرای احتمالاً نامتناهی σ' را می‌توان تعریف کرد. خودکاره ویرایش $A=(Q, q_0, \delta)$ در یک سامانه با مجموعه کنش‌های Σ، یک اجرای ورودی $\sigma \in \Sigma^{\infty}$ را به $\sigma' \in \Sigma^{\infty}$ تبدیل می‌کند، که با نماد $(q_0, \sigma) \Downarrow_A \sigma'$ نمایش داده می‌شود، اگر و فقط اگر

$$\forall q' \in Q: \forall \sigma'' \in \Sigma^{\infty}: \forall \tau \in \Sigma^*:$$
$$((q_0, \sigma) \overset{\tau}{\Longrightarrow}_A (q', \sigma')) \overset{\tau}{\Longrightarrow} \tau \preceq \sigma'$$

و

$$\forall \tau \preceq \sigma': \exists q' \in Q: \exists \sigma'' \in \Sigma^{\infty}: (q_0, \sigma) \overset{\tau}{\Longrightarrow}_A (q', \sigma'')$$

در واقع، خودکاره A زمانی که از (q_0, σ) شروع به کار می‌کند و پس از یک گذار چندگامی به یک (q', σ'') می‌رسد، تنها پیشوندهای σ' را خروجی می‌دهد. علاوه بر این، همه پیشوندهای σ' را خروجی می‌دهد. برای ناظر M، منظور از $M(\sigma)$ دنباله کنش‌هایی است که M در پاسخ به اجرای ورودی σ تولید کرده است. پس اگر M با خودکاره A بازنمایی شده است، رابطه زیر برقرار است:

$$M(\sigma) = \sigma' \Leftrightarrow (q_0.\sigma) \Downarrow_A \sigma'$$

[158] Reflexivity

[159] Transitivity

عملیاتی که یک ناظر زمان‌اجرا می‌تواند انجام دهد توسط مجموعه‌ای از قواعد، که از آن به عنوان پارادایم اعمال[۱۶۰] نام برده می‌شود، محدود شده است. در همه پارادایم‌های اعمال، درستی باید وجود داشته باشد. به این معنا که هر اجرا پس از عبور از ناظر، باید خاصیت امنیتی را برآورده کند. در حقیقت، تفاوت تعابیر پارادایم‌های اعمال در تفسیر آن‌ها از شفافیت است. به این منظور که ناظر تا چه حد مجاز است که یک اجرای ورودی را تغییر دهد. در ادامه تعابیر مرسوم $_{=}$effectively و دقیق بیان می‌شوند.

ناظر M خاصیت $\psi \subseteq \Sigma^{\infty}$ را $_{=}$effectively اعمال می‌کند اگر و فقط اگر

$$\forall \sigma \in \Sigma^{\infty} : M(\sigma) \in \psi \ \wedge (\sigma \in \psi \ \rightarrow M(\sigma) = \sigma)$$

منظور از بیان صوری بالا این است که علاوه بر درستی، اگر اجرایی خاصیت امنیتی را برآورده می‌کرد، اجرای خروجی ناظر از نظر معناشناخت هم‌ارز و مساوی با اجرای ورودی باشد. اثبات شده است [۳۳] که ناظرها می‌توانند خاصیت‌های معقول تجدید نامتناهی را $_{=}$effectively اعمال کنند. منظور از معقول بودن یک خاصیت $\psi \subseteq \Sigma^{\infty}$ آن است که شرایط زیر برقرار باشد:

$$\varepsilon \in \square \ \wedge \ \Sigma^{*} \cap \psi \ is \ decidable$$

بنابراین، خاصیت‌های معقول تجدید نامتناهی، کرانی پایین برای خاصیت‌های قابل اعمال در پارادایم $_{=}$effectively هستند. گرچه خاصیت‌هایی وجود دارند که تحت پارادایم $_{=}$effectively توسط ناظرهای زمان‌اجرا قابل اعمال نیستند. به عنوان مثال، خاصیتی که در آن دسترس‌پذیری منابع برای سامانه‌ها با n منبع مجزا مورد بررسی باشد و قید کند که همه منابع بازشده باید در نهایت، بسته شوند. این خط‌مشی توسط ناظر زمان‌اجرا قابل اعمال نیست. زیرا هرگاه ناظر یک پیشوند نامعتبر ببیند، که در این‌جا یعنی پیشوندی که آخرین کنشش بازکردن یک منبع باشد، ناظر تنها می‌تواند از قابلیت توقیف‌کردن کنش استفاده کند تا زمانی‌که کنشی را دریافت کند که بیانگر بستن آن منبع بازشده باشد. در آن زمان است که ناظر می‌تواند کنش توقیف‌شده را درج کند. اما ممکن است بلافاصله پس از کنش

[160] Enforcement Paradigm

بستن منبع، کنش دیگری دریافت کند که در آن دوباره منبع را باز می‌کند. به این ترتیب، ناظر نمی‌تواند کنش‌های توقیف‌شده را درج کند.

اعمال دقیق مشابه اعمال effectively$_{=}$ است با این تفاوت که ناظر را محدود می‌کند تا در هر اجرای ورودی معتبر، خروجی کنش را بلافاصله پس از دریافت کنش ورودی تولید کند. به بیان صوری، ناظر M که توسط خودکاره A بازنمایی می‌شود، خاصیت $\psi \subseteq \Sigma^{\infty}$ را به طور دقیق[۱۶۱] اعمال می‌کند اگر و فقط اگر

$$\forall \sigma \in \Sigma^{\infty} : M(\sigma) \in \psi \quad \wedge$$

$$\sigma \in \Psi \Longrightarrow \forall i \in \mathbb{N}: \exists q \in Q: (q_0, \sigma) \overset{\sigma[..i]}{\Longrightarrow}_A (q, \sigma[i+1..])$$

می‌توان نتیجه گرفت [۲۴] که مجموعه خاصیت‌های قابل اعمال در پارادایم دقیق برابر است با خاصیت‌های ایمنی معقول. از آن‌جایی که هر خاصیت ایمنی نیز یک خاصیت تجدید نامتناهی است، مجموعه خاصیت‌های قابل اعمال در پارادایم effectively$_{=}$ شامل مجموعه خاصیت‌های قابل اعمال در پارادایم دقیق خواهد بود.

۲-۶ بررسی تأثیر اطلاعات ایستا در توانایی ناظرها

تا این‌جا ناظرهای زمان‌اجرا اطلاعاتی از اجراهای دیگر برنامه نداشتند [۴۶]. حال با در نظر گرفتن اطلاعات ایستا برای ناظرها به بررسی توانایی و قدرت اعمال آن‌ها پرداخته می‌شود. می‌توان تأثیر در اختیار داشتن اطلاعات ایستا را مشابه کسب اطلاعات درباره اجراهای مرتبط دیگر در زمان‌اجرا دانست که هر دو در تصمیم‌گیری ناظر با توجه به پیشوندی از اجرا که تاکنون مشاهده کرده است، کمک می‌کند. به طور کلی، این‌که یک اجرای داده‌شده، یک اجرای ممکن از برنامه‌ای که کد منبع آن در اختیار است، تصمیم‌ناپذیر است. به همین دلیل، به جای استفاده مستقیم از کد منبع، معمولاً از یک تحلیل محافظه‌کارانه از آن استفاده می‌شود که تقریبی محاسبه‌پذیر از اجراهای ممکن برنامه ارائه

[161] Precisely

می‌کند. پس مجموعه اجراهای حاصل از تحلیل ایستای کد منبع، فوق‌مجموعه[۱۶۲] مجموعه اجراهای واقعی برنامه است.

در مقاله [۴۶]، ناظر زمان‌اجرا به شکلی مدل می‌شود که در دو فاز عمل می‌کند. در فاز اول، ناظر مجموعه‌ای از اجراها که تقریبی از مجموعه اجراهای ممکن برنامه هدف است را دریافت می‌کند و قواعدی را برای تبدیل اجراهای منفرد استنتاج می‌کند.

تعریف – یک ناظر زمان‌اجرا M، یک تابع محاسبه‌پذیر $M: \mathcal{P}(\Sigma^{\infty}) \to (\Sigma^{\infty} \to \Sigma^{\infty})$ است که هر مجموعه اجرای داده‌شده S را به یک تابع $M(S): \Sigma^{\infty} \to \Sigma^{\infty}$ نگاشت می‌کند که بیانگر یک خودکاره ویرایش $A_S=(Q, q_0, \delta)$ است؛ یعنی $M(S)(\sigma)=\sigma'$ اگر و فقط اگر $(q_0, \sigma)\Downarrow_{A_S} \sigma'$. به این ترتیب، ناظر M، برای یک مجموعه اجرای S، مجموعه $S' \subseteq \Sigma^{\infty}$ را به $S'' \subseteq \Sigma^{\infty}$ تبدیل می‌کند اگر S'' تصویر S' تحت M(S) باشد. به بیان دیگر، $S'' = M(S)(S') = \{M(S)(\sigma) | \sigma \in S'\}$.

ناظری که پیشتر معرفی شده بود، هیچ اطلاعاتی درباره اجراهای ممکن برنامه هدف نداشت و برای همه برنامه‌ها به طور یکسان عمل می‌کرد. البته می‌توان این‌گونه ناظرها را نیز با تعریف ارائه‌شده مدل کرد با این شرط که برای همه مجموعه اجراها خودکاره مشابهی تولید کند.

در ادامه، پارادایم‌های جدیدی [۴۶] معرفی می‌شود که هم برای خاصیت‌ها و هم برای ناخاصیت‌ها می‌توان به کار بست. پارادایم‌های $\text{effectively}^{d}_{=}$ و precisely^{d} مطرح می‌شوند که d در آن‌ها بیانگر پارامتری است که میزان دقت تقریب داده‌شده از اجراهای ممکن برنامه هدف را مشخص می‌کند.

۶-۲-۱ پارادایم $\text{effectively}^{d}_{=}$

منظور از پارادایم $\text{effectively}^{d}_{=}$ این است که تفاضل متقارن بین دو مجموعه تقریب (S) و هر مجموعه اجرای داده‌شده برای نظارت (S') کمتر از مقدار d باشد، و همان تعبیر پارادایم $\text{effectively}_{=}$ لحاظ شود. به طور مشابه، پارادایم precisely^{d} نیز تعریف شده است.

[162] Superset

اثبات می‌شود که هر خط‌مشی قابل اعمال در پارادایم $\text{effectively}_{=}$، یک خاصیت است. در ادامه به بیان قدرت ناظرها با در نظر گرفتن اطلاعات ایستا در پارادایم $\text{effectively}^{d}_{=}$ پرداخته می‌شود. برای سه حالت $d=\infty$، $d\in\mathbb{Z}^{+}$ و $d=0$ قدرت ناظرها را می‌توان بررسی کرد. البته به طور شهودی مشخص است که مجموعه خط‌مشی‌های قابل اعمال $\text{effectively}^{0}_{=}$ شامل دو حالت دیگر می‌شود. می‌توان ثابت کرد که هر خط‌مشی قابل اعمال در $\text{effectively}^{\infty}_{=}$، تحت پالایش بسته و یک خاصیت است. به همین دلیل، ناخاصیت‌ها را نمی‌توان در پارادایم $\text{effectively}^{\infty}_{=}$ اعمال کرد. طبق قضیه‌ای در این مقاله، خط‌مشی‌ای که قابل اعمال در $\text{effectively}_{=}$ باشد، در $\text{effectively}^{\infty}_{=}$ قابل اعمال است و برعکس.

مشخص است که هر خط‌مشی قابل اعمال در $\text{effectively}^{\infty}_{=}$، در $\text{effectively}^{d\in\mathbb{Z}+}_{=}$ نیز قابل اعمال است. ضمناً اثبات شده است که برای هر خط‌مشی P قابل اعمال در $\text{effectively}^{d\in\mathbb{Z}+}_{=}$، خاصیتی وجود دارد که همه مجموعه‌های متناهی اجراهای برآورده‌کننده P را شامل می‌شود؛ یعنی این خط‌مشی‌ها نشانه‌هایی از خاصیت‌ها را دارند و تفاوت آن‌ها در برنامه‌های امنی است که چندین اجرای نامتناهی دارند. به همین دلیل، بسیاری از ناخاصیت‌ها مانند خط‌مشی‌های جریان اطلاعات و خط‌مشی میانگین زمان پاسخ، در $\text{effectively}^{d\in\mathbb{Z}+}_{=}$ قابل اعمال نیستند.

همان‌طور که گفته شد، در پارادایم $\text{effectively}^{0}_{=}$ فرض بر آن است که تقریب داده‌شده به ناظر از اجراهای ممکن، دقیق باشد. می‌توان نشان داد که مجموعه خط‌مشی‌های قابل اعمال در این پارادایم، فوق‌مجموعه محضی[۱۶۳] از مجموعه خط‌مشی‌های قابل اعمال در دیگر پارادایم‌های اعمال موثر است. خاصیت عدم خاتمه[۱۶۴] و بعضی از خاصیت‌های ۲-ایمنی [۴۷] در $\text{effectively}^{0}_{=}$ قابل اعمال هستند. در خاصیت عدم خاتمه، هر اجرای برنامه باید یا دنباله تهی (ε) یا دنباله‌ای نامتناهی از کنش‌ها باشد. ناظر برای اعمال این خاصیت، بی‌شمار کنش را به انتهای یک اجرای خاتمه‌دار اضافه می‌کند. به طور کلی برای اعمال این خاصیت، ناظر یا باید منتظر باشد که در آینده، پیشوندی از اجرای نامتناهی ببیند یا به تعداد نامتناهی به یک اجرای متناهی، کنش اضافه کند. پس اگر صبر کند تا در آینده پیشوند اجرای نامتناهی را ببیند، ممکن است هیچ‌گاه این اتفاق رخ ندهد (نقض درستی) و اگر بی‌شمار کنش را درج

[163] Proper (Strict) Superset

[164] Non-Termination

کند، شفافیت نقض می‌شود. پس این خاصیت گرچه در $\text{effectively}^{d\in\mathbb{Z}+}_{=}$ قابل اعمال نیست، در $\text{effectively}^{0}_{=}$ می‌تواند اعمال شود.

دسته دیگری از خط‌مشی‌ها که در $\text{effectively}^{0}_{=}$ قابل اعمال است، زیرکلاسی از خط‌مشی‌های ۲-ایمنی است. بعضی از مهم‌ترین خط‌مشی‌های جریان اطلاعات مانند عدم تداخل Goguen-Messeguer [۷] و قطعیت مشاهده‌ای[165] [۵۰]، از خاصیت‌های ۲-ایمنی هستند. مجموعه اجراهای ممکن هر برنامه‌ی ناقض یک خط‌مشی ۲-ایمنی، تعدادی مشاهده نامعتبر غیرقابل اصلاح شامل حداکثر دو اجرای متناهی را گسترش می‌دهد. به بیان صوری، خط‌مشی P، یک خط‌مشی ۲-ایمنی است اگر و فقط اگر برای هر $S \subseteq \Sigma^{\infty}$،

$$S \notin P \Leftrightarrow \exists S' \subseteq \Sigma^{*}: (S' \preceq S \wedge |S'| \leq 2 \wedge (\forall S'' \subseteq \Sigma^{\infty}: S' \preceq S'' \Longrightarrow S'' \notin P))$$

پس خط‌مشی ۲-ایمنی، خط‌مشی‌ای است که اگر یک برنامه آن را نقض کند، یک مجموعه اجراهای متناهی با حداکثر دو عضو وجود خواهد داشت که هر اجرا، پیشوند اجراهای برنامه است و هر مجموعه اجرای دیگری که حاصل گسترش مجموعه اجراهای متناهی باشد، خط‌مشی را برآورده نمی‌کند. همچنین، اگر یک خط‌مشی ۲-ایمنی خاصیت نباشد، به آن یک ناخاصیت ۲-ایمنی گفته می‌شود.

علاوه بر این، می‌توان نشان داد که خط‌مشی P یک خط‌مشی ۲-ایمنی است اگر و فقط اگر

$$\forall S \subseteq \Sigma^{\infty}: S \in P \Leftrightarrow \forall \tau.\tau' \in pref(S): \{\tau.\tau'\} \in P$$

یا به بیان دیگر، اگر P یک خط‌مشی ۲-ایمنی باشد، برای هر $S \notin P$، یک مشاهده بد $\{\tau.\tau'\} \notin P$ وجود دارد که $\{\tau.\tau'\} \preceq S$. پس برای یک مجموعه اجراهای داده‌شده S، ناظر در پارادایم $\text{effectively}^{0}_{=}$ می‌تواند خط‌مشی ۲-ایمنی P را اعمال کند. با خواندن پیشوندهای اجراهای منفرد τ و وارسی این‌که هیچ $\tau' \in pref(S)$ای وجود ندارد که $\{\tau.\tau'\}$ یک مشاهده بد باشد، این عمل محقق می‌شود. پس

[165] Observational Determinism

زیرکلاسی از خط‌مشی‌های ۲-ایمنی که این وارسی در آن‌ها محاسبه‌پذیر است، در پارادایم $\text{effectively}^0_=$ قابل اعمال خواهد بود.

ناظری که در [۴۶] برای اعمال این‌گونه خط‌مشی‌ها ارائه شده است، مبتنی بر ایده خودترکیبی موازی [۴۷] است. طبق این ایده، می‌توان راستی‌آزمایی یک ناخاصیت ۲-ایمنی روی یک سامانه که مجموعه اجراهای آن S است را به راستی‌آزمایی یک خاصیت ایمنی روی سامانه‌ای با اجراهای S×S کاهش داد.

یکی از ناخاصیت‌های قابل اعمال در $\text{effectively}^0_=$، قطعیت مشاهده‌ای است [۴۶]. منظور از این خط‌مشی آن است که یک برنامه امن باید در دید مشاهده‌گرهای سطح پایین عدم قطعیت نداشته باشد. به بیان دیگر، خروجی‌های سطح پایین برنامه باید تنها تابعی از ورودی‌های سطح پایین آن برنامه باشند. پس برای هر دو اجرایی از برنامه که i کنش اول آن یکسان است، باید ۱+iامین کنش در هر دو یا ورودی باشد یا خروجی‌های یکسان باشد. به بیان صوری داریم:

$$\begin{aligned}\text{OD} = \{S \subseteq \Sigma^\infty \mid \forall \sigma, \sigma' \in S: \forall i \in \mathbb{Z}^+: \\ \sigma_L[..\, i-1] = \sigma'_L[..\, i-1] \Longrightarrow \sigma_L[i] =_L \sigma'_L[i]\}\end{aligned}$$

نشان داده می‌شود که با توجه به محاسبه‌پذیر بودن تابع وارسی مطرح‌شده برای خط‌مشی‌های ۲-ایمنی در خط‌مشی قطعیت مشاهده‌ای، این خط‌مشی در پارادایم $\text{effectively}^0_=$ قابل اعمال است.

۶-۲-۲ پارادایم precisely^d

در پاردایم اعمال دقیق، برای اجراهای معتبر برنامه، ناظر باید هر کنش ورودی را قبل از دریافت کنش ورودی بعدی به عنوان خروجی ارائه کند؛ یعنی برای حفظ شفافیت، باید دقیقاً در همان گام، کنش ورودی را در خروجی تکرار کند حال پارادایم جدیدی به نام precisely^d مطرح می‌شود [۴۶]. ناظر M خط‌مشی امنیتی P را در پاردایم precisely^d اعمال می‌کند اگر و فقط اگر برای هر مجموعه اجراهای $S \subseteq \Sigma^\infty$ که مجموعه پیشوندهای pref(S) آن محاسبه‌پذیر است، و برای هر $S' \subseteq S$ با تفاضل متقارن کمتر از d،

$$(1)\ M(S)(S') \in P \text{ and}$$
$$(2)\ S' \in P \Longrightarrow (\forall \sigma \in S' \colon \forall i \in \mathbb{N} \colon \exists q \in Q \colon (q_0, \sigma) \sigma \overset{[..i]}{\Longrightarrow}_{\mathcal{A}_S} (q, \sigma[i+1..]))$$

مشخص است که هر خط‌مشی قابل اعمال در پارادایم دقیق، در پارادایم effectively$_{=}$ نیز قابل اعمال است. به عنوان نتیجه این گزاره، نمی‌توان ناخاصیت‌ها را در پارادایم دقیق اعمال کرد. حال به بررسی قدرت ناظرها در پارادایم‌های precisely$^{\infty}$ و precisely$^{d \in \mathbb{Z}+}$ پرداخته می‌شود.

طبق تعریف، به وضوح می‌توان دید که هر خط‌مشی قابل اعمال در precisely$^{\infty}$، در پارادایم precisely$^{d \in \mathbb{Z}+}$ نیز قابل اعمال است. اثبات شده است که ناظرهای زمان‌اجرا می‌توانند خط‌مشی‌هایی را در precisely$^{\infty}$ اعمال کنند که توسط ناظرهای سنتی در پارادایم دقیق قابل اعمال باشند. این دسته از خط‌مشی‌ها همان خاصیت‌های ایمنی معقول[۱۶۶] هستند. به عبارت دیگر، خط‌مشی P را می‌توان در پارادایم دقیق اعمال کرد اگر و فقط اگر در پارادایم precisely$^{\infty}$ قابل اعمال باشد.

همان‌طور که در تعریف نیز مطرح شد، پارادایم‌های دقیق سخت‌گیرانه‌تر از پارادایم‌های مؤثر متناظر خود هستند. بنابراین، مجموعه خط‌مشی‌های قابل اعمال در پارادایم‌های precisely$^{\infty}$ و precisely$^{d \in \mathbb{Z}+}$، به ترتیب زیرمجموعه محض مجموعه خط‌مشی‌های قابل اعمال در پارادایم‌های effectively$_{=}^{\infty}$ و effectively$_{=}^{d \in \mathbb{Z}+}$ است.

در پارادایم precisely^{0}، فرض بر آن است که تقریب ارائه‌شده برای مجموعه اجراهای ممکن برنامه دقیق است. بنابراین ناظر در این پارادایم می‌تواند خط‌مشی‌هایی که در پارادایم precisely$^{d \in \mathbb{Z}+}$ قابل اعمال نبود را اعمال کند. از ناظرهای پیشنهادشده و قضایای استفاده‌شده در پارادایم effectively$_{=}^{0}$ می‌توان نتیجه گرفت که خط‌مشی‌هایی مانند عدم خاتمه، قطعیت مشاهده‌ای و زیرکلاس مطرح‌شده از خط‌مشی‌های ۲-ایمنی در پارادایم precisely^{0} قابل اعمال هستند. البته خاصیت‌ها و ناخاصیت‌هایی هستند که گرچه در effectively$_{=}^{0}$ می‌توان آن‌ها را اعمال کرد، اما در پارادایم precisely^{0} قابل اعمال نیستند.

[166] Reasonable Safety Property

۶-۳ ارتباط بین پارادایم‌های اعمال معرفی‌شده

برای سادگی در جمع‌بندی قدرت ناظرها در پارادایم‌های معرفی‌شده [۴۶]، از علائم اختصاری استفاده می‌شود. برای مجموعه‌های خط‌مشی‌های قابل اعمال در پارادایم‌های $\text{effectively}_{=}$، $\text{effectively}_{=}^{\infty}$، $\text{effectively}_{=}^{d\in\mathbb{Z}+}$، $\text{effectively}_{=}^{0}$، precisely، $\text{precisely}^{\infty}$، $\text{precisely}^{d\in\mathbb{Z}+}$ و precisely^{0} به ترتیب از نمادهای $\text{EFF}_{=}$، $\text{EFF}_{=}^{\infty}$، $\text{EFF}_{=}^{d\in\mathbb{Z}+}$، $\text{EFF}_{=}^{0}$، PRCS، PRCS^{∞}، $\text{PRCS}^{d\in\mathbb{Z}+}$ و PRCS^{0} استفاده می‌شود.

همان‌طور که پیشتر اشاره شد، $\text{EFF}_{=} = \text{EFF}_{=}^{\infty}$. بنابراین، $\text{EFF}_{=}^{\infty}$ همه خاصیت‌های معقول تجدید نامتناهی را شامل می‌شود اما همه ناخاصیت‌ها را در بر ندارد. همچنین، می‌توان گفت که ناخاصیت‌ها $\text{EFF}_{=}^{d\in\mathbb{Z}+}$ نیز نیستند. شاید این حدس که $\text{EFF}_{=}^{\infty} = \text{EFF}_{=}^{d\in\mathbb{Z}+}$ درست باشد. همچنین داریم:

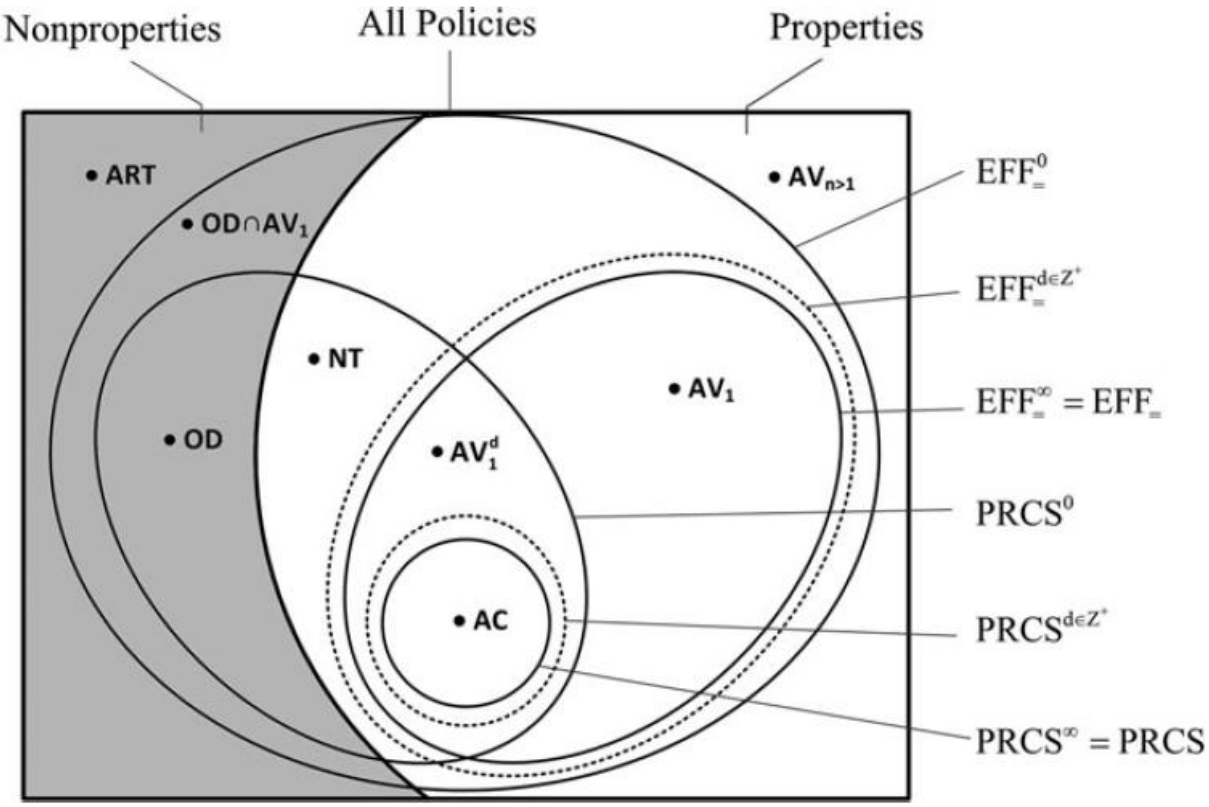
$$\text{EFF}_{=} = \text{EFF}_{=}^{\infty} \subseteq \text{EFF}^{d\in\mathbb{Z}^+} \subsetneq \text{EFF}_{=}^{0}$$

از طرف دیگر، مجموعه PRCS معادل با PRCS^{∞} است. بنابراین، PRCS^{∞} شامل خاصیت‌های ایمنی معقول مانند خط‌مشی‌های کنترل دسترسی می‌شود. همچنین، می‌توان گفت که هیچ ناخاصیتی در $\text{PRCS}^{d\in\mathbb{Z}+}$ وجود ندارد. علاوه بر این، PRCS^{0} فوق‌مجموعه محضی از $\text{PRCS}^{d\in\mathbb{Z}+}$ است. پس:

$$\text{PRCS} = \text{PRCS}^{\infty} \subseteq \text{PRCS}^{d\in\mathbb{Z}^+} \subsetneq \text{PRCS}^{0}.$$

به وضوح می‌توان دید که برای هر $d\in\mathbb{N}\cup\{\infty\}$، داریم $\text{PRCS}^{d}\subsetneq\text{EFF}_{=}^{d}$. همچنین، PRCS^{∞} در هر دو مجموعه PRCS^{0} و EFF^{∞} وجود دارد و داریم $\text{PRCS}^{\infty} \subsetneq \text{EFF}^{\infty} \cap \text{PRCS}^{0}$.

در شکل زیر، ارتباط بین خط‌مشی‌های قابل اعمال در پارادایم‌های اعمال گوناگون مشخص شده است. نقاط مشخص‌شده در شکل، نمونه‌هایی از خط‌مشی‌ها در هر ناحیه هستند.



شکل ۹ – ارتباط بین خط‌مشی‌های قابل اعمال در پارادایم‌های اعمال مطرح‌شده در [۴۶]

به طور خلاصه، در مقاله [۴۶] به سرشت‌نمایی ظرفیت ناظرهای زمان‌اجرا برای اعمال خط‌مشی‌های امنیتی پرداخته شده است. ناظرهای زمان‌اجرا با فرض در اختیار داشتن اطلاعات ایستا و تقریبی از رفتار ممکن برنامه هدف، توانایی بیشتری دارند. برای بیان دقیق این مطلب، پارادایم‌های اعمال جدیدی معرفی شد که برای خط‌مشی‌ها، چه خاصیت و چه ناخاصیت، قابل بررسی باشند. پاردایم‌های مؤثر و دقیق برای این زمینه بازتعریف شدند.

طبق میزان دقتی که از تقریب اجراهای ممکن برنامه در اختیار ناظر قرار داده می‌شود، قدرت ناظر نیز تغییر خواهد کرد. نتایج نشان داد که در شرایطی که ناظر اطلاعات دقیق از اجراهای ممکن را در اختیار ندارد، نمی‌تواند ناخاصیت‌ها را اعمال کند. علاوه بر این، ناظرهایی که اطلاعات ایستای دقیق نیز دارند، مجموعه‌های مشخصی از خاصیت‌ها و ناخاصیت‌ها را می‌توانند اعمال کنند که در شرایط دیگر قابل اعمال نیستند. البته کماکان خط‌مشی‌هایی وجود دارند که حتی با در اختیار داشتن دانش دقیق از اجراهای ممکن برنامه، ناظر توانایی اعمال آن‌ها را ندارد.

لازم به یادآوری است که هنوز گام‌های اولیه سرشت‌نمایی خط‌مشی‌های امنیتی قابل اعمال توسط ناظرهایی با اطلاعات بیشتری از صرف رویدادهای زمان‌اجرا برداشته می‌شود. فضای انجام پژوهش‌ها در این حوزه باز است. در کارهای آینده می‌توان به بررسی پارادایم‌های دیگر پرداخت که تعبیر آن‌ها از شفافیت، اعمال تغییرات حداقلی در برنامه باشد، مستقل از این‌که برنامه امن است یا خیر. همچنین سرشت‌نمایی خط‌مشی‌های امنیتی با مفاهیم مطرح‌شده در زمینه ریاضیات و توپولوژی پژوهش بعدی مناسبی خواهد بود.

فصل هفتم

بررسی تکنیک‌ها و پیاده‌سازی‌های اعمال زمان‌اجرا

بررسی تکنیک‌ها و پیاده‌سازی‌های اعمال زمان‌اجرا

تکنیک‌های زمان‌اجرا نوید صحت و انعطاف‌پذیری بیشتری در اعمال خط‌مشی‌های امنیتی را داده‌اند. گرچه مکانیزم‌های ایستای اعمال امنیت پیش از این مورد مطالعه و دسته‌بندی قرار گرفته است، اما [۲۳] اولین کار در مکانیزم اعمال زمان‌اجرا است. در این کار، محدودیت‌ها و مزایای تکنیک‌های اعمال و پیاده‌سازی آن‌ها مورد بحث قرار می‌گیرد. معیارهایی نظیر سطح انتزاع، خط‌مشی‌های اعمال‌شده و تضمین‌های امنیتی از جمله معیارهای مقایسه تکنیک‌های مختلف مورد استفاده در این کار است.

استفاده و اجرای برنامه‌های کاربردی بدون در اختیار داشن کد برنامه بسیار رایج است و همین باعث به‌وجود آمدن ریسک‌های امنیتی می‌شود. به طور مثال، ۲۸٪ از برنامه‌های کاربردی اندروید به اطلاعات حساس دسترسی دارند.

رفتار مجاز و غیرمجاز برنامه‌ها در خط‌مشی امنیتی مشخص می‌شود. تطابق سیستم با خط‌مشی‌های امنیتی باعث افزایش محافظت از آن خواهد بود. منظور از اعمال خط‌مشی این است که اطمینان داده شود که خط‌مشی‌ها در یک سیستم برآورده می‌شوند حتی اگر برنامه مورد اعتماد نباشد. این کار می‌تواند پیش از اجرای برنامه، در طول اجرای برنامه یا هم قبل و هم در حین اجرا صورت بگیرد. یک ناظر امنیتی، یا ناظر اجرا یا اعمال‌گر، مجموعه‌ای شامل یک یا چند مکانیزم است که وظیفه آن نظارت بر اجرای برنامه و واکنش در برابر رویدادهای بدخواه است.

اعمال خط‌مشی امنیتی حوزه گسترده‌ای در روش‌های صوری دارد. نظریه خودکاره‌های[۱۶۷] حالت‌متناهی گام‌های بزرگی در چگونگی توصیف و اعمال کلاس بزرگی از خط‌مشی‌های امنیتی در برنامه غیرقابل اعتماد به وجود آورده است. یک خودکاره یک ماشین حالت است که حالت‌های امن و گذارهای بین آن‌ها را برای یک برنامه داده‌شده مدل‌سازی می‌کند. در اعمال زمان‌اجرا، همراه با برنامه اجرا می‌شود و هرگاه برنامه عملی را اجرا کرد که مغایر با خط‌مشی امنیتی بود، خودکاره سیگنالی تولید می‌کند که نشان می‌دهد این گذار حالت امن نیست. البته این‌که چگونه یک ناظر مبتنی بر خودکاره می‌تواند اجرای برنامه را تغییر دهد، محل پژوهش و بررسی است.

[167] Automata

اعمال خط‌مشی از منظر تحلیل ایستا؛ یعنی تحلیل کد برنامه به منظور وارسی آن برنامه که آیا خط‌مشی مورد نظر را برآورده می‌کند یا خیر، موضوع مقالات و مطالعات بسیاری بوده است اما از این تعبیر نمی‌توان برای اعمال زمان‌اجرا استفاده کرد.

در [۲۳] از دو مفهوم *تکنیک اعمال*[۱۶۸] و *پیاده‌سازی اعمال*[۱۶۹] استفاده می‌شود. یک تکنیک اعمال، یا همان مکانیزم امنیتی، راهی کلی برای اعمال مجموعه‌ای از خط‌مشی‌ها است. یک پیاده‌سازی اعمال مصداقی از یک تکنیک اعمال روی یک هدف مشخص است و از ابزارهای مشخصی نیز در آن استفاده می‌شود. پژوهشگر باید معناشناخت خط‌مشی را بداند و طبق آن مکانیزم‌های مناسب برای حیطه[۱۷۰] آن خط‌مشی انتخاب نماید.

منظور از سیستم محیطی است که در آن برنامه‌ها اجرا می‌شوند. به این معنا که در یک سیستم که ماشینی است که دارای منابع مختلفی است، برنامه‌های مورد اعتماد و غیرمورد اعتماد وجود دارند و همگی به دنبال استفاده از منابع سیستم هستند. به این ترتیب لازم است برنامه‌های غیرمورد اعتماد توسط ابزارهای امنیتی تحلیل و بررسی شوند تا از رفتارهای بدخواهانه جلوگیری شود. به این برنامه‌ها اصطلاحاً هدف[۱۷۱] گفته می‌شود. حال خط‌مشی‌های امنیتی را می‌توان محدودیت‌ها و قیودی روی رفتارهای هدف‌ها دانست.

طبق تعریف Bishop در [۳]، یک خط‌مشی گزاره‌ای مشخص از بایدها و نبایدها است. اگر سیستم همواره در حالت‌های مجاز باشد و کاربران فقط قادر به انجام عملیات مجاز باشند، به آن سیستم امن گفته می‌شود. در صورتی که سیستم به حالت غیرمجاز برود یا اگر کاربر بتواند یک عمل غیرمجاز را با موفقیت انحام دهد، سیستم را ناامن می‌نامند.

طبق تعریف دیگری [۲۳]، یک خط‌مشی امنیتی گزاره‌ای است که از یک دارایی در برابر استفاده غیرمجاز محافظت می‌کند. در تعابیر صوری، Schneider یک خط‌مشی امنیتی را به عنوان محدودیتی

---

[168] Enforcement Technique

[169] Enforcement Implementation

[170] Scope

[171] Target

روی یک اجرای برنامه تعریف می‌کند. در [۲] سه نوع خط‌مشی عنوان می‌شود که عبارتند از خط‌مشی‌های کنترل دسترسی، که درباره عملیات کاربران روی منابع مطرح می‌شوند؛ خط‌مشی‌های جریان اطلاعات، که درباره داده‌هایی که توسط یک برنامه قابل مشاهده و استنتاج هستند محدودیت گذاشته می‌شود؛ و خط‌مشی‌های دسترس‌پذیری[۱۷۲]، که درباره کاربرانی است که استفاده از منابع یک سیستم برای کاربران دیگر را مختل می‌کنند.

هرگاه یک برنامه از خط‌مشی امنیتی تخطی کرد، گفته می‌شود که یک نقض امنیت[۱۷۳] رخ داده است. اعمال خط‌مشی امنیتی مجموعه اقدامات برای نگه‌داری سیستم در وضعیت سازگاری با خط‌مشی است که ممکن است به صورت صریح یا ضمنی در خط‌مشی بیان شده باشد. می‌توان اعمال خط‌مشی امنیتی را به عنوان روالی از اقدامات در گام‌های زیر دانست:

۱. اعمال‌گر باید برنامه هدف را از نظر عملیات مرتبط با خط‌مشی نظارت کند. ممکن است این نظارت شامل تغییر بخشی از برنامه برای تولید داده اضافی باشد.

۲. اعمال‌گر ارزیابی کند که برنامه هدف در حال نقض خط‌مشی است.

۳. طبق تصمیم گرفته‌شده در گام دوم، تعدادی از اقدامات بعدی توسط اعمال‌گر انجام شود که این اقدامات می‌تواند تنبیهی یا اصلاحی باشد.

در روش‌های صوری، مهم‌ترین مکانیزم برای نظارت بر اجرای یک برنامه هدف و اعمال یک خط‌مشی به آن اجرا، خودکاره امنیتی[۱۷۴] است. خودکاره امنیتی یک ماشین حالت‌متناهی است که اجرا را به کمک عناصر زیر مدل می‌کند:

- تعدادی حالت خودکاره که بعضی از آن‌های حالت آغازین هستند.
- تعدادی نماد ورودی
- یک تابع گذار

[172] Availability

[173] Security Violation

[174] Security Automaton

از دیدگاه سنتی، این خودکاره همگام با برنامه هدف اجرا می‌شود و در هر گام از اجرای برنامه هدف، ورودی‌ای به خودکاره داده می‌شود و آن نیز فقط طبق تابع گذار داده شده، به حالت جدیدی می‌رود.

انواع مختلفی از خودکاره‌های امنیتی در این حوزه وجود دارند. خودکاره قطع‌کننده[۱۷۵] که دنباله غیرمجاز از عمل‌ها را تشخیص داده و زمانی که برنامه می‌خواهد آن‌ها را انجام دهد، اجرا را متوقف[۱۷۶] می‌کند [۲]. نوع دیگر خودکاره ویرایش[۱۷۷] است که در آن خودکاره‌های درج[۱۷۸] و توقیف[۱۷۹] با یکدیگر ترکیب شده‌اند تا برنامه هدف زمانی که می‌خواهد دستوری را اجرا کند که ناقض امنیت است، تغییر یابد [۱۹]. اخیراً نوع دیگری از خودکاره‌ها به نام خودکاره نتایج اجباری[۱۸۰] ارائه شده است که مدل صوریِ واقع‌گرایانه‌تری به شمار می‌رود. از این جهت که Ligatti و Reddy [۱۷] مشاهده کردند که خودکاره ویرایش غیرعملی است و نمی‌توان در عمل آن را پیاده‌سازی کرد. زیرا خودکاره ویرایش با فرض قابلیت پیش‌بینی و حافظه میان‌گیر[۱۸۱] نامحدود مطرح شده است.

اعمال خط‌مشی می‌تواند قبل از اجرای برنامه هدف، در حین اجرای آن، یا هم قبل و هم بعد از اجرا صورت بگیرد. گام نظارت در اعمال خط‌مشی در زمان اجرا می‌تواند با به‌کارگیری شاخه‌ای از تحلیل برنامه به نام تحلیل پویای برنامه[۱۸۲] انجام شود. در روش ترکیبی[۱۸۳] اعمال که قبل و در حین اجرای برنامه مطرح می‌شود، گرفتن اطلاعاتی درباره برنامه قبل از اجرای آن، از مزایای شاخه دیگری از تحلیل برنامه؛ یعنی تحلیل ایستای برنامه[۱۸۴]، به شمار می‌رود.

---

[175] Truncation Automaton

[176] Halt

[177] Edit Automaton

[178] Insertion Automaton

[179] Suppression Automaton

[180] Mandatory Results Automaton

[181] Buffer

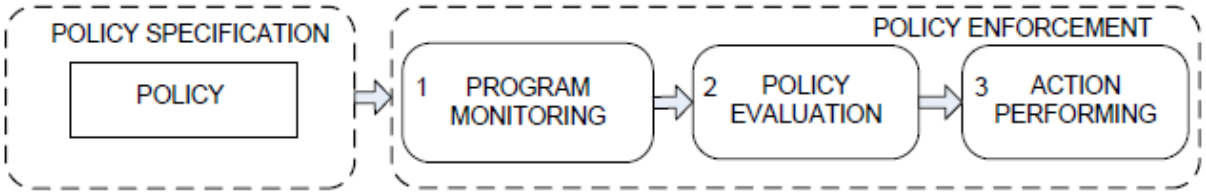
[182] Dynamic Program Analysis

[183] Hybrid

[184] Static Program Analysis

تحلیل ایستای برنامه به دنبال الگوها و ویژگی‌هایی است که برای یک مجموعه متناهی از مسیرهای اجرای یک برنامه برقرار باشد [۲۳]. تحلیل ایستا چندین رفتار ممکن برنامه را در نظر می‌گیرد و یک مدل ایستا از کد را برای ارزیابی آن می‌سازد. به این ترتیب تحلیلی که نتواند ویژگی‌ای را درباره برنامه اثبات کند، آن برنامه را به عنوان یک نقض احتمالی گزارش می‌کند. از این رو است که تحلیل ایستا دارای مثبت‌های کاذب است.

تحلیل پویای برنامه ویژگی‌های یک برنامه را مبتنی بر اطلاعات ناشی از اجرای آن وارسی می‌کند. این تحلیل از کدهای کامپایل‌شده یا اجراشدنی[۱۸۵] استفاده می‌کند. همچنین، تحلیل پویا را می‌توان به دو تحلیل آفلاین[۱۸۶]، که دنباله‌های اجرای برنامه را تحلیل می‌کند؛ و تحلیل زمان‌اجرا، که اجرای واقعی برنامه را تحلیل می‌کند، دسته‌بندی کرد. برخلاف تحلیل ایستا، تحلیل پویا دارای مثبت کاذب کم‌تری است، زیرا فقط بخشی از برنامه که با توجه به ورودی داده‌شده اجرا می‌شود را نظارت می‌کند. تحلیل پویای یک برنامه در هر زمان بسیار به داده ورودی وابسته است. زیرا ممکن است برنامه برای یک مجموعه ورودی امن باشد، اما برای مجموعه‌ای دیگر چنین نباشد. چالش جدی برای یک تحلیل پویای خوب، آزمون داده‌های مرتبط است و به همین دلیل، ابزارهای تحلیل پویا دارای منفی‌های کاذب هستند؛ به این معنا که ممکن است برنامه امن اعلام شود در حالی‌که در واقع چنین نیست.



شکل ۱۰ – شمای کلی نحوه اعمال خط‌مشی توسط ناظرهای برنامه [۲۳]

ناظر مرجع[۱۸۷] مدلی مفهومی است که ریشه اصلی در سیستم‌های عامل است و بعدها در امنیت و در کنترل دسترسی نیز به کار برده شد. هدف آن جلوگیری از کاربران غیرمجاز از دسترسی به یک

[185] Executable Code

[186] Offline Analysis

[187] Reference Monitor

سیستم بود. هرگاه یک کاربر منبعی را درخواست می‌کند، ناظر مرجع دخالت کرده و تعیین می‌کند که انجام این کار مجاز است یا نه و تنها پس از گذراندن بررسی‌های لازم اجازه انجام کار داده می‌شود. به همین دلیل، یک ناظر مرجع باید حتماً این شرایط را داشته باشد که: (۱) نتوان آن را دور زد؛ به این معنا که همه دسترسی‌های کاربر کنترل شود و هیچ‌یک از آن‌ها بدون دخالت ناظر مرجع انجام نشود. (۲) ناظر مرجع توسط موجودیت‌های خارجی قابل دستکاری نباشد. (۳) به اندازه کافی کوچک باشد تا بتوان آن را راستی‌آزمایی کرد. این تعبیر از ناظر مرجع به هسته امنیتی[۱۸۸] مربوط است و بخشی از مرکز محاسبات قابل اعتماد (TCB)[۱۸۹] است.

یکی از رویکردهای اخیر، استفاده از یک روش ترکیبی است که در آن تحلیل ایستا در زمان کامپایل مکمل ابزارهای تحلیل ایستا در زمان اجرا می‌شود.

۷-۱ معیارهای مقایسه تکنیک‌های اعمال

برای مقایسه تکنیک‌های اعمال می‌توان از معیارهایی مانند مقصود[۱۹۰] (تکنیک اعمال می‌خواهد چه چیزی را محافظت کند)، سطح انتزاع (حیطه و موقعیت)، نوع (چگونگی تأثیر اعمال به اجرای برنامه) و اندازه برنامه تحلیل‌شده[۱۹۱] ، کلاس خط‌مشی‌های اعمال‌شده (خاصیت یا ناخاصیت) و تضمین[۱۹۲] بررسی کرد [۲۳]. همچنین برای پیاده‌سازی هر تکنیک معیارهایی نظیر زبان خط‌مشی مورد استفاده، مدل اعتماد و سربار کارایی آن مطرح هستند.

---

[188] Security Kernel

[189] Trusted Computing Base

[190] Objective

[191] Locality

[192] Guarantees

جدول ۱ - معیارهای مقایسه تکنیک‌ها و پیاده‌سازی‌های اعمال خط‌مشی‌های امنیتی [۲۳]

CRITERIA FOR ENFORCEMENT TECHNIQUES	CRITERIA FOR ENFORCEMENT IMPLEMENTATIONS
T0. OBJECTIVE **T1. ABSTRACTION LEVEL** **T2. LOCALITY** **T3. TYPE** T4. CLASS OF POLICY ENFORCED T5. GUARANTEES	I1. TRUST MODEL I2. POLICY LANGUAGE I3. OVERHEAD

اکثر تکنیک‌های اعمال از مکانیزمی به نام مداخله‌گری[۱۹۳] برای تشخیص خط‌مشی مرتبط با رویدادها استفاده می‌کنند. رویداد مرتبط تا زمانی که تصمیمی گرفته شود و عمل لازم انجام شود، مسدود می‌ماند. می‌توان تکنیک‌ها را بر اساس این‌که ناظر امنیتی اجرا را در هنگام بروز رفتار بدخواهانه متوقف می‌کند یا ادامه می‌دهد، به دو دسته کلی تقسیم‌بندی کرد. به ناظرهای دسته اول، بازشناس[۱۹۴] گفته می‌شود و به ناظرهای دسته دوم که برنامه هدف بدخواه را به یک برنامه سازگار با خط‌مشی تبدیل می‌کنند، تمیزکننده[۱۹۵] اطلاق می‌شود. مشخص است که بازشناس‌ها زیرکلاسی از تمیزکننده‌ها هستند که عملیات تبدیل برنامه در آن وجود ندارد؛ اما تفکیک این دو دسته از ناظرها به دلیل ایجاد تفکیک بین خودکاره‌های سنتی[۱۹۶] (یا بازشناس‌های اجرا) و خودکاره‌های ویرایش[۱۹۷] (یا تبدیل‌کنندگان اجرا) است.

دسته‌بندی و کلاس‌بندی خط‌مشی‌های امنیتی در زمینه کارهای صوری صورت گرفته است. Schneider نشان داد که همه خط‌مشی‌ها قابل اعمال نیستند و خودکاره امنیتی فقط می‌تواند خاصیت‌های ایمنی را اعمال کند [۲]. Hamlen در [۱۴] به بررسی کلاس‌های مختلفی از خاصیت‌های امنیتی پرداخته است و نشان داده است که بعضی از آن‌ها به طور ایستا، بعضی دیگر در زمان اجرا و دسته دیگر به کمک بازنویسی برنامه قابل اعمال هستند. همچنین نشان داد که تعدادی از خط‌مشی‌ها قابل اعمال نیستند. در [۳۳] ناظر اجرایی سنتی پیشنهادشده توسط Schneider، که پیشتر با نام بازشناس معرفی شده بود، را مورد بررسی قرار داده است. Ligatti و همکاران ناظر سنتی را به ناظری

---

[193] Interception

[194] Recognizer

[195] Sanitizer

[196] Traditional Automata

[197] Edit Automata

گسترش می‌دهند که می‌تواند برای تصحیح رفتار بدخواهانه برنامه هدف، عملیات درج و توقیف داشته باشد.

در [۲۳] از دسته‌بندی اولیه ارائه‌شده توسط Schneider استفاده شده است که خط‌مشی‌های امنیتی را به خط‌مشی‌های کنترل دسترسی[۱۹۸] و خط‌مشی‌های جریان اطلاعات[۱۹۹] تقسیم‌بندی می‌کند. خط‌مشی‌های کنترل دسترسی قیودی روی یک برنامه هدف برای دسترسی به منابع سیستم اعمال می‌کند. از طرف دیگر، خط‌مشی‌های جریان اطلاعات از نشت داده به موجودیت‌های ناخواسته جلوگیری می‌کند تا نتوانند درباره رفتار برنامه قضاوتی داشته باشند.

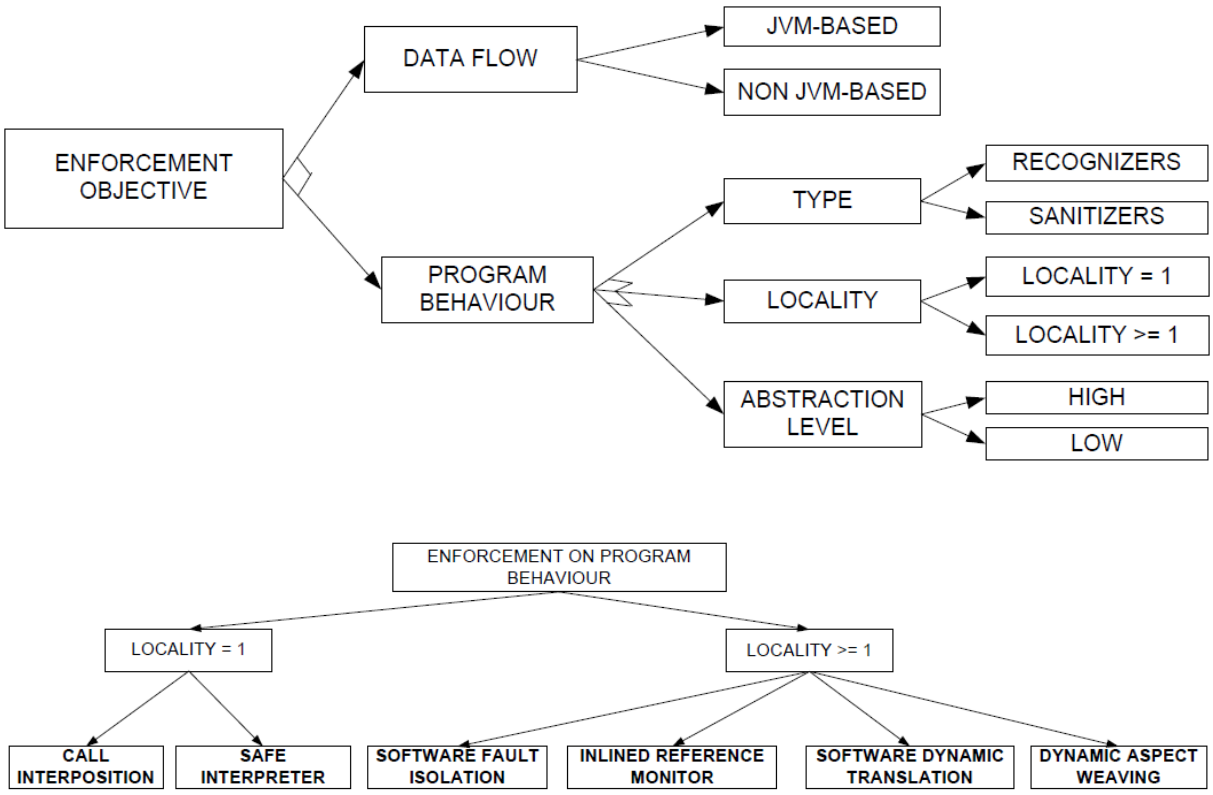
همان‌طور که پیشتر اشاره شد، برای مقایسه پیاده‌سازی‌های انجام‌شده از تکنیک‌ها، معیارهایی ارائه شده است. این معیارها به معیارهای مطرح‌شده برای مقایسه تکنیک‌ها بسیار وابسته است. مؤلفه‌های مورد اعتماد و سربار به سطح انتزاع تکنیک، زبان خط‌مشی به نوع تکنیک و کلاس خط‌مشی‌هایی که باید اعمال کند، ارتباط نزدیکی دارند.

منظور از معیار اول، مدل اعتماد، این است که موجودیت‌های سیستم که به آن‌ها اعتماد می‌شود و باید حفاظت شوند، کدام‌یک هستند. به این ترتیب می‌توان مقدار و چگونگی مؤلفه‌های مورداعتماد در یک پیاده‌سازی را درک کرد. در خصوص معیار زبان خط‌مشی، به میزان بیانگری[۲۰۰] زبان خط‌مشی و راحتی استفاده آن توجه می‌شود. از طرف دیگر، هرگاه که بحث از مکانیزم‌های زمان‌اجرا می‌شود، تأثیر آن‌ها بر عملکرد کلی برنامه کاربردی اهمیت می‌یابد. در پیاده‌سازی‌های تکنیک‌های اعمال زمان‌اجرا به ازای هر بار اجرای برنامه هدف، از آن‌ها استفاده می‌شود؛ در حالی‌که در اِعمال ایستا چنین نیست. همچنین مقایسه دقیق و ارزیابی میزان سربار پیاده‌سازی‌های مختلف دشوار است، زیرا فرض‌ها و بسترهای آزمون متفاوتی دارند. در شکل ۱۱ دسته‌بندی مختلف مکانیزم‌های اعمال خط‌مشی آمده است.

[198] Access Control Policies

[199] Information Flow Policies

[200] Expressiveness



شکل ۱۱- (الف) و (ب) معیارهای دسته‌بندی تکنیک‌های اعمال زمان‌اجرا [۲۳]

۲-۷ دسته‌بندی تکنیک‌ها

در شکل بالا می‌توان از منظر مقصود یا کلاس خط‌مشی‌هایی که اعمال می‌شوند، تکنیک‌ها را به دو دسته کلی تقسیم کرد: (۱) آن‌هایی که قیودی روی رفتار برنامه، مستقل از داده‌های برنامه می‌گذارند (۲) آن‌هایی که قیدهایی روی داده برنامه، مستقل از جریان برنامه اعمال می‌کنند. هدف دسته اول (دسته پایین در شکل الف) جلوگیری از خرابی سیستم توسط کد برنامه بدخواه است و شبیه *گودال ماسه*[۲۰۱] هستند که رویکردی کلی برای کمینه‌کردن اثر برنامه غیرمورد اعتماد روی سیستم است. دسته دیگری از تکنیک‌ها به انتشار داده تمرکز دارند. تکنیک‌های جریان داده معمولاً تمیزکننده‌هایی هستند با اندازه برنامه تحلیل‌شده بزرگتر یا مساوی یک. جریان اطلاعات نیز خط‌مشی‌ای است که روی چندین

[201] Sandbox

اجرا، به جای یک اجرای منفرد، تعریف می‌شود. به همین دلیل برای اعمال جریان اطلاعات دنباله‌ای از رویدادها در یک زمان تحلیل می‌شوند.

در شکل ب، به اعمال‌گرهای مبتنی بر رفتار برنامه از دید اندازه تحلیل‌شده توجه شده است.

۷-۲-۱ مداخله فراخوانی سامانه

برای تکنیک‌هایی که با مداخله‌گری به ازای هر دستور یا فراخوانی به اعمال خط‌مشی می‌پردازند، این اندازه یک، و برای دیگر تکنیک‌ها حداقل یک است. مداخله فراخوانی[۲۰۲] تکنیک اعمالی است که ناظر صریحاً وقوع فراخوانی‌های خاصی را مدنظر دارد و آن‌ها را مسدود می‌کند یا تغییر می‌دهد. این تکنیک در سطح فراخوانی سیستم یا بالاتر قابل انجام است.

جدول ۲ - خلاصه‌ای از تکنیک‌ها و پیاده‌سازی‌های روش مداخله فراخوانی سامانه [۲۳]

Criterion	System call implementations
Abstraction level	Operating system
Guarantees	mediation (full if in kernel), tamper-proof
Trusted components	OS, system call wrappers, libraries
Policy class	access control
Policy language	very low level, not user-friendly
Overheads	very low (ms), depends on no.system calls

۷-۲-۲ مفسر ایمن

مفسرهای ایمن[۲۰۳] نیز به ازای هر دستور اجرا می‌شوند. در این تکنیک، یک لایه مجازی تعامل‌های بین یک برنامه در حال اجرا با پردازنده مرکزی را میانجی‌گری می‌کند. بزرگترین مزیت این روش آن است که برنامه‌های غیرمورد اعتماد نمی‌توانند مستقیماً به منابع سیستم دسترسی یابند. سطح

[202] Call Interposition

[203] Safe Interpreter

انتزاع این تکنیک‌ها بالا است و خط‌مشی‌های این‌گونه به راحتی قابل فهم است. از نظر سربار از تکنیک‌های نوع قبلی بدتر هستند. زیرا تفسیر کد کارایی بسیار پایین‌تری نسبت به فراخوانی سیستم دارد. از همین رو است که مفسرها برای برنامه‌های کاربردی پیچیده استفاده نمی‌شود. گرچه سربار بعضی از پیاده‌سازی‌ها به یک درصد می‌رسد، اما برای تکنیک چنداجرایی[۲۰۴]، ترکیب مفسر و هر اجرای موازی سرباری بین ۲۵ تا ۲۰۰ درصد خواهد داشت.

جدول ۳ - خلاصه‌ای از تکنیک‌ها و پیاده‌سازی‌های روش مفسر ایمن [۲۳]

Criterion	Safe interpreter implementations
Abstraction level	Application level
Guarantees	tamper-proof, but not non-bypassability
Trusted components	browser, helper modules,interpreter
Policy class	access control
Policy language	scripting languages, or customized
Overheads	from 1 to 25% (peaks at 200%)

تکنیک‌های با اندازه برنامه تحلیل‌شده بزرگتر از یک به بازنویسی برنامه مرتبط هستند. بازنویسی برنامه تکنیکی است که هدف آن رفع نقض‌های خط‌مشی امنیتی به کمک تغییر بخش‌هایی از برنامه است. برخلاف روش‌های قبلی که به ازای هر دستور در زمان انجام می‌شدند، این تغییرات تنها یک‌بار برای کل برنامه اعمال می‌شوند. بازنویسی برنامه می‌تواند در سطح زبان سطح بالا یا دودویی صورت بگیرد. بازنویس دودویی، کد دودویی برنامه را می‌گیرد و کد را تبدیل و بهینه‌سازی می‌کند. دو نوع برای بازنویسی برنامه وجود دارد: بازنویسی ایستا[۲۰۵] و بازنویسی پویا[۲۰۶]. در بازنویسی ایستا، کد دودویی را روی دیسک تغییر می‌دهد. به این ترتیب ابزار بازنویس تنها یک‌بار برای اجراهای متعدد برنامه نیاز است و سربار ثابتی خواهد داشت. اما در بازنویسی پویا، مقادیر دودویی در حافظه تغییر می‌کنند. ابزار باید در

[204] Multi-Execution Technique

[205] Static Rewriting

[206] Dynamic Rewriting

زمان‌اجرا حضور داشته باشد و درج یا حذف دستور همزمان با اجرای برنامه انجام می‌شود. این تکنیک‌ها را می‌توان به دسته‌های مختلف که در ادامه مطرح شده‌اند، تقسیم‌بندی کرد.

۷-۲-۳ ایزوله‌کردن خطای نرم‌افزار

ایزوله‌کردن خطای نرم‌افزار[۲۰۷] آدرس‌های حافظه را در کد شی[۲۰۸] یا اسمبلر تغییر می‌دهد تا از خواندن‌ها و نوشتن‌ها و پرش به آدرس‌های خارج از ناحیه تعریف‌شده توسط خط‌مشی جلوگیری کند. این تکنیک‌ها مستقل از زبان مبدأ هستند و ایمنی کد سطح پایین را فراهم می‌کنند. هنگامی که یک برنامه کاربردی مجاز باشد تا به صورت پویا مؤلفه‌های غیرمورد اعتماد را بارگذاری کند، این‌که آن مؤلفه‌ها سیستم میزبان را خراب نکند اهمیت دارد. بنابراین نیاز است تا این چنین مؤلفه‌هایی را درون مؤلفه‌های خطا[۲۰۹] ایزوله کرد. مدل ایزوله‌کردن خطای نرم‌افزار یک کپسوله‌سازی است که کد شی نتواند روی آدرس‌های حافظه خارج از بازه خود عملیاتی انجام دهد. این تکنیک بیشتر در زمان بارگذاری[۲۱۰]، نسبت به زمان اجرا، صورت می‌گیرد. طبق نظر Schneider، این تکنیک را نمی‌توان مکانیزم نظارت زمان‌اجرا دانست. زیرا برنامه هدف را قبل از این‌که واقعاً اجرا شود، دچار تغییراتی می‌شود. با این وجود، دستوراتی که توسط این تکنیک درج می‌شود را می‌توان با یک خودکاره امنیتی پیاده‌سازی کرد. به طور کلی، تکنیک‌های ایزوله‌سازی خطای نرم‌افزار به کنترل دسترسی سطح پایین کدهای غیرمورد اعتماد توجه دارد. از مزایای این تکنیک، سربار نه چندان زیاد آن است.

---

[207] Software Fault Isolation

[208] Object Code

[209] Fault Modules

[210] Load Time

جدول ۴ - خلاصه‌ای از تکنیک‌ها و پیاده‌سازی‌های روش ایزوله‌کردن خطای نرم‌افزار [۲۳]

Criteria	SFI implementations
Abstraction	OS and platform
Guarantees	nonbypassability, tamproofness
Trust components	OS, rewriter, compiler
Policy class	access control
Policy language	high-level (Naccio) otherwise very low level
Overheads	low to medium, e.g., 9-45%

۷-۲-۴ ناظرهای مرجع درخط

ناظرهای مرجع درخط[۲۱۱] نظارت اجرا را با برنامه‌های غیرمورد اعتماد تلفیق می‌کنند. در شکل زیر تفاوت بین ناظر مرجع درخط و بازنویس برنامه قابل مشاهده است. این ناظرها می‌توانند بازشناس یا تمیزکننده باشند. یک ناظر مرجع درخط از یک ابزار بازنویس مورد اعتماد استفاده می‌کنند که کدهای امنیتی را در برنامه هدف درج می‌کند تا از نقض خط‌مشی جلوگیری شود [۲۱]. این نوع ناظرها به سه نوع اطلاعات زیر برای اعمال نیاز دارند:

- رویدادهای امنیتی عملیاتی هستند که توسط خط‌مشی امنیتی حساس اعلام شده‌اند؛ مانند فراخوانی‌های سیستم

- حالت امنیتی که به هر نوع اطلاعاتی اطلاق می‌شود که منطق خط‌مشی برای تصمیم‌گیری نیاز دارد؛ مانند تاریخچه اجراها

- به‌روزرسانی امنیتی که به اقدامی که برنامه باید در قبال وقوع رویداد امنیتی اتخاذ کند که می‌تواند به‌روزرسانی حالت امنیتی باشد. این اقدام می‌تواند از ارسال سیگنال نقض خط‌مشی تا مجموعه‌ای از اقدامات اصلاحی باشد.

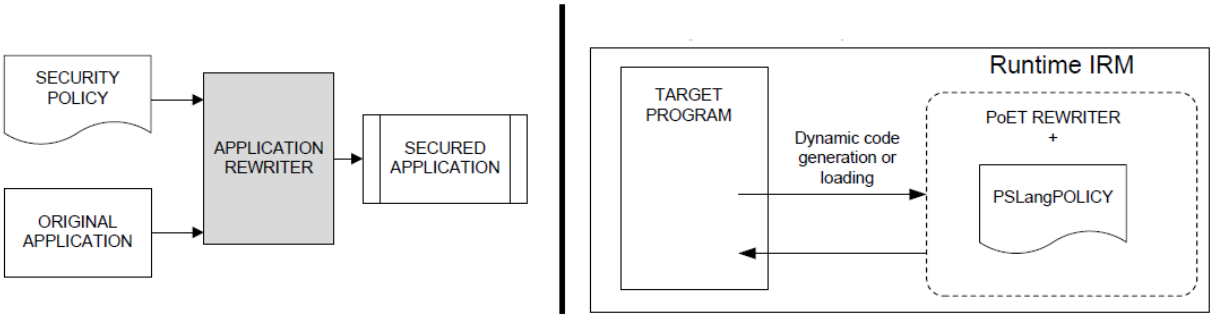
[211] Inline Reference Monitor

توانایی اعمال خودکاره امنیتی از متوقف‌کردن اجرای برنامه هدف تا عملیات توقیف یا درج می‌تواند باشد. خودکاره ویرایش که توسط Ligatti و همکاران در [۳۵] معرفی شد، می‌تواند اجرای برنامه هدف را قطع کند، یا دستوری به آن اضافه و درج کند، و یا تغییر دهد. باید توجه داشت که بازنویس‌های برنامه، برنامه‌ها را تغییر می‌دهند؛ در حالی‌که خودکاره‌های ویرایش می‌توانند گام‌های یک اجرا را تغییر دهند. معرفی خودکاره‌های ویرایش باعث افزایش قدرت ناظرها شده است. خودکاره ویرایش می‌تواند وانمود کند که اجرای برنامه هدف مجاز است تا زمانی‌که ناظر در نهایت تصمیم بگیرد که اجرا معتبر بوده است یا نه. به عبارت دیگر، دنباله‌ای از دستورات غیرمورد اعتماد توقیف شده و فقط در صورتی که خط‌مشی را نقض نمی‌کنند، بازدرج شوند. با توجه به بررسی‌های انجام‌شده در مقالات [۱۴]، [۱۹]، [۵۱] می‌توان این نتایج را برای دستاوردهای جدید مطرح کرد:

- اگر یک خط‌مشی در زمان‌اجرا قابل اعمال باشد، آنگاه توسط یک ناظر مرجع درخط نیز قابل اعمال است.

- بیشتر خط‌مشی‌های قابل اعمال به صورت ایستا را می‌توان در زمان‌اجرا نیز اعمال کرد.

- بعضی از خاصیت‌های ناایمنی[۲۱۲] می‌توانند در زمان اجرا اعمال شوند که به توانایی‌های محاسباتی خودکاره امنیتی در مدل‌سازی اجرا بستگی دارد.



شکل ۱۲ –مقایسه بازنویسی برنامه و ناظر مرجع درخط [۲۳]

[212] Non-Safety Properties

ناظرهای مرجع درخط معمولا به رویدادها نزدیک‌تر به کاربرد، مانند فراخوانی‌ها متد و بازگشت‌ها، ایجاد ریسه و رویدادهای امنیتی خاص تمرکز دارند. این ناظرها روی کدهای میانی کار می‌کنند و به همین دلیل، مختص ابزارهای زمان‌اجرا مانند ماشین مجازی جاوا و دات‌نت هستند.

۵-۲-۷ ترجمه پویای نرم‌افزار

ترجمه پویای نرم‌افزار[۲۱۳] برای امنیت تکنیک کلی‌تری است که در زمان اجرا از کدکامپایل‌شده به کد دودویی ترجمه می‌کند. این تکنیک هر خط از کد از یک زبان را به دستورات زبان ماشین ترجمه می‌کند و اگر از مفسری استفاده شده باشد، آن‌ها را اجرا می‌کند. مفسرهای ایمن اجرای یک دستور را تبدیل می‌کند، در حالی‌که ترجمه پویای نرم‌افزار کل برنامه یا بخشی از آن را تبدیل می‌کند. ماشین‌های مجازی[۲۱۴] نمونه خوبی از مترجم‌های پویا است. دستورات را می‌گیرند، آن را به دستوراتی ترجمه می‌کنند و سپس نتیجه را آماده اجرا می‌کنند. از جمله کاربردهای ترجمه نرم‌افزار می‌توان به ترجمه دودویی، بهینه‌سازهای پویا[۲۱۵] و اشکال‌زداها[۲۱۶] اشاره کرد.

جدول ۵ - خلاصه‌ای از تکنیک‌ها و پیاده‌سازی‌های روش ناظر مرجع درخط [۲۳]

Criteria	IRM implementations
Abstraction	application and platform
Guarantees	mediation, integrity, tamproofness
Trust components	rewriter, policy compiler
Policy class	access control
Policy language	security automata or Java-like
Overheads	low to medium, e.g., 0.1-30%

[213] Software Dynamic Translation

[214] Virtual Machines

[215] Dynamic Optimizers

[216] Debuggers

با وجود ماشین مجازی جاوا و مترجم‌های دودویی، یک لایه مجازی‌سازی توسط ترجمه پویا مطرح می‌شود که به امنیت و قابلیت حمل[۲۱۷] بیشتر کمک می‌کند. تکنیک‌های ترجمه پویای نرم‌افزار تضمینی قوی برای عدم قابلیت دورزدن مکانیزم امنیتی می‌دهد. همچنین پیاده‌سازی‌های این تکنیک کارایی بسیار پایین‌تری نسبت به تکنیک مداخله فراخوانی‌های سیستم دارد. به همین دلیل است که از این تکنیک برای برنامه‌های پیچیده استفاده نمی‌شود.

جدول ۶ - خلاصه‌ای از تکنیک‌ها و پیاده‌سازی‌های روش ترجمه پویای نرم‌افزار [۲۳]

Criteria	SDT implementations
Abstraction level	runtime and application logic level
Guarantees	nonbypassable, tamperproof
Trusted comp.	OS,interpreter,compiler
Policy class	access control
Policy language	low-level, or custom
Overheads	low to medium e.g., 2-30%

۶-۲-۷ بافنده‌های پویای جنبه

بافنده‌های پویای جنبه[۲۱۸] از دیگر تکنیک‌های مطرح‌شده هستند. برنامه‌نویسی جنبه‌گرا[۲۱۹] یکی از پارادایم‌های مهندسی نرم‌افزار است که به جداسازی نگرانی‌ها[۲۲۰] از تکامل نرم‌افزار تأکید دارد. بعضی از نیازمندی‌های برنامه کاربردی مانند امنیت آن، نگرانی‌هایی ورای کل برنامه است و اضافه‌کردن یا حذف آن‌ها باعث ایجاد تغییر در کل برنامه می‌شود. رویکرد برنامه‌نویسی جنبه‌گرا برای حل این مشکل، *بافتن*[۲۲۱] برنامه اولیه با یک یا چند جنبه یا نگرانی است. به این معنا که از ابتدا با در نظر گرفتن آن

[217] Portability

[218] Dynamic Aspect Weavers

[219] Aspect-Oriented Programming

[۲۲۰] Concern

[221] Weave

جنبه یا نگرانی، نرم‌افزار توسعه یابد. از مزایای این رویکرد در بحث امنیت می‌توان به این موارد اشاره کرد: (۱) یک وسیله سراسری برای وارسی امنیت فراهم می‌شود، زیرا تمامی کدهای مربوط به امنیت در یک مکان خواهد بود. (۲) نگرانی‌های امنیتی در توسعه نرم‌افزار جداسازی شده است و لازم نیست توسعه‌دهنده در طول توسعه برنامه به مسائل امنیتی فکر کند، و (۳) می‌توان از خط‌مشی‌های امنیتی در برنامه‌های کاربردی دیگر بازاستفاده کرد. به عنوان مثال، برای اضافه کردن مشخصه تصدیق اصالت باید تقریبا همه کلاس‌های یک برنامه شی‌گرا را تغییر داد، حال آن‌که در این رویکرد این مشکل وجود نخواهد داشت.

جنبه‌ها می‌توانند به دو شیوه در برنامه‌های کاربردی بافته شوند: مقیدکردن[۲۲۲] زمان‌کامپایل و مقیدکردن زمان‌اجرا. بافتن ایستای جنبه زمانی اتفاق می‌افتد که یک بافنده جنبه، فایل کد منبع یک جنبه را با کد منبع یک کلاس ادغام کند. اما بافتن پویا در زمان اجرا رخ می‌دهد و جنبه‌ها را در حین اجرا فعال و غیرفعال می‌کند. به این ترتیب جنبه‌ها می‌توانند در لحظات مختلفی از زمان اجرا ایجاد شوند اما لازم است تا برنامه کاربردی قادر به پشتیبانی از این امکان باشد. با توجه به این‌که این گزارش به مکانیزم‌های امنیتی زمان‌اجرا توجه دارد، به شیوه دوم از مقیدکردن پرداخته می‌شود.

بافتن پویای جنبه تکنیکی در سطح برنامه کاربردی است. گرچه بیانگری و انعطاف‌پذیری خط‌مشی‌های تعریف‌شده توسط کاربر در این تکنیک بالا است، اما تضمین خوبی برای عدم دورزدن مکانیزم داده نمی‌شود.

[222] Binding

جدول ۷ - خلاصه‌ای از تکنیک‌ها و پیاده‌سازی‌های روش بافتن پویای جنبه [۲۳]

Criteria	Dynamic weaving implementations
Abstraction level	application logic level
Guarantees	tamperproof
Trusted components	rewriter, interpreter, compiler, aspects
Policy class	access control
Policy language	automata, very high level
Overheads	high or very high e.g., 50-100%

تکنیک‌های مبتنی بر برنامه‌نویسی جنبه‌گرا سربار قابل توجهی را تحمیل می‌کنند که البته به نوع خط‌مشی مورد استفاده، تعداد وارسی‌های پویا و متد بافتن بستگی دارد.

۷-۲-۷اعمال روی جریان داده

خط‌مشی‌هایی از جنس «هیچ داده‌ای از این فایل نباید از طریق ایمیل ارسال شود» را نمی‌توان به وسیله کنترل دسترسی بیان کرد. زیرا کنترل دسترسی نمی‌تواند چگونگی پردازش اطلاعات در یک فایل را ردیابی کند. کنترل جریان اطلاعات[۲۲۳] به همین منظور مطرح شده است. طبق این خط‌مشی، افراد غیرمجاز نباید به اطلاعات حساس دسترسی پیدا کنند. یا به عبارت دیگر، هیچ نشتی‌ای نباید به سطح پایین وجود داشته باشد. می‌توان با تحلیل پویای جریان داده[۲۲۴]، خط‌مشی‌های کنترل جریان اطلاعات را اعمال کرد. این تحلیل می‌تواند در سطوح مختلف انتزاع مانند سیستم عامل، زمان‌اجرا، کتابخانه‌ها و برنامه کاربردی صورت بگیرد.

برای جلوگیری از حملاتی مانند تزریق SQL باید از کنترل جریان داده استفاده شود. این کنترل در زمان‌اجرا در سه گام لکه‌دارکردن[۲۲۵]، ردیابی[۲۲۶] و اظهارکردن[۲۲۷] انجام می‌شود. در گام اول، داده‌ای که

[223] Information Flow Control

[224] Dynamic Data Flow Analysis

[225] Tainting

از منابع غیرمورد اعتماد آمده است، نشانه‌گذاری می‌شود. در گام ردیابی، همه عملیات بعدی روی داده‌های لکه‌دارشده ردیابی می‌شوند، و در گام پایانی، در صورتی که داده نشانه‌گذاری‌شده در عملیات غیرمجاز استفاده شده باشد، بروز نقض امنیتی اظهار می‌شود.

اعمال کنترل جریان اطلاعات بیشتر به صورت ایستا انجام می‌شود. یکی از مهم‌ترین نمونه‌ها که اعمال کنترل جریان اطلاعات به صورت ایستا بوده است، JFlow است. اخیراً تحلیل‌های آلودگی پویا[۲۲۸] برای بررسی جریان‌های ضمنی مورد استفاده قرار گرفته‌اند. کارهای انجام‌شده در این بخش را می‌توان به رویکردهای مبتنی بر JVM[۲۲۹] و رویکردهای غیر مبتنی بر JVM دسته‌بندی کرد [۲۳].

JFlow یکی از ابزارهای مبتنی بر JVM است. در این زبان، اعمال‌گر به صورت ایستا و پویا جریان داده را به کمک حاشیه‌نویسی‌ها[۲۳۰] وارسی می‌کند، و برای متغیرها از برچسب‌های پویا استفاده می‌کند. سپس برای هر دسترسی متد به داده، کنترل دسترسی پویا انجام می‌دهد.

در رویکرد غیرمبتنی بر JVM، تمرکز بیشتری روی کامپایلر، کتابخانه‌های زمان‌اجرا یا سیستم عامل وجود دارد. این‌گونه ابزارها معمولاً به کد منبع برنامه نیاز دارند. به همین دلیل نمی‌توان آن‌ها را کاملاً جزو رویکردهای زمان‌اجرا به شمار آورد. RIFLE از جمله این ابزارها است. این ابزار دودوییِ برنامه کاربردی عادی را به حالت دودویی دیگری تبدیل می‌کند که روی یک معماری مجموعه دستورات خاصی اجرا می‌شود تا طبق آن بتوان جریان‌های صریح و ضمنی داده را ردیابی کرد. یک رویکرد کاملاً پویا در ابزار TaintDroid معرفی شده است. این ابزار که یک راه‌حل امنیتی زمان‌اجرا برای گوشی‌های هوشمند اندروید است، نیازی به در اختیارداشتن کد منبع برنامه کاربردی ندارد. TaintDroid مسیری که داده‌های حساس از برنامه‌های کاربردی مختلف می‌گذرند را به کمک ماشین مجازی Dalvik ردیابی می‌کند. همه آلودگی‌ها در یک نقشه آلودگی مجازی ذخیره می‌شود و در زمان‌اجرا، وقوع رویدادهای مرتبط با آن‌ها، مانند فراخوانی یک متد که با داده‌های حساس کار می‌کند، وارسی می‌شود.

---

[226] Tracking

[227] Asserting

[228] Dynamic Taint Analysis

[229] Java Virtual Machine

[230] Annotations

ترکیب کنترل جریان اطلاعات و ناظرهای مرجع درخط از ایده‌های اخیر محسوب می‌شود. Le Guernic [۵۲] دو ناظر عدم تداخل، مبتنی بر خودکاره ویرایش و معناشناختی مخصوص اجراهای نظارت‌شده ارائه کرده است. به این ترتیب ترکیبی از تحلیل ایستا و پویا را برای برآورده‌سازی عدم تداخل پیشنهاد شده است. درستی و شفافیت جزئی هر دو تکنیک اثبات شده است و این ایده‌ها علاوه بر تشخیص جریان‌های اطلاعات، برای تصحیح جریان‌های غیرمجاز در زمان‌اجرا مورد استفاده قرار گرفته است. همچنین، Sabelfeld در [۲۴] تحلیل پویای محض را برای کنترل جریان اطلاعات ارائه می‌کند. ضمناً اثبات می‌کند که هم تحلیل ایستا و هم اعمال پویا عدم تداخل غیرحساس به خاتمه را برآورده می‌کنند. برای این ویژگی، احتمال این‌که مهاجم یک مقدار محرمانه بزرگتر از یک بیت را بفهمد، ناچیز است [۵۳].

در ادامه رویکردهای ذکرشده در معیارهای مختلف بررسی شده‌اند [۲۳]. سربار پیاده‌سازی‌های کنترل جریان اطلاعات پویا بین ۳۰ تا ۲۰۰ درصد برآورد شده‌اند. شاید بهترین ابزار از این منظر، TaintDroid باشد. زیرا سربار ۲۷ درصدی از این ابزار گزارش شده است.

جدول ۸ - خلاصه‌ای از تکنیک‌ها و پیاده‌سازی‌های روش ردیابی جریان داده [۲۳]

Criterion	Data Flow Trackers
Abstraction level	OS, runtime, platform, application
Guarantees	full mediation but not always proven
Policy class	both access and usage control
Trust components	the OS, runtime or platform libraries
Policy language	complex, Java-like syntax in best case
Overheads	very high with minimum overheads around 30%

۷-۳ بحث روی اعمال زمان‌اجرا

به طور کلی، اعمال‌گرها برای اعمال یک خط‌مشی سه گام اصلی دارند: (۱) نظارت بر اجرای برنامه یا جریان داده، (۲) تشخیص این‌که چیز ناخواسته‌ای رخ داده است یا در حال وقوع است، و (۳) اقدام برای ایجاد یک اثر مطلوب روی برنامه هدف. مدل اعمال که در آن هر عملیات حساسی مسدود

می‌شود، همواره پیشگیرانه است و به وجود قیود در خط‌مشی برای پیش‌بینی همه علائم رفتار بدخواهانه نیازمند است.

مدل اعمال که در آن برنامه هنگام وقوع رفتار ناخواسته متوقف شود، به دلیل سخت‌گیری بیش از اندازه ممکن است عملی نباشد. به همین دلیل، دو رویکرد دیگر می‌تواند جایگزین شود: اول آن‌که برنامه هدف در یک محیط امن اجرا شود که همه فراخوانی‌های سیستم نظارت می‌شود. فراهم‌کردن چنین محیط امنی خود هزینه‌بر است. دیگر آن‌که کد برنامه هدف طوری بازنویسی شود که رفتار بدی نداشته باشد. مشکل روش بازنویسی این است که به مدل خودکاره امنیتی وابستگی دارد. این در حالی است که خودکاره امنیتی نمی‌تواند به درستی یک اجرای برنامه را تصحیح کند. در مدل‌های اعمال مبتنی بر خودکاره، ناظر می‌تواند از اجراهای قبلی برنامه با در نظر گرفتن محدودیت‌های حافظه‌ای برای تصمیم‌گیری استفاده کند اما فرایند تصمیم‌گیری کاملاً ایزوله شده است؛ به این معنی که از موجودیت دیگری از سیستم نمی‌تواند ورودی بگیرد. از دیگر محدودیت‌های خودکاره‌های امنیتی این است که فرض آن‌ها بر این است که خط‌مشی امنیتی از قبل آماده است.

به طور کلی، سربار مکانیزم متناسب است با سطح انتزاع و روشی که آن مکانیزم کار می‌کند [۲۳]. به این ترتیب که هر چه از سطح ماشین و سیستم عامل دور شویم، سربار بیشتر خواهد شد. پیاده‌سازی‌های مداخله‌گری فراخوانی‌های سیستم بهترین کارایی را دارند. مفسرهای ایمن بین یک تا ۲۵ درصد، ایزوله‌سازی خطای نرم‌افزار از ۹ تا ۴۵ درصد، ناظرهای مرجع درخط از یک دهم تا ۳۰ درصد و مترجم‌های پویای نرم‌افزار از دو تا ۳۰ درصد سربار به کارایی سیستم اضافه می‌کنند. اما روش‌های بافتن جنبه و ردیاب‌های جریان داده سرباری بیش از ۳۰ درصد دارند.

فصل هشتم

مقایسه مکانیزم‌های پویای جریان اطلاعات

در این فصل، مقایسه‌ای روی مکانیزم‌های دیگر پویا جریان اطلاعات با توجه به دو بُعد درستی[۲۳۱] و شفافیت[۲۳۲] انجام می‌شود. برای سادگی، از لفظ *ناظر* برای همه مکانیزم‌های مطرح‌شده استفاده می‌شود [۵۴]؛ در حالی‌که مشخصاً بعضی از این مکانیزم‌ها طبق تعریف، ناظر زمان‌اجرا محسوب نمی‌شوند.

۸-۱ مقایسه تعاریف مختلف عدم‌تداخل

برای درستی، از تعریف استاندارد عدم تداخل غیرحساس به خاتمه[۲۳۳] TINI استفاده شده که اجازه وقوع کانال‌های خاتمه را می‌دهد. در حالیکه فرقی بین اینکه کانال خاتمه در خود برنامه اصلی وجود داشته است یا توسط یک ناظر اضافه شده است، قائل نمی‌شود. در این کار، یک تعبیر قوی‌تری به نام عدم تداخل آگاه از خاتمه[۲۳۴] (TANI) نیز ارائه می‌شود که همین مسئله بالا را بررسی می‌کند و هم بنابراین، اجازه می‌دهد تا ارزیابی بهتری از امنیتی که ناظرهای مختلف تضمین می‌کنند، داشته باشیم.

برای شفافیت، تعابیر متفاوتی که در کارهای گذشته مطرح شده است، مورد بررسی قرار می‌گیرد. یک تعبیر مرسوم مورد استفاده در کارهای گذشته به تنهایی کافی نیست، از آنجا که تعیین می‌کند که ناظری بهتر است که اجراهای ناامن را بپذیرد و درباره دانش مهاجم بحث می‌شود. برای تمایز قائل‌شدن بین رفتارهای ناظرها روی اجراهای امن و ناامن، دو تعبیر شفافیت حقیقی[۲۳۵] و شفافیت کاذب[۲۳۶] را مطرح می‌کنیم. با این تعابیر، ناظرهایی که قبلا با یکدیگر قابل مقایسه نبودند را نیز می‌توان مقایسه کرد.

---

[231] Soundness

[232] Transparency

[233] Termination-Insensitive Noninterference

[234] Termination-Aware Noninterference

[235] True Transparency

[236] False Transparency

پنج دسته از ناظرهای جریان اطلاعات بررسی می‌شوند که عبارتند از [۵۴]: ارتقا-بدون-حساسیت[۲۳۷] (NSU) [۳۲]، ارتقای آسان‌گیر[۲۳۸] (PU) [۵۵]، ناظر ترکیبی[۲۳۹] (HM) [۵۲]، چنداجرایی امن[۲۴۰] (SME) [۴] و وجه‌های چندگانه[۲۴۱] (MF) [۵۶].

در سال‌های اخیر با توجه به طبیعت پویا و تعداد زیاد آسیب‌پذیری‌های موجود در برنامه‌های وب، مکانیزم‌های اعمال پویایی در قالب ناظرهای جریان اطلاعات مطرح شده‌اند. در ناظرهای زمان‌اجرا، دو ویژگی ناظرها به طور خاص مورد توجه است: درستی و شفافیت.

یک ناظر جریان اطلاعات دارای درستی است اگر تضمین دهد که خروجی‌های قابل مشاهده برآورده‌کننده خط‌مشی جریان اطلاعات مورد نظر است. در بیان عدم تداخل، اگر اجراها از ورودی‌های قابل مشاهده یکسان آغاز می‌شوند، ناظر باید این اطمینان را بدهد که خروجی‌های قابل مشاهده نیز یکسان خواهند بود.

باید توجه داشت که بعضی از تکنیک‌های نظارت کانال‌های خاتمه جدیدی را مطرح می‌کند که در سایر تکنیک‌ها وجود ندارد.

عدم تداخل حساس به خاتمه[۲۴۲] (TSNI) یک خط‌مشی قوی‌ای است که اجازه وقوع هیچ‌گونه کانال خاتمه‌ای را نمی‌دهد. از آن جهت این خط‌مشی را حساس به خاتمه گویند که اجازه نمی‌دهد تا مهاجم اطلاعات محرمانه‌ای از واگرایی[۲۴۳] برنامه به دست آورد. البته بسیاری از ناظرهای جریان اطلاعات این خط‌مشی را برآورده نمی‌کنند. به این ترتیب، نمی‌توان به خوبی ناظرهای مختلف را با یکدیگر مقایسه کرد. بنابراین، تعبیری از عدم تداخل، که از عدم تداخل غیرحساس به خاتمه قوی‌تر ولی از حساس به خاتمه ضعیف‌تر است، به نام عدم تداخل آگاه به خاتمه ارائه می‌شود. در این خط‌مشی این

---

237 No-Sensitive-Upgrade

238 Permissive-Upgrade

239 Hybrid Monitor

240 Secure-Multi Execution

241 Multiple Facets

242 Termination-Sensitive Noninterference

243 Divergence

مسئله مورد توجه قرار می‌گیرد که ناظر کانال خاتمه جدیدی را مطرح نکند. همان‌طور که در این مقاله به دست آمده است، HM، SME و MF خط‌مشی TANI را برآورده می‌کنند در حالی‌که NSU و PU قادر به برآورده‌کردن این خط‌مشی نیستند.

مثال ۱-۸ - اگر برنامه if h=0 then l=1; output l; را در نظر بگیرید، NSU اجرای این برنامه را با حالت آغازین [h=1, l=0] مجاز می‌داند و دیگر اجراها را متوقف می‌کند. پس همین باعث مطرح‌شدن یک کانال خاتمه جدید، به واسطه حضور ناظر، می‌شود.

یک ناظر جریان اطلاعات شفاف است اگر اجرای برنامه‌ای با خط‌مشی سازگار باشد، معناشناخت برنامه حفظ شود. در بیان عدم تداخل، ناظر باید همان خروجی‌ای را تولید کند که یک اجرای برنامه اصلی با مقداری تنها وابسته به ورودی‌های قابل مشاهده تولید می‌کند.

برای مقایسه رفتار ناظرها، مفاهیم و معیارها متفاوتی در کارهای گذشته مطرح شده است؛ از جمله دقت[۲۴۴]، آسان‌گیربودن[۲۴۵] و شفافیت. آسان‌گیربودن یک ناظر معیار خوبی برای مقایسه ناظرها نیست. زیرا با این معیار ناظری بهتر است که اجراهای ناامن را می‌پذیرد و بنابراین دانش مهاجم مورد بحث است. به همین خاطر، دو مفهوم شفافیت حقیقی و شفافیت کاذب مطرح می‌شود تا بتوان بین رفتارهای مختلف برنامه تمایز قائل شد. شفافیت حقیقی به تعبیر استاندارد موجود از شفافیت در زمینه نظارت زمان‌اجرا مربوط می‌شود. توانایی یک ناظر برای حفاظت معناشناخت اجراهای امن را شفافیت حقیقی می‌نامند. همچنین، یک ناظر جریان اطلاعات شفاف کاذب است اگر در حالی‌که خط‌مشی امنیتی مورد نظر را برآورده نمی‌شود، معناشناخت اجرای برنامه اصلی را حفظ کند. ممکن است در نگاه اول شفافیت کاذب با درستی متناقض برسد اما این گونه نیست. زیرا جریان اطلاعات یک خاصیت روی یک اجرا نیست، ولی یک خاصیت روی چندین اجرا است، که فوق‌خاصیت نیز نامیده می‌شود.

ثابت می‌شود که HM از نظر TSNI دقیق‌تر از NSU است؛ به این معنا که شفافیت حقیقی بیشتری برای مجموعه برنامه‌های از نظر TSNI امن دارد. از طرفی NSU دارای شفافیت کاذب بیشتری نسبت به HM است [۵۴].

---

[244] Precision

[245] Permissiveness

در [۵۴]، مکانیزم‌های پویا برای اعمال امنیت جریان اطلاعات مدنظر بوده است. باید توجه داشت که طبق تعاریف، نمی‌توان همه این پنج دسته نام‌برده را ناظر نامید، اما به دلیل سادگی در کار، به اصطلاح ناظر نامیده می‌شوند. ناظرهای در نظر گرفته‌شده شامل ناظرهای پویای محض، مانند NSU و PU، ناظرهای ترکیبی به شکل مطرح‌شده در [۵۲]، چنداجرایی امن و ناظر وجه‌های چندگانه هستند. همه این ناظرها دست‌کم عدم تداخل غیرحساس به خاتمه را برآورده می‌کنند. البته ناظرهای دیگری مانند چنداجرایی امن، عدم تداخل حساس به خاتمه و زمان را هم اعمال می‌کنند. می‌دانیم که TINI اجازه وقوع کانال‌های خاتمه را می‌دهد. پس اجازه می‌دهد تا ناظرها نیز کانال‌های خاتمه جدیدی را مطرح کنند که در برنامه اصلی وجود نداشته است.

اما تعبیر جدیدی که در [۵۴] تعریف شده است، مفهوم عدم تداخل آگاه به خاتمه یا TANI است که اجازه مطرح‌کردن کانال خاتمه جدیدی را به ناظر نمی‌دهد. به طور شهودی، ناظر باید با همه حافظه‌هایی که دارای سطح پایین معادل هستند و برنامه اصلی با آن‌ها خاتمه می‌یابد، به صورت یکسانی برخورد کند. به این معنا که ناظر یا باید نتیجه مشابهی برای همه حافظه‌ها تولید کند یا برای همه آن‌ها، واگرا شود. در شرایطی که برنامه اصلی همواره واگراست، TANI زمانی برقرار است که ناظر نیز همواره واگرا باشد یا ناظر همیشه با مقدار مشابهی خاتمه یابد.

برای مثال، در همان برنامه if h=0 then l=1; output l;، برای برآورده‌شدن TANI لازم است که در [h=0, l=0] و [h=1, l=0] که از نظر مقدار سطح پایین معادل هستند و برنامه اصلی به ازای این حافظه‌ها خاتمه می‌یابد، ناظر یکسان رفتار کند؛ یعنی یا برای هر دو حافظه خروجی یکسان تولید کند یا برای هر دو، واگرا شود.

طبق تعاریف ارائه‌شده، به وضوح می‌توان دید که TSNI یک نوع قوی از عدم تداخل است که TINI را دربردارد. ضمناً با بیان قضیه زیر ارتباط بین خط‌مشی‌های TSNI، TANI و TINI مشخص می‌شود.

قضیه – در صورتی که یک ناظر قادر به اعمال TSNI برای همه برنامه‌ها باشد، پس قادر به اعمال TANI نیز خواهد بود و در صورتی که قادر به برقراری TANI برای همه برنامه‌ها باشد، می‌تواند TINI را نیز اعمال کند. یا به بیان صوری، $TSNI(M) \Rightarrow TANI(M)$ and $TANI(M) \Rightarrow TINI(M)$، که در آن M بیانگر ناظر است.

۸-۲ مکانیزم‌های پویای اعمال امنیت جریان اطلاعات

برای مقایسه ناظرهای ذکرشده، ابتدا همه آن‌ها براساس یک زبان ساده امری مدل می‌شوند که در ادامه نحو و معناشناخت زبان آمده است.

$$
\begin{aligned}
P &::= S;\ \mathsf{output}\ x \\
S &::= \mathsf{skip} \mid x := e \mid S_1; S_2 \mid \mathsf{if}\ x\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2 \mid \mathsf{while}\ x\ \mathsf{do}\ S \\
e &::= v \mid x \mid e_1 \oplus e_2
\end{aligned}
$$

$$
\text{SKIP}\ \frac{}{(\mathsf{skip}, \mu) \Downarrow \mu} \qquad \text{ASSIGN}\ \frac{}{(x := e, \mu) \Downarrow \mu[x \mapsto [\![e]\!]_\mu]} \qquad \text{SEQ}\ \frac{(S_1, \mu) \Downarrow \mu' \qquad (S_2, \mu') \Downarrow \mu''}{(S_1; S_2, \mu) \Downarrow \mu''}
$$

$$
\text{IF}\ \frac{[\![x]\!]_\mu = \alpha \qquad (S_\alpha, \mu) \Downarrow \mu'}{(\mathsf{if}\ x\ \mathsf{then}\ S_{true}\ \mathsf{else}\ S_{false}, \mu) \Downarrow \mu'} \qquad \text{WHILE}\ \frac{(\mathsf{if}\ x\ \mathsf{then}\ S; \mathsf{while}\ x\ \mathsf{do}\ S\ \mathsf{else}\ \mathsf{skip}, \mu) \Downarrow \mu'}{(\mathsf{while}\ x\ \mathsf{do}\ S, \mu) \Downarrow \mu'}
$$

$$
\text{OUTPUT}\ \frac{[\![x]\!]_\mu = v}{(\mathsf{output}\ x, \mu) \Downarrow (v, \mu)}
$$

$$
\text{where } [\![x]\!]_\mu = \mu(x),\ [\![v]\!]_\mu = v \text{ and } [\![e_1 \oplus e_2]\!]_\mu = [\![e_1]\!]_\mu \oplus [\![e_2]\!]_\mu
$$

شکل ۱۳ –نحو و معناشناخت زبان مورد استفاده در [۵۴]

در معناشناخت بالا، pc همان شمارنده برنامه، M نام ناظر و Γ یک محیط امنیتی است که متغیرها را به سطوح امنیتی نگاشت می‌دهد. فرض بر این است که تنها خروجی‌های تولیدشده توسط برنامه که دارای سطح امنیتی L یا پایین هستند توسط مهاجم قابل مشاهده‌اند.

در ادامه انواع ناظرها مورد بررسی قرار می‌گیرند:

۸-۲-۱ ارتقا-بدون-حساسیت (NSU)

رویکرد ارتقا-بدون-حساسیت که ابتدا در [۳۲] مطرح شد، بر پایه یک ناظر پویای محض است که فقط یک اجرا از برنامه را کنترل می‌کند. برای جلوگیری از جریان‌های اطلاعات ضمنی، NSU به متغیرهای سطح پایین، اجازه هیچ ارتقایی را در زمینه[۲۴۶] امنیتی سطح بالا نمی‌دهد. در برنامه مثال ۱،

[246] Context

اجرایی از برنامه که در آن h=0 باشد، توسط NSU مسدود خواهد شد. زیرا متغیر سطح پایین l در زمینه سطح بالا -همان زمینه ناشی از شرط h=0- به‌روز می‌شود.

معناشناخت NSU در شکل زیر آمده است.

$$\text{SKIP}\ \frac{}{pc \vdash (\Gamma, \mathsf{skip}, \mu) \Downarrow_{\mathbf{NSU}} (\Gamma, \mu)}$$

$$\text{ASSIGN}\ \frac{[\![e]\!]_\mu = v \qquad pc \sqsubseteq \Gamma(x) \qquad \Gamma' = \Gamma[x \mapsto \Gamma(e) \sqcup pc]}{pc \vdash (\Gamma, x := e, \mu) \Downarrow_{\mathbf{NSU}} (\Gamma', \mu[x \mapsto v])}$$

$$\text{SEQ}\ \frac{pc \vdash (\Gamma, S_1, \mu) \Downarrow_{\mathbf{NSU}} (\Gamma', \mu') \qquad pc \vdash (\Gamma', S_2, \mu') \Downarrow_{\mathbf{NSU}} (\Gamma'', \mu'')}{pc \vdash (\Gamma, S_1; S_2, \mu) \Downarrow_{\mathbf{NSU}} (\Gamma'', \mu'')}$$

$$\text{IF}\ \frac{[\![x]\!]_\mu = \alpha \qquad pc \sqcup \Gamma(x) \vdash (\Gamma, S_\alpha, \mu) \Downarrow_{\mathbf{NSU}} (\Gamma', \mu')}{pc \vdash (\Gamma, \mathsf{if}\ x\ \mathsf{then}\ S_{true}\ \mathsf{else}\ S_{false}, \mu) \Downarrow_{\mathbf{NSU}} (\Gamma', \mu')}$$

$$\text{WHILE}\ \frac{pc \vdash (\Gamma, \mathsf{if}\ x\ \mathsf{then}\ S; \mathsf{while}\ x\ \mathsf{do}\ S\ \mathsf{else}\ \mathsf{skip}, \mu) \Downarrow_{\mathbf{NSU}} (\Gamma', \mu')}{pc \vdash (\Gamma, \mathsf{while}\ x\ \mathsf{do}\ S, \mu) \Downarrow_{\mathbf{NSU}} (\Gamma', \mu')}$$

$$\text{OUTPUT}\ \frac{[\![x]\!]_\mu = v \qquad \Gamma(x) = L}{L \vdash (\Gamma, \mathsf{output}\ x, \mu) \Downarrow_{\mathbf{NSU}} (v, \Gamma, \mu)}$$

شکل ۱۴ - معناشناخت NSU [۵۴]

همان‌طور که در شکل بالا مشخص است، ایده اصلی رویکرد NSU در قاعده ASSIGN مطرح می‌شود؛ یعنی ناظر ارتقاهای حساس را زمانی مسدود می‌کند که سطح شمارنده برنامه pc پایین‌تر از سطح متغیر در حال انتساب نباشد.

اثبات شده است که NSU عدم تداخل غیرحساس به خاتمه را می‌تواند اعمال کند. همچنین، می‌توان با مثالی دید که NSU، عدم تداخل آگاه به خاتمه را نمی‌تواند اعمال کند. اگر برنامه مثال۱-۸ را با حافظه اولیه [h=1, l=0] در نظر بگیرید، ناظر تنها در این شرایط حافظه خاتمه می‌یابد، در حالی‌که برنامه اصلی به ازای هر دو حافظه که از دید سطح پایین معادل هستند، خاتمه پیدا می‌کند. به این ترتیب، NSU قادر به اعمال خط‌مشی TANI نیست.

۸-۲-۲ارتقای آسان‌گیر (PU)

گرچه رویکرد NSU برای اعمال TINI کافی است، اما اغلب اجرای یک برنامه را به صورت پیشگیرانه مسدود می‌کند. از این رو، Austin و Falnagan [۵۷] یک راهبرد کمتر سخت‌گیرانه‌ای به نام ارتقای

$$\text{ASSIGN}\ \frac{[\![e]\!]_\mu = v \qquad \Gamma' = \Gamma[x \mapsto \Gamma(e) \sqcup \mathsf{lift}(pc, \Gamma(x))]}{pc \vdash (\Gamma, x := e, \mu) \Downarrow_{\mathbf{PU}} (\Gamma', \mu[x \mapsto v])}$$

$$\text{IF}\ \frac{\Gamma(x) \neq P \qquad [\![x]\!]_\mu = \alpha \qquad pc \sqcup \Gamma(x) \vdash (\Gamma, S_\alpha, \mu) \Downarrow_{\mathbf{PU}} (\Gamma', \mu')}{pc \vdash (\Gamma, \text{if } x \text{ then } S_{true} \text{ else } S_{false}, \mu) \Downarrow_{\mathbf{PU}} (\Gamma', \mu')}$$

where

$$\mathsf{lift}(pc, l) = \begin{cases} L & \text{if } pc = L \\ H & \text{if } pc = H \wedge l = H \\ P & \text{if } pc = H \wedge l \neq H \end{cases}$$

شکل ۱۵ - معناشناخت PU [۵۴]

آسان‌گیر را معرفی کردند. برخلاف NSU، در این رویکرد اجازه انتساب به متغیرهای سطح پایین در زمینه سطح بالا داده می‌شود اما متغیر به‌روزشده را به عنوان *نشت جزئی* یا 'P' برچسب می‌زند. برچسب P به این معناست که محتوای متغیر سطح بالاست اما ممکن است در اجراهای دیگر سطح پایین باشد. اگر در ادامه اجرا، انشعابی وجود داشته باشد که به یک متغیر با برچسب P وابسته باشد، ناظر اجرا را متوقف می‌کند. در ادامه معناشناخت رویکرد ارتقای آسان‌گیر آمده است.

تفاوت معناشناخت PU با NSU تنها در قاعده ASSIGN و IF است. اثبات می‌شود که PU می‌تواند TINI را اعمال کند. گرچه با توجه به این‌که مکانیزم PU در ایجاد کانال‌های خاتمه جدید مشابه NSU است، پس PU نیز قادر به اعمال TANI نیست.

۸-۲-۳ناظر ترکیبی (HM)

اولین بار Le Guernic و همکارانش [۵۲] یک ناظر ترکیبی برای کنترل جریان اطلاعات ارائه کرد که تحلیل ایستا و پویا را با یکدیگر تلفیق کرده بود. این مکانیزم در هر شرط برنامه، انشعابی که اجرا نمی‌شود را تحلیل ایستا می‌کند و همه متغیرهایی که ممکن است در آن شاخه به‌روز شوند را جمع‌آوری

می‌کند. سپس سطح امنیتی چنین متغیرهایی به سطح شرط آزمون ارتقا می‌یابد و به این ترتیب از نشت اطلاعات جلوگیری می‌شود. معناشناخت HM نیز مشابه NSU است و فقط در قواعد زیر متفاوت است.

$$\text{ASSIGN}\ \frac{[\![e]\!]_\mu = v \qquad \Gamma' = \Gamma[x \mapsto pc \sqcup \Gamma(e)]}{pc \vdash (\Gamma, x := e, \mu) \Downarrow_{\mathbf{HM}} (\Gamma', \mu[x \mapsto v])}$$

$$\text{IF}\ \frac{\begin{array}{c}\Gamma'' = \mathsf{Analysis}(S_{\neg\alpha}, pc \sqcup \Gamma(x), \Gamma) \\ [\![x]\!]_\mu = \alpha \qquad pc \sqcup \Gamma(x) \vdash (\Gamma, S_\alpha, \mu) \Downarrow_{\mathbf{HM}} (\Gamma', \mu')\end{array}}{pc \vdash (\Gamma, \text{if } x \text{ then } S_{true} \text{ else } S_{false}, \mu) \Downarrow_{\mathbf{HM}} (\Gamma' \sqcup \Gamma'', \mu')}$$

$$\text{OUTPUT}\ \frac{\Gamma(x) = L \Rightarrow v = [\![x]\!]_\mu \qquad \Gamma(x) \neq L \Rightarrow v = \mathsf{def}}{L \vdash (\Gamma, \text{output } x, \mu) \Downarrow_{\mathbf{HM}} (v, \Gamma, \mu)}$$

شکل ۱۶ – معناشناخت HM [۵۴]

در همان مقاله [۵۲]، اثبات شده است که HM توانایی اعمال TINI را دارد. حال در این مقاله اثبات می‌شود که HM می‌تواند TANI را هم اعمال کند.

۸-۲-۴ چنداجرایی امن (SME)

این روش برای اولین بار توسط Devriese و Piessens در [۴] ارائه شد. ایده اصلی چنداجرایی امن این است که برنامه به ازای هر سطح امنیتی یک بار اجرا شود. در هر اجرا فقط ورودی‌های قابل مشاهده برای آن سطح امنیتی دریافت می‌شود و برای متغیرهای ورودی غیرقابل مشاهده برای آن سطح، مقدار ثابت def قرار داده می‌شود. اجراهای مختلف با یک زمان‌بند با اولویت پایین اجرا می‌شود تا از نشت‌های ناشی از اجراهای سطح بالا جلوگیری شود. به همین خاطر SME قادر به اعمال TSNI خواهد بود.

$$\text{SME}\ \frac{(P, \mu|_\Gamma) \Downarrow (v, \mu') \qquad \mu''' = \begin{cases}\mu' \odot_\Gamma \mu'' & \text{if } \exists \mu''.(P, \mu) \Downarrow (v', \mu'') \\ \mu' \odot_\Gamma \bot & \text{otherwise}\end{cases}}{pc \vdash (\Gamma, P, \mu) \Downarrow_{\mathbf{SME}} (v, \Gamma, \mu''')}$$

$$\text{where}\quad \mu|_\Gamma(x) = \begin{cases}\mu(x) & \Gamma(x) = L \\ \mathsf{def} & \Gamma(x) = H\end{cases} \qquad \mu' \odot_\Gamma \mu''(x) = \begin{cases}\mu'(x) & \Gamma(x) = L \\ \mu''(x) & \Gamma(x) = H\end{cases}$$

شکل ۱۷ - معناشناخت SME [۵۴]

معناشناخت چنداجرایی امن در شکل بالا آمده است. نکته قابل توجه آن است که اگر اجرای مربوط به سطح امنیتی H خاتمه نیابد، معناشناخت SME خاتمه خواهد یافت. در [۴] اثبات شده است که SME قادر به اعمال TSNI است. پس بنابر قضیه‌ای که پیشتر مطرح شد، این روش توانایی اعمال TANI را نیز دارد.

۸-۲-۵وجه‌های چندگانه (MF)

Austin و Falnagan در [۵۶] رویکرد وجه‌های چندگانه را مطرح کردند. در روش وجه‌های چندگانه، هر متغیر به ازای هر سطح امنیتی، به یک مقدار یا وجه نگاشت می‌شود. هر مقدار متناسب است با دید متغیر از نقطه نظر مشاهده‌گرهای در سطح مختلف امنیتی. ایده اصلی در این روش این است که اگر یک ارتقای حساس وجود دارد، معناشناخت MF وجه قابل مشاهده را به‌روز نکند. در غیر این صورت، اگر هیچ ارتقای حساسی موجود نبود، MF با توجه به معناشناخت اصلی برنامه به‌روزرسانی‌ها را انجام دهد.

در برنامه if h=0 then l=0 else l=0; output l; مقدار قابل مشاهده از دید سطح پایین L، یا وجه L از متغیر l، همیشه برابر با مقدار اولیه متغیر l است. زیرا روش MF یک متغیر سطح پایین را در زمینه امنیتی سطح بالا به‌روز نمی‌کند. بنابراین همه اجراهای این برنامه که با l=1 آغاز می‌شوند، مستقل از مقدار h، خروجی l=1 را تولید خواهند کرد. معناشناخت MF نیز در ادامه آمده است.

هر مقدار وجه‌دار را با $\langle v_1 : v_2 \rangle$ نمایش داده می‌شود که زوجی از مقادیر v_1 و v_2 است. مقدار اول بیانگر دید یک مشاهده‌گر در سطح H یا بالا است و مقدار دوم از دید مشاهده‌گری در سطح L یا پایین. اثبات می‌شود که MF قادر به اعمال TINI [۵۶] و TANI [۵۴] است.

به بیان ساده، دقت به معنای تعداد دفعاتی است که یک ناظر برنامه‌های امن را متوقف می‌کند یا تغییر می‌دهد. رویکردهای متفاوتی برای مقایسه دقت ناظرها وجود دارد که تعاریفی مانند «دقت»، «آسان‌گیربودن» و «شفافیت» از جمله آن‌هاست.

همان‌طور که پیشتر نیز مطرح شد، در حوزه نظارت زمان‌اجرا یک ناظر باید در هنگام اعمال یک خط‌مشی امنیتی درباره درستی و شفافیت آن تضمین دهد. شفافیت [۳۵] به این معنا که هرگاه یک اجرا از برنامه خاصیت مورد نظر را برآورده می‌کرد، ناظر باید همان اجرا را بدون تغییر به عنوان خروجی دهد.

۸-۳ تعابیر مختلف دقت و شفافیت

انواع مختلفی از مفاهیم دقت در کارهای گذشته موجود است که در ادامه به آن‌ها پرداخته می‌شود.

$$\text{MF Rule}\ \boxed{\frac{pc \vdash (\Gamma, P, \mu\uparrow_\Gamma) \downarrow_{MF} (\langle v_1 : v_2\rangle, \Gamma', \hat{\mu})}{pc \vdash (\Gamma, P, \mu) \Downarrow_{\mathbf{MF}} (v_2, \Gamma', \hat{\mu}\downarrow_{\Gamma'})}}\quad \text{SKIP}\ \frac{}{pc \vdash (\Gamma, \mathsf{skip}, \hat{\mu}) \downarrow_{MF} (\Gamma, \hat{\mu})}$$

$$[e]\hat{\mu} = \langle v_1 : v_2\rangle \qquad \hat{v} = \begin{cases} \langle v_1 : \hat{\mu}(x)_2\rangle & \text{if } pc = H \wedge \Gamma(x) = L \\ \langle v_1 : v_2\rangle & \text{if } pc = L \vee \Gamma(x) \neq L \end{cases}$$

$$\Gamma'(y) = \begin{cases} \Gamma(e) & \text{if } pc = L \wedge y = x \\ \Gamma(y) & \text{otherwise} \end{cases}$$

$$\text{ASSIGN}\ \frac{}{pc \vdash (\Gamma, x := e, \hat{\mu}) \downarrow_{MF} (\Gamma', \hat{\mu}[x \mapsto \hat{v}])}$$

$$\text{SEQ}\ \frac{pc \vdash (\Gamma, S_1, \hat{\mu}) \downarrow_{MF} (\Gamma', \hat{\mu}') \qquad pc \vdash (\Gamma', S_2, \hat{\mu}') \downarrow_{MF} (\Gamma'', \hat{\mu}'')}{pc \vdash (\Gamma, S_1; S_2, \hat{\mu}) \downarrow_{MF} (\Gamma'', \hat{\mu}'')}$$

$$\text{IF-HIGH}\ \frac{[x]\hat{\mu} = \langle \alpha_1 : \alpha_2\rangle \qquad pc = H \vee \Gamma(x) = H \qquad H \vdash (\Gamma, S_{\alpha_1}, \hat{\mu}) \downarrow_{MF} (\Gamma', \hat{\mu}')}{pc \vdash (\Gamma, \mathsf{if}\ x\ \mathsf{then}\ S_{true}\ \mathsf{else}\ S_{false}, \hat{\mu}) \downarrow_{MF} (\Gamma', \hat{\mu}')}$$

$$\text{IF-LOW}\ \frac{\begin{array}{c}[x]\hat{\mu} = \langle \alpha_1 : \alpha_2\rangle \qquad pc = L \wedge \Gamma(x) = L \\ L \vdash (\Gamma, S_{\alpha_1}, \hat{\mu}) \downarrow_{MF} (\Gamma', \hat{\mu}_1) \qquad L \vdash (\Gamma, S_{\alpha_2}, \hat{\mu}) \downarrow_{MF} (\Gamma', \hat{\mu}_2)\end{array}}{pc \vdash (\Gamma, \mathsf{if}\ x\ \mathsf{then}\ S_{true}\ \mathsf{else}\ S_{false}, \mu) \downarrow_{MF} (\Gamma', \hat{\mu}_1 \otimes_\Gamma \hat{\mu}_2)}$$

$$\text{WHILE}\ \frac{pc \vdash (\Gamma, \mathsf{if}\ x\ \mathsf{then}\ S; \mathsf{while}\ x\ \mathsf{do}\ S\ \mathsf{else\ skip}, \hat{\mu}) \downarrow_{MF} (\Gamma', \hat{\mu}')}{pc \vdash (\Gamma, \mathsf{while}\ x\ \mathsf{do}\ S, \hat{\mu}) \downarrow_{MF} (\Gamma', \hat{\mu}')}$$

$$\text{OUTPUT}\ \frac{[x]\hat{\mu} = \hat{v}}{L \vdash (\Gamma, \mathsf{output}\ x, \hat{\mu}) \downarrow_{MF} (\hat{v}, \Gamma, \hat{\mu})}$$

where

$$\begin{array}{ll} [\hat{v}]\hat{\mu} & = \hat{v}, \quad [x]\hat{\mu} = \hat{\mu}(x) \\ [e_1 \oplus e_2]\hat{\mu} & = \langle v_1 \oplus v_2 : v_1' \oplus v_2'\rangle, \ \text{where } [e_1]\hat{\mu} = \langle v_1 : v_1'\rangle, [e_2]\hat{\mu} = \langle v_2 : v_2'\rangle \end{array}$$

$$\hat{\mu}_1 \otimes_\Gamma \hat{\mu}_2(x) = \begin{cases} \hat{\mu}_1(x) & \text{if } \Gamma(x) = H \\ \langle \hat{\mu}_1(x)_1 : \hat{\mu}_2(x)_2\rangle & \text{if } \Gamma(x) = L \end{cases}$$

$$\mu\uparrow_\Gamma(x) = \begin{cases} \langle \mu(x) : \mu(x)\rangle & \text{if } \Gamma(x) = L \\ \langle \mu(x) : \bot\rangle & \text{if } \Gamma(x) = H \end{cases} \qquad \hat{\mu}\downarrow_\Gamma(x) = \begin{cases} \hat{\mu}(x)_1 & \text{if } \Gamma(x) = H \\ \hat{\mu}(x)_2 & \text{if } \Gamma(x) = L \end{cases}$$

شکل ۱۸ - معناشناخت MF [۵۴]

دقت (در مقابل برنامه‌های خوش‌نوع[۲۴۷]): در [۵۲] یک ناظر ترکیبی مطرح شده است که اثبات می‌شود همه اجراهای یک برنامه خوش‌نوع تحت یک نوع‌سامانه غیرحساس به جریان[۲۴۸] مشابه با کار Volpano [۱۵] را می‌پذیرد. همچنین در مقاله [۵۵] اثبات شده است که ناظر ترکیبی ارائه‌شده، همه اجراهای یک برنامه که از نظر نوع‌سامانه حساس به جریان خوش‌نوع هستند را معتبر به حساب می‌آورد.

دقت (در مقابل برنامه‌های امن): در [۴] تعبیر قوی‌تری به نام *دقت* مطرح شده است؛ به این معنا که یک ناظر باید همه اجراهای همه برنامه‌های امن را بپذیرد تا ناظری *دقیق* تلقی شود. این مفهوم از آن جهت قوی‌تر است که نه تنها ناظر باید قادر به تشخیص اجراهای برنامه‌های خوش‌نوع باشد، بلکه برنامه‌های امن دیگری که خوش‌نوع نیستند نیز مطرح می‌شوند. Devriese و Piessens اثبات کرده‌اند که تضمین این چنین دقتی برای روش چنداجرایی امن در برابر برنامه‌های TSNI وجود دارد.

شفافیت (در مقابل اجراهای امن): در [۵۸] ناظری بر مبنای روش SME ارائه شده است که حتی برای برنامه‌ای که ناامن باشد، *شفافیت برای TSNI* را برآورده می‌کند.

آسان‌گیربودن (در مقابل اجراهای پذیرفته‌شده توسط دیگر ناظرها): در [۶] ناظر ترکیبی پیشنهادی با ناظر ترکیبی دیگری مقایسه شده است که در آن تحلیل ایستای دقیق‌تری انجام می‌شود و یک قضیه *دقت بهبودیافته* اثبات می‌شود که بیان می‌کند هرگاه ناظر ترکیبی اول اجرایی را بپذیرد، ناظر دوم نیز آن را خواهد پذیرفت. در کار دیگری از تعبیری به نام *آسان‌گیربودن* نام برده می‌شود و به مقایسه مجموعه‌های اجراهای پذیرفته‌شده می‌پردازد. منظور آن است که ناظری آسان‌گیرتر از دیگری است که اگر مجموعه اجراهای پذیرفته‌شده‌اش شامل مجموعه‌ای از اجراهای پذیرفته‌شده ناظر دیگر باشد.

مشخص شد که در کارهای قبلی، معیارهای مقایسه ناظرها متفاوت بوده است [۵۴]. حال با تعریف دو تعبیر از شفافیت، سعی در ارائه معیاری برای مقایسه ناظرهای گوناگون با یکدیگر دارد؛ شفافیت حقیقی که اجراهای امن، و شفافیت کاذب که اجراهای ناامن پذیرفته‌شده توسط یک ناظر را تعریف می‌کند.

---

[247] Well-typed Programs

[248] Flow-insensitive Type System

۸-۳-۱ تعبیر شفافیت حقیقی و کاذب

تعریف شفافیت حقیقی برای TINI این‌گونه خواهد بود که یک ناظر شفاف حقیقی است اگر همه اجراهای TINI یک برنامه را بپذیرد.

تعریف – ناظر M شفاف حقیقی است اگر برای هر برنامه P، هر حافظه μ و μ’ و خروجی v، رابطه زیر برقرار باشد.

$$TINI(P,\mu_{\mathrm{L}}) \wedge (P,\mu) \Downarrow (v,\mu') \Rightarrow (P,\mu) \Downarrow_M (v,\mu')$$

یک خودکاره قطع‌کننده[۲۴۹] نمی‌تواند بیش از خاصیت‌های ایمنی محاسبه‌پذیر را تشخیص دهد [۱۴]. با توجه به این‌که عدم تداخل را می‌توان به یک خاصیت ایمنی که محاسبه‌پذیر نیست کاهش داد، و NSU و PU توسط خودکاره قطع‌کننده قابل مدل‌سازی هستند، پس این رویکردها شفاف حقیقی نخواهند بود. در [۵۴] نیز نشان داده می‌شود که ناظرهای ذکرشده که توسط خودکاره قطع‌کننده قابل مدل‌سازی نیستند نیز شفافیت حقیقی برای TINI ندارند.

با توجه به مثال‌های نقض ذکرشده در [۵۴]، HM، MF و SME برای TINI شفاف حقیقی نیستند. گرچه هیچ‌یک از ناظرهای مطرح‌شده برای TINI شفاف حقیقی نبودند اما می‌توان با تعریف یک شفافیت حقیقی نسبی، به مقایسه رفتار ناظرهای جریان اطلاعات در هنگام مواجهه با اجراهای امن پرداخت.

اگر یک برنامه را با P و یک ناظر را با M نشان دهیم، مجموعه‌ای از حافظه‌های اولیه را تعریف می‌کنیم که منجر به خاتمه امن اجراهای برنامه P می‌شوند و ناظر M آن اجراها را تغییر نمی‌دهد:

$$\mathcal{T}(M,P) = \{\mu \mid TINI(P,\mu_{\mathrm{L}}) \wedge \exists \mu', v.\ (P,\mu) \Downarrow (v,\mu') \Rightarrow (P,\mu) \Downarrow_M (v,\mu')\}$$

تعریف شفافیت حقیقی نسبی – ناظر A از ناظر B شفاف حقیقی‌تر است، و با $A \sqsupseteq_T B$ نوشته می‌شود، اگر برای هر برنامه P این رابطه برقرار باشد:

$$\mathcal{T}(A,P) \supseteq \mathcal{T}(B,P)$$

[249] Truncated Automata

در [۵۷] اثبات شده است که MF از PU و PU از NSU شفاف حقیقی‌تر است یا به بیان دیگر،

$$MF \supseteq_{\mathcal{T}} PU \supseteq_{\mathcal{T}} NSU$$

همچنین با بیان مثال‌های نقض، نشان داده می‌شود که ناظرهای دیگر مورد بررسی را نمی‌توان از این جهت مقایسه کرد. پس برای مقایسه این ناظرها نیاز به معیار دیگری داریم.

تعریف دقت – ناظر M دقیق است اگر به ازای هر برنامه P رابطه زیر برقرار باشد:

$$TINI(P) \wedge \forall \mu.(\exists \mu', v.(P,\mu) \Downarrow (v,\mu') \Rightarrow (P,\mu) \Downarrow_M (v,\mu'))$$

طبق این تعریف لازم است تا همه اجراهای برنامه‌های امن توسط ناظر پذیرفته شود. با توجه به این‌که NSU، PU، HM و MF شفاف حقیقی نیستند، پس دقیق نیز نیستند. از طرفی، گرچه SME برای TINI دقیق نیست ولی برای TSNI دقیق خواهد بود.

مجموعه‌ای از برنامه‌های TINI به نام P تعریف می‌شود به طوری که یک ناظر همه اجراهای P را می‌پذیرد و به شرح زیر است:

$$\mathcal{P}(M) = \{P \mid TINI(P) \wedge \forall \mu.(\exists \mu', v.(P,\mu) \Downarrow (v,\mu') \Rightarrow (P,\mu) \Downarrow_M (v,\mu'))\}$$

تعریف دقت نسبی – ناظر A از ناظر B دقیق‌تر است، و به شکل $A \supseteq_P B$ نوشته می‌شود، اگر $\mathcal{P}(A) \supseteq \mathcal{P}(B)$.

طبق این تعریف و مشاهده مثال‌های نقض، می‌توان گفت که هیچ‌یک از ناظرهای مورد بررسی در رابطه دقت نسبی نیز نیستند. به همین دلیل، تعریف قوی‌تری از دقت نسبی برای برنامه‌های TSNI مطرح می‌شود.

$$\mathcal{P}^*(M) = \{P \mid TSNI(P) \wedge \forall \mu.(\exists \mu', v.(P,\mu) \Downarrow (v,\mu') \Rightarrow (P,\mu) \Downarrow_M (v,\mu'))\}$$

تعریف دقت TSNI نسبی – ناظر A از ناظر B با توجه به TSNI دقیق‌تر است، و $A \supseteq^*_P B$ نوشته می‌شود، اگر $\mathcal{P}^*(A) \supseteq \mathcal{P}^*(B)$.

قضیه – برای همه برنامه‌های بدون کد مرده[۲۵۰]، داریم

$$HM \supseteq^*_{\mathcal{P}} NSU, HM \supseteq^*_{\mathcal{P}} PU$$

همچنین با توجه به اینکه برای SME اثبات شده است [۴] که برای برنامه‌های TSNI دقیق است، پس SME از هر ناظر دیگری از نظر TSNI دقیق‌تر است.

برای مقایسه ناظرها برحسب تعداد اجراهای ناامنی که می‌پذیرند، تعبیری به نام شفافیت کاذب تعریف می‌شود. باید توجه داشت که شفافیت کاذب، درستی را نقض نمی‌کند.

تعریف شفافیت کاذب – ناظر M شفاف کاذب است اگر برای هر برنامه P، به ازای هر اجرا با شروع از یک حافظه μ و با پایان در حافظه μ' با مقدار v، رابطه زیر برقرار باشد:

$$\neg TINI(P,\mu) \wedge (P,\mu) \Downarrow (v,\mu') \Rightarrow (P,\mu) \Downarrow_M (v,\mu')$$

می‌توان مجموعه‌ای از حافظه‌های اولیه که برنامه P خاتمه یافته و ناظر M برای آن برنامه شفاف کاذب است را چنین تعریف کرد:

$$\mathcal{F}(M,P) = \{\mu \mid \neg TINI(P,\mu_{\mathrm{L}}) \wedge \exists \mu', v.(P,\mu) \Downarrow (v,\mu') \Rightarrow (P,\mu) \Downarrow_M (v,\mu')\}$$

تعریف شفافیت کاذب نسبی – ناظر A از ناظر B دارای شفافیت کاذب بیشتری است، و با $A \sqsupseteq_{\mathcal{F}} B$ نمایش داده می‌شود، اگر برای هر برنامه P، رابطه $\boldsymbol{\mathcal{F}}(A, P) \supseteq \boldsymbol{\mathcal{F}}(B, P)$ برقرار باشد.

قضیه – روابط زیر برقرار است:

$NSU \sqsupseteq_{\mathcal{F}} HM$، $PU \sqsupseteq_{\mathcal{F}} NSU$، $PU \sqsupseteq_{\mathcal{F}} HM$، $SME \sqsupseteq_{\mathcal{F}} HM$، $MF \sqsupseteq_{\mathcal{F}} NSU$ ، $MF \sqsupseteq_{\mathcal{F}} PU$، و $MF \sqsupseteq_{\mathcal{F}} HM$.

۸-۴ مقایسه مکانیزم‌ها

برای بررسی و مقایسه ناظرهای جریان اطلاعات در زمینه درستی کاری به صورت مشخص انجام نشده است. با توجه به این‌که TSNI شکل قوی‌تری از عدم تداخل است که TINI را نیز شامل می‌شود و

[۲۵۰] بخشی از کد منبع برنامه که اجرا می‌شود اما نتیجه آن در هیچ محاسبات دیگری استفاده نمی‌شود.

اکثر ناظرهای موجود، درستی آن‌ها فقط برای TINI اثبات شده است، از نظر درستی می‌توان SME را از دیگران متمایز ساخت. زیرا اثبات شده است که SME برای TSNI درستی دارد.

از جنبه شفافیت، در [۴] دقت SME برای TSNI و در [۵۸] شفافیت حقیقی SME برای TSNI اثبات شده است. به این ترتیب می‌توان گفت که SME یک اعمال مؤثر[۲۵۱] برای TSNI است.

در [۵۴] نشان داده شد که هیچ‌یک از ناظرهای بررسی‌شده برای TINI شفاف حقیقی نیستند. همچنین بیان شد که HM دقیق‌تر از NSU و PU برای TSNI است و NSU و PU شفافیت کاذب بیشتری از HM دارد.

در [۵۶]، [۵۷] اثبات شده است که PU از NSU و MF از PU آسان‌گیرتر است. همچنین در مقاله فعلی بررسی شد که نه تنها MF شفاف حقیقی‌تر از NSU و PU است، بلکه MF به شدت شفافیت کاذب بیشتری نسبت به NSU و PU دارد.

	NSU	PU	HM	SME	MF
NSU		$\not\supseteq_{\mathcal{P}} \not\supseteq_{\mathcal{F}}$	$\not\supseteq_{\mathcal{P}} \supseteq_{\mathcal{F}}$	$\not\supseteq_{\mathcal{P}} \not\supseteq_{\mathcal{F}}$	$\not\supseteq_{\mathcal{P}} \not\supseteq_{\mathcal{F}}$
PU	$\supseteq_{\mathcal{T}}$ $\supseteq_{\mathcal{F}}$		$\not\supseteq_{\mathcal{P}} \supseteq_{\mathcal{F}}$	$\not\supseteq_{\mathcal{P}} \not\supseteq_{\mathcal{F}}$	$\not\supseteq_{\mathcal{P}} \not\supseteq_{\mathcal{F}}$
HM	$\supseteq^*_{\mathcal{P}} \not\supseteq_{\mathcal{F}}$	$\supseteq^*_{\mathcal{P}} \not\supseteq_{\mathcal{F}}$		$\not\supseteq_{\mathcal{P}} \not\supseteq_{\mathcal{F}}$	$\not\supseteq_{\mathcal{P}} \not\supseteq_{\mathcal{F}}$
SME	$\supseteq^*_{\mathcal{P}} \not\supseteq_{\mathcal{F}}$	$\supseteq^*_{\mathcal{P}} \not\supseteq_{\mathcal{F}}$	$\supseteq^*_{\mathcal{P}} \supseteq_{\mathcal{F}}$		$\supseteq^*_{\mathcal{P}} \not\supseteq_{\mathcal{F}}$
MF	$\supseteq_{\mathcal{T}} \supseteq_{\mathcal{F}}$	$\supseteq_{\mathcal{T}} \supseteq_{\mathcal{F}}$	$\not\supseteq_{\mathcal{P}} \supseteq_{\mathcal{F}}$	$\not\supseteq_{\mathcal{P}} \not\supseteq_{\mathcal{F}}$	

$\supseteq_{\mathcal{T}}$ more true TINI transparent than
$\supseteq_{\mathcal{P}}$ more TINI precise than ($\not\supseteq_{\mathcal{P}} \implies \not\supseteq_{\mathcal{T}}$)
$\supseteq^*_{\mathcal{P}}$ more TSNI precise than
$\supseteq_{\mathcal{F}}$ more false TINI transparent than
Monitor is TANI
Monitor is TSNI, hence TANI

شکل ۱۹ – مقایسه مکانیزم‌های اعمال پویای امنیت جریان اطلاعات [۵۴]

به این ترتیب، مفهوم جدید درستی: عدم تداخل آگاه به خاتمه (TANI) معرفی شد و بیان گردید که HM، SME و MF خطمشی امنیتی TANI را برآورده می‌کنند، اما NSU و PU قادر به انجام این کار نیستند.

علاوه بر این، طبق تعبیر شفافیت حقیقی برای هیچ‌کدام از ناظرهای بررسی‌شده برقرار نبودند. به همین خاطر، تعبیر ضعیف‌تر شفافیت حقیقی نسبی ارائه شد که به معنای این است که کدام ناظر به

[251] Effective Enforcement

شفاف‌بودن نزدیک‌تر است. حتی تعبیر ضعیف‌تری به نام دقت ارائه شد که رفتار ناظرها را برای برنامه‌های امن مقایسه می‌کند. و نشان داده شد که HM از NSU و PU از نظر TSNI دقیق‌تر است؛ در حالی‌که پیشتر مقایسه این ناظرها ممکن نبود. همچنین، نشان داده شد که تعبیر مرسوم آسان‌گیربودن ترکیبی از شفافیت حقیقی و کاذب نسبی است و با این معیار می‌توان همه ناظرها را با یکدیگر مقایسه کرد.

فصل نهم

جمع‌بندی، مسائل باز و پروژه کارشناسی ارشد

جمع‌بندی، مسائل باز و پروژه کارشناسی ارشد

در این گزارش به تعریف خط‌مشی‌های امنیتی جریان اطلاعات، و عدم تداخل به عنوان معروف‌ترین آن‌ها پرداخته شد. با توجه به این‌که خط‌مشی‌های امنیتی را می‌توان به دو دسته خاصیت و فوق‌خاصیت، و یا به تعبیری ناخاصیت، تقسیم‌بندی کرد، خط‌مشی‌های جریان اطلاعات از آن جهت که معمولاً با گزاره‌ای روی بیش از یک اجرا قابل بیان هستند، ناخاصیت به شمار می‌روند. همین امر باعث ایجاد محدویت در نحوه اعمال آن‌ها توسط مکانیزم‌های امنیتی می‌شود به طوری‌که اغلب، به جای اعمال خط‌مشی، تقریبی از آن توسط مکانیزم اعمال می‌شود.

از این رو، بررسی مکانیزم‌های مختلف برای اعمال انواع خط‌مشی‌ها حائز اهمیت است. انواع مکانیزم‌های امنیتی را می‌توان به دو دسته کلی ایستا و پویا دسته‌بندی کرد که ویژگی اصلی مکانیزم‌های ایستا، تحلیل ایستای کد منبع برنامه قبل از اجرا و محافظه‌کارانه‌بودن آن‌ها است. از طرف دیگر، در مکانیزم‌های پویا روش‌های نظارت بر اجرا دسته اصلی آن‌ها را شامل می‌شود. این مکانیزم‌ها با توجه به میزان اطلاعاتی که از برنامه و اجراهای آن به دست می‌آورد، توانایی‌های متفاوتی برای اعمال خط‌مشی‌ها دارند. به طور کلی، با توجه به در اختیار داشتن اطلاعات زمان اجرا و تحلیل کد منبع برنامه در مکانیزم‌های پویا، می‌توان قدرت بیشتری نسبت به مکانیزم‌های ایستا قائل شد؛ گرچه سربار اضافی این روش‌ها باعث کاهش کارایی آن‌ها می‌شود.

در این گزارش به بررسی مطالعات مختلف انجام‌شده در حوزه اعمال پویای خط‌مشی‌های امنیتی پرداخته شد. همچنین، این‌که پاسخ به این پرسش که چه خط‌مشی‌هایی توسط ناظرهای اجرایی قابل اعمال هستند، مورد بحث قرار گرفت. سه مبحث کلی شامل تأثیر میزان دسترسی ناظر به اطلاعات دیگر مانند اطلاعات ایستا و امکانات ناظر در مواجهه با نقض خط‌مشی، تأثیر محدودیت‌هایی مانند محدودیت‌های حافظه‌ای و محاسباتی در عملکرد ناظرها و بررسی اثر تعریف و تعبیر اعمال در گستره خط‌مشی‌های قابل اعمال توسط این مکانیزم بررسی شد. نتایج این تحلیل‌ها مهر تأییدی بر این تصور بود که ناظرها می‌توانند مجموعه بزرگی از خط‌مشی‌های امنیتی را اعمال کنند.

با این حال، کماکان سوالات پژوهشی متفاوتی در این حوزه قابل طرح است. ارتباط بین سطح انتزاع و خط‌مشی‌های قابل اعمال اولین آن‌ها است. این‌که آیا ناظرها در عمل می‌توانند با اطلاعات و

امکانات در دسترس چگونه رفتار کنند و چه مجموعه‌ای از خط‌مشی‌ها را اعمال کنند. دیگر آن‌که نحوه انتزاع و بیان دیگری غیر از بررسی دنباله‌های اجرا وجود دارد یا خیر. به طور کلی، باید به دنبال موازنه مناسبی از انتزاع و اعمال بود.

همان‌طور که بارها نیز در این گزارش عنوان شد، تحلیل ایستا یکی از ارکان مهم و تأثیرگذار در گسترش بازه خط‌مشی‌های قابل اعمال توسط ناظرها به شمار می‌رود. این‌که حداقل اطلاعات ایستای لازم برای اعمال یک خط‌مشی چه مقدار است، از دیگر سوالات پژوهشی این حوزه خواهد بود. گرچه اولین گام‌های آن در مقاله [۴۶] صورت گرفته است.

خودکاره ویرایش که به عنوان قوی‌ترین مدل ناظرها، با توقیف نامحدود کنش‌ها و بازدرج آن‌ها مطرح شد. اما همواره این امکان وجود ندارد که ناظر کنشی را توقیف کند و با بررسی ادامه دنباله اجرا، از امن‌بودن آن اطمینان حاصل کند. این موضوع باعث کاهش توانایی خودکاره ویرایش می‌شود اما این‌که دقیقاً چه میزان روی مجموعه خط‌مشی‌های قابل اعمال تأثیرگذار است، مشخص نیست. زیرا ممکن است کنش‌هایی باشند که از نظر امنیتی بسیار حائز اهمیت باشند و ایجاد چنین تغییراتی در دنباله اجرا، خود نقض امنیتی محسوب شود. بررسی قدرت اعمال خودکاره‌های ویرایش با این فرض که بعضی از کنش‌ها قابل توقیف یا درج نباشند، یکی دیگر از کارهای آینده این حوزه محسوب می‌شود.

یکی از فرض‌های مشترک بین مکانیزم‌های مطرح‌شده، ثابت‌بودن خط‌مشی‌های امنیتی است. تغییر خط‌مشی پس از تعیین آن برای یک مکانیزم زمان‌اجرا بسیار سخت یا در بعضی موارد نشدنی است. به این معنا که ممکن است مکانیزم نظارت نیاز به بازپیکربندی داشته باشد. از این رو پژوهش بیشتری برای انعطاف‌پذیری و مقیاس‌پذیری مکانیزم‌های اعمال باید صورت گیرد. توجه به همروندی و برنامه‌های چندریسه‌ای از دیگر محورهای پژوهش‌های آینده این حوزه به شمار می‌رود. زیرا یکی از نقاط ضعف خودکاره‌های امنیتی، عدم توانایی مواجهه با این‌گونه برنامه‌ها محسوب می‌شود. یکنواخت‌سازی نحوه ارزیابی سربار تکنیک‌های اعمال نیز حائز اهمیت است. تاکنون معیارهای مشترکی بین همه ابزارهای اعمال ارائه نشده است که بتوان با دقت کافی این ابزارها را با یکدیگر مقایسه کرد.

دو معیار درستی و کامل‌بودن برای مقایسه انواع روش‌های اعمال به کار می‌رود. به دلیل ناکامل‌بودن مکانیزم‌های ارائه‌شده فعلی، پژوهش‌ها برای دسته‌بندی انواع خط‌مشی‌ها و مکانیزم‌ها در شرایط و فرضیات مختلف کماکان ادامه دارد.

پروژه کارشناسی ارشد تعریف‌شده در این حوزه با عنوان « بهبود مکانیزم‌های مبتنی بر چنداجرایی برای اِعمال خط‌مشی‌های جریان اطلاعات» مطرح شده است که شرح آن در ادامه آمده است.

خط‌مشی امنیتی، تعریفی از امن‌بودن یک سامانه یا برنامه را ارائه می‌دهد که رفتارهای مجاز و غیرمجاز، در آن مشخص می‌شود. خط‌مشی‌های جریان اطلاعات، خط‌مشی‌های محرمانگی و صحت هستند که انتشار داده‌ها را در برنامه کنترل می‌کنند. یکی از خط‌مشی‌های محرمانگیِ مهم برای امنیت جریان اطلاعات، عدم تداخل است. 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.

خط‌مشی‌های امنیتی را می‌توان به دو دسته خاصیت و فوق خاصیت تقسیم‌بندی کرد. همان‌طور که می‌دانیم، یک سامانه، شامل مجموعه‌ای از اجراها است. یک خط‌مشی امنیتی را خاصیت می‌نامند اگر بتوان آن را با مجموعه‌ای از اجراهای دارای رفتار مجاز بیان کرد. برای نمونه، می‌توان به خط‌مشی‌های کنترل دسترسی اشاره کرد. خط‌مشی‌هایی مانند عدم تداخل، خاصیت نیستند؛ یعنی آن‌ها را باید با مجموعه توانیِ مجموعه اجراها بیان کرد. به این‌گونه خط‌مشی‌ها، فوق خاصیت گفته می‌شود [۴۷]. بنابراین، روش اِعمال خاصیت‌ها با نحوه اعمال فوق خاصیت‌ها متفاوت است.

اِعمال خط‌مشی‌های جریان اطلاعات، یک مسئله چالش‌برانگیز است. مکانیزم‌های اِعمال، به دنبال دست‌یابی به این اهداف هستند [۴]: ۱) درستی[۲۵۲]: اجازه وقوع جریان غیرمجاز اطلاعات در طول اجرا داده نشود. ۲) دقت[۲۵۳]: از اجرای امن برنامه‌ها جلوگیری نشود. ۳) عملی‌بودن[۲۵۴]: هزینه اعمال مکانیزم قابل قبول باشد. هزینه‌ها ممکن است در زمان توسعه، استقرار[۲۵۵] یا اجرای برنامه باشد. گرچه تلاش‌های بسیاری در دهه‌های اخیر برای پاسخ به این مسئله شده است، اما کماکان مکانیزم‌های اعمالی که به طور همزمان به همه این اهداف دست یابند، مطرح نشده است.

---

[252] Soundness

[253] Precision

[254] Practicality

[255] Deployment

دو دسته کلی برای مکانیزم‌های اعمال خط‌مشی‌های جریان اطلاعات وجود دارد. از جمله رویکردهای ایستا می‌توان به مکانیزم‌های مبتنی بر نوع[۲۵۶] [۱۵] و مکانیزم‌های مبتنی بر راستی‌آزمایی[۲۵۷] [۱۶] اشاره کرد. این‌گونه مکانیزم‌ها، دارای درستی هستند و هزینه‌ای در زمان اجرا یا استقرار تحمیل نمی‌کنند. با این حال، مکانیزم‌های مبتنی بر نوع دقیق نیستند و ممکن است برنامه‌های امن زیادی توسط آن‌ها پذیرفته نشوند و محافظه‌کار هستند. اما مکانیزم‌های مبتنی بر راستی‌آزمایی، برحسب کامل‌بودن[۲۵۸] منطق برنامه، ممکن است از دقت کامل برخوردار باشند[۴]. همچنین، هم مکانیزم‌های مبتنی بر نوع و هم مکانیزم‌های مبتنی بر راستی‌آزمایی، هزینه زمان توسعه زیادی دارند.

رویکردهای پویا، که در سال‌های اخیر توجه بیشتری به آن‌ها شده است، شامل ناظرهای زمان‌اجرا[۲۵۹] [۵۷]، [۶] و تکنیک اجرای چندباره امن (SME) [۱۸] می‌شود. مکانیزم‌های ذکرشده درستی دارند و می‌توانند نسبت به بعضی از مکانیزم‌های ایستا، برای خط‌مشی‌های بیشتری دقت را فراهم کنند. به عنوان نمونه، ناظرهای زمان‌اجرا، برنامه‌های کمتری نسبت به مکانیزم‌های مبتنی بر نوع را رد می‌کنند. در نظارت زمان‌اجرا، برخلاف مکانیزم‌های ایستا، حالت‌های برنامه‌ی در حال اجرا توسط ناظر بررسی شده و در صورت امکان ورود به حالت ناامن ادامه اجرا متوقف خواهد شد یا با اعمال تغییراتی به اجرای امن تبدیل می‌شود. البته اثبات شده است [۵۸] که عدم تداخل، با توجه به این‌که یک خاصیت ایمنی[۲۶۰] نیست، توسط ناظرهای اجرا قابل اعمال نیست.

مفهوم اصلی تکنیک چنداجرایی امن [۱۸] آن است که به ازای هر سطح امنیتی، یک اجرا از برنامه انجام شود. به این ترتیب که ورودی‌ها، در اجراهای مربوط به سطح امنیتی خود یا بالاتر، مقدار می‌گیرند و در غیر این صورت، با مقادیر پیش‌فرض جایگزین می‌شوند. خروجی‌ها نیز فقط در اجرای مربوط به سطح امنیتی خود تولید می‌شوند. ضمناً با توجه به این‌که اجراهای سطح بالا از ورودی‌های سطح پایین هم استفاده می‌کنند، اثرات جانبی ورودی‌ها نیز در نظر گرفته می‌شوند. با توجه به تفکیک یک اجرای

[256] Type-based

[257] Verification-based

[258] Completeness

[259] Execution Monitor

[260] Safety Property

برنامه به چندین اجرا به ازای هر سطح امنیتی، استراتژی زمان‌بندی[۲۶۱] این اجراها از نکات مهم این روش به شمار می‌رود. نشان داده می‌شود که این روش درستی را تضمین می‌کند.

چنداجرایی امن هزینه توسعه ندارد، اما چنداجرایی امن را نمی‌توان به سادگی اعمال کرد زیرا همه پیاده‌سازی‌های چنداجرایی امن، نیازمند ایجاد اصلاح‌هایی در زیرساخت محاسباتی مانند سیستم عامل، مرورگر وب و یا ماشین مجازی است [۴]. نکته دیگر آن که در تکنیک چنداجرایی، امکان تشخیص تغییر معناشناخت[۲۶۲] برنامه وجود ندارد و هیچ تضمینی برای ترتیب نسبی خروجی‌های سطوح مختلف امنیتی نمی‌دهد[۵۸].

در این پژوهش، با بررسی مزایا و معایب مکانیزم‌های موجودِ اعمال خط‌مشی‌های جریان اطلاعات، می‌خواهیم دقت مکانیزم‌های مبتنی بر چنداجرایی را افزایش دهیم. به دنبال آن هستیم که برای یک زبان برنامه‌نویسی مدل، با به کارگیری تکنیک چنداجرایی و استفاده از روش‌های دیگر مانند خودترکیبی [۱۶] و نظارت زمان اجرا، مکانیزم بهتری در این خصوص ارائه کنیم. با انجام این پژوهش، از حیث دقت مکانیزم، اعمال مبتنی بر تکنیک چنداجرایی برای فوق خاصیت محرمانگی جریان اطلاعات را بهبود خواهیم داد و با اثبات صوری درستی و دقت مکانیزم بهبودیافته، به ارزیابی و مقایسه آن با سایر مکانیزم‌ها تحت خط‌مشی‌های مختلف می‌پردازیم. به این ترتیب، توسعه‌دهندگان برنامه‌های کاربردی، هزینه کم‌تری برای تضمین امنیت نرم‌افزار متحمل خواهند شد.

برای پاسخ‌گویی به سوال پژوهش، ابتدا باید با بررسی خط‌مشی‌های مطرح‌شده در سابقه علمی این حوزه، خط‌مشی جریان اطلاعات مورد نظر را تعیین کرد. انتخاب این خط‌مشی، بستگی به سطح انتزاع و موازنه[۲۶۳] کاربردی‌بودن-مدل‌بودن خط‌مشی دارد. ضمناً باید محدودیت‌های تکنیک چنداجرایی برای اعمال خط‌مشی‌ها را نیز در نظر داشت. در ادامه، متناسب با خط‌مشی انتخاب‌شده، یک زبان برنامه‌نویسی مدل برای بیان صوری آن خط‌مشی معرفی می‌شود. بیان صوری نحو[۲۶۴] و معناشناخت آن زبان، گام بعدی است. پس از آن، با بررسی مکانیزم‌های پویا، مقایسه مکانیزم‌های اعمال مختلف و

[261] Scheduling Strategy

[262] Semantics

[263] Trade-off

[264] Syntax

دسته‌بندی نقاط قوت و ضعف هر یک، با توجه به عناصر زبان برنامه‌نویسی مطرح‌شده، مکانیزمی مبتنی بر تکنیک چنداجرایی، برای اعمال فوق خاصیت امنیتی جریان اطلاعات ارائه خواهد شد که در برابر مکانیزم‌های موجود، از منظر دقت، برتری داشته باشد. همان‌طور که قبل‌تر اشاره شد، اصلی‌ترین معیارهای مقایسه مکانیزم‌های اعمال، همان درستی و دقت روش ارائه‌شده خواهد بود. به همین منظور، برای راستی‌آزمایی مکانیزم ارائه‌شده، از رویکرد اثبات درستی و دقت استفاده خواهد شد.

منابع و مراجع

[1] J. McLean, “A general theory of composition for trace sets closed under selective interleaving functions,” *Res. Secur. Privacy, 1994. Proceedings., 1994 IEEE Comput. Soc. Symp.*, pp. 79–93, 1994.

[2] F. B. Schneider, “Enforceable security policies,” *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 1, pp. 30–50, 2000.

[3] M. Bishop, *Computer Security: Art and Science*, 2nd ed. Addison-Wesley, 2003.

[4] G. Barthe, J. M. Crespo, D. Devriese, F. Piessens, and E. Rivas, “Secure multi-execution through static program transformation,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2012, vol. 7273 LNCS, pp. 186–202.

[5] E. Cohen, “Information Transmission in Computational Systems,” *Proc. Sixth ACM Symp. Oper. Syst. Princ.*, no. November, pp. 133–139, 1977.

[6] G. Le Guernic, “Confidentiality enforcement using dynamic information flow analyses,” PhD Thesis, Kansas State University, 2007.

[7] J. A. Goguen and J. Meseguer, “Security Policies and Security Models,” *Secur. Privacy, IEEE Symp.*, vol. 0, p. 11+, 1982.

[8] J. McLean, “Security models and information flow,” *Res. Secur. Privacy, 1990. Proceedings., 1990 IEEE Comput. Soc. Symp.*, pp. 180–187, 1990.

[9] D. Sutherland, “A model of information,” in *Proc. 9th National Computer Security Conference*, 1986, pp. 175–183.

[10] A. Sabelfeld and A. C. Myers, “Language-based information-flow security,” *IEEE J. Sel. Areas Commun.*, vol. 21, no. 1, pp. 5–19, 2003.

[11] D. E. Denning, “A lattice model of secure information flow,” *Commun. ACM*, vol. 19, no. 5, pp. 236–243, 1976.

[12] 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: Softwa, no. 10.3233/978-1-61499-028-4-319, 2012, pp. 319–347.

[13] S. Hunt and D. Sands, “On flow-sensitive security types,” *ACM SIGPLAN Not.*, vol. 41, no. 1, pp. 79–90, 2006.

[14] K. W. Hamlen, F. B. Schneider, K. W. Hamlen, F. B. Schneider, and G. Morrisett, “Computability Classes for Enforcement Mechanisms,” *ACM Trans. Program. Lang. Syst.*, vol. 28, no. 1, pp. 175–205, 2006.

[15] D. Volpano, C. Irvine, and G. Smith, “A sound type system for secure flow analysis,” *J. Comput. Secur.*, vol. 4, no. 2/3, p. 167, 1996.

[16] G. Barthe, P. R. D’Argenio, and T. Rezk, “Secure information flow by self-composition,” *Proceedings. 17th IEEE Comput. Secur. Found. Work. 2004.*, pp. 1–52, 2004.

[17] J. Ligatti and S. Reddy, “A theory of runtime enforcement, with results,” in *European Symposium on Research in Computer Security*, 2010, pp. 87–100.

[18] D. Devriese and F. Piessens, “Noninterference through secure multi-execution,” in *Proceedings* -

IEEE Symposium on Security and Privacy, 2010, pp. 109–124.

[19] J. Ligatti, L. Bauer, and D. Walker, "Enforcing non-safety security policies with program monitors," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2005, vol. 3679 LNCS, pp. 355–373.

[20] C. E. Irvine, "The reference monitor concept as a unifying principle in computer security education," Technical Report, NAVAL Postgraduate Sschool Monterey CA Dept of Computer Science, 1999.

[21] Ú. Erlingsson, "The inlined reference monitor approach to security policy enforcement," phdthesis, Cornell University, 2004.

[22] M. Viswanathan, "Foundations for the Run-time Analysis of Software systems," PhD Thesis, University of Pennsylvania, 2000.

[23] G. Gheorghe and B. Crispo, "A survey of runtime policy enforcement techniques and implementations," University of Trento, Technical Report # DISI-11-477, 2011.

[24] A. Sabelfeld and A. Russo, "From dynamic to static and back: riding the roller coaster of Information-flow control research," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2010, vol. 5947 LNCS, pp. 352–365.

[25] L. Zheng and A. C. Myers, "Dynamic security labels and static information flow control," in *International Journal of Information Security*, 2007, vol. 6, no. 2–3, pp. 67–84.

[26] N. Broberg and D. Sands, "Flow-sensitive semantics for dynamic information flow policies," *Proc. ACM SIGPLAN Fourth Work. Program. Lang. Anal. Secur. - PLAS '09*, p. 101, 2009.

[27] P. Shroff, S. F. Smith, and M. Thober, "Dynamic dependency monitoring to secure information flow," in *Proceedings - IEEE Computer Security Foundations Symposium*, 2007, pp. 203–217.

[28] D. E. Denning and P. J. Denning, "Certification of programs for secure information flow," *Commun. ACM*, vol. 20, pp. 504–513, 1977.

[29] G. Smith and D. Volpano, "Secure information flow in a multi-threaded imperative language," in *Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 1998, pp. 355–364.

[30] D. Volpano and G. Smith, "Probabilistic noninterference in a concurrent language," in *Proceedings of the Computer Security Foundations Workshop*, 1998, pp. 34–43.

[31] a. Sabelfeld and D. Sands, "Probabilistic noninterference for multi-threaded programs," *Proc. 13th IEEE Comput. Secur. Found. Work. CSFW-13*, pp. 200–214, 2000.

[32] S. A. Zdancewic, "Programming languages for information security," phdthesis, Cornell University, 2002.

[33] J. Ligatti, L. Bauer, and D. Walker, "Run-Time Enforcement of Nonsafety Policies," *ACM Trans. Inf. Syst. Secur.*, vol. 12, no. 3, pp. 1–41, 2009.

[34] L. Bauer, J. Ligatti, and D. Walker, "More enforceable security policies," in *Proceedings of the Workshop on Foundations of Computer Security (FCS02), Copenhagen, Denmark*, 2002.

[35] J. Ligatti, L. Bauer, and D. Walker, "Edit automata: Enforcement mechanisms for run-time security policies," *Int. J. Inf. Secur.*, vol. 4, no. 1–2, pp. 2–16, 2005.

[36] C. Talhi, N. Tawbi, and M. Debbabi, "Execution monitoring enforcement under memory-limitation constraints," *Inf. Comput.*, vol. 206, no. 2–4, pp. 158–184, 2008.

[37] R. Khoury and N. Tawbi, “Which security policies are enforceable by runtime monitors? A survey,” *Computer Science Review*, vol. 6, no. 1. pp. 27–45, 2012.

[38] M. Kim, S. Kannan, I. Lee, O. Sokolsky, and M. Viswanathan, “Computational analysis of run-time monitoring: Fundamentals of java-MaC,” in *Electronic Notes in Theoretical Computer Science*, 2002, vol. 70, no. 4, pp. 85–99.

[39] A. Lamei, “Formal Characterization of Security Policy Enforcement through Program Rewriting,” PhD Thesis, Amirkabir University of Technology, 2016.

[40] P. W. L. Fong, “Access control by tracking shallow execution history,” in *Proceedings - IEEE Symposium on Security and Privacy*, 2004, vol. 2004, pp. 43–55.

[41] D. Beauquier, J. Cohen, and R. Lanotte, “Security policies enforcement using finite and pushdown edit automata,” *Int. J. Inf. Secur.*, vol. 12, no. 4, pp. 319–336, 2013.

[42] N. Bielova and F. Massacci, “Do you really mean what you actually enforced?,” *Int. J. Inf. Secur.*, vol. 10, no. 4, pp. 239–254, 2011.

[43] R. Khoury and N. Tawbi, “Corrective enforcement of security policies,” in *International Workshop on Formal Aspects in Security and Trust*, 2010, pp. 176–190.

[44] R. Khoury and N. Tawbi, “Using equivalence relations for corrective enforcement of security policies,” in *International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security*, 2010, pp. 139–154.

[45] H. Chabot, R. Khoury, and N. Tawbi, “Extending the enforcement power of truncation monitors using static analysis,” *Comput. Secur.*, vol. 30, no. 4, pp. 194–207, 2011.

[46] F. Imanimehr and M. S. Fallah, “How Powerful Are Run-Time Monitors with Static Information?,” *The Computer Journal*, 2016.

[47] M. R. Clarkson and F. B. Schneider, “Hyperproperties,” *J. Comput. Secur.*, vol. 18, no. 6, pp. 1157–1210, 2010.

[48] R. Khoury and N. Tawbi, “Corrective enforcement: a new paradigm of security policy enforcement by monitors,” *ACM Trans. Inf. Syst. Secur.*, vol. 15, no. 2, p. 10, 2012.

[49] F. B. Schneider, M. Ngo, F. Massacci, D. Milushev, and F. Piessens, “Runtime Enforcement of Security Policies on Black Box Reactive Programs,” *ACM SIGPLAN-SIGACT Symp. Princ. Program. Lang.*, vol. 3, no. 1, pp. 43–54, 2015.

[50] S. Zdancewic and A. C. Myers, “Observational determinism for concurrent program security,” in *Computer Security Foundations Workshop, 2003. Proceedings. 16th IEEE*, 2003, pp. 29–43.

[51] F. B. Schneider, G. Morrisett, and R. Harper, “A language-based approach to security,” in *Informatics*, 2001, pp. 86–101.

[52] G. Le Guernic, A. Banerjee, T. Jensen, and D. A. Schmidt, “Automata-based confidentiality monitoring,” in *Annual Asian Computing Science Conference*, 2006, pp. 75–89.

[53] A. Askarov, S. Hunt, A. Sabelfeld, and D. Sands, “Termination-insensitive noninterference leaks more than just a bit,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2008, vol. 5283 LNCS, pp. 333–348.

[54] N. Bielova and T. Rezk, “A taxonomy of information flow monitors,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2016, vol. 9635, pp. 46–67.

[55] A. Russo and A. Sabelfeld, “Dynamic vs. static flow-sensitive security analysis,” in *Proceedings - IEEE Computer Security Foundations Symposium*, 2010, pp. 186–199.

[56] T. H. Austin and C. Flanagan, “Multiple facets for dynamic information flow,” *Proc. 39th Annu. ACM SIGPLAN-SIGACT Symp. Princ. Program. Lang. - POPL ’12*, vol. 47, no. 1, p. 165, 2012.

[57] T. H. Austin and C. Flanagan, “Permissive dynamic information flow analysis,” in *Proceedings of the 5th ACM SIGPLAN Workshop on Programming Languages and Analysis for Security - PLAS ’10*, 2010, p. 3.

[58] D. Zanarini, M. Jaskelioff, and A. Russo, “Precise enforcement of confidentiality for reactive systems,” in *Proceedings of the Computer Security Foundations Workshop*, 2013, pp. 18–32.

Amirkabir University of Technology
(Tehran Polytechnic)

Computer and Information Technology Engineering Department

Seminar Report

Title

Dynamic Enforcement of Security Hyperproperties

By
Seyed Mohammad Mehdi Ahmadpanah

Supervisor
Dr. Mehran S. Fallah

Course Instructor
Dr. Babak Sadeghiyan

September 2016