علوم رایانش و فناوری اطلاعات
نشریه علمی انجمن کامپیوتر ایران
مجلد ۱۷، شماره ۲، ۱۳۹۸
صفحات ۶۱-۵۲
**مقاله عادی**

# قرنطینه‌سازی تروجان‌های سخت‌افزاری در پردازنده‌های عام‌منظوره با روش‌های نرم-افزاری مبتنی بر مترجم

فرزانه قطب‌الدینی[۱*] و علی جهانیان[۲]

*نویسنده مسئول، دریافت: ۹۹/۰۱/۰۴، بازنگری: ۹۹/۰۲/۱۴، پذیرش: ۹۹/۰۳/۰۲

[۱] دانشجوی کارشناسی ارشد دانشکده مهندسی و علوم کامپیوتر، دانشگاه شهید بهشتی، تهران، ایران
[۲] دانشیار دانشکده مهندسی و علوم کامپیوتر، دانشگاه شهید بهشتی، تهران، ایران

## چکیده

با جهانی شدن فرآیند طراحی و ساخت نیمه هادی، مدارهای مجتمع به‌طور فزاینده‌ای به فعالیت‌های مخرب و تغییرات آسیب‌پذیر می‌شوند. در بسیاری از سامانه‌های صنعتی، شواهدی در مورد ناامنی بخش‌هایی از سیستم مشاهده شده است، اما معمولاً تغییر اجزای سیستم و جایگزینی قطعات نامطمئن چندان ساده نیست. در بسیاری از شرایط قطعه جایگزین مطمئن برای اجزای مشکوک وجود ندارد و یا تغییر معماری سیستم و دست‌کاری آن با ریسک بالایی همراه است. در این شرایط ممکن است بتوان با قرنطینه‌سازی تروجان به کمک تغییر نرم‌افزار سیستم، تروجان را غیرفعال کرد و عملاً لایه امن نرم‌افزاری روی بستر سخت‌افزاری ناامن ایجاد نمود.

در این مقاله قرنطینه کردن یک تروجان در بانک ثبات - که از رایج‌ترین نوع تروجان در پردازنده‌ها می‌باشد - با روش‌های نرم‌افزاری مبتنی بر مترجم به‌منظور اجرای امن یک برنامه بر روی بستر سخت‌افزاری ناامن ارزیابی می‌شود. ایده ارائه‌شده روی پردازنده عام‌منظوره پیاده‌سازی شده و مورد ارزیابی قرارگرفته است. نتایج شبیه‌سازی نشان می‌دهد که روش‌های قرنطینه‌سازی پیشنهادی می‌تواند به‌طور مؤثر با سربار زمان اجرای مناسب به‌منظور بالا بردن امنیت پردازنده مورد استفاده قرار گیرد.

**کلمات کلیدی:** پردازنده، قرنطینه‌کردن تروجان سخت‌افزاری، مترجم.

## ۱- مقدمه

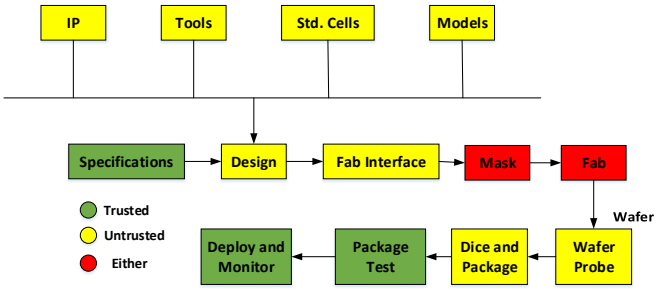
ریزپردازنده هسته اصلی محاسبات در یک سامانه دیجیتال است که عملکرد آن در امنیت و یکپارچگی کل سیستم شامل بخش‌های مختلف سخت‌افزاری و نرم‌افزاری آن تأثیر بسزایی دارد [۱]. با توجه به غیرمتمرکز بودن فرآیند طراحی و ساخت محصولات میکروالکترونیک، نگرانی‌هایی در مورد ویژگی‌های سخت‌افزاری مخرب عمدی و اسب‌های تروجان[۱] وجود دارد [۲].

اخیراً در چرخه‌ی تولید مدارهای مجتمع، بخش عمده‌ی تراشه‌ها توسط کارخانه‌های ساخت طرف سوم[۲] تولید می‌شود. علاوه بر این، در روند طراحی تراشه، مالکیت معنوی طرف سوم[۳] نیز استفاده می‌گردد. شکل ۱، اعتبار هر مرحله در چرخه تولید تراشه را نشان می‌دهد. هر یک از بخش‌های درگیر در طراحی و ساخت مدار مجتمع[۴]، یک فرصت برای حضور یک مهاجم است تا تروجان‌های سخت‌افزاری را درج کند [۳]. با توجه به شکل ۱ به‌طورکلی تولید مدار مجتمع شامل سه دسته مراحل طراحی، ساخت و اعتبارسنجی می‌شود. مرحله طراحی با مشخصات آغاز می‌شود که شامل قراردادها و محدودیت‌های طراحی است. طراحی مدار مجتمع اغلب شامل مالکیت معنوی طرف سوم است. سرویس‌های برون‌سپاری طراحی و آزمایش مانند ابزارهای طراحی خودکار الکترونیکی[۵] توسط فروشنده‌های مختلف ارائه می‌شود. بخش‌هایی که با رنگ سبز در شکل ۱ مشخص شده‌اند، مراحل امن فرض شده‌اند. بخش‌های زرد رنگ نواحی ناامن هستند و نواحی قرمز رنگ شامل بخش‌هایی هستند که در شرایط خاص می‌توانند ناامن فرض شوند.

در بسیاری از سامانه‌های واقعی شواهدی برای وجود ناامنی‌های سخت‌افزاری در سیستم مشاهده شده است، اما تغییر سخت‌افزار و جایگزینی قطعات مشکوک بسیار مشکل است. دلایلی که باعث می‌شود جایگزینی قطعات مشکوک ساده نباشد، عبارت‌اند از:

- در بسیاری از شرایط اصولاً قطعه جایگزین وجود ندارد، به‌عنوان‌مثال در سامانه‌های صنعتی علیرغم وجود گزارش‌های متعدد در مورد امنیت کنترل‌کننده‌های قابل‌برنامه‌ریزی منطقی[۶] صنعتی برخی برندها، قطعه مشابه یا سازگار با کنترل‌کننده قابل‌برنامه‌ریزی منطقی این برندها وجود ندارد که قابل جایگزینی باشد.
- اصولاً دست‌کاری یک سیستم در حال کار به هر دلیل (ازجمله نگرانی‌های امنیتی) از دید مهندسان تصمیمی با ریسک بالا به‌حساب می‌آید. پیچیدگی سامانه‌های پردازشی و تفاوت‌های کوچک بین محصولات مشابه ممکن است باعث مشکلات جدی در سامانه‌های حساس شود.
- هر نوع دست‌کاری در یک سیستم دیجیتال نیاز به طی شدن بخش بزرگی از پروسه تست و تصدیق سیستم دارد که هم هزینه و هم ریسک بالایی دارد.

در این شرایط، یک راه‌کار میانی این است که با قرنطینه‌سازی نواحی یا کارکردهای مشکوک در قطعه نامطمئن، تروجان را قرنطینه کرد و عملاً یک لایه ایمن روی بستر سخت‌افزاری ناامن ایجاد کرد.




شکل ۱- مراحل آسیب‌پذیر از چرخه تولید یک مدار مجتمع [۳]

تأکید این نکته لازم است که هدف این مقاله یافتن تروجان در طرح نیست، بلکه روش ارائه‌شده در این مقاله در شرایطی کاربرد دارد که شواهدی برای وجود نواحی مشکوک یا تروجان‌های مخرب داده در سیستم پردازشی مشاهده شده است و طراح سیستم به دنبال روش‌هایی برای تغییر نرم‌افزار سیستم، با هدف قرنطینه‌سازی سخت‌افزارهای مشکوک می‌باشد، به‌طوری‌که عملکرد عادی برنامه حفظ شود و سربار زمان اجرا هم قابل‌قبول باشد. با توجه به اینکه گزارش‌های متعددی در مورد درج تروجان در سیستم ثبات پردازنده‌ها گزارش شده است [۴-۶] ، ما این نوع از تروجان را به‌عنوان مورد مطالعاتی جهت ارزیابی ایده خود استفاده کرده‌ایم. البته ایده ارائه‌شده برای مقابله با حملات تروجان‌های سخت‌افزاری در حافظه و واحد محاسبه و منطق[۷] و رمزگشا هم قابل اعمال است. در واقع ما فرض کرده‌ایم که شواهدی در مورد اشکالات مشکوک در ثبات‌های پردازنده وجود دارد و هدف قرنطینه‌سازی این ثبات‌ها می‌باشد. در این مقاله روش‌هایی مبتنی بر مترجم برای این نوع قرنطینه‌سازی با کمترین سربار ارائه‌شده است.

ساختار این مقاله به صورت زیر است. فصل ۲ به معرفی تروجان‌های شناخته شده موجود در پردازنده‌ها می‌پردازد. در فصل ۳ روش‌های قرنطینه‌سازی تروجان شناخته شده، شرح داده می‌شود. در فصل ۴ به تحلیل نتایج به‌دست آمده و مقایسه سربار زمان اجرا پرداخته می‌شود و در نهایت در فصل ۵ نتیجه‌گیری از مقاله ارائه می‌گردد.

## ۲- تروجان‌ها در پردازنده‌های عام منظوره

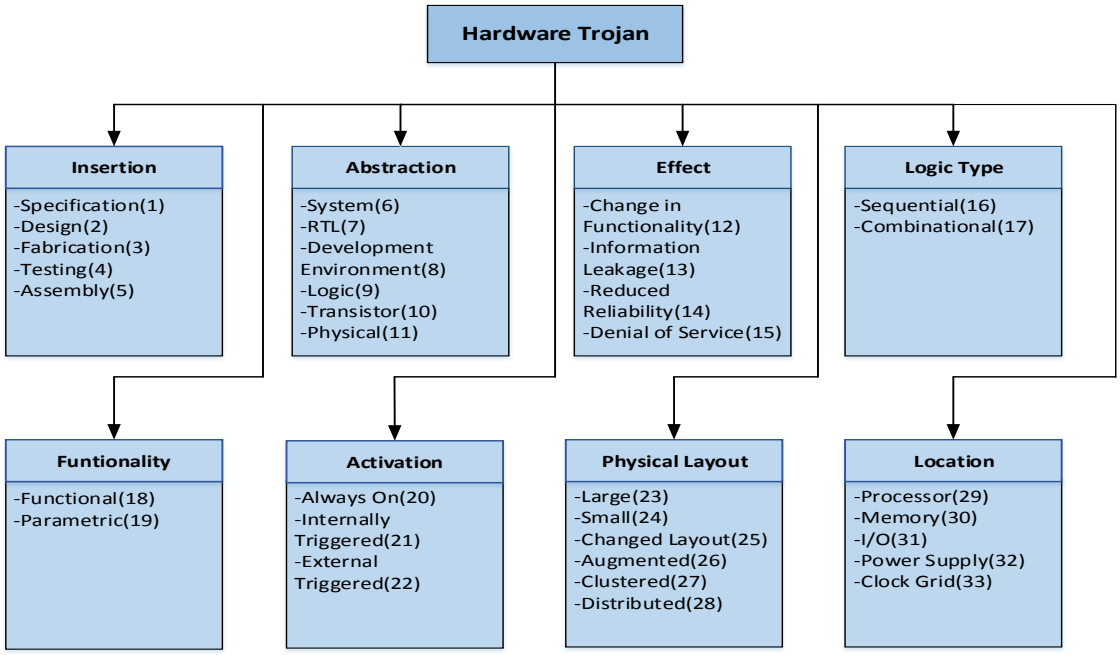
یک تروجان سخت‌افزاری به‌عنوان تغییرات مخرب، ناخواسته و عمدی در یک مدار الکترونیکی تعریف شده است. چنین تغییراتی به‌طور بالقوه می‌تواند اثرات مختلفی به شرح زیر داشته باشد [۷]:

- **تغییر عملکرد:** یک تروجان سخت‌افزاری می‌تواند عملکرد یک مدار را تغییر دهد و باعث انجام عملیات مخرب و غیرمجاز مانند دور زدن الگوریتم‌های رمزنگاری، افزایش سطح دسترسی[۸]، انکار سرویس و ... شود.
- **تضعیف عملکرد:** یک تروجان سخت‌افزاری همچنین می‌تواند باعث آسیب رساندن به عملکرد IC شود و موجب خرابی آن شود، که به‌طور بالقوه می‌تواند سیستم بحرانی را که IC در آن بکار گرفته شده است، به خطر بیندازد. چنین اثراتی می‌تواند به‌صورت افزایش / کاهش در تأخیر مسیر، تزریق خطا و غیره باشد
- **نشت اطلاعات:** تروجان‌ها همچنین می‌توانند امنیت ارائه‌شده توسط الگوریتم‌های رمزنگاری را به خطر بیندازند یا به‌طور مستقیم اطلاعات حساسی که توسط IC مدیریت می‌شوند را نشت دهد. نشت اطلاعات حساس می‌تواند شامل کلیدهای رمزنگاری یا سایر اطلاعات حساس از طریق debug یا پورت‌های ورودی / خروجی، کانال‌های جانبی (تأخیر، power) و غیره باشد.

طبقه‌بندی مدارهای تروجان به شکل‌های مختلف ارائه‌شده است و همچنان که انواع تروجان و حملات جدیدتر کشف می‌شوند، این دسته‌بندی‌ها نیز تغییر می‌کنند [۸]. یک مدل جامع برای طبقه‌بندی تروجان که برخلاف مدل‌های قبلی که مرحله مشخصات[۹] و طراحی مورد اعتماد فرض می‌شدند، اما در این طبقه‌بندی آسیب‌پذیری این مراحل نیز مورد توجه قرار گرفته است. این روش طبقه‌بندی تروجان بر اساس هشت دسته کلیدی است: درج (چه زمانی تروجان درج شده است؟)، انتزاع (تروجان در کجا و چه سطحی قرار داده شده؟)، اثر (تروجان چه‌کاری انجام می‌دهد؟)، نوع منطق (از چه نوع مدار منطقی استفاده می‌کند؟)، عملکرد (عملکرد آن چگونه است؟)، فعال‌سازی (چگونه فعال می‌شود؟)، طرح فیزیکی (در چه مقیاسی ظاهر می‌شود؟) و محل (کجا قرار گرفته است؟) این طبقه‌بندی در شکل ۲ نشان داده شده است.

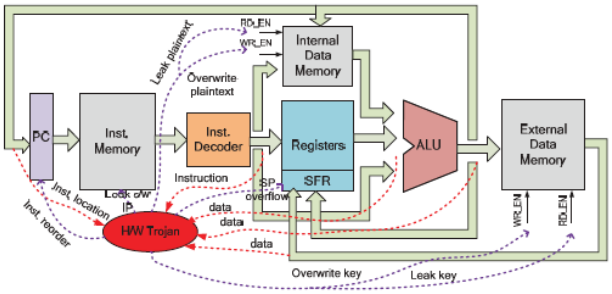
با توجه به هدف مقاله ارائه‌شده، ما بر روی تروجان‌های موجود در پردازنده‌ها تمرکز داریم. یک نوع خاص از تروجان‌های سخت‌افزاری در پردازنده نهفته که می‌تواند به‌وسیله نرم‌افزار یا داده‌ها فعال شود تا اطلاعات حساس را نشت دهد توسط ونگ و همکاران ارائه‌شده است [۹]. در این مدل حمله، مهاجم با طراحی یک تروجان با سربار کم و شناسایی دشوار می‌تواند اطلاعات ارزشمند از یک سیستم به کار گرفته شده را به‌دست آورد. این اطلاعات ارزشمند می‌تواند شامل کلیدهای مخفی ذخیره شده در یک پردازنده، کد در حال اجرا بر روی پردازنده یا داده‌ی در حال پردازش بر روی آن باشد.

طراحی این نوع تروجان به‌گونه‌ای است که می‌تواند از کد نرم‌افزار اجرایی بر روی پردازنده سوءاستفاده کند. تروجان‌های سخت‌افزاری قابل بهره‌برداری از نرم‌افزار به‌گونه‌ای طراحی می‌شوند که بتوانند حملات همه‌منظوره‌ای را با تأثیرات عملکردی مختلف که به‌وسیله نرم‌افزار مخرب تعریف می‌شود را انجام دهند. این تروجان‌ها برای پردازنده‌های همه‌منظوره و پردازنده‌های نهفته پیچیده مناسب هستند. این پردازنده‌ها ویژگی‌های امنیتی پشتیبانی سخت‌افزاری را دارند. حملات مختلفی می‌توانند بر اساس خراب کردن این ویژگی‌های امنیتی انجام شوند.




شکل ۲- طبقه‌بندی تروجان بر اساس هشت دسته کلیدی [۸]

فعال‌سازی این تروجان ترتیبی به‌وسیله ترکیبی از ترتیب دستورالعمل‌ها و داده‌ها به‌منظور انواع خرابکاری در سیستم و نشت اطلاعات حساس مانند IPهای نرم‌افزاری، کلید مخفی استفاده شده در عملیات رمزنگاری در پردازنده و بد عمل کردن سیستم انجام می‌گیرد. شکل ۲ انواع روش‌های فعال‌سازی و نحوه عملکرد این نوع تروجان را نشان می‌دهد.




شکل ۳- انواع شرایط فعال‌سازی و نحوه عملکرد تروجان پیشنهاد شده در یک پردازنده نهفته [۹]

با توجه به شکل ۲ به دلیل اینکه میکروکنترلر ۸۰۵۱ یک معماری چند سیکلی با استفاده از یک ماشین حالت متناهی[۱۰] برای کنترل اجرای دستورالعمل دارد، مهاجم می‌تواند ناظر ترتیب فعال‌ساز تروجان را در منطق کنترل میکروکنترلر ۸۰۵۱ قرار دهد. با توجه به [۹] منطق کنترل میکروکنترلر ۸۰۵۱ در رمزگشا آن قرار دارد. بنابراین از دستورالعمل‌های اجرایی و داده‌های استفاده شده توسط پردازنده ۸۰۵۱ برای فراهم کردن شرایط فعال‌سازی تروجان استفاده می‌شود. در این روش، فرآیند فعال‌سازی تروجان به‌وسیله مهاجم قابل کنترل است و همچنین به دلیل اینکه چندین دستورالعمل یا داده تا زمانی که ویژگی فعال‌سازی تروجان را داشته باشند تروجان مورد نظر را فعال می‌سازند؛ این فرآیند فعال‌سازی انعطاف‌پذیر است. برای یک پردازنده که یک سیستم عامل با چندین برنامه کاربر را اجرا می‌کند، طراحی شرایط فعال‌سازی تروجان سخت‌افزاری به چندین روش می‌تواند انجام شود به‌طوری‌که سه روش فعال‌سازی ذکر شده می‌توانند در مکانیزم فعال‌سازی تروجان بکار برده شوند.

در این مدل حمله، مهاجم برای طراحی تروجان دو بخش شرایط فعال‌سازی تروجان و نحوه عملکرد تروجان را در نظر می‌گیرد. برای مثال فرض می‌شود پردازنده ۸۰۵۱ یک برنامه اختصاصی رمزنگاری RC5 را اجرا می‌کند. بنابراین شرایط فعال‌سازی تروجان باید توانایی استفاده از این برنامه خاص را داشته باشد. از سوی دیگر ما باید مطمئن شویم که این شرط را فقط مهاجم می‌داند و فقط توسط مهاجم قابل کنترل است و در طول عملکرد عادی برنامه فعال نمی‌شود. بنابراین ما نمی‌توانیم یک ترتیب ساده از دستورالعمل‌ها را به‌عنوان شرط فعال‌سازی تروجان در نظر بگیریم.

بنابراین مهاجم یک ترکیب از ترتیب دستورالعمل‌ها و داده را در نظر می‌گیرد. به‌طور خاص، با در نظر گرفتن یک ترتیب از دستورالعمل‌ها، یک ترتیب از داده‌های متن ساده[۱۱] را به‌دست می‌آورد که با ترتیب داده‌های از پیش تعیین شده برای فعال‌سازی تروجان مقایسه می‌شوند.

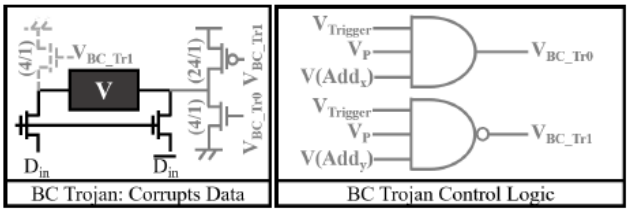
به‌این‌ترتیب با توجه به شکل ۲ مهاجم در رمزگشا پردازنده ۸۰۵۱ یک ماشین حالت متناهی تروجان به‌منظور نظارت ترتیب دستورالعمل‌ها قرار می‌دهد تا اجرای یک قطعه کد خاص را مشاهده کند و با توجه به آن قطعه کد، اطلاعات مورد نظر را نشت دهد. به محض مشاهده یک قطعه کد خاص و به دلیل عدم آگاهی در مورد پیاده‌سازی نرم‌افزاری الگوریتم رمزنگاری، داده به‌جای اینکه از گذرگاه داده گرفته شود؛ از ورودی‌های واحد محاسبه و منطق گرفته می‌شود. اگر ترتیب داده‌های از پیش تعریف شده، توسط ماشین حالت متناهی تروجان کامل مشاهده شود؛ ماشین حالت متناهی عملکرد تروجان را فعال می‌کند. در حین گرفتن هر کلمه متن خام یک مقایسه با داده‌ی از پیش تعریف شده انجام خواهد شد و بر اساس نتیجه این مقایسه و تطابق یک حالت روبه‌جلو حرکت خواهد شد و در صورت عدم تطابق ترتیب حالت‌ها، ماشین حالت متناهی به حالت اولیه برمی‌گردد. اگر تمام ترتیب متن ساده از پیش تعریف شده مشاهده شود، ماشین حالت متناهی عملکرد تروجان را فعال خواهد کرد. در صورت فعال شدن تروجان علاوه بر نشت اطلاعات حساس، انواع خرابکاری در سیستم مانند نوشتن غیرمجاز در حافظه، تغییر مقدار اشاره‌گر پشته به‌منظور تغییر مکان آدرس برگشت از زیروال و اشاره‌کردن به یک زیروال مخرب و تغییر کلید رمزنگاری در ورودی واحد محاسبه و منطق می‌تواند انجام شود.

نوع دیگری از تروجان‌های موجود در پردازنده که مربوط به حوزه اعتماد سخت‌افزار[۱۲] می‌باشد؛ در [۱۰] ارائه‌شده است. یک روش طراحی سیستماتیک تروجان

سخت‌افزاری بنام DeTrust با شرط فعال‌سازی ضمنی مخفی در برابر همه تکنیک‌های تأیید اعتماد سخت‌افزار، مقاوم است. این تروجان نشان می‌دهد که روش‌های درستی‌سنجی قبلی مانند VeriTrust [۱۱] و FANCI [۱۲] دارای محدودیت‌هایی هستند. این روش یک حمله به یک پردازنده را طراحی و پیاده‌سازی می‌کند که قادر است این تکنیک‌های تأیید اعتماد سخت‌افزار را دور بزند درحالی‌که از این مراحل تأیید عملکرد بدون شناخته شدن عبور می‌کند. هدف DeTrust این است که شرط فعال‌سازی خود را به نحوی طراحی کند تا بتواند در برابر روش‌های تأیید اعتماد ناشناخته بماند و درعین‌حال عملکرد مخرب تروجان حفظ شود. عملکرد این نوع تروجان می‌تواند به‌صورت فعال کردن یک زمان‌سنج، غیرفعال کردن وقفه، به خطر انداختن داده، به خطر انداختن اشاره‌گر پشته، به خطر انداختن آدرس حافظه، به خطر انداختن دستورالعمل، دست‌کاری آدرس، دسترسی به حافظه‌ها و اجرای کدهای مخرب باشد.

نوع دیگری از تروجان‌های سخت‌افزاری که اطلاعات حساس موجود در بانک ثبات یک ریزپردازنده را دست‌کاری می‌کنند و یا نشت می‌دهند در [۴] ذکر شده‌اند. یک تروجان می‌تواند فیلد سطح دسترسی فعلی (CPL) را در فایل ثبات Code Segment تغییر دهد. مهاجم می‌تواند به‌وسیله دست‌کاری ورودی بانک ثبات که حالت اجرایی پردازنده را ذخیره می‌کند؛ کنترل حالت هسته[۱۳] را به‌دست بگیرد و عملیات غیر مجاز را اجرا کند.

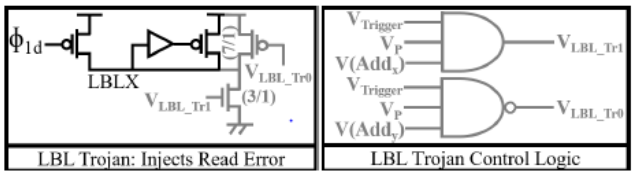
همچنین یک تروجان در بانک ثبات می‌تواند در زمانی که سلول‌های بانک ثبات در حال نگهداری هستند، بیت‌های رجیسترها را به "۰" یا "۱" تنظیم کند. شکل ۴-الف یک تروجان سلول بیتی را نشان می‌دهد که می‌تواند بیت ذخیره شده را خراب کند. شکل ۴-ب منطق تولید سیگنال کنترل را برای تروجان BC نشان می‌دهد. زمانی که راه‌انداز تروجان فعال می‌شود (Vtrigger=1)، آدرس نوشتن (Addx) با الگوی داده $P_{SET}$، $V_{BC\text{-}Tr0}$ را تنظیم می‌کند. بنابراین می‌تواند ترانزیستور NMOS را روشن کند و گره داده بانک ثبات را به زمین اتصال کوتاه کند. همچنین اگر Addy با الگوی داده‌ی $P_{SET}$ نوشته شود، $V_{BC\text{-}Tr1}$ را تنظیم خواهد کرد که می‌توان از آن برای اتصال گره داده به Vdd از طریق ترانزیستور PMOS استفاده کرد [۴].




(الف) (ب)

شکل ۴- تروجان سلول بیتی: (الف) مدار Payload؛ (ب) مدار trigger [۴].

همچنین در زمان خواندن بانک ثبات، خطاهای خواندن قابل تزریق هستند. یعنی هنگام خواندن داده، داده صفر با استفاده از تروجان پیشنهادی (Local Bitline) به‌عنوان ۱ خوانده می‌شود و داده ۱ به‌عنوان صفر خوانده می‌شود.




(الف) (ب)

شکل ۵- تروجان Local Bitline: (الف) مدار Payload؛ (ب) مدار trigger [۴].

بر طبق شکل ۵ تروجان‌های NMOS و PMOS برای تزریق خطای خواندن ۱-<۰ و ۰->۱ مورد نیاز هستند. ترانزیستورهای NMOS و PMOS به‌ترتیب با $V_{LBL\text{-}Tr1}$ و $V_{LBL\text{-}Tr0}$ روشن می‌شوند. $V_{LBL\text{-}Tr0}$ و $V_{LBL\text{-}Tr1}$ فقط زمانی مؤثر واقع می‌شوند که $\Phi_{ld}$ در طول عمل خواندن high باشد [۴].

ثبات‌های عام‌منظوره نیز می‌توانند به‌عنوان ثبات‌های قربانی توسط تروجان‌ها مورد حمله قرار بگیرند. فعال‌سازی این نوع تروجان می‌تواند از طریق یک فرآیند قربانی که به آدرس فعال‌ساز دسترسی دارد، انجام شود. تعداد دسترسی‌های به آدرس فعال‌ساز توسط ورودی کاربر کنترل می‌شود. مهاجم می‌تواند تروجان را از طریق ورودی فعال کند. در صورت فعال شدن تروجان، یک عملیات محاسباتی غلط توسط دستورالعملی که از آن ثبات عام‌منظوره استفاده می‌کند، انجام می‌گیرد. همچنین در موارد آدرس‌دهی غیرمستقیم ثباتی که از ثبات قربانی استفاده می‌شود، مهاجم می‌تواند اشاره‌گر آدرس را به یک آدرس خاص در فضای فرآیند قربانی تنظیم کند و از آن آدرس داده حساس را نشت دهد. همچنین تأثیر تروجان بر روی ثبات‌هایی مانند instruction pointer، stack pointer و base pointer ممکن است برای خراب کردن control-flow برنامه استفاده شود [۴].

یک روش شناسایی تروجان‌های مخرب داده موجود در IP ها که مجموعه‌ی راه‌های معتبر V به‌روزرسانی مقدار ثبات R را در نظر می‌گیرد و یک ویژگی[۱۴] برای آن می‌نویسد در [۵] ارائه‌شده است. همچنین با استفاده از یک ابزار و با کمک این ویژگی به شناسایی تروجان مخرب مقدار ثبات می‌پردازد. اگر ابزار یک نمونه را شناسایی نکند که این ویژگی را در تعداد T کلاک نقض کند؛ غیر از آن‌هایی که در مجموعه V هستند هیچ توالی ورودی دیگری وجود ندارد که مقدار R را تغییر دهد. بااین‌حال، اگر ابزار نمونه خروجی‌ای را نشان دهد که این ویژگی را نقض می‌کند، مقدار ثبات R می‌تواند خراب شود [۵].

روش‌های مقابله با تروجان شامل دو رویکرد پیشگیری از درج تروجان و یافتن آن بعد از ساخت تراشه می‌باشد. روش‌های پیشگیری شامل مبهم کردن[۱۵] و تغییر مدار اصلی تراشه به‌منظور ایجاد یک طرح امن، کمک به یک روش تشخیص تروجان و یا ایجاد زنجیره تولید قابل اعتماد است [۱۳]. یک روش مبهم کردن بر اساس افزایش تعداد حالت‌های قابل دستیابی مدار اصلی در [۱۴] ارائه‌شده است. این حالت‌ها به دو بخش فضای حالت اصلی و فضای حالت ایزوله تقسیم می‌شوند. ورود به فضای حالت اصلی با استفاده از یک الگوی ورودی خاص (به‌عنوان مثال کلید مخفی) حاصل می‌شود. با هر الگوی ورودی اشتباه، تراشه در فضای حالت ایزوله قرار می‌گیرد. این فضای حالت ایزوله به‌گونه‌ای طراحی شده است که پس از ورود نمی‌توان از آن خارج شد و خروجی‌ها هرگز درست نمی‌شوند.

نمونه‌ای از کاهش حملات تروجان‌های سخت‌افزاری در پردازنده‌های نهفته با استفاده از یک مبهم‌کننده[۱۶] مبتنی بر سخت‌افزار در [۱۵] ارائه‌شده است. همان‌گونه که در این بخش ذکر شد، برای فعال کردن تروجان از طریق نرم‌افزار، مهاجم باید بتواند یک دنباله دقیق از رویدادهای قابل شناسایی توسط بخش ماشه[۱۷] تروجان مانند مقادیر خاص سیگنال، ترتیب داده و دستورالعمل یا حالت پردازنده را ایجاد کند. این کار نیازمند اجرای کد بر روی سخت‌افزار است. این کد به‌طور خاص برای تولید این رویدادها طراحی شده است. روش مبهم کردن پیشنهاد شده در [۱۵] می‌تواند به‌طور مستقیم در یک پردازنده خط لوله برای برنامه‌های نهفته اجرا شود. این تکنیک، ترتیب و جریان دستورالعمل‌های اجرایی ماشین در زمان اجرا را تغییر می‌دهد به‌طوری‌که از ایجاد رویداد فعال‌سازی ماشه تروجان و فعال کردن مدار مخرب به‌وسیله نرم‌افزار جلوگیری می‌کند. این ایده، مبهم کردن را بر طبق یک الگوریتم که پس از تولید پیکربندی شده است، انجام می‌دهد. در این روش، با وجود آگاهی مهاجم از وجود مبهم‌کننده، نمی‌تواند جریان دستورالعمل‌های اجرایی را کنترل کند، زیرا جریان و ترتیب دستورالعمل‌های اجرایی به پیکربندی الگوریتم بستگی دارد [۱۵].

همچنین می‌توان تروجان‌ها را با استفاده از روش‌های غیر مخرب در زمان اجرای برنامه بر روی تراشه شناسایی کرد. برای مثال، یک روش زمان اجرا می‌تواند سیاست‌های دسترسی قانونی به حافظه را مشخص کند. این سیاست‌ها در یک ماژول سخت‌افزاری قابل پیکربندی مجدد[۱۸]، سنتز می‌شوند. این ماژول سخت‌افزاری، قانونی بودن هر درخواست از یک ماژول مسیر داده را برای دسترسی به حافظه، تعیین می‌کند. این روش به یک تکنیک توسعه داده شد تا کنترل‌کننده‌های امنیتی بر پایه سخت‌افزار را تولید کند تا بتوانند اجزای مخرب پردازنده را در زمان اجرا شناسایی کنند [۱۶].

یک روش تشخیص تروجان در یک سیستم حاوی چندین مؤلفه که به‌وسیله یک گذرگاه AXI4-Lite متصل شده‌اند با استفاده از تجزیه و تحلیل کامل مجموعه AXI4-Lite assertion توسط ARM ارائه‌شده است[۱۷]. تروجان‌هایی که در don't care های RTL مخفی هستند؛ می‌توانند زیرساخت‌های گذرگاه تراشه را تغییر دهند تا اطلاعات را در طی چرخه‌های بیکاری گذرگاه نشت دهند. روش یافتن تروجان[۱۷]. مجموعه‌ای از ادعاهای ارائه‌شده توسط ARM را برای پروتکل گذرگاه AMBA AXI4-Lite تجزیه و تحلیل می‌کند تا شرایط غیرقابل اطمینان سیگنال‌های اتصال گذرگاه را در سیستم نشان دهد. این گذرگاه شامل تعدادی master و slave است. عناصر master واحدهای عملکردی گذرگاه هستند که در SystemVerilog نوشته شده‌اند. Slave های گذرگاه، جمع‌کننده‌های ۸ بیتی ساده هستند که عملوندها و نتایج محاسبات آن‌ها از طریق عملیات خواندن و نوشتن در یک ثبات کنترل شده توسط سیگنال‌های AXI4-Lite، قابل دسترسی هستند. یک مثال از نشت اطلاعات توسط تروجان مربوط به زمانی است که طرح در reset قرار دارد (reset-n فعال پایین است) و خروجی RDATA بر اساس سیگنال‌های ARVALID(سیگنال معتبر بودن آدرس خواندن) و ARADDR (آدرس خواندن) تغییر کند. به‌محض بررسی حالت ماشین خواندن، اگر در حالت «خواندن بیکار»، ARVALID تنظیم شده باشد، به آدرس داده شده توسط ARADDR دسترسی یافته و RDATA به‌روز می‌شود. هنگامی‌که reset فعال می‌شود، حالت ماشین مجبور است در حالت «خواندن بیکار» باقی بماند. بنابراین اگر سیگنال reset در پردازنده نگه داشته شود، اما سیگنال‌هایARVALID و ARADDR در حال نوسان باشند، داده‌های مکان‌های مختلف رجیستر در RDATA نمایان می‌شود. RDATA یک سیگنال ۳۲ بیتی است اما coprocessor حداکثر از ۸ بیت برای پیاده‌سازی رجیسترها استفاده می‌کند. در زمان reset، بیت‌های استفاده نشده RDATA صفر نمی‌شوند. به همین دلیل، تجزیه و تحلیل‌ها این مسئله را به‌عنوان منبع بالقوه نشت اطلاعات معرفی می‌کنند، زیرا یک تروجان می‌تواند به این بیت‌های استفاده نشده هر مقداری ازجمله سیگنال‌های محرمانه داخلی را اختصاص دهد [۶]. روش یافتن تروجان ارائه‌شده در [۶] سیگنال‌های گذرگاه مورد استفاده برای پیاده‌سازی تروجانی که داده را در بیت‌های استفاده نشده رجیستر در جمع‌کننده Coprocessor می‌نویسد، مشخص می‌کند. طرح آدرس‌دهی باعث می‌شود تا رجیسترها ۳۲ بیتی باشند؛ اما حداکثر اندازه رجیستر استفاده شده توسط coprocessor ۸ بیت است. از فضای استفاده نشده می‌توان برای مخفی‌کردن داده‌های تروجان استفاده کرد. داده‌های مخرب ذخیره شده در بیت‌های استفاده نشده رجیستر می‌توانند توسط یک مؤلفه مخرب دیگر گذرگاه و با استفاده از یک تراکنش خواندن معمولی، خوانده شوند. در [۶] بر اساس تجزیه و تحلیل کد منبع RTL پردازنده منبع باز OR1200، یکی از روش‌های رسمی، روش بررسی مدل، برای تشخیص تروجان سخت‌افزاری پیشنهاد شده است. با توجه به طبقه‌بندی تروجان‌های سخت‌افزاری در شکل ۲، تروجانی که برای یافتن مورد نظر است؛ متعلق به مرحله طراحی، دارای توصیف رفتاری و فعال‌ساز داخلی است و عملکرد پردازنده را تغییر می‌دهد. این نوع تروجان دارای منطق ترکیبی است. ایده اصلی روش یافتن تروجان Model checking، که یکی از روش‌های درستی‌سنجی صوری است، استفاده از یک زبان توصیفی برای بیان خواص مورد نظر سیستم است. آزمایش‌هایی بر روی واحد محاسبه و منطق (ALU) پردازنده OR1200 منبع باز انجام شده است. یک ALU معمولاً عملیات or، xor، shift، گسترش بیت، جمع، تفریق و سایر عملیات را انجام می‌دهد. در آزمایش طرح بدون تروجان ورودی a و b مطابق با طراحی مورد انتظار، خروجی a xor b را می‌دهد. سپس آزمایش‌هایی با تروجان سخت‌افزاری دارای منطق ترکیبی انجام می‌شود.

عملکرد تروجان سخت‌افزاری به‌صورت XOR بیت a[31] از عملوند ورودی a در ALU با بیت b[20] از عملوند ورودی b است. نتیجه بعد از XOR معکوس خواهد شد. اگر نتیجه‌ی XOR دو بیت a[31] و b[20] ۱ باشد، تروجان سخت‌افزاری فعال نخواهد شد و نتیجه عملیات xor دو عملوند ورودی، صحیح خواهد بود. اگر نتیجه‌ی XOR دو بیت a[31] و b[20] صفر باشد، تروجان سخت‌افزاری فعال خواهد شد و نتیجه عملیات xor دو عملوند ورودی صفر خواهد بود. بنابراین پس از افزودن تروجان سخت‌افزاری به کد منبع وریلاگ، آزمایش‌ها به شکست می‌انجامند. بنابراین، هنگامی‌که ورودی a=0[31] و b=1[20]، شرایط فعال‌سازی تروجان سخت‌افزاری دارای منطق ترکیبی را برآورده کنند، این تروجان می‌تواند باعث تغییر نتایج خروجی شود [۶].

نتایج آزمایش‌ها در [۶] نشان می‌دهد که روش model checking ارائه‌شده می‌تواند تروجان سخت‌افزاری را که در مرحله طراحی به کد منبع اضافه شده است را به‌طور مؤثر تشخیص دهد.

تکنیک‌های نظارتی زمان اجرا بر روی محاسبات می‌تواند به‌طور قابل‌توجهی اثر فاجعه‌بار تروجان‌ها را کاهش دهد. غیرفعال کردن تراشه پس از تشخیص منطق مخرب و یا دور زدن منطق مخرب برای انجام عملیات قابل‌اعتماد، با وجود برخی از اجزای غیرقابل‌اعتماد، امکان‌پذیر است. ناظرهای امنیتی قابل پیکربندی به‌صورت منطق قابل پیکربندی مجدد، برای کنترل‌کردن عملیات به سیستم‌های بر روی تراشه اضافه می‌شوند. این ناظرها طوری پیکربندی می‌شوند که حالت‌های متناهی ماشین را پیاده‌سازی کنند و رفتار سیگنال‌ها را بررسی می‌کنند تا بتوانند رفتار غیرمنتظره یا غیرقانونی ایجاد شده توسط یک تروجان مانند دسترسی به فضای حافظه محافظت شده یا ورود به حالت آزمون در طول عملیات عادی را تشخیص دهند. منطق مخرب در صورت شناسایی غیرفعال خواهد شد و هسته منطق قابل پیکربندی مجدد عملکرد صحیح منطق مخرب را در مدار اصلی پیاده‌سازی خواهد کرد [۱۸].

از دیگر تکنیک‌های نظارتی زمان اجرا، زمان‌بندی و اجرای تغییرات معادل عملکردی است که توسط کامپایل‌های مختلف و یا تغییرات الگوریتم‌های مختلف به دست می‌آید. مقایسه نتایج به‌دست‌آمده از اجرا بر روی مؤلفه‌های پردازشی مختلف و تشخیص عدم تطابق به تشخیص تروجان کمک می‌کند. این ایده شبیه برنامه‌نویسی چند نسخه‌ای است که به‌صورت تولید و اجرای چندین نسخه عملکردی معادل از یک برنامه برای دستیابی به قابلیت اطمینان بالا در حضور خطاهای نرم‌افزاری است [۱۸].

در سیستم‌های مبتنی بر ریزپردازنده، یک راه‌حل نرم‌افزاری می‌تواند فعالیت تروجان را کشف کند و تحمل اثرات تروجان می‌تواند رویکرد حفاظتی مؤثری باشد. راه‌حل‌هایی مانند حذف مدارهای مشکوک و جایگزین کردن یک استثنای نرم‌افزاری باعث دور زدن تروجان‌های سخت‌افزاری مخرب می‌شود. این نوع تروجان‌های سخت‌افزاری که با روش‌های نرم‌افزاری دور زده می‌شوند، از نظر هدف مخربی که دارند شبیه تروجان‌های نرم‌افزاری هستند و در هسته‌های مالکیت معنوی سخت‌افزاری یا کد دستورالعمل اجرایی بر روی پردازنده نهفته درج می‌شوند [۱۸].

تجزیه و تحلیل کانال جانبی[۱۹]، به‌عنوان ابزاری برای حمله به الگوریتم‌های رمزنگاری مورد استفاده قرار می‌گیرد. همچنین تجزیه و تحلیل کانال جانبی می‌تواند برای یافتن تروجان مورد استفاده قرار گیرد. به دلیل وجود تروجان‌های سخت‌افزاری در یک مدار آلوده به تروجان، طرح مدار آلوده به تروجان با طرح مدار اصلی بدون

تروجان از نظر فیزیکی متفاوت خواهد بود. محدودیت اصلی در این روش نیاز به وجود تراشه طلایی بدون تروجان است [۱۳].

درحالی‌که ممکن است روش‌های تجزیه و تحلیل کانال جانبی تا حدودی در تشخیص تروجان‌ها موفق باشند، دستیابی به پوشش بالا در هر دروازه یا شبکه و استخراج سیگنال‌های کوچک و غیرعادی کانال‌های جانبی تروجان‌های سخت‌افزاری در حضور فرآیندها و تغییرات محیطی، کاری دشوار است [۱۹].

به‌منظور اجرای امن برنامه بر روی یک بستر سخت‌افزاری، رویکرد جدیدی که اخیراً در برخی پردازنده‌های ARM و در سری Cortex-M و Cortex-A استفاده می‌شود، فن‌آوری TrustZone است [۲۰]. این فن‌آوری برای مجزا کردن سخت-افزار در سیستم‌های نهفته معرفی شده است. داده‌ها در یک محیط مجزا توسط سخت‌افزار مورد اعتماد پردازش می‌شوند. برای جلوگیری از نشت اطلاعات، تکنیک-های جداسازی سخت‌افزاری باعث ایجاد محیط اجرایی قابل‌اطمینان برای حفاظت از اطلاعات و برنامه‌های حساس می‌شود. زمان اجرای برنامه به دو محیط مجزای secure world و normal world تقسیم می‌شود [۲۱] و [۲۲].

این کار در سطح گذرگاه فیزیکی با استفاده از فن‌آوری TrustZone به‌عنوان یک قالب امنیتی در پردازنده ARM ارائه‌شده است. اصل کار قرنطینه‌سازی سخت-افزار این است که جهان عادی نمی‌تواند به‌طور مستقیم به جهان امن دسترسی پیدا کند و این امر به‌وسیله مجزا سازی در سطح گذرگاه تضمین شده است. یک عامل امنیتی در دنیای امن بررسی‌های امنیتی لازم را انجام می‌دهد و درخواست دسترسی را قبول یا رد می‌کند. تبادل اطلاعات بین جهان امن و عادی می‌تواند توسط یک حافظه مشترک اختصاص داده شده در جهان عادی انجام شود که برای هر دو جهان قابل دسترسی است. درواقع توسعه‌دهنده نرم‌افزار باید دو برنامه امن و عادی ایجاد کند و آن‌ها را به دو جهان برای استفاده از چارچوب TrustZone بفرستد. این کار پیچیدگی توسعه نرم‌افزار را افزایش می‌دهد. بنابراین معماری مجزا کردن سخت-افزاری به یک توسعه‌دهنده نرم‌افزار نیاز دارد تا نحوه تقسیم داده‌ها و کد را بر اساس ویژگی‌های امنیتی آن‌ها انجام دهد [۲۱].

یکی از روش‌هایی که از تکنولوژی TrustZone استفاده می‌کند، تکنیکی بنام EvoIsolator، یک چارچوب برای برش برنامه کامل برای ایجاد امنیت بر اساس جداسازی سخت‌افزاری، است. دنیای امن و دنیای عادی در چارچوب TrustZone توسط یک رابط مجزاسازی سخت‌افزاری در سطح گذرگاه از یکدیگر جدا شده‌اند. منابع ذخیره شده در دنیای امن به‌عنوان بخشی از پایه محاسبات قابل‌اعتماد (TCB) محسوب می‌شوند و هرگونه خطا یا آسیب‌پذیری امنیتی در جهان امن می‌تواند تمام سیستم را به خطر بیندازد. روش EvoIsolator با برنامه اصلی و مجموعه‌ای از متغیرهای حساس شروع می‌کند. برش‌های تصادفی امن و نرمال اولیه را تولید می-کند که این برش‌ها از یک مجموعه آزمون امنیتی عبور می‌کنند. EvoIsolator با استفاده از برنامه‌نویسی ژنتیکی، محاسبات حساس را در دنیای امن و محاسبات غیرحساس باقی‌مانده را در جهان عادی قرار می‌دهد تا بهترین پیکربندی کد را پیدا کند. TZOptimizer[۲۰] کد را دوباره مرتب می‌کند تا سربار ارتباط بین جهان امن و جهان عادی را کاهش دهد. یک مجموعه آزمایش نیز شامل جفت‌های ورودی-خروجی طراحی شده برای تشخیص نشت اطلاعات و دیگر اشکالات انجام می‌شود [۲۲].

## ۳- روش پیشنهادی قرنطینه کردن تروجان

چنان‌که گفته شد، در بسیاری از کاربردها علیرغم اطلاع کامل یا ناقص از وجود ناامنی‌های سخت‌افزاری در سیستم، امکان رفع اشکال یا تغییر سخت‌افزار مشکوک با هدف حذف تروجان وجود ندارد یا با ریسک بالا همراه است.

در این مقاله هدف ارائه روشی کارآمد برای قرنطینه‌سازی تروجان‌های احتمالی شناخته شده در طرح است. لازم به ذکر است هدف این مقاله یافتن تروجان نیست، بلکه فرض آن است اطلاعاتی در مورد تروجان طرح وجود دارد و می‌خواهیم در لایه نرم‌افزار آن را قرنطینه کنیم. در واقع فرض می‌کنیم برنامه‌نویس می‌تواند بخش‌هایی از کد که ممکن است تحت تأثیر تروجان دچار اشکال شوند را تشخیص دهد. هرکدام از این نواحی در این مقاله با عنوان «ناحیه بحرانی-امنیتی» شناخته می‌شوند.

در ادامه این فصل با ارائه دو روش قرنطینه‌سازی تروجان، سعی می‌کنیم در «نواحی بحرانی-امنیتی» تروجان را غیرفعال کنیم. در این نواحی سعی می‌شود از مؤلفه‌های ناهنجار پردازنده استفاده نشود و از مؤلفه‌های شناخته شده امن پردازنده به‌جای مؤلفه‌های ناامن مورد هدف تروجان استفاده شود. در روش اول قرنطینه‌سازی مبتنی بر تهیه نسخه پشتیبان از مقدار ثبات قربانی در پشته، به دلیل سروکار داشتن با حافظه‌ی پشته سربار کد اضافه شده به کد برنامه و سربار زمان اجرای برنامه زیاد خواهد شد. به همین دلیل با ارائه روش دوم قرنطینه‌سازی مبتنی بر تغییر نام ثبات سعی کردیم این سربارها را تا حدود زیادی کاهش دهیم.[۲۱]

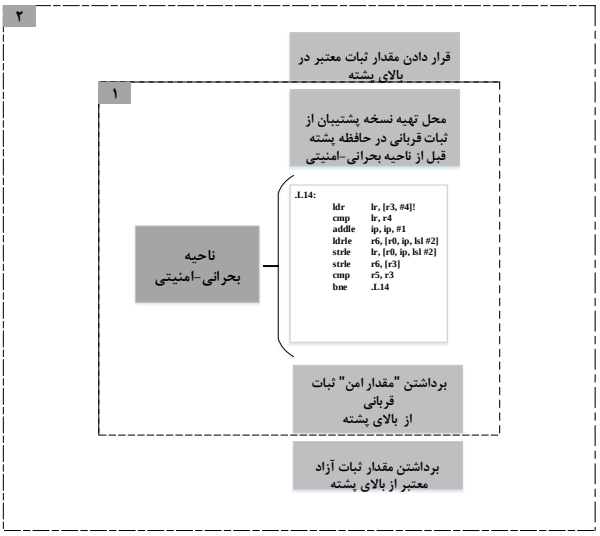
### ۳-۱- ارائه روش اول قرنطینه‌سازی: تهیه نسخه پشتیبان از مقدار ثبات در پشته

در اولین روش قرنطینه‌سازی ارائه‌شده در این مقاله با روش‌های تولید کد در مترجم arm-none-eabi-gcc و در سطح کد اسمبلی به ارائه‌ی یک روش قرنطینه‌سازی تروجان با استفاده از تهیه نسخه پشتیبان از ثبات‌های مورد هدف تروجان در حافظه امن قبل از «ناحیه بحرانی-امنیتی» می‌پردازیم. به‌این‌ترتیب با پردازش اتوماتیک کد اسمبلی ایجاد شده، قبل از «ناحیه بحرانی-امنیتی» یک نسخه پشتیبان از مقدار صحیح ثبات قربانی در حافظه امن (پشته) ذخیره می‌شود. در ابتدای «ناحیه بحرانی-امنیتی» از ثبات‌های آزاد امن به‌منظور برداشتن مقدار ثبات قربانی از بالای پشته و استفاده از آن ثبات آزاد به‌جای ثبات قربانی در تمام ناحیه بحرانی-امنیتی استفاده می‌شود. ثبات آزاد، یک ثبات امن با مقدار معتبر یا نامعتبر است که در ناحیه بحرانی-امنیتی کد اسمبلی مورد نظر بدون استفاده می‌باشد. بنابراین پیاده‌سازی این روش به تجزیه و تحلیل بازه عمر ثبات‌ها[۲۲] برای به‌دست آوردن لیست ثبات‌های آزاد در هر بلاک کد اسمبلی بستگی دارد. تجزیه و تحلیل بازه عمر ثبات‌ها و ایجاد تغییرات و اضافه کردن دستورالعمل‌های push و pop در کد اسمبلی برنامه به‌منظور قرنطینه‌کردن تروجان، به‌صورت خودکار با پس‌پردازش فایل متنی کد اسمبلی انجام می‌گیرد.

اگر ثبات آزاد با مقدار نامعتبر در ناحیه بحرانی-امنیتی موجود باشد، از آن به‌منظور برداشتن مقدار ثبات قربانی از بالای پشته و استفاده از آن به‌جای ثبات قربانی در ناحیه بحرانی-امنیتی استفاده می‌کنیم و بعد از ناحیه بحرانی-امنیتی مقدار امن و سالم ثبات قربانی از بالای پشته برداشته می‌شود تا برای ادامه برنامه که امن فرض می‌شود از آن ثبات استفاده شود. در صورت عدم وجود ثبات آزاد نامعتبر در ناحیه بحرانی-امنیتی، مجبور به استفاده از ثبات‌های آزاد با مقدار معتبر هستیم. ثبات آزاد معتبر ثباتی است که دارای مقدار معتبر بوده و در بلاک‌های کد اسمبلی بعد از ناحیه بحرانی-امنیتی از آن استفاده می‌شود. به‌منظور قرنطینه‌سازی تروجان در این شرایط قبل از ناحیه بحرانی-امنیتی ابتدا مقدار معتبر ثبات آزاد در بالای پشته قرار می‌گیرد و سپس مقدار ثبات قربانی در بالای پشته قرار می‌گیرد. در ناحیه بحرانی-امنیتی از ثبات آزاد معتبر برای برداشتن مقدار صحیح ثبات قربانی از بالای پشته استفاده می‌شود. در تمام ناحیه بحرانی-امنیتی از ثبات آزاد معتبر به‌جای ثبات قربانی استفاده می‌کنیم. بعد از ناحیه بحرانی-امنیتی ابتدا مقدار امن و سالم ثبات قربانی از بالای پشته برداشته می‌شود. سپس مقدار اصلی ثبات آزاد معتبر از بالای پشته برداشته می‌شود. شکل ۶ دیدگاه کلی روش قرنطینه‌سازی «تهیه نسخه پشتیبان از مقدار ثبات در پشته» را نشان می‌دهد. کلیه مراحل تحلیل بازه عمر ثبات‌ها و تغییر کد اسمبلی، به‌صورت اتوماتیک در مرحله تولید کد انجام می‌گیرد. قطعه کد ۱ نیز مثالی از روش قرنطینه‌سازی «تهیه نسخه پشتیبان از مقدار ثبات در

پشته» را نشان می‌دهد که در ناحیه بحرانی-امنیتی از یک ثبات آزاد با مقدار معتبر، r10، به‌جای ثبات قربانی، r3، استفاده شده است.

مزیت این روش این است که همیشه یک نسخه صحیح از مقدار ثبات قربانی قبل از خراب شدن مقدار آن در حافظه پشته وجود خواهد داشت. معایب آن حجم زیاد کد اضافه شده به کد اسمبلی برنامه به دلیل دستورالعمل‌های push و pop اضافه شده به کد اسمبلی می‌باشد. علاوه بر این، زمان اجرای دستورالعمل‌های push و pop به دلیل مراجعه به حافظه‌ی پشته زیاد است. بنابراین سربار زمان اجرای اضافه شده به زمان اجرای برنامه نیز زیاد خواهد شد و این عیب اصلی روش «تهیه نسخه پشتیبان از مقدار ثبات در پشته» است.




شکل ۶- دیدگاه کلی روش قرنطینه‌سازی «تهیه نسخه پشتیبان از مقدار ثبات در پشته» در حالت ۱) استفاده از ثبات‌های آزاد نامعتبر امن و ۲) استفاده از ثبات‌های آزاد معتبر امن

```
        push    {r10} @The free valid register is pushed onto the stack
        push    {r3}  @The victim register is pushed onto the stack
.L9:
        pop     {r10} @The safe value of the victim register is popped
@from the stack by the free valid register
        ldr     lr, [r10, #4]! @The victim register(r3) is substituted
@with the free valid register(r10)
        cmp     lr, ip
        addlt   r2, r2, #1
        cmp     r4, r10 @The victim register is substituted with the
@free valid register
        push    {r10} @Push the value of the victim register onto the
@stack
        bne     .L9
        pop     {r3} @Pop the safe value of the victim register from
@the stack to be used in continuing code blocks that are safe
        pop     {r10} @Pop the main value of the free valid register
@from the stack
```

قطعه کد ۱- استفاده از ثبات آزاد با مقدار معتبر، r10، به‌جای ثبات قربانی، r3، در ناحیه بحرانی-امنیتی با استفاده از روش قرنطینه‌سازی «تهیه نسخه پشتیبان از مقدار ثبات در پشته»

با توجه به قطعه کد ۱، پیاده‌سازی این روش نیازمند دقت بالا در تعیین محل قرار دادن دستورالعمل‌های push و pop در نواحی کد اسمبلی می‌باشد. به عبارتی به ازای هر دستورالعمل push اضافه شده در کد به‌منظور قرار دادن مقدار ثبات در بالای پشته، باید دستورالعمل pop متناظر با آن نیز به کد اسمبلی اضافه شود. با توجه به ساختار حلقه در کد اسمبلی و ساختار پشته، دستورالعمل‌های push و pop را در حلقه تکرار قرار می‌دهیم.

بنابراین با توجه به اینکه تغییر کد در سطح اسمبلی پیچیده است و نیاز به دانش کافی از ساختار و نواحی کد اسمبلی دارد؛ در صورت نداشتن دانش کافی از نحوه عملکرد دستورالعمل‌ها و ساختار حلقه تکرار و پشته در کد اسمبلی، احتمال بروز خطای لاجیکی بالا خواهد بود.

## ۳-۲- ارائه روش دوم قرنطینه‌سازی: تغییر نام ثبات در نواحی ناامن کد

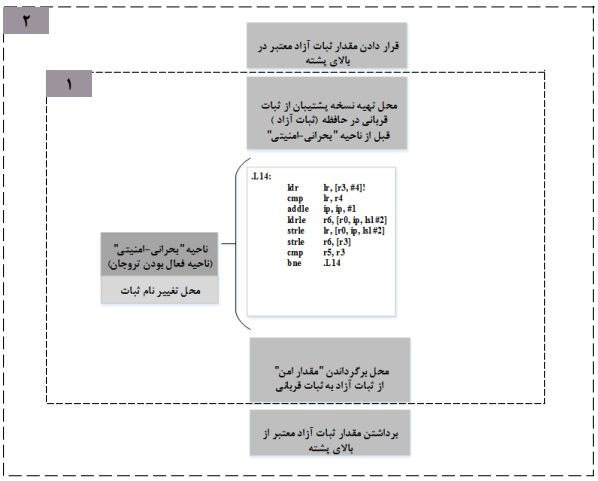
در روش دوم قرنطینه‌سازی ارائه‌شده در این مقاله، به تغییر نام ثبات‌های مورد هدف تروجان در نواحی بحرانی-امنیتی می‌پردازیم. در این روش یک مجموعه ثبات آزاد که امن فرض می‌شوند و در آینده نزدیک کمتر مورد استفاده قرار می‌گیرند؛ در نواحی ناامن کد برای ثبات‌های قربانی رزرو می‌شوند. در ناحیه بحرانی-امنیتی از ثبات آزاد امن به‌جای ثبات قربانی استفاده می‌کنیم. به‌بیان‌دیگر در آن ناحیه کد اسمبلی، «تغییر نام ثبات» را انجام می‌دهیم. پس از ناحیه بحرانی-امنیتی که تروجان غیرفعال است، مقدار سالم و امن ثبات قربانی را به آن برمی‌گردانیم.

تفاوت این روش با روش قبل این است که به‌جای اینکه قبل از ناحیه بحرانی-امنیتی مقدار سالم ثبات قربانی را در بالای پشته ذخیره کنیم، مقدار آن را در یک ثبات آزاد و امن قرار می‌دهیم. بنابراین پیاده‌سازی این روش نیز نیازمند تجزیه و تحلیل بازه عمر ثبات‌ها است. در اولین روش قرنطینه‌سازی، تهیه نسخه پشتیبان از مقدار ثبات در پشته، به دلیل اضافه شدن دستورالعمل‌های push و pop به کد اسمبلی برنامه، سربار زمان اجرا و سربار حجم کد اضافه شده به کد اسمبلی زیاد است. به همین دلیل با ارائه روش دوم قرنطینه‌سازی می‌توان سربار زمان اجرا و سربار حجم کد را به‌طور قابل‌ملاحظه‌ای کاهش داد. زیرا در این روش فقط قبل و بعد از ناحیه بحرانی-امنیتی یک دستورالعمل "mov" به ترتیب به‌منظور انتقال محتوای ثبات قربانی به ثبات آزاد امن و انتقال محتوای ثبات آزاد امن به ثبات قربانی، به کد اسمبلی اضافه می‌شود.

همانند روش قبل در صورت عدم وجود ثبات آزاد با مقدار نامعتبر، مجبور به استفاده از ثبات‌های آزاد با مقدار معتبر به‌جای ثبات‌های قربانی هستیم. در این شرایط ابتدا مقدار ثبات آزاد معتبر در بالای پشته قرار می‌گیرد. سپس مقدار ثبات قربانی به ثبات آزاد معتبر منتقل می‌شود. در تمام ناحیه بحرانی-امنیتی از ثبات آزاد معتبر به‌جای ثبات قربانی استفاده می‌کنیم. به عبارتی از تغییر نام ثبات در ناحیه بحرانی-امنیتی استفاده می‌کنیم. بعد از ناحیه بحرانی-امنیتی ابتدا مقدار سالم و امن ثبات قربانی از ثبات آزاد معتبر به آن برگردانده می‌شود. سپس مقدار اصلی ثبات آزاد معتبر از بالای پشته برداشته می‌شود. در این حالت، فقط قبل و بعد از ناحیه بحرانی-امنیتی دستورالعمل‌های push و pop به کد اسمبلی اضافه می‌شوند. درحالی‌که در روش اول مجبور به اضافه کردن دستورالعمل‌های push و pop در داخل ناحیه بحرانی-امنیتی نیز بودیم. بنابراین در این حالت روش دوم به روش اول تبدیل نخواهد شد و به دلیل دستورالعمل‌های اضافه شده کم‌تر، سربار روش دوم حتی در این حالت از روش اول کمتر خواهد بود. شکل ۷ دیدگاه کلی روش قرنطینه-سازی «تغییر نام ثبات در نواحی ناامن کد» را نشان می‌دهد. کلیه مراحل تحلیل بازه عمر ثبات‌ها و تغییر کد اسمبلی، به‌صورت اتوماتیک در مرحله تولید کد انجام می‌گیرد. قطعه کد ۲ نیز مثالی از روش قرنطینه‌سازی «تغییر نام ثبات در نواحی ناامن کد» را نشان می‌دهد که در ناحیه بحرانی-امنیتی از یک ثبات آزاد با مقدار معتبر، r10، به‌جای ثبات قربانی، r3، استفاده شده است.

با توجه به قطعه کد ۲، پیاده‌سازی این روش نیز نیازمند دقت بالا در تعیین محل قرار دادن دستورالعمل‌های push و pop و همچنین دستورالعمل mov در نواحی کد اسمبلی می‌باشد. در این روش نیز همانند روش «تهیه نسخه پشتیبان از مقدار ثبات در پشته» به ازای هر دستورالعمل push اضافه شده در کد به‌منظور قرار دادن مقدار ثبات در بالای پشته، باید دستورالعمل pop متناظر با آن نیز به کد اسمبلی اضافه شود. همچنین بایستی به ساختارهای حلقه تکرار و پشته در کد

اسمبلی و محل قرار گرفتن دستورالعمل‌های push و pop در داخل حلقه تکرار توجه شود. تعیین محل قرار گرفتن دستورالعمل mov نیز حائز اهمیت است.




شکل ۷- دیدگاه کلی روش قرنطینه‌سازی «تغییر نام ثبات در نواحی ناامن کد» ۱) استفاده از ثبات‌های آزاد نامعتبر ۲) استفاده از ثبات‌های آزاد معتبر

```
push    {r10}           @The free valid register value is pushed
@onto the stack
mov     r10, r3         @The victim register value(r3) is moved to
@the free valid register(r10)
.L9:
ldr     lr, [r10, #4]!  @ r3 is renamed to r10
cmp     lr, ip
addlt   r2, r2, #1
cmp     r4, r10         @ r3 is renamed to r10
bne     .L9
mov     r3, r10         @Safe value is returned to r3
pop     {r10}           @Pop the main value of  the free valid
@register from the stack
```

قطعه کد ۲- استفاده از ثبات آزاد با مقدار معتبر، r10، به‌جای ثبات قربانی، r3، در ناحیه بحرانی-امنیتی با استفاده از روش قرنطینه‌سازی «تغییر نام ثبات در نواحی ناامن کد»

به‌طورکلی میزان بروز خطا در روش «تغییر نام ثبات در نواحی ناامن کد» از روش «تهیه نسخه پشتیبان از مقدار ثبات در پشته» کمتر است. زیرا در روش «تهیه نسخه پشتیبان از مقدار ثبات در پشته» از دستورالعمل‌های push و pop در ناحیه بحرانی-امنیتی استفاده کردیم که این کار نیازمند داشتن دانش دقیق از ساختار پشته و ساختار حلقه تکرار است. درحالی‌که در روش «تغییر نام ثبات در نواحی ناامن کد» دستورالعمل‌های push و pop خارج از ناحیه بحرانی-امنیتی به کد اسمبلی اضافه می‌شوند. بنابراین احتمال بروز خطا نسبت به روش «تهیه نسخه پشتیبان از مقدار ثبات در پشته» کم‌تر است.

یک مکانیسم قرنطینه‌سازی با استفاده از گزینه‌های مترجم نیز وجود دارد. با به کار بردن گزینه‌های مترجم و ذکر نام ثبات قربانی، مترجم در هنگام تولید کد از استفاده از آن ثبات در کل برنامه صرف‌نظر می‌کند. بنابراین برنامه با استفاده از مؤلفه‌های امن اجرا می‌شود. همان‌طور که در قطعه کد ۳ نشان داده شده است؛ با به کار بردن گزینه -ffixed-RegisterName از استفاده از ثبات‌های r4، r8 و r10 در کل برنامه خودداری می‌شود و در سراسر اجرای برنامه بر روی پردازنده هدف از این ثبات‌ها استفاده نمی‌شود. به‌این‌ترتیب تروجان مخرب داده که محتوای این ثبات‌ها را در هنگام فعال‌شدن خراب می‌کند، قرنطینه خواهد شد.

```
arm-none-eabi-gcc  -S  -ffixed-r4 // R4 is isolated in created assembly code
arm-none-eabi-gcc  -S  -ffixed-r8 // R8 is isolated in created assembly code
arm-none-eabi-gcc  -S  ffixed-r10// R10 is isolated in created assembly code
```

قطعه کد ۳- قرنطینه کردن ثبات‌های قربانی r4، r8 و r10 در کل برنامه با استفاده از گزینه‌های کامپایلر

این روش زمانی مناسب است که پیچیدگی برنامه بالا نباشد زیرا زمانی که پیچیدگی برنامه زیاد است به دلیل تعداد زیاد ثبات‌های عملیاتی استفاده شده در کد اسمبلی برنامه و کمبود تعداد ثبات‌های آزاد برای جایگزینی ثبات‌های قربانی، برخلاف ذکر نام ثبات قربانی در گزینه مترجم -ffixed-RegisterName، مترجم از آن ثبات برای تولید کد اسمبلی برنامه استفاده خواهد کرد.

مزیت این روش نسبت به دو روش «تهیه نسخه پشتیبان از مقدار ثبات در پشته» و «تغییر نام ثبات در نواحی ناامن کد»، دقت بالای آن می‌باشد. احتمال بروز خطا در دستورالعمل‌های اضافه شده توسط کامپایلر به کد اسمبلی برنامه، صفر خواهد بود. بنابراین در برنامه‌های با پیچیدگی پایین، این روش بهینه‌ترین روش است.

## ۴- نتایج آزمایش‌ها

این بخش نتایج شبیه‌سازی قرنطینه‌سازی ثبات‌ها در پردازنده ARM7 را نشان می‌دهد. برنامه‌ها در مترجم arm-none-eabi-gcc کامپایل شده‌اند. شبیه‌سازی‌ها در محیط نرم‌افزار ModelSim SE-64 10.5 انجام شده‌اند، به این صورت که یک مدل کامل از پردازنده فوق ایجاد شده و تروجان در آن درج گردیده است. سپس پنج الگوریتم مرتب‌سازی به‌عنوان برنامه‌های آزمون روی آن اجرا شده‌اند.

در روش اول قرنطینه‌سازی «تهیه نسخه پشتیبان از مقدار ثبات در پشته»، در ابتدای ناحیه بحرانی-امنیتی، محتوای ثبات قربانی در بالای پشته ذخیره و در انتهای ناحیه این مقدار در ثبات قربانی بازیابی می‌گردد. نتایج آزمایش اندازه‌گیری سربار زمان اجرای این روش در جدول ۱ نشان داده شده است. Nr تعداد ثبات قرنطینه‌شده در کل برنامه را نشان می‌دهد. نکته قابل توجه این است که اثر منفی روی توان عملیاتی پردازنده با سربار زمان اجرا نمایان می‌شود.

با توجه به جدول ۱، با افزایش تعداد ثبات‌های قربانی، تعداد دستورالعمل‌های push و pop اضافه شده به کد اسمبلی برنامه افزایش خواهد یافت و به دلیل اینکه این دستورالعمل‌ها باعث دسترسی به حافظه (پشته) می‌شوند، سربار زمان اجرای برنامه در برنامه‌های با پیچیدگی بالا به‌طور فزاینده‌ای افزایش خواهد یافت. به همین دلیل این روش به‌طورکلی نمی‌تواند روش کارآمدی باشد. بنابراین با پیشنهاد روش دوم قرنطینه‌سازی «تغییر نام ثبات در نواحی ناامن کد» سربار زمان اجرا را نسبت به روش اول کاهش می‌دهیم. در روش «تغییر نام ثبات در نواحی ناامن کد» که روش اصلی ارائه‌شده در این مقاله می‌باشد؛ در ابتدای بلوک بحرانی-امنیتی، بازه عمر ثبات‌های سیستم تحلیل می‌گردد و ثبات‌های آزادی که در آینده نزدیک کمتر مورد استفاده قرار می‌گیرند، برای تهیه نسخه پشتیبان اطلاعات مورد استفاده قرار می‌گیرند. نتایج آزمایش اندازه‌گیری سربار زمان اجرای این روش در جدول ۲ نشان داده شده است.

با توجه به جدول ۲ به‌طورکلی سربار زمان اجرا در روش «تغییر نام ثبات در نواحی ناامن کد» ناچیز است. این روش با توجه به اضافه شدن دستورالعمل‌های "mov" به کد اسمبلی برنامه اصلی قبل و بعد از ناحیه بحرانی امنیتی، سربار زمان اجرای ناچیزی خواهد داشت زیرا فقط یک‌بار هرکدام از این دستورالعمل‌ها اجرا خواهند شد.

جدول ۱- نتایج شبیه‌سازی سربار زمان اجرا برحسب تعداد ثبات قرنطینه شده در روش اول «تهیه نسخه پشتیبان از مقدار ثبات در پشته»

| تعداد ثبات / نام برنامه | ۰ | ۱ | ۲ | ۳ | ۴ |
|---|---|---|---|---|---|
| QuickSort | ۰ | ٪ ۲۶/۹۶ | ٪ ۲۸ | ٪ ۲۹ | ٪ ۴۹ |
| CombSort | ۰ | ٪ ۲۲ | ٪ ۲۸ | ٪ ۳۲ | ٪ ۳۵ |
| MergeSort | ۰ | ٪ ۱/۷۹ | ٪ ۲/۳ | ٪ ۱۲/۱۷ | ٪ ۱۳/۹۶ |
| TimSort | ۰ | ٪ ۱/۴۹ | ٪ ۲۳/۷۲ | ٪ ۲۸ | ٪۲۹/۵۴ |
| CycleSort | ۰ | ٪ ۰/۹۳ | ٪ ۱/۱۹ | ٪ ۲/۱۳ | ٪ ۲/۱۳ |
| میانگین (٪) | ۰ | ٪ ۱۰/۶۳ | ٪ ۱۷ | ٪ ۲۱ | ٪ ۲۶ |

با توجه به دو روش قرنطینه‌سازی ارائه‌شده در این مقاله، انعطاف‌پذیری برنامه در این دو روش تحت تأثیر قرار نخواهد گرفت زیرا اعمال تغییرات در این دو روش در کد اسمبلی برنامه خواهد بود و ما بعد از کامپایل برنامه و تبدیل آن به کد اسمبلی این تغییرات را اعمال خواهیم کرد.

جدول ۲- نتایج شبیه‌سازی سربار زمان اجرا برحسب تعداد ثبات قرنطینه شده در روش «تغییر نام ثبات در نواحی ناامن کد»

| تعداد ثبات / نام برنامه | ۰ | ۱ | ۲ | ۳ | ۴ |
|---|---|---|---|---|---|
| QuickSort | ۰ | ٪ ۱/۲۸ | ٪ ۱/۳۲ | ٪ ۱/۳۵ | ٪ ۱/۳۵ |
| CombSort | ۰ | ٪ ۰/۱۶ | ٪ ۰/۳۲ | ٪ ۰/۳۲ | ٪ ۰/۳۴ |
| MergeSort | ۰ | ٪ ۰/۰۲ | ٪ ۰/۰۴ | ٪ ۰/۰۵ | ٪ ۲/۹ |
| TimSort | ۰ | ٪ ۱/۸۲ | ٪ ۱/۸۳ | ٪ ۳/۶۴ | ٪ ۳/۶۶ |
| CycleSort | ۰ | ٪ ۰/۰۱۴ | ٪ ۰/۰۲۸ | ٪ ۰/۰۴۲ | ٪ ۰/۰۴۲ |
| میانگین (٪) | ۰ | ٪ ۰/۶۶ | ٪ ۰/۷۱ | ٪ ۱/۰۸ | ٪ ۱/۶۶ |

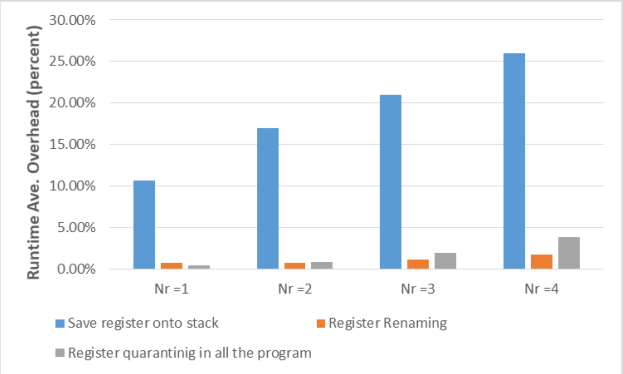
در روش آخر که در گزینه‌های مترجم موجود می‌باشد؛ در ابتدای برنامه با شناسایی ثبات‌های قربانی احتمالی، از سوییچ‌های مترجم به نحوی استفاده شد که در کل برنامه از ثبات‌های قربانی استفاده نشود. نتایج آزمایش این روش در جدول ۳ نشان داده شده است.

جدول ۳- نتایج شبیه‌سازی سربار زمان اجرا برحسب تعداد ثبات قرنطینه شده در روش «قرنطینه‌سازی ثبات در کل برنامه»

| تعداد ثبات / نام برنامه | ۰ | ۱ | ۲ | ۳ | ۴ |
|---|---|---|---|---|---|
| QuickSort | ۰ | ٪ ۱/۵ | ٪ ۲/۳ | ٪ ۳/۸۹ | ٪ ۶/۱۷ |
| CombSort | ۰ | ٪ ۰/۰۱۳ | ٪ ۰/۰۲ | ٪ ۰/۰۲ | ٪ ۰/۰۸ |
| MergeSort | ۰ | ٪ ۰/۲۶ | ٪ ۱/۸۸ | ٪ ۳/۹۲ | ٪ ۹/۴۷ |
| TimSort | ۰ | ٪ ۰/۰۰۴ | ٪ ۰/۰۶ | ٪ ۰/۹۴ | ٪ ۲/۱۵ |
| CycleSort | ۰ | ٪ ۰ | ٪ ۰/۰۰۷ | ٪ ۰/۷ | ٪ ۱/۴۲ |
| میانگین (٪) | ۰ | ٪ ۰/۳۶ | ٪ ۰/۸۵ | ٪ ۱/۸۹ | ٪ ۳/۸۶ |

نتایج جدول ۳ نشان می‌دهد که در روش «قرنطینه‌سازی ثبات در کل برنامه» با افزایش تعداد ثبات قربانی، به دلیل کاهش تعداد ثبات عملیاتی در برنامه اصلی، سربار زمان اجرا افزایش یافته است. هر چه پیچیدگی برنامه بیش‌تر باشد، با افزایش تعداد ثبات‌های قربانی در برنامه اصلی، مترجم در صورت مواجهه با کمبود ثبات‌های آزاد برای جایگزینی ثبات‌های قربانی، شیوه تولید کد اسمبلی را تغییر می‌دهد و در نتیجه آن تعداد دستورالعمل‌ها در کد اسمبلی تولید شده جدید افزایش می‌یابد و بنابراین سربار زمان اجرا نیز افزایش خواهد یافت. مقایسه نتایج سربار این روش با روش «تغییر نام ثبات در نواحی ناامن کد» نشان می‌دهد که هرچه پیچیدگی برنامه بیش‌تر باشد و تعداد ثبات‌های قربانی افزایش یابد (Nr=3,4,...)، سربار زمان اجرا در روش «تغییر نام ثبات در نواحی ناامن کد» به مراتب از روش «قرنطینه‌سازی ثبات در کل برنامه» کمتر می‌باشد. ذکر این نکته حائز اهمیت است که به ازای تعداد ثبات قربانی بیش‌تر (Nr=5,6,...) در برنامه‌های با پیچیدگی بالا، در روش «قرنطینه-سازی ثبات در کل برنامه» به دلیل اینکه تعداد ثبات‌های عملیاتی استفاده شده در برنامه زیاد هستند و با کمبود ثبات‌های آزاد برای جایگزینی ثبات‌های قربانی مواجه هستیم؛ برخلاف قرنطینه‌سازی ثبات در کل برنامه با گزینه‌های مترجم، مترجم مجبور به استفاده از ثبات‌های قربانی برای تولید کد اسمبلی می‌شود.

شکل ۸ نمودار مقایسه سه روش قرنطینه‌سازی را از نظر میانگین سربار زمان اجرا به ازای تعداد ثبات مختلف نشان می‌دهد. شواهد نشان می‌دهد که با افزایش تعداد ثبات‌های قربانی (Nr=2,3) سربار زمان اجرای روش «تغییر نام ثبات در نواحی ناامن» کد نسبت به دو روش دیگر قرنطینه‌سازی کمتر است.




شکل ۸- نمودار مقایسه میانگین سربار زمان اجرای سه روش قرنطینه‌سازی به ازای تعداد ثبات قرنطینه شده مختلف

در مقایسه بین عملکرد دو روش قرنطینه‌سازی «تغییر نام ثبات در نواحی ناامن کد» و «قرنطینه‌سازی ثبات در کل برنامه» میزان سربار بین 0.3 درصد به ازای یک ثبات قرنطینه شده تا 2.2- درصد به ازای چهار ثبات قرنطینه شده است. پیش‌بینی می‌شود به ازای تعداد ثبات قربانی بیش‌تر (Nr=4,5,...) سربار زمان اجرای روش «تغییر نام ثبات در نواحی ناامن کد» از روش «قرنطینه‌سازی ثبات در کل برنامه» کم‌تر باشد. زیرا زمان اجرای دستورات اضافه شده به کد اسمبلی برنامه، زمان اجرای دستورالعمل‌های mov، در روش«تغییر نام ثبات در نواحی ناامن کد» نسبت به زمان اجرای دستورالعمل‌های اضافه شده توسط مترجم به کد اسمبلی برنامه در روش «قرنطینه‌سازی ثبات در کل برنامه» کم‌تر است. بنابراین استفاده از روش «تغییر نام ثبات در نواحی ناامن کد» به‌منظور قرنطینه‌سازی تروجان، نسبت به دو روش دیگر بهتر می‌باشد.

## ۵- نتیجه‌گیری

در این مقاله با قرنطینه‌کردن تروجان‌های بانک ثبات ریزپردازنده، اجرای امن یک نرم‌افزار بر روی یک سخت‌افزار ناامن را فراهم کردیم. سربار زمان اجرا برای هر برنامه به پیچیدگی آن برنامه آزمون و تعداد ثبات استفاده شده در برنامه بستگی دارد. روش «تهیه نسخه پشتیبان از مقدار ثبات در پشته» با توجه به ذخیره مقدار رجیستر در پشته و بارگذاری مقدار رجیستر از پشته دارای سربار زمان اجرای زیادی

است. روش «تغییر نام ثبات در نواحی ناامن کد» با توجه به اینکه فقط انتقال و جایگزینی رجیستر به‌وسیله یک رجیستر دیگر صورت می‌گیرد؛ نسبت به دو روش قرنطینه‌سازی دیگر، دارای سربار زمان اجرای کم‌تری است و کارایی بهتری دارد. روش «قرنطینه‌سازی ثبات در کل برنامه» برای برنامه‌های آزمون با پیچیدگی پایین مناسب است و در برنامه‌های با پیچیدگی بالا به دلیل کمبود ثبات‌های آزاد امن جایگزین برای ثبات‌های قربانی، کارایی لازم را ندارد.

## ۶- مراجع


[۱] پ. طالبیان و ع. جهانیان، "قرنطینه کردن تروجان‌های بانک ثبات در میکروپروسسورهای همه‌منظوره با استفاده از برنامه‌نویسی امن،" *اولین کنفرانس بین المللی دستاوردهای نوین پژوهشی در مهندسی برق و کامپیوتر*، ۱۳۹۵.

[2] S. Bhunia, M. Abramovici, D. Agrawal, P. Bradley, M. S. Hsiao, J. Plusquellic and M. Tehranipoor, "Protection against hardware trojan attacks: Towards a comprehensive solution," *IEEE Design & Test*, vol. 30, no. 3, pp. 6-17, 2013.

[3] D. Wang, L. Wu, X. Zhang, and XJ. Wu, "A novel hardware Trojan design based on one-hot code," *IEEE 6th International Symposium on Digital Forensic and Security (ISDFS)*, pp. 1-5, 2018.

[4] M. N. I. Khan, A. De, and S. Ghosh, "RF-Trojan: Leaking Kernel Data Using Register File Trojan," *arXiv preprint arXiv:1904.07144*, 2019.

[5] J. Rajendran, V. Vedula, and R. Karri, "Detecting malicious modifications of data in third-party intellectual property cores," *52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1-6, 2015.

[6] J. Zhao, "Case Study: Discovering Hardware Trojans Based on model checking," *In Proceedings of the 8th International Conference on Communication and Network Security*, pp. 64-68, 2018.

[7] M. Tehranipoor, and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," *IEEE design & test of computers*, vol. 27, no. 1, pp. 10-25, 2010.

[8] S. Moein, S. Khan, T. A Gulliver, F. Gebali, and M. W. El-Kharashi, "An attribute based classification of hardware trojans," *IEEE Tenth International Conference on Computer Engineering & Systems (ICCES)*, pp. 351-356. 2015.

[9] X. Wang,, T. Mal-Sarkar, A. Krishna, S. Narasimhan, and S. Bhunia, "Software exploitable hardware Trojans in embedded processor," *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pp. 55-58, 2012.

[10] J. Zhang, F. Yuan, and Q. Xu, "Detrust: Defeating hardware trust verification with stealthy implicitly-triggered hardware trojans," *In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pp. 153-166, 2014.

[11] J. Zhang, F. Yuan, L. Wei, Y. Liu, and Q. Xu, "VeriTrust: Verification for hardware trust," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 7, pp. 1148-1161, 2015.

[12] A. Waksman, M. Suozzo, and S. Sethumadhavan, "FANCI: identification of stealthy malicious logic using boolean functional analysis," *In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 697-708. 2013.

[13] X. T. Ngo, V. P. Hoang, and H. Le Duc, "Hardware Trojan Threat and Its Countermeasures," *IEEE 5th NAFOSTED Conference on Information and Computer Science (NICS)*, pp. 35-40, 2018.

[14] R. S. Chakraborty, and S. Bhunia, "Security against hardware Trojan through a novel application of design obfuscation," *IEEE/ACM International Conference on Computer-Aided Design-Digest of Technical Papers*, pp. 113-116, 2009.

[15] A. Marcelli, E. Sanchez, L. Sasselli, and G. Squillero, "On the Mitigation of Hardware Trojan Attacks in Embedded Processors by Exploiting a Hardware-Based Obfuscator," *IEEE 3rd International Verification and Security Workshop (IVSW)*, pp. 31-37, 2018.

[16] M. Bilzor, T. Huffmire, C. Irvine, and T. Levin, "Evaluating security requirements in a general-purpose processor by combining assertion checkers with code coverage," *IEEE International Symposium on Hardware-Oriented Security and Trust*, pp. 49-54, 2012.

[17] N. Fern, and K. T. T. Cheng, "Evaluating Assertion Set Completeness to Expose Hardware Trojans and Verification Blindspots," *IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 402-407, 2019.

[18] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan, "Hardware Trojan attacks: threat analysis and countermeasures," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1229-1247, 2014.

[19] K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, and M. Tehranipoor, "Hardware trojans: Lessons learned after one decade of research," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 22, no. 1, pp. 1-23, 2016.

[20] D. Soubra, "How to get a handle on TrustZone for ARMv8-M software development," https://www.embedded-computing.com, March 2017.

[21] M. Ye, J. Sherman, W. Srisa-An, and S. Wei, "TZSlicer: Security-aware dynamic program slicing for hardware isolation," *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 17-24, 2018.

[22] M. Ye, M. B. Cohen, W. Srisa-an, and S. Wei, "EvoIsolator: Evolving program slices for hardware isolation based security," *International Symposium on Search Based Software Engineering*, Springer, Cham, pp. 377-382, 2018.



**فرزانه قطب‌الدینی** مدرک کارشناسی خود را در سال ۱۳۹۵ از دانشگاه شهید باهنر کرمان و مدرک کارشناسی ارشد خود را در سال ۱۳۹۹ از دانشگاه شهید بهشتی اخذ کرده‌اند. زمینه تحقیقاتی ایشان تولید کد امن بر پایه کامپایلر است.

آدرس پست الکترونیکی ایشان عبارت است از:

f.ghobaddini@sbu.ac.ir



**دکتر علی جهانیان** مدرک کارشناسی خود را در سال ۱۳۷۵ از دانشگاه تهران و مدارک کارشناسی ارشد و دکترای خود را به ترتیب در سالهای ۱۳۷۷ و ۱۳۸۶ از دانشگاه صنعتی امیرکبیر اخذ کرده‌اند. ایشان اکنون دانشیار دانشکده مهندسی و علوم کامپیوتر دانشگاه شهید بهشتی هستند. زمینه تحقیقاتی ایشان امنیت سخت افزار و طراحی سخت افزارهای امن است.

آدرس پست الکترونیکی ایشان عبارت است از:

jahanian@sbu.ac.ir


[1] Trojan Horses
[2] Third-party foundry
[3] Intellectual Property(IP)
[4] Integrated Circuit(IC)
[5] Electronic Design Automation(EDA)
[6] Programmable Logic Controller(PLC)
[7] Arithmetic Logic Unit(ALU)
[8] Privilege escalation
[9] Specification
[10] Finite State Machine(FSM)
[11] Plaintext
[12] hardware trust
[13] kernel
[14] property
[15] Obfuscation
[16] obfuscator
[17] Trigger
[18] Reconfigurable
[19] Side channel
[20] TrustZone Optimizer
[21] Register Renaming
[22] Register Lifetime

The CSI Journal on Computing Science and Information Technology

The CSI Journal on
Computing Science and
Information Technology
Vol. 17, No.2, 2020

# Quarantining hardware Trojans in general-purpose processors by compiler-based software methods

**Farzane Ghotbadini, Ali Jahanian**

Faculty of Electrical and Computer Engineering, Shahid Beheshti University, Tehran, Iran.

## Abstract

Integrated circuits are becoming increasingly vulnerable to the malicious activities and alternations due to the globalization of the semiconductor design and fabrication process. In many Industrial systems, there is some reported evidences of the system insecurity, but there is no possibility to change or replace the infected part of the system. In many conditions, there is no safe component to substitute malicious elements and/or changing the system architecture and also modifying it may have high level of risk. In this situation, quarantining the Trojan in the system software is a promising solution to deactivate the Trojan and create a secure software layer over an insecure hardware platform. In this paper Trojan quarantining in the register bank of a microprocessor, which are the most common type of Trojan in the microprocessors, will be assessed. The proposed idea has implemented in the general-purpose processor and is evaluated. According to the experimental results, the proposed methods can be efficiently utilized with acceptable runtime overhead in order to increase the microprocessor security.

**Keywords**: Processor, Quarantining hardware Trojan, Compiler