بررسی روش‌های نرم‌افزاری وارسی روند کنترلی در سیستم‌های نهفته بحرانی-ایمن

جواد یوسفی[1]، یاسر صداقت[2]
آزمایشگاه سیستم‌های نهفته توزیع‌شده اتکاپذیر (DDEmS)
دانشکده مهندسی کامپیوتر، دانشگاه فردوسی مشهد، مشهد، ایران
javad.yousefi@stu.um.ac.ir [1], y_sedaghat@um.ac.ir [2]

چکیده - امروزه استفاده از پردازنده‌های همه‌منظوره‌ی تجاری در کاربردهای بحرانی-ایمن از جمله کاربردهای فضایی، هواپیمایی و نظامی بسیار مورد توجه قرار گرفته است. با توجه به این که رخداد خرابی در سیستم‌های بحرانی-ایمن منجر به فجایع جانی، مالی و زیست‌محیطی می‌شود، جهت جلوگیری از خرابی، لازم است که اشکال‌های رخ داده در این سیستم‌ها هر چه سریع‌تر کشف شوند. با توجه به این که احتمال رخداد اشکال‌های گذرا 10 تا 30 برابر بیشتر از احتمال رخداد اشکال‌های دائمی و ادواری است و تا حدود 77 درصد اشکال‌های گذرا باعث رخداد خطاهای روند کنترلی در پردازنده‌ها می‌شوند، روش‌های وارسی روندکنترلی جهت بالا بردن قابلیت اطمینان پردازنده‌ها از اهمیت ویژه‌ای برخوردار هستند. در بین انواع روش‌های وارسی روند کنترلی، به علت هزینه‌بر بودن و انعطاف‌پذیر نبودن روش‌های سخت‌افزاری، روش‌های نرم‌افزاری بیشتر مورد توجه هستند. در این مقاله، به معرفی و بررسی خطاهای روند کنترلی، انواع آن، روش‌های نرم‌افزاری وارسی روند کنترلی جهت استفاده در سیستم‌های بحرانی-ایمن و چالش‌های موجود در این روش‌ها پرداخته شده است.

کلید واژه- سیستم‌های بحرانی-ایمن، تحمل‌پذیری اشکال، اشکال‌های گذرا، وارسی روند کنترلی.

1- مقدمه

امروزه اجرای بسیاری از وظایف بحرانی و پردازش اطلاعات مهم بر عهده سیستم‌های نهفته[1] است. منظور از سیستم نهفته، سیستمی است مبتنی بر ریزپردازنده[2] که به منظور کنترل عملیات یک سیستم بزرگتر در آن قرار گرفته است. از این سیستم‌ها در بسیاری از کاربردهای خاص مانند کاربردهای فضایی، هواپیمایی، نظامی و تجهیزات پزشکی و کاربردهای عمومی‌تر مانند اجزای کنترلی خودرو، ماشین‌آلات صنعتی، وسایل خانگی و همچنین در بسیاری از کاربردهای بی‌درنگ[3] استفاده می‌شود [1, 2].

رخداد خرابی در کاربردهای بحرانی-ایمن[4] مانند کاربردهای فضایی، هواپیمایی و نظامی مخاطرات بسیاری از قبیل خطرات جانی، مالی، زیست-محیطی و یا از دست رفتن حجم وسیعی از اطلاعات را در پی دارد. پس قابلیت اطمینان[5] یک نیاز اساسی در این سیستم‌ها می‌باشد [3]. جهت طراحی یک سیستم با قابلیت اطمینان بالا دو رویکرد مختلف وجود دارد، در برخی موارد از پردازنده‌های خاص-منظوره که جهت کاربردهای کامپیوتری اتکاپذیری[6] طراحی و ساخته شده‌اند، استفاده می‌شود؛ اما با توجه به قیمت بالا و طولانی شدن مدت زمان ارائه محصول به بازار معمولاً استفاده از این پردازنده‌ها در سیستم‌های نهفته، توجیه‌پذیر نیست و لذا امروزه استفاده از پردازنده‌های عام-منظوره تجاری[7] (COTS) بسیار مورد توجه قرار گرفته است [4]. با توجه به اینکه سیستم‌های بحرانی-ایمن در محیط‌های عملیاتی پرخطر فعالیت می‌کنند و از این نظر مستعد اشکال می‌باشند، افزودن قابلیت‌های کشف خطا به پردازنده‌های عام-منظوره تجاری جهت استفاده از این پردازنده‌ها در کاربردهای بحرانی-ایمن، بسیار حائز اهمیت می‌باشد [3].

اشکال‌ها به لحاظ طول عمر در سه دسته اشکال‌های دائمی[8]، گذرا[9] و ادواری[10] قرار می‌گیرند. با توجه به این که احتمال رخداد اشکال‌های گذرا 10 تا 30 برابر بیشتر از احتمال رخداد اشکال‌های دائمی و ادواری است، جهت بالا بردن قابلیت اطمینان سیستم‌ها، اشکال‌های گذرا بیشتر از اشکال‌های دائمی و

[1] Embedded Systems
[2] Microprocessor
[3] Real-Time
[4] Safety-Critical
[5] Reliability
[6] Dependability
[7] Commercial Off The Shelf
[8] Permanent
[9] Transient
[10] Intermittent

ادواری مدنظر قرار دارند. اشکال‌های گذرا، اشکال‌هایی هستند که به صورت موقت و لحظه‌ای در سیستم ظاهر شده و به واسطه تاثیرات محیطی مانند تداخل‌های الکترومغناطیس و ذرات آلفا پدید می‌آیند، رخداد این نوع اشکال‌ها در برخی اجزای سخت‌افزاری مانند شمارنده برنامه[11]، مدارهای آدرس و عناصر حافظه می‌تواند باعث رخداد خطاهای روند کنترلی [12] (CFE) شود [5, 6]. طبق گزارش‌های ارائه شده، تا حدود 77 درصد اشکال‌های گذرا باعث رخداد خطاهای روند کنترلی شده و بقیه به خطاهای داده[13] تبدیل می‌شوند [7]. خطاهای روند کنترلی، به خطاهایی گفته می‌شود که باعث نقض رفتار زمان اجرای برنامه شوند یا به بیان ساده‌تر خطایی است که باعث تغییر در روند اجرای برنامه شود و دستورات بر خلاف آنچه که پیش‌بینی شده است، اجرا شوند [8]. به همین دلیل، روش‌های وارسی روند کنترلی، جهت کشف خطاهای روند کنترلی از اهمیت ویژه‌ای برخوردار است. در این مقاله پس از معرفی دقیق این نوع خطاها و نحوه مدل کردن آن‌ها، روش‌های کشف خطای روند کنترلی پیشین و چالش‌های موجود در این روش‌ها، بررسی خواهند شد.

2- وارسی روند کنترلی

هر اشکال در پردازنده می‌تواند یا در حافظه سیستم و مدارهای آدرس و یا در ثبات‌های داخلی پردازنده، مانند اشاره‌گرهای پشته و دستورالعمل خود را نمایان کند و باعث تغییر روند کنترلی برنامه شود [8].

جهت تشخیص انواع خطاهای روند کنترلی، روش‌ها و تکنیک‌های مختلفی از دهه 1980 تاکنون ارائه شده است که در سه دسته کلی روش‌های سخت‌افزاری، نرم‌افزاری و ترکیب سخت‌افزار و نرم‌افزار قرار می‌گیرند [7]. روش‌های سخت‌افزاری وارسی روند کنترلی، با به‌کارگیری سخت‌افزار اضافی عمل کشف خطاهای روند کنترلی را انجام می‌دهند. روش‌های نرم‌افزاری با افزودن کد به برنامه و بررسی صحت اجرای روند برنامه در نقاط خاص همراهند. در روش‌های ترکیبی نیز از یک سخت‌افزار کوچک یا از امکانات سخت‌افزاری پردازنده‌ها به همراه مختصری افزونگی نرم‌افزاری جهت انجام وارسی روند کنترلی استفاده می‌شود [4, 8].

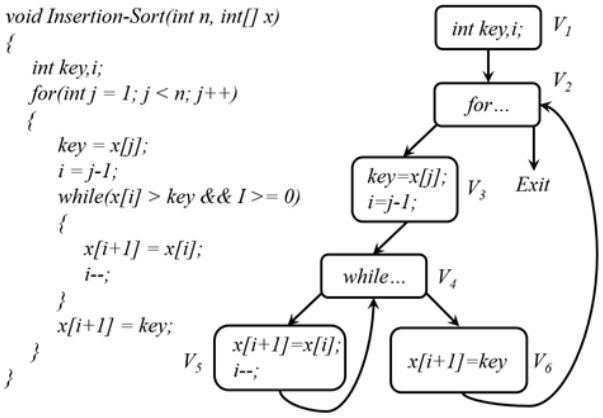
روش‌های سخت‌افزاری وارسی روند کنترلی در مقایسه با روش‌های نرم‌افزاری، پوشش کشف خطای[14] بهتر، تأخیر تشخیص خطای[15] کمتر و هزینه بیشتری دارند. به لحاظ انعطاف‌پذیر بودن، روش‌های نرم‌افزاری انعطاف‌پذیرترند و قابلیت پیاده‌سازی روی سیستم‌های مختلف را دارند و به سخت‌افزار وابسته نمی‌باشند، اما روش‌های سخت‌افزاری با توجه به وابستگی‌شان به سخت‌افزار، نیازمند پیاده‌سازی مجدد در سیستم‌های مختلف هستند ودر نتیجه انعطاف‌پذیری کمتری دارند [7, 9, 10]، همچنین روش‌های نرم‌افزاری در مقایسه با روش‌های سخت‌افزاری پتانسیل بالایی در تحمیل سربار کارایی[16] (افزایش زمان اجرا) دارند [11].

2-1- نظارت بر امضا

اغلب روش‌های وارسی روند کنترلی، مبتنی بر روش نظارت بر امضا[17] هستند [3, 5-20]. در روش نظارت بر امضا، برنامه به تعدادی بلوک پایه[18] تقسیم شده، هر بلوک پایه محدوده‌ای از کد را شامل می‌شود که عاری از پرش باشد (به جز دستور اول بلوک که می‌تواند مقصد پرش باشد و آخرین دستور بلوک که می‌تواند مبدأ پرش باشد). گراف روند کنترلی[19] (CFG) برنامه بر اساس ارتباط‌های بین بلوک‌های پایه و انشعاب‌های برنامه تشکیل می‌شود به طوری که رئوس[20] این گراف را بلوک‌های پایه و یال‌های[21] آن را انشعاب‌ها و پرش‌های برنامه تشکیل می‌دهند [7]؛ به عنوان مثال در شکل 1 نحوه تشکیل گراف روند کنترلی برنامه و بلوک‌های پایه برای تابع مرتب‌سازی درجی، نمایش داده شده است.

[11] Program Counter
[12] Control Flow Error
[13] Data Error

[14] Error Detection Coverage
[15] Error Detection Latency
[16] Performance Overhead
[17] Signature Monitoring
[18] Basic Block
[19] Control Flow Graph
[20] Vertex
[21] Edge

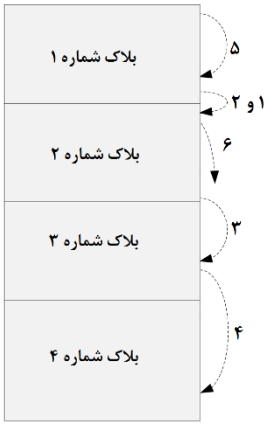


شکل 1: ساخت گراف روند کنترلی از روی کد برنامه

پرش‌های غیرمجاز داخل و بین بلـوک‌هـای پایـه، منجـر بـه شش نوع خطای روند کنترلی زیر می‌شوند:

1- پرش غیرمجاز از انتهای یک بلوک پایه به ابتـدای یـک بلوک پایه دیگر
2- پرش مجاز ولی غیرصحیح از انتهای یک بلوک پایـه بـه ابتدای یک بلوک پایه دیگر
3- پرش از انتهای یک بلوک پایه به میانه یک بلـوک پایـه دیگر
4- پرش از میانه یک بلوک پایه به هر نقطه از یـک بلـوک پایه دیگر
5- پرش از داخل یک بلوک پایه به یک نقطه داخل همـان بلوک پایه
6- پرش از هر نقطه داخل یک بلوک پایه بـه مکـانی بـین بلوک‌های پایه (مانند پرش غیرمجاز از یک بلـوک پایـه به یک فضای استفاده نشده از حافظه)

به خطاهایی مانند خطاهـای نـوع 1، 2، 3 و 4 خطـای بـین بلوکی[22] و به خطاهایی مانند خطای نوع 5، خطای درون بلوکی[23] گفته مـی‌شـود [7]. در شـکل 2 انـواع خطاهـای رونـد کنترلـی ممکن نمایش داده شده است.



شکل 2: خطاهای روند کنترلی ممکن بین بلوک‌های پایه

اکثر روش‌های وارسی رونـد کنترلـی مبتنـی بـر نظـارت بـر امضا[24] پیاده‌سازی می‌شوند، بنابراین در زمان کامپایل برنامه، پس از تشکیل گراف روند کنترلی، به هـر گـره در گـراف، یـک امضـا تخصیص داده می‌شود و این امضا در محلی ذخیـره مـی‌شـود. در طول اجرای برنامه و با گذر از هر بلوک پایه، امضایی ساخته شده و با امضای زمان کامپایل آن گره مقایسه می‌شود، در صورتی کـه امضاها یکسان نباشند، رخداد خطای روند کنترلی تشـخیص داده می‌شود. در روش‌های سخت‌افزاری وظیفه سـاخت امضـای زمـان اجرا و مقایسه آن با امضای زمان کامپایل بر عهده یک سخت‌افزار اضـافی ماننـد پردازنـده مراقـب اسـت [7]، امـا در روش‌هـای نرم‌افزاری این کار به صورت کاملاً نرم‌افزاری و با افزودن کدهایی به برنامه انجام می‌شود و نیاز بـه هـیچ‌گونـه سـخت‌افـزار اضـافی نیست [11].

امضاها یا به صورت اختیاری تخصیص داده می‌شوند کـه بـه امضاهای تخصیص‌یافته[25] معروف‌اند و یا از کد دودویی و یا آدرس دستورالعمل‌ها مشتق می‌شوند که امضای مشتق‌شـده یـا امضـای استنتاج‌شده[26] نامیده مـی‌شـوند. بـه عنـوان مثـال در روش‌هـای SIC[27] و CFCSS[28] از امضای تخصـیص‌یافتـه اسـتفاده شـده و در روش‌هـای PSA[29]، SCFC[30]، SIS[31]، ASIS[32]، CSM[33]، OSLC[34]

[22] Inter-Block
[23] Intra-Block

[24] Signature Monitoring
[25] Assigned Signature
[26] Derived Signature
[27] Structural Integrity Checking
[28] Control-Flow Checking by Software Signatures
[29] Path Signature Analysis
[30] Software-based Control Flow Checking
[31] Signatured Instruction Streams
[32] Asynchronous SIS
[33] Continuous Signature Monitoring
[34] On-line Signature Learning and Checking

و ISC[35] از امضای مشتق‌شده استفاده شده است [7].

در روش‌های نرم‌افزاری، وظیفه ساخت امضای زمان اجرا و مقایسه آن با امضای زمان کامپایل بر عهده قطعه کدهایی است که به برنامه افزوده می‌شوند. نوع و تعداد کدهایی که به برنامه افزوده می‌شود در روش‌های مختلف متفاوت است و هر روش با توجه به کدهای افزونه می‌تواند تشخیص هر کدام از انواع خطاهای روند کنترلی را پوشش دهد [11, 14]. در ادامه جهت آشنایی با نحوه تشخیص خطا در روش‌های وارسی روند کنترلی، چند نمونه از این روش‌ها که جهت کاربردهای بحرانی-ایمن مناسب‌تر هستند، بررسی شده‌اند. این روش‌ها، هر کدام از شیوه متفاوتی جهت تشخیص خطاهای روند کنترلی استفاده کرده‌اند.

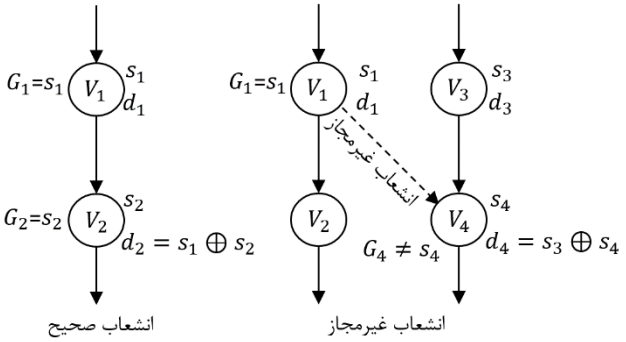
2-2- روش CFCSS

روش CFCSS از روش‌های نظارت بر امضا جهت کشف خطاهای روند کنترلی بین بلوکی است، در این روش با به‌کارگیری یک ثبات اختصاصی GSR[36] که دربردارنده امضای G است، روند کنترلی برنامه را وارسی می‌نماید. G امضای زمان اجرای گره‌ای است که شامل دستورالعملی که اکنون در حال اجرا شدن هست، می‌باشد. در این روش برنامه به تعدادی بلوک پایه تقسیم می‌شود، هر بلوک پایه با v_i نمایش داده می‌شود و در زمان کامپایل به هر بلوک پایه یک امضای منحصر به فرد S_i تخصیص داده می‌شود. در حالت اجرای صحیح و بدون خطای برنامه، باید G_i برابر S_i باشد و در غیر این صورت خطای روند کنترلی رخ داده است. وقتی کنترل برنامه از یک بلوک پایه به بلوک پایه دیگر منتقل می‌شود مقدار G توسط تابع امضای f در گره مقصد انشعاب به‌روز می‌شود. تابع f جهت به‌روزرسانی G از دو مقدار امضای گره قبلی (گره منبع انشعاب) و امضای گره کنونی (گره مقصد انشعاب) استفاده می‌کند. از این دو مقدار به این دلیل استفاده می‌شود که این دو مقدار برای هر انشعاب، منحصر به فرد هستند. تابع f با رابطه‌های 1 و 2 تعریف می‌شود:

$$f \equiv f(G, d_i) = G \oplus d_i \qquad (1)$$

$$d_d \equiv s_s \oplus s_d \qquad (2)$$

که d_d اختلاف امضای دو بلوک پایه مبدأ و مقصد انشعاب است. قبل از اینکه انشعاب از s به d اجرا شود، G حاوی مقدار G_s است و پس از انشعاب مقدار G با مقدار $f(G_s , d_d)$ به‌روز می‌شود. در صورتی که مقدار به‌روزشده G برابر مقدار امضای گره مقصد انشعاب (s_d) باشد، خطایی رخ نداده است، و در غیر این صورت خطای روند کنترلی رخ داده است. در شکل 3 دو نوع انشعاب صحیح و غیرمجاز و نحوه تشخیص انشعاب غیرمجاز در این روش نمایش داده شده است.



شکل 3: تشخیص انشعاب غیرمجاز در روش CFCSS [11]

این روش همچنین قادر به تشخیص خطا در پرش به گره‌هایی با چند مقصد (Branch-Fan-In) می‌باشد [11].

3-2- روش SCFC

SCFC یک روش نرم‌افزاری محض، مبتنی بر نظارت بر امضا جهت وارسی روند کنترلی برنامه است. در این روش هم مانند دیگر روش‌های نظارت بر امضا، به هر بلوک پایه یک امضای S_i تخصیص داده می‌شود که این امضا نمایانگر گره‌های مقصد گره i می‌باشد، در واقع امضای هر بلوک پایه از روی گراف روند کنترلی برنامه مشتق می‌شود. روش SCFC در هر بلوک پایه چهار دستورالعمل قرار می‌دهد. اولین دستورالعمل، دستور کنترل[37] است که وظیفه بررسی جریان ورودی در ابتدای هر بلوک پایه را بر عهده دارد، هدف از قرار دادن این دستورالعمل در بلوک‌های پایه، کشف پرش‌های غیرمجاز به ابتدای آنهاست. دومین دستوری که به هر بلوک پایه افزوده می‌شود، دستوری با نام وارسی[38] است که وظیفه کشف خطای روند کنترلی، زمانی که پرش از یک بلوک پایه به میانه بلوک پایه دیگر رخ دهد، را بر عهده دارد. دستور سوم، دستور به‌روزرسانی[39] است که وظیفه‌ی به‌روزرسانی امضای زمان اجرا را برعهده دارد. این دستور در میانه بلوک پایه قرار می‌گیرد تا خطاهای درون‌بلوکی را کشف نماید.

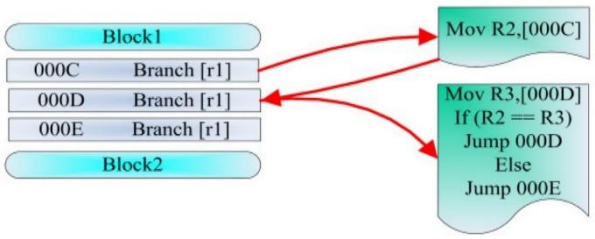
[35] Implicit Signature Checking
[36] Global Signature-Registers

[37] Control Instruction
[38] Check Instruction
[39] Update Instruction

آخرین دستوری که اضافه می‌شود، دستور خروج است که در انتهای بلوک پایه قرار می‌گیرد و وظیفه به‌روزرسانی متغیر ID را بر عهده دارد که نگهدارنده مقدار امضای بلوک کنونی است. در این روش با توجه به این که دو دستورالعمل در میانه برنامه قرار گرفته‌اند، مقداری از خطاهای روند کنترلی درون‌بلوکی هم تشخیص داده می‌شوند [7].

2-4- روش BTMR

در این روش هر دستور انشعاب سه نسخه نویسی می‌شود و یک روال نرم‌افزاری احضار می‌شود تا صحت دستور انشعاب را بررسی کند. در زمان اجرای یک برنامه، زمانی که یک دستور انشعاب اجرا می‌شود، با دومین دستور انشعاب افزونه در روال مقایسه می‌شود. اگر عدم تطابقی مشاهده شود، سومین دستور انشعاب افزونه، به عنوان دستور انشعاب عاری از اشکال در نظر گرفته می‌شود. در غیر این صورت هیچ خطائی رخ نداده است. در شکل 4 نحوه کار این روش مشاهده می‌شود [4].



شکل 4: نحوه کار روش BTMR [4]

انواع اشکالی که توسط این روش کشف می‌شوند، بیشتر محدود به دستورات انشعاب می‌باشند. تغییر کد عملیاتی[40] یک دستور انشعاب به یک دستور غیرانشعاب، تغییر در آدرس مقصد یکی از کپی‌های دستور انشعاب، تبدیل کد عملیاتی یکی از دستورات انشعاب به کد عملیاتی دستور انشعاب دیگر از جمله نمونه اشکال‌هایی هستند که در این روش به طور کامل کشف می‌شوند. به منظور کشف تغییر دستورات غیرانشعابی به انشعابی، در این روش دستوراتی که احتمال تبدیل آن‌ها به دستورات انشعابی بیشتر است دونسخه‌نویسی می‌شوند. روال نرم‌افزاری مورد نظر، پس از اجرای اولین نسخه فراخوانی می‌شود و در صورتی که اولین دستور از نوع انشعابی باشد، خطا رخ داده است و روال به دومین نسخه اعتماد می‌کند و آن را اجرا می‌کند [4].

[40] Op-Code

2-5- وارسی روند کنترلی در پردازنده‌های چندهسته

در [8] یک روش نرم‌افزاری جهت کشف خطاهای روند کنترلی در سیستم‌های با معماری چندهسته‌ای[41] ارائه شده است. در واقع این مقاله با این رویکرد نوشته شده است که با ایجاد تغییراتی در ساختار روش‌های قبلی وارسی روند کنترلی نرم‌افزاری و استفاده از امکانات پردازنده‌های چندهسته‌ای، از سربار روش‌های قبلی کاسته و این روش‌ها را مناسب پردازنده‌های چندهسته‌ای نماید. همچنین در این روش سعی شده با کشف خطاهای محتمل‌تر، پوشش کشف خطا را بالا برده شود. در این مقاله، ابتدا میزان رخداد انواع خطاهای روند کنترلی در چند محک[42] مختلف محاسبه شده و طبق محاسبات صورت‌گرفته حدود 74٪ خطاها مربوط به پرش غیرمجاز از یک بلوک پایه به فضای خارج از برنامه است، که این نوع خطاها اکثراً توسط سیستم عامل کشف می‌شود. خطاهای دیگری که میزان رخ دادن آن‌ها نسبت به بقیه بیشتر است، به ترتیب عبارتند از: انشعاب غیرمجاز از یک بلوک پایه به یک بلوک پایه دیگر و انشعاب غیرمجاز داخل یک بلوک پایه که به ترتیب میزان رخ دادن آن‌ها 11٪ و 10٪ است. جهت تقلیل اثرات منفی افزودن کد به برنامه و سربار کارایی، این روش سعی در مدل کردن روش‌های سخت‌افزاری کرده است. در روش‌های سخت‌افزاری معمولاً یک سخت‌افزار خارجی مانند پردازنده مراقب به طور موازی با برنامه اصلی، رفتار آن را وارسی می‌نماید. جهت مدل کردن روش‌های سخت‌افزاری، در روش ارائه شده در این مقاله، از یک نخ مراقب[43] استفاده می‌نماید که وظیفه وارسی امضاها را در زمان اجرای نخ‌های اصلی بر عهده دارد. با توجه به این‌که موازی‌سازی در برنامه‌ها به طور خیلی گسترده پیشرفت نکرده و معمولاً در پردازنده‌های چندهسته، هسته‌های خالی داریم، نخ مراقب به صورت موازی با نخ‌های اصلی در یک هسته بیکار فعالیت می‌کند. با افزودن نخ مراقب به برنامه، ارتباط‌های بین نخ‌هابه دلیل ارتباط بین نخ مراقب و نخ‌های اصلی، زیاد خواهد شد، جهت کاهش این ارتباط‌ها، یک آرایه برای ذخیره کردن امضاها در نظر گرفته شده و تا زمانی که همه مقادیر آرایه به‌روز نشود، نخ مراقب نباید امضاها را وارسی نماید، پس در نتیجه، ارتباط‌های بین نخ مراقب و نخ‌های اصلی کاهش می‌یابد و از

[41] Multicore Architecture
[42] Benchmark
[43] Watchdog Thread

طرفی تأخیر تشخیص خطا افزایش می‌یابد. در این روش جهت کشف خطاهای درون‌بلوکی دستورات افزونه را به جای این‌که در ابتدا یا پایان بلوک پایه قرار دهد در میانه بلوک‌های پایه قرار می‌دهد، با انجام این کار برخی خطاهای درون‌بلوکی تشخیص داده می‌شوند و در عوض برخی خطاهای بین بلوکی کشف نمی‌شوند. در گراف روند کنترلی برنامه‌های چندنخی باید ارتباط‌ها و وابستگی‌های بین نخ‌های یک برنامه مدنظر قرار گیرند تا تشخیص خطای روند کنترلی به صورت موثرتری انجام شود. در این روش به ازای هر نخ یک گراف روند کنترلی ایجاد شده و تعدادی یال که مربوط به ارتباط نخ‌ها با یکدیگر است در این گراف موجود است [8, 21].

3- چالش‌های روش‌های موجود

همانطور که پیش از این گفته شد، روش‌های نرم‌افزاری وارسی روند کنترلی، مبتنی بر افزونگی کد هستند و در نتیجه نسبت به روش‌های سخت‌افزاری سربار کارایی بالایی دارند. در برنامه‌ها به طور میانگین در هر بلوک پایه حدود سه تا پنج دستورالعمل وجود دارد که افزودن تعداد زیاد دستورات افزونه موجب بالا رفتن بیش از حد سربار کارایی و حافظه شده [7]، به طوری که در این روش‌ها سربار کارایی بین 40٪ تا 200٪ افزایش می‌یابد، با توجه به اینکه سربار کارایی باعث اجرای کندتر برنامه شده و در اغلب سیستم‌های بحرانی-ایمن مسأله زمان اجرا، مسأله مهمی به شمار می‌آید، در نتیجه میزان سربار کارایی از اهمیت ویژه‌ای برخوردار خواهد بود.

مشکل مهم دیگری که همه روش‌های وارسی روند کنترلی با آن مواجه‌اند، عدم کشف همه انواع خطاهای روند کنترلی است. در اکثر روش‌های ارائه شده خطاهای درون‌بلوکی کشف نمی‌شوند و در برخی روش‌ها مانند SCFC یا روش ارائه شده در [8] که به کشف این‌گونه خطاها می‌پردازند، میزان کمی از این‌گونه خطاها کشف شده و بقیه کشف نشده باقی می‌مانند [7] و یا اینکه کشف این خطا باعث عدم کشف خطاهای نوع دیگر می‌شود [8]. همچنین در هیچ‌کدام از روش‌هایی که تاکنون ارائه شده به کشف خطاهای روند کنترلی نوع 2 که بر اثر پرش مجاز ولی غیرصحیح از انتهای یک بلوک پایه به ابتدای یک بلوک پایه دیگر به وجود می‌آیند، پرداخته نشده است.

تأخیر تشخیص خطا، یعنی فاصله بین رخداد خطا و کشف آن خطا توسط روش وارسی روند کنترلی، در ارزیابی روش‌ها از اهمیت ویژه‌ای برخوردار است، با توجه به اینکه، رخداد خرابی در سیستم‌های بحرانی-ایمن موجب فجایع مختلف می‌شود، باید در این‌گونه سیستم‌ها پس از رخداد خطا، کشف و بازیابی از خطا هرچه سریع‌تر انجام پذیرد و از رخداد خرابی جلوگیری شود. از این جهت تأخیر تشخیص خطا، به عنوان یک عامل مهم در این روش‌ها معرفی شده و در روش‌های ارائه شده تاکنون بهبود این چالش خیلی مدنظر قرار نگرفته است.

از بین روش‌های نرم‌افزاری وارسی روند کنترلی که تاکنون ارائه شده است، چند روش که جهت استفاده در کاربردهای بحرانی-ایمن مناسب‌تر هستند انتخاب شده و در جدول 1 از

جدول 1: مقایسه برخی روش‌های نرم‌افزاری وارسی روند کنترلی ارائه شده

نام	روش وارسی	پوشش کشف خطا	سربار کارایی	سربار حافظه	محیط اجرا	کشف خطای درون‌بلوکی
GTCFC [16]	ساخت درخت روند کنترلی و استفاده از ویژگی‌های درخت	٪91.5	52.7 ٪	-	سیستم نهفته ماهواره ZDPS-1A	خیر
ACCE [22]	دونسخه‌نویسی تمام دستورات	٪88	٪20	٪120	Intel	خیر
CFCSS [11]	استفاده از امضاهای روند اجرا	٪84.2	٪46	٪47	MIPS	خیر
SCFC [7]	استفاده از امضاهای روند اجرا در میانه بلوک‌های پایه	٪95.1	٪41	٪43	PhyCORE-MPC555	آری
روش [9]	اجرای همزمان کد در یک هسته دیگر پردازنده	٪95	٪45	٪73	MIPS	خیر
BTMR [4]	سه نسخه نویسی دستورات انشعاب رای‌گیری اکثریت در وارسی نقاط پرش	٪96	٪10	٪29	LEON2	خیر
روش [8]	استفاده از امضاهای روند اجرا در میانه بلوک‌های پایه در پردازنده چندهسته‌ای	٪90.8	٪18	٪56	Intel	آری

نقطه نظر میزان پوشش کشف خطا، سربار کارایی، سربار حافظه، امکان کشف خطاهای روند کنترلی درون‌بلوکی و محیطی که برنامه در آن اجرا شده است، با یکدیگر مقایسه شده‌اند. یک روش مناسب وارسی روند کنترلی، باید قادر به تشخیص همه انواع خطاهای روند کنترلی باشد و افزایش در میزان تشخیص خطا نباید موجب افزایش بیش از اندازه سربارهای کارایی و حافظه شود. برای انتخاب یک روش مناسب جهت استفاده در سیستم‌های بحرانی-ایمن باید با توجه به ویژگی‌های سیستم موردنظر و ایجاد مصالحه بین میزان پوشش کشف خطا و سربارهای کارایی و حافظه روشی که برای کاربرد موردنظر مناسب‌تر است، را انتخاب نمود. به عنوان مثال، ممکن است در یک کاربرد خاص، میزان پوشش کشف خطا نسبت به سربار کارایی از اهمیت بیشتری برخوردار باشد، در نتیجه باید روشی انتخاب شود که بیشترین پوشش کشف خطا را داراست.

4- نتیجه‌گیری

در این مقاله، به بررسی روش‌های نرم‌افزاری وارسی روند کنترلی از روش‌های رفتاری کشف خطای گذرا پرداخته شد. همانطورکه گفته شد، روش‌های نرم‌افزاری پتانسیل بالایی در تحمیل سربار کارایی (افزایش زمان اجرا) دارند؛ این تنزل کارایی ناشی از افزایش زمان اجرا باعث خواهد شد که تلاش‌ها و هزینه-های سنگین طراحی پردازنده‌های مدرن با هدف کاهش زمان اجرا به هدر برود، همچنین افزایش زمان اجرا در کاربردهای بی‌درنگ و بحرانی-ایمن منجر به عواقب سنگینی خواهد شد. لذا با بررسی دقیق‌تر محدودیت‌های پردازنده‌های جدید، می‌توان به این نتیجه دست یافت که امروزه نیاز دستیابی به روش‌ها و عناصر پردازشی با کارایی بالا، به شدت با محدودیت‌های ناشی از تحمیل هزینه‌های مختلف گره خورده است. بنابراین ارائه روشی به عنوان جایگزینی مناسب برای روش‌های سنتی و با درنظر گرفتن محدودیت‌های موجود، جهت افزایش قابلیت اطمینان سیستم‌های امروزی به عنوان یکی از مهم‌ترین دغدغه‌های طراحان و سازندگان سیستم‌های بحرانی-ایمن به حساب می‌آید.

5- مراجع

[1] Hennessy J.L. and D. A. Patterson, Computer architecture: *a quantitative approach*: Elsevier, 2007.

[2] P. Pop, "Analysis and Synthesis of Communication-Intensive Heterogeneous Real-Time Systems," PhD Dissertation, Department of Computer and Information Science, Linköping, Sweden, 2003.

[3] M. A. Schuette and J. P. Shen, "Processor control flow monitoring using signatured instruction streams," *IEEE Trans. on Computers*, Vol. 100, pp. 264-276, 1987.

[4] N.F. Ghalaty, M. Fazeli, H.I. Rad, and S.G. Miremadi, "Software-based control flow error detection and correction using branch triplication," *Proc. of 17th IEEE International On-Line Testing Symposium*, pp.214-217, 13-15 July 2011.

[5] Y. Sedaghat, S. G. Miremadi, and M. Fazeli, "A software-based error detection technique using encoded signatures," Proc. of *21st IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'06),* pp. 389-400, 4-6 October 2006.

[6] T. Boroomandnezhad and M. A. Azgomi, "An efficient control-flow checking technique for the detection of soft-errors in embedded software," *Journal of Computers & Electrical Engineering, Elsevier*, Vol. 39, Issue 4, pp. 1320-1332, 2013.

[7] S. Asghari, H. Taheri, H. Pedram and O. Kaynak, "Software-based Control Flow Checking against transient faults in industrial environments," *IEEE Trans. on Industrial Informatics*, Vol. 10, pp. 481-490, 2014.

[8] M. Maghsoudloo, H. R. Zarandi, and N. Khoshavi, "An efficient adaptive software-implemented technique to detect control-flow errors in multi-core architectures," *Journal of Microelectronics Reliability*, Vol. 52, pp. 2812-2828, 2012.

[9] N. Khoshavi, H. R. Zarandi, and M. Maghsoudloo, "Control-flow error detection using combining basic and program-level checking in commodity multi-core architectures," *Proc. of 6th IEEE International Symposium on Industrial Embedded Systems (SIES'11),* pp. 103-106, 15-17 June 2011.

[10] S.A. Asghari, A. Abdi, O. Kaynak, H. Taheri, and H. Pedram, "An effective control flow checking method for multitask processing in harsh environments," *World Scientific Journal of Circuits, Systems, and Computers*, Vol. 22, Issue 8, September 2013.

[11] N. Oh, P.P. Shirvani, and E. J. McCluskey, "Control-flow checking by software signatures," *IEEE Transactions on Reliability*, Vol. 51, pp. 111-122, 2002.

[12] N. Farazmand, M. Fazeli, and S. G. Miremadi, "FEDC: Control Flow Error Detection and Correction for embedded systems without program interruption," *Proc. of 3rd International Conference on Availability, Reliability and Security (ARES 08)*, pp. 33-38, 4-7 March 2008.

[13] A. Benso, S. Di Carlo, G. Di Natale, P. Prinetto, and L. Tagliaferri, "Control-flow checking via regular expressions," *Proc. of 10th Asian Test Symposium (ATS'01),* pp. 299-303, 19-21 November 2001.

[14] M. Jafari-Nodoushan, S. G. Miremadi, and A. Ejlali, "Control-flow checking using branch instructions," *Proc. of IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (EUC'08)*, pp. 66-72.17-20 December 2008.

[15] A. Li and B. Hong, "Software implemented transient fault detection in space computer," *Aerospace science and technology*, Vol. 11, pp. 245-252, 2007.

[16] M. Yang, H. Wang, Y. Zheng, and Z. Jin, "Graph-tree-based software control flow checking for COTS processors on pico-satellites," *Chinese Journal of Aeronautics*, vol. 26, pp. 413-422, 2013.

[17] N. J. Warter and W. Hwu, "A software based approach to achieving optimal performance for signature control flow checking," *Proc. of 20th International Symposium Fault-Tolerant Computing (FTCS-20)*, pp. 442-449, 26-28 June 1990.

[18] Y. Wu, G. Gu, S. Huang ,and J. Ni, "Control flow checking algorithm using soft-basedintra-/inter-block assigned-signature," *Proc. of 2nd International Multi-Symposiums on Computer and Computational Sciences (IMSCCS'07)*, pp. 412-415, 13-15 August 2007.

[19] Y.-X. Wu, G.-C. Gu, and K.-H .Wang, "An improved cfcss control flow checking algorithm," *Proc. of IEEE International Workshop on Anti-counterfeiting, Security, Identification (ASID'07),* pp. 284-287, 16-18 April 2007.

[20] A. Mahmood and E. J. McCluskey, "Concurrent error detection using watchdog processors-a survey," *IEEE Transactions on Computers*, Vol. 37, pp. 160-174, 1988.

[21] M. Maghsoudloo, H.-R. Zarandi, S.P. Mozafari, and N. Khoshavi, "Soft error detection technique in multi-threaded architectures

using control-flow monitoring," *Proc. of 14th Euromicro Conference on Digital System Design (DSD'11)*, pp.789-792, 31 August-2 September 2011.
[22] R. Vemu, S. Gurumurthy, and J.A. Abraham, "ACCE: Automatic correction of control-flow errors," *IEEE International Test Conference*, pp.1-10, 21-26 Oct. 2007.