نشریه علمی «پدافند الکترونیکی و سایبری»
سال هفتم، شماره ۱، بهار ۱۳۹۸؛ ص ۱۳۲-۱۲۵

ارایه یک حمله زمانی جدید بر روی پردازنده ARM و پیاده‌سازی عملی آن بر روی برد Raspberrypi3

وحید معراجی[۱]، هادی سلیمانی[۲*]

۱- کارشناس ارشد و ۲- استادیار، گروه امنیت شبکه و رمزنگاری، پژوهشکده فضای مجازی، دانشگاه شهیدبهشتی، تهران

(دریافت: ۹۷/۶/۱۹، پذیرش: ۹۸/۳/۲۸)

چکیده

دسته مهمی از حملات کانال جانبی از این حقیقت استفاده می‌کنند که حافظه نهان منجر به تغییرات زمانی در اجرای الگوریتم‌های رمزنگاری و در نتیجه نشت اطلاعات می‌شود. علی‌رغم آن‌که حملات کانال جانبی مبتنی‌بر وجود حافظه نهان به علت کارایی زیاد، از مهمترین و پرکاربردین حملات کانال جانبی محسوب می‌شوند، نسبت به سایر حملات کانال جانبی کمتر مورد مطالعه قرار گرفته‌اند. بر همین اساس تحقیقات گسترده‌ای توسط جامعه رمزنگاری در حوزه حملات کانال جانبی مبتنی‌بر وجود حافظه نهان صورت گرفته است. تمرکز تحقیقات صورت گرفته عمدتا بررسی امنیت اجرای الگوریتم‌های رمزنگاری توسط پردازنده‌های اینتل و پنتیوم بوده است که با توجه به ساختار متفاوت حافظه نهان در پردازنده‌های مختلف، قابل تعمیم به پرازنده‌های پرکاربرد دیگر، نظیر ARM نیست. در پاسخ به این چالش، تحقیقات جدید به اعمال حملات کانال جانبی مبتنی‌بر حافظه نهان بر روی پردازنده‌های انواع تلفن‌های همراه و بردهای کاربردی دیگر از جمله پردازنده‌های ARM، معطوف شده است. متفاوت بودن ساختار حافظه نهان و عدم پشتیبانی از بعضی دستورات مورد نیاز جهت اجرای حمله‌های مبتنی‌بر حافظه نهان، اجرای این دسته از حملات را بر روی پردازنده‌های ARM دشوار کرده است. در این مقاله ما برای اولین بار به بررسی اعمال حمله کانال‌جانبی زمانی مبتنی‌بر حافظه نهان با استفاده از رویداد تصادم بر روی یکی از پردازنده‌های ARM می‌پردازیم. حمله ارایه‌شده در این مقاله، از نوع حملات کاربردی time-driven است که در آن مهاجم تنها باید زمان اجرای عملیات رمزنگاری را اندازه‌گیری کند و برخلاف حملات access-driven، مهاجم نیازی به دسترسی به حافظه نهان قربانی را ندارد. همچنین حمله ارائه‌شده را با استفاده از یک برد پراستفاده در صنعت با نام Raspberrypi3 که مجری سامانه‌عامل رزبین است، پیاده‌سازی کردیم که نتایج نشان‌دهنده صحت حمله ارائه شده است.

کلیدواژه‌ها: الگوریتم رمزنگاری AES، پردازنده cortex-A53، حمله تصادم، معماری Exclusive، دستور clflush، سیاست جایگزینی

۱. مقدمه

دلیل استفاده از حافظه نهان در پردازنده‌های امروزی، افزایش سرعت عملکرد آن‌ها با دسترسی سریع به اطلاعات مورد نیازشان است. در صورت وجود اطلاعات مورد نیاز پردازنده در حافظه نهان، اصطلاحا تصادم رخ داده و این اطلاعات در اختیار پردازنده قرار می‌گیرد. در صورت عدم وجود این اطلاعات در این حافظه یک عدم‌تصادم رخ داده و این رویداد سبب انتقال یک بلوک داده از حافظه اصلی به حافظه نهان می‌شود. تفاوت زمانی بین این دو رویداد، پایه اصلی حمله‌های کانال جانبی مبتنی‌بر حافظه نهان است. حمله‌های مبتنی‌بر حافظه نهان در حالت کلی به دو دسته تقسیم می‌شوند:

۱- حمله‌های time-driven: در این حملات، مهاجم از آدرس داده‌های مربوط به سامانه رمزنگاری مطلع نبوده و تنها با آگاهی از ظرفیت لایه‌های حافظه نهان و خط‌های تقسیم‌کننده این حافظه، اقدام به اندازه‌گیری زمان اجرای سامانه رمزنگاری به‌ازای ورودی‌های مختلف نموده و در نهایت با انجام یک‌سری محاسبات آماری در جهت استخراج کلید تلاش می‌کند.

۲- حمله‌های access-driven: مهاجم در این نوع حمله‌ها علاوه‌بر آگاهی از ظرفیت لایه‌ها و خطوط حافظه نهان، به آدرس داده‌های الگوریتم رمزنگاری در حافظه‌های فیزیکی و حافظه مجازی نیز دسترسی دارد. علاوه‌بر موارد ذکرشده، اندازه‌گیری زمان در این حملات نیز می‌تواند در حد اندازه‌گیری زمان فراخوانی یک بلوک از داده‌ها تغییر نماید.

*رایانامه نویسنده پاسخگو: h_soleimany@sbu.ac.ir

مقالات [۱-۳] از جمله مقالاتی هستند که به اجرای حمله‌های time-driven برکامپیوترهای شخصی می‌پردازند. مقاله‌های [۴-۶] نیز سه نمونه از بهترین موارد بررسی و اجرای حمله‌های access-driven بر روی پردازنده‌های اینتل هستند. از مقاله [۷] که به اجرای یک حمله کانال‌جانبی نهان با استفاده از ویژگی تصادم انبوه[1] بر روی یک برد ARM9 می‌پردازد، می‌توان به‌عنوان اولین و یکی از شناخته‌شده‌ترین حمله‌های کانال جانبی مبتنی‌بر حافظه نهان، بر روی پردازنده‌های ARM، نام برد. پس از آن نیز [۸-۹] به اجرای حمله زمانی Bernstein بر روی پردازنده‌های cortex-A8,cortex-A9 اقدام نموده‌اند. اخیرا نیز در مقاله [۱۰] اقدام به بررسی و پیاده‌سازی تمام حملات access-driven مطرح، بر روی پردازنده‌های cortex-A53, cortex-A57 شده است.

۱-۱. چالش‌های اعمال حمله Time-Driven بر روی پردازنده cortex-A53

اجرای حمله موردنظر بر روی پردازنده مذکور نسبت به پردازنده‌های مورد حمله قبلی با سه چالش مهم روبه‌رو است:

۱- پشتیبانی نکردن از دستور clflush، جهت خالی کردن عناصر جدول‌های مراجعه‌ای سامانه رمزنگاری AES پس از هر بار اجرای سامانه رمزنگاری در حمله موردنظر.

۲- Exclusive بودن لایه آخر حافظه نهان در پردازنده موردحمله.

۳- شبه‌تصادفی بودن سیاست جایگزینی در لایه آخر حافظه نهان پردازنده مورد حمله. در بخش‌های بعدی به تاثیر این چالش‌ها در نتیجه حمله مذکور و نحوه رفع آن‌ها اشاره خواهد شد.

۱-۲. نوآوری مقاله

هدف این مقاله بررسی و پیاده‌سازی یک حمله time-driven مبتنی‌بر تصادم روی دور اول سامانه رمزنگاری AES است که با استفاده از کتابخانه openssl در برد Raspberrypi3 اجرا می‌شود، است. پردازنده پشتیبان این برد cortex-A53 است. حمله موردنظر با در نظر گرفتن تاثیر بروز تصادم در حافظه نهان بر روی زمان اجرای الگوریتم رمزنگاری، سعی در استخراج بیت‌هایی از کلید سامانه رمزنگاری AES می‌نماید. بدین منظور راهکارهایی به‌منظور فائق آمدن بر چالش‌های ارائه‌شده در بخش ۱-۱- ارائه می‌کنیم و نشان می‌دهیم که می‌توان با استفاده از راهکارهای پیشنهادی یک حمله time-driven مبتنی‌بر تصادم را بر روی پردازنده‌های ARM اعمال کرد. در نهایت حمله توصیف‌شده را با استفاده از یک برد پرکاربرد در اتوماسیون صنعتی با نام Raspberrypi3 که مجری سامانه‌عامل رزبین[2] است، پیاده‌سازی کرده و نتائج حاصله را به‌صورت عملی بررسی می‌کنیم.

ساختار مقاله بدین صورت است که ابتدا در بخش ۲ مقدمات و پیش‌زمینه‌های حمله مورد بررسی ارائه می‌شوند. سپس در بخش ۳ به تفصیل در خصوص راه‌کارهای اعمال حمله time-driven مبتنی‌بر تصادم در پردازنده‌های ARM را شرح می‌دهیم. در بخش ۴، نتایج به‌دست‌آمده از پیاده‌سازی این حمله با استفاده از راه‌کارهای ارائه‌شده بررسی می‌شوند. در نهایت بخش ۵، به جمع‌بندی و بررسی کارهای آینده اختصاص یافته است.

۲. پیش‌زمینه‌ها

۲-۱. حافظه نهان

حافظه نهان، یک حافظه واسط بین رم و پردازنده است. این حافظه وظیفه کاهش زمان دسترسی پردازنده به اطلاعات مورد نیازش را دارد. عمدتا پردازنده‌های امروزی از چندین دسته تشکیل شده‌اند. هر یک از این دسته‌ها نیز از چندین خط ساخته شده‌اند. با فراخوانی یک داده مشخص در یک آدرس معین توسط پردازنده در صورت عدم وجود اطلاعات موردنظر در حافظه نهان، علاوه‌بر مقدار فراخوانی شده، داده‌های آدرس بعدی نیز متناسب با ظرفیت خطوط این حافظه به یکی از دسته‌ها منتقل می‌شوند. انتخاب دسته و خط موردنظر جهت انتقال بلوک اطلاعات، متناسب با شماره آدرس داده فراخوانی‌شده صورت می‌گیرد. فیزیکی یا مجازی بودن هریک از آدرس‌ها جهت انتخاب دسته‌ها و خطوط در پردازنده‌های مختلف، متفاوت است. به‌طور مثال در پردازنده‌های اینتل برای انتخاب دسته و خط میزبان از آدرس‌های حافظه مجازی و در پردازنده‌های ARM از آدرس‌های فیزیکی استفاده می‌شود.

با پر شدن هر یک از دسته‌های حافظه نهان از سیاست‌های مختلفی جهت جایگزینی اطلاعات جدید در خطوط هر دسته استفاده می‌شود. پردازنده‌های اینتل از سیاست LRU در هریک از لایه‌های خود برای جایگزینی اطلاعات جدید در دسته‌ها استفاده می‌کنند. طبق این سیاست خطی هر دسته که زودتر از بقیه پر شده‌است، با ورود اطلاعات جدید خالی خواهد شد. پردازنده‌های ARM در لایه اول خود از سیاست LRU و در لایه آخر از سیاست شبه‌تصادفی استفاده می‌کنند. این ویژگی پردازنده‌های ARM، نیاز به افزایش تعداد نمونه‌های اندازه‌گیری زمان در اجرای حمله‌های کانال جانبی نهان را افزایش می‌دهد.

1- Wide Collision

2- Rasbian

متناسب با نوع ارتباط اطلاعات موجود در لایه‌های اولیه حافظه نهان و لایه آخر، می‌توان سه نوع تقسیم‌بندی Exclusive، inclusive و non-inclusive برای این حافظه‌ها در نظر گرفت. در حافظه‌های Exclusive در هر لحظه از زمان اطلاعات خطوط لایه‌های اولیه و لایه آخر با یکدیگر متفاوت هستند. در حافظه‌های نهان inclusive لایه آخر حافظه نهان علاوه‌بر اطلاعات مربوط به خود، اطلاعات لایه‌های اولیه را نیز در خود نگه می‌دارد. در حافظه‌های non-inclusive برخی اطلاعات لایه‌های اولیه و لایه آخر یکسان و برخی متفاوت است. پردازنده‌های اینتل از حافظه‌های inclusive و پردازنده‌ها ARM از هر دو مورد non-inclusive و Exclusive استفاده می‌کنند. پردازنده cortex-A53 نیز از حافظه Exclusive استفاده می‌نماید.

در پردازنده‌های چندهسته‌ای لایه آخر حافظه نهان بین تمامی هسته‌ها مشترک است، یعنی هر هسته می‌تواند از داده‌های منتقل‌شده به لایه آخر توسط هسته‌های دیگر استفاده کند. علاوه‌بر این در برد Raspberry pi 3 پروتکل انسجام[1] نیز برقرار است. مطابق با ویژگی پروتکل انسجام، درصورت مشترک بودن لایه آخر حافظه نهان بین تمام هسته‌ها، هرگونه تغییر در اطلاعات مشترک دوهسته (چه لایه آخر و چه لایه‌های اولیه) توسط یکی از آن‌ها موجب تغییر اطلاعات هسته دیگر نیز می‌شود. از این ویژگی موجود در پردازنده موردنظر، برای مرتفع کردن یکی از چالش‌های موجود در حمله زمانی مورد نظر استفاده می‌شود.

۲-۲. پیاده‌سازی نرم‌افزاری سامانه رمزنگاری AES

سامانه رمزنگاری AES شامل ده دور متوالی است. هر دور از این سامانه رمزنگاری ۱۶ مقدار x^i را به‌عنوان ورودی دریافت کرده و با اجرای عملیات‌هایی جبری شامل جمع بیتی با کلید هر دور و عبور از بخش‌های sub-byte و shift row و Mix-column، ۱۶ مقدار x^{i+1} را به‌عنوان خروجی تحویل می‌دهد. در نه دور اول سامانه رمزنگاری هر چهار عملیات بیان‌شده بر روی هریک از ورودی‌های هر دور صورت می‌گیرد. دور آخر فقط به اعمال سه عملیات اول جمع بیتی کلید و sub-byte و shift row بر روی ورودی‌ها می‌پردازد. در پیاده‌سازی نرم‌افزاری سامانه AES، سه عملیات آخر هریک از نه دور اول با استفاده از چهار جدول مراجعه T^0، T^1، T^2 و T^3 موجود در رابطه (۱)، به‌صورت رابطه (۲) پیاده‌سازی می‌شوند.

$$T1(Z)=\begin{pmatrix}3\bullet S(Z)\\2\bullet S(Z)\\S(Z)\\S(Z)\end{pmatrix}.T0(Z)=\begin{pmatrix}2\bullet S(Z)\\S(Z)\\S(Z)\\3\bullet S(Z)\end{pmatrix} \quad (۱)$$

$$T3(Z)=\begin{pmatrix}S(Z)\\S(Z)\\3*S(Z)\\2*S(Z)\end{pmatrix}.T2(Z)=\begin{pmatrix}S(Z)\\3*S(Z)\\2*S(Z)\\S(Z)\end{pmatrix}$$

دور آخر سامانه رمزنگاری نیز با استفاده از جدول مراجعه‌ای $T4$ که ساختار آن در رابطه (۳) قابل مشاهده است، پیاده‌سازی می‌شود.

$$T4(Z)=\begin{pmatrix}S(Z)\\S(Z)\\S(Z)\\S(Z)\end{pmatrix} \quad (۳)$$

۲-۳. حمله Time-driven مبتنی‌بر تصادم

۲-۳-۱. مدل حمله

پیاده‌سازی سامانه رمزنگاری AES با استفاده از جدول‌های مراجعه‌ای، سبب ۳۶ بار فراخوانی هریک از ۴ جدول مورد استفاده در ۹ دور اول سامانه خواهد شد. با هر بار فراخوانی یکی از $T^j[x^i \oplus k^i]$ ها، علاوه‌بر عنصر فراخوانی‌شده، & عنصر دیگر از جدول T^j به یکی از خطوط حافظه نهان منتقل می‌شود. مقدار & به ظرفیت هر خط از حافظه نهان بستگی دارد. به‌طور مثال با توجه به ۶۴ بایتی بودن خطوط حافظه نهان در پردازنده cortex-A53، به‌ازای هر فراخوانی ۱۶ عنصر از هر جدول مراجعه‌ای درون حافظه نهان قرار می‌گیرند. در صورتی که مقدار یکی از ۳۵ فراخوانی باقی‌مانده T^j به طور مثال $T^j[x^{''i} \oplus k''i]$، با یکی از مقادیر انتقالی به حافظه نهان توسط فراخوانی قبلی برابر باشد، یک تصادم رخ داده و موجب جلوگیری از بروز یک عدم‌تصادم خواهد شد. در نتیجه در صورت بروز چنین رویدادی، زمان اجرای سامانه رمزنگاری نسبت به حالت‌های دیگر کم‌تر خواهد بود. ایجاد تصادم بین $T^j[x^i \oplus k^i]$ و $T^j[x^{'i} \oplus k^{'i}]$ به‌معنای برقراری (۴) است. با توجه به توصیفات بالا مهاجم می‌تواند با استفاده از (۵) و با آگاهی از $x^i, x^{'i}$ هایی که موجب تصادم بین دو فراخوانی مذکور می‌شود به l بیت بزرگ‌تر از مقدار $<k^i \oplus k^{'i}>$ دست یابد.

$$< x^i \oplus k^i >=< x'^i \oplus k'^i > \quad (۴)$$

$$< k^i \oplus k'^i >=< x'^i \oplus x'^i > \quad (۵)$$

مقدار l نیز به‌صورت رابطه (۶) محاسبه می‌شود.

$$l = 8 - \log^{\&} 2 \quad (۶)$$

۲-۴-۱. پیاده‌سازی حمله زمانی تصادم بر روی دور اول سامانه رمزنگاری AES

با توجه به توضیحات بخش قبلی و مقادیر ورودی جدول‌های مراجعه در دور اول سامانه رمزنگاری AES که به‌صورت $x^i = P^i \oplus k^i$ هستند، اجرای حمله زمانی متاثر از تصادم بر

1-Coherency Protocol

روی جدول‌های مراجعه در این دور از سامانه رمزنگاری به‌علت در دسترس بودن مقادیر ورودی سامانه و ارتباط نزدیک با کلیداصلی سامانه رمزنگاری یک حمله مناسب شناخته می‌شود. مطابق با (۲) هر چهار مورد x^i مشخص در دور اول، موجب فراخوانی عناصر یک جدول مراجعه یکتا می‌شوند. به‌طور مثال x^0,x^4,x^8,x^{12} موجب فراخوانی عناصرجدول T^0 می‌شوند. مهاجم جهت استخراج $k^i \oplus k^j$ از طریق جستجوی تصادم بین $T^k[p^i \oplus k^i]$ و $T^k[p^j \oplus k^j]$ در دور اول، به‌صورت زیر عمل می‌کند:

$$< p^i \oplus p^j >=< k^i \oplus k^j > \quad (۷)$$

با توجه به ۶۴ بایتی بودن ظرفیت پردازنده مورد حمله در این مقاله، مهاجم با بررسی وقوع شش تصادم ممکن برای هریک از جدول‌های مراجعه در دور اول به ۴ بیت بزرگ‌تر از هر بایت کلید دست پیدا می‌کند. به‌طور مثال، با کاوش شش تصادم ممکن:

$$(T^0[p^0 \oplus k^0], T^0[p^4 \oplus k^4])$$
$$(T^0[p^0 \oplus k^0], T^0[p^8 \oplus k^8])$$
$$(T^0[p^0 \oplus k^0], T^0[p^{12} \oplus k^{12}])$$
$$(T^0[p^4 \oplus k^4], T^0[p^8 \oplus k^8])$$
$$(T^0[p^4 \oplus k^4], T^0[p^{12} \oplus k^{12}])$$
$$(T^0[p^{12} \oplus k^{12}], T^0[p^8 \oplus k^8])$$

در جدول مراجعه‌ای اول، با استفاده از گام‌های بیان‌شده می‌توان، اقدام به استخراج ۴ بیت بزرگ‌تر از کلیدهای k^0، k^4، k^8 k^{12} نمود. با اجرای این حمله بر جدول‌های مراجعه‌ای دیگر در دور اول و استخراج ۴ بیت بزرگ‌تر هر یک از بایت‌های کلید، جستجوی کلید AES-128 به 2^{80} قابل تقلیل است.

۲-۵. نحوه اندازه‌گیری زمان در پردازنده موردنظر

یکی از مهم‌ترین ملزومات اجرای حمله‌های کانال‌جانبی مبتنی‌بر حافظه نهان با استفاده از کانال زمان، استفاده از یک دستور و یا واحد فیزیکی مناسب جهت اندازه‌گیری زمان، و ایجاد تمایز بین بروز تصادم و عدم‌تصادم است. در اجرای حمله‌های زمانی بر روی پردازنده‌های اینتل، از دستور rdtsc جهت اندازه‌گیری زمان استفاده می‌شود. پردازنده‌های مبتنی‌بر معماری ARMv7-A دستور خاصی جهت استفاده برای اندازه‌گیری زمان ارائه نکرده‌اند. پردازنده‌های مبتنی‌بر این معماری از جمله پردازنده مورد نظر این مقاله، دارای یک واحد مشاهده و نظارت عملیاتی جهت مشاهده و نظارت کردن عملکرد پردازنده خود هستند. ثبات شمارنده سیکل‌های پردازنده یکی از قسمت‌های واحد مذکور است.

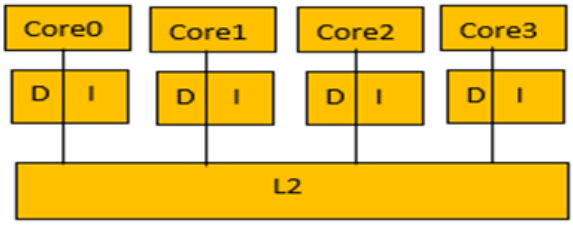
در حمله زمانی ارائه‌شده بر پردازنده موردنظر، جهت شناسایی حالت‌های وقوع تصادم از این رجیستر استفاده شده‌است. جهت استفاده از رجیستر موردنظر، نیاز به ساخت و اعمال یک ماژول کرنل است، که این عملیات در کرنل سامانه عامل رزیبن قابل اجرا است.

۳. بررسی چالش‌های موجود در اجرای حمله تصادم بر پردازنده cortex-A53

با توجه به گام‌ها و راهبردهای حمله زمانی مبتنی‌بر تصادم، توانایی در ایجاد یک تمایز قوی بین بروز و عدم بروز تصادم و پاک‌کردن کامل و مناسب تمام لایه‌های حافظه نهان برای هر بار اجرای سامانه رمزنگاری، دو مورد از مهم‌ترین عوامل تاثیرگذار بر نتیجه حمله هستند. Exclusive بودن حافظه نهان در پردازنده cortex-A53، سبب ایجاد اختلال در تمایز بین تصادم و عدم‌تصادم می‌شود. در کنار Exclusive بودن حافظه نهان در این پردازنده، پشتیبانی نکردن از دستور clflush و شبه‌تصادفی بودن سیاست جایگزینی در لایه آخر این حافظه نیز، اطمینان از خالی شدن کامل حافظه نهان را مختل می‌سازد. در ادامه این بخش به توصیف تاثیر موارد مذکور در نتیجه حمله، وراه‌های مقابله با آن‌ها پرداخته خواهد شد.

۳-۱. بررسی تاثیر منفی Exclusive بودن حافظه نهان بر تمایز بین بروز تصادم و عدم تصادم

اکثر پردازنده‌های ARM از حافظه‌های نهان دو لایه استفاده می‌کنند. در همه این پردازنده‌ها لایه اول شامل دو قسمت لایه اطلاعات و لایه دستور بوده و هر هسته، لایه اول مخصوص به خود را دارا هست. لایه دوم حافظه نهان نیز در بین تمامی هسته‌ها مشترک است. شکل (۱): ساختار حافظه نهان در پردازنده‌های نمونه‌ای از ساختار حافظه نهان این پردازنده‌ها را نشان می‌دهد. در صورت inclusive بودن حافظه نهان در این پردازنده‌ها، مهاجم از توانایی هر هسته برای دسترسی به اطلاعات منتقل‌شده به حافظه نهان از طریق هسته‌های دیگر مطمئن است. یعنی مهاجمی که به دنبال تصادم بین دو فراخوانی $T^k[p^j \oplus k^j]$ و $T^k[p^i \oplus k^i]$ است، از بروز تصادم در صورت برابر بودن یکی از عناصر $T^k[p^j \oplus k^j]$، با عناصر انتقالی توسط $T^k[p^i \oplus k^i]$ اطمینان دارد.



شکل (۱): ساختار حافظه نهان در پردازنده‌های ARM [۱۰]

دلیل این اطمینان، قرار گرفتن قطعی اطلاعات مربوط به فراخوانی اول در هر دو لایه این حافظه به دلیل نوع کارکرد حافظه‌های inclusive است. در حافظه‌های Exclusive، بر خلاف حافظه‌های inclusive، اطمینان از بروز تصادم بین دو فراخوانی مدنظر در صورت برابری مقدار فراخوانی دوم، با یکی از عناصر انتقالی به حافظه نهان توسط فراخوانی اول وجود ندارد. این عدم اطمینان به این دلیل است، که در صورت فراخوانی $T^k[p^j \oplus k^j]$ و $T^k[p^i \oplus k^i]$ از هسته‌های مختلف در پردازنده‌های دارای حافظه Exclusive، به سبب قرار گرفتن عناصر انتقالی توسط فراخوانی اول در لایه اول مخصوص به هسته خود و عدم انتقال آن‌ها به لایه دوم، فراخوانی اطلاعات آن، از هسته دیگر امکان‌پذیر نبوده و عدم‌تصادم رخ خواهد داد. این عدم‌تصادم در صورت Exclusive بودن هر دو قسمت اطلاعات و دستورالعمل حافظه نهان رخ داده و در صورت inclusive بودن قسمت دستورالعمل حافظه نهان، فراخوانی اطلاعات از هسته‌های دیگر از قاعده‌ای متفاوت پیروی می‌کند. در صورت inclusive بودن بخش‌های دستورالعمل حافظه نهان یک پردازنده، با فراخوانی اطلاعات مخصوص یک هسته توسط هسته دیگر، انتقال یک کپی از دستور فراخوانی اطلاعات از قسمت دستورالعمل لایه اول هسته فراخوانی‌کننده به بخش دستورالعمل لایه دوم و قسمت دستورالعمل لایه اول هسته فراخوانی‌شده صورت می‌گیرد. با انتقال دستورالعمل فراخوانی به بخش دستورالعمل لایه اول هسته موردنظر، اطلاعات فراخوانی‌شده به لایه دوم منتقل شده و دستورالعمل‌ها از لایه اول پاک می‌شوند. این امر سبب در دسترس قرار گرفتن این اطلاعات توسط همه هسته‌ها می‌شود و دیگر عدم‌تصادمی رخ نخواهد داد. طی شدن روال دسترسی یک هسته به اطلاعات هسته دیگر زمانی بیش‌تر از زمان وقوع یک تصادم مصرف می‌کند. این در حالی است که به علت برقراری پروتکل انسجام در بین هسته‌های پردازنده مورد حمله، زمان این دسترسی، زمانی کم‌تر از زمان دسترسی به حافظه اصلی خواهد بود. در نتیجه با بالا بردن نمونه‌های اندازه‌گیری زمان اجرای الگوریتم در حمله موردنظر، می‌توان بین زمان دسترسی به حافظه اصلی و زمان دسترسی بین هسته‌ای تمایز قائل شد.

۳-۲. عدم پشتیبانی از دستور clflush

یکی از مراحل مهم اجرای حمله‌های کانال‌جانبی مبتنی‌بر حافظه نهان بر اساس رویداد تصادم، پاک کردن تمام لایه‌های حافظه نهان از اطلاعات مربوط به جدول‌های مراجعه‌ای، پس از هربار اندازه‌گیری زمان اجرای آن است. این مهم در پردازنده‌های اینتل با استفاده از دستور clflush صورت می‌گیرد. در پردازنده‌های مبتنی‌بر معماری ARM7-A دستور خاصی جهت پاک کردن حافظه نهان، وجود ندارد. جهت پیشبرد حمله مورد نظر بر روی این پردازنده‌ها باید به دنبال یک جایگزین مناسب برای این دستور بود. یکی از راه‌کارهای جایگزین برای پاک کردن اطلاعات خطوط یک دسته مشخص، فراخوانی اطلاعاتی (به‌جز عناصر جدول‌های مراجعه‌ای) است که در صورت انتقال به حافظه نهان درون دسته مورد نظر قرار می‌گیرند. به‌طور مثال در پردازنده‌هایی که از سیاست جایگزینی LRU استفاده می‌کنند، فراخوانی عناصر n آدرس مختلف که درصورت فراخوانی درون یک دسته n خطی مشخص قرار می‌گیرند، سبب خالی شدن تمام دسته مورد نظر از عناصر قبلی می‌شود. به‌علت شبه‌تصادفی بودن سیاست جایگزینی در لایه آخر پردازنده مورد حمله، خالی کردن هر دسته از حافظه نهان پیچیده‌تر از فراخوانی تنها n مقدار خواهد بود. [۱۱] به بررسی رویکردهای مختلف جهت خالی‌کردن دسته‌های حافظه نهان پردازنده‌های گوناگون اقدام کرده و به ارایه یک الگوی مناسب و قابل اجرا در بسیاری از پردازنده‌ها، پرداخته‌است. در الگوریتم (۱) ساختار این الگو قابل مشاهده است.

الگوریتم (۱)

$$for\ (s = 0; s \leq S - D; s += L)\ \{$$

$$for\ (c = 0; c \leq C;\ \ c += 1)\ \{$$

$$for\ (d = 0; d \leq D; d += 1)\{$$

$$a^*[s + d]\}\}\}$$

مهاجم با انتخاب S آدرس مختلف که فراخوانی عناصر آن‌ها سبب انتقال به دسته مورد نظرش می‌شود، اقدام به انتخاب C,D,L های متفاوت کرده و با فراخوانی عناصر قبلی موجود در آن دسته، میزان موفقیت راهبرد را می‌سنجد. جدول (۱) نتیجه چند نمونه از این راهبردها بر روی پردازنده cortex-A53 را نشان می‌دهد. انتخاب فیزیکی یا مجازی بودن آدرس‌ها به نوع پردازنده بستگی دارد. در پردازنده cortex-A53 از آدرس‌های فیزیکی برای اجرای این راهبرد استفاده می‌شود. دسترسی به آدرس‌های فیزیکی داده‌های هر برنامه در سامانه عامل رزبین از طریق /proc/self/pagemap امکان‌پذیر است. در حمله موردنظر راهبرد اول جدول (۱) بر روی تمام دسته‌های لایه آخر پردازنده اعمال می‌شود.

جدول (۱): نتایج اجرای راهبرد تخلیه به‌ازای مقادیر مختلف

درصد موفقیت	D	L	S
٪۱۰۰	۵	۲	۲۳
٪۱۰۰	۶	۴	۲۳
٪۹۹/۹۹	۶	۱	۲۲
٪۹۹/۹۳	۶	۱	۲۱
٪۹۹/۴۴	۶	۴	۲۰
٪۹۹/۱۰	۱	۱	۸۰۰
٪۷۰/۷۸	۱	۱	۴۸

۳-۳. بررسی تاثیر منفی Exclusive بودن حافظه نهان در موفقیت راهبردی اعمالی بر پردازنده

با توجه به این که در حافظه‌های Exclusive، انتقال اطلاعات لایه اول حافظه نهان به لایه دوم، وابسته به پر شدن کامل دسته مربوط به آن داده‌ها در لایه اول و یا فراخوانی آن‌ها از طریق هسته‌های دیگر است، احتمال باقی ماندن بعضی عناصر جدول مراجعه در لایه اول وجود دارد. به همین جهت با توجه به LRU بودن سیاست جایگزینی لایه اول پردازنده cortex-A53، در راستای خالی کردن هر دسته از این لایه باید متناسب با تعداد خطوط هر دسته، اقدام به فراخوانی آدرس‌های مناسب نمود. با توجه به چهار خطی بودن هریک از دسته‌ها در لایه اول و مجزا بودن این لایه برای هریک از چهار هسته پردازنده، باید فراخوانی‌ها به‌گونه‌ای صورت بپذیرد که هر هسته حداقل چهار فراخوانی را به خود اختصاص بدهد. بدین منظور از (۸) برای شناسایی تعداد فراخوانی‌های لازم جهت اختصاص حداقل m فراخوانی به هریک از هسته‌های یک پردازنده r هسته‌ای برای پاک شدن هر دسته m خطی از لایه اول، استفاده می‌شود.

در حمله صورت گرفته بر روی پردازنده مورد نظر با توجه r = 4، m = 4 از n = 20 جهت خالی کردن هر یک از دسته‌های لایه اول حافظه نهان استفاده شده است.

$$\frac{\binom{n+r-m-1}{r-1}}{\binom{n+r-1}{r-1}} > \frac{1}{2} \quad (۸)$$

۴. پیاده‌سازی و بررسی نتایج عملی حمله

در این بخش، پیاده‌سازی عملی حمله و نتائج آن را ارائه خواهیم کرد.

۴-۱. پیاده‌سازی عملی حمله تصادم

بستر استفاده‌شده در این مقاله جهت اجرای یک حمله تصادم، یک برد کاربردی در اتوماسیون صنعتی با نام Raspberry pi3 است.

۴-۱-۱. نحوه اندازه‌گیری زمان

جهت اندازه‌گیری زمان با استفاده از ثبات رجیستر شمارنده سیکل در این حمله، به اعمال یک ماژول هسته در سامانه عامل رزبین اقدام شده است. الگوریتم (۲) و الگوریتم (۳) دستورات لازم جهت اعمال یک ماژول به هسته سامانه عامل را نشان می‌دهند. با استفاده از دستور sudo make و sudo insmod nn.ko در فایل شامل کدهای الگوریتم (۲) و الگوریتم (۳) می‌توان از ثبات رجیستر شمارنده سیکل برای اندازه‌گیری زمان اجرای سامانه رمزنگاری استفاده نمود.

الگوریتم (۲)

nn.c:

```
#include < linux/module.h >
#include < linux/kernel.h >
1: void enable_ccr(void * info){
2: asm volatile ("mcr p15.0.%0. c9.c14.0"
: : "r" (1));
3: asm volatile ("mcr p15.0.%0. c9.c12.0
\t\n" : : "r" (1));
4: asm volatile ("mcr p15.0.%0. c9.c12.1
\t\n" : : "r" (0x80000000)); }
5: int init_module(void){
6: on_each_cpu(enable_ccr. NULL. 0);
7: printk (KERN_INFO "userspace access to
CCR enabled\n");
8: return 0; }
9: void cleanup_module(void){}
```

الگوریتم (۳)

```
1: obj − m += nn.o
2: KDIR = /usr/src/linux − headers
−4.9.59 − v7 +
```

3: all:

4: $\$(MAKE) -C \$(KDIR) SUBDIRS = \$(PWD) modules$

5: $clean$:

6: $rm -rf *.o *.ko *.mod.* *.symvers *.order$

۴-۱-۲. تخلیه حافظه نهان

همان‌طور که در بخش ۲-۳ نیز بیان‌شد، در این حمله از راهبرد اول جهت تخلیه هریک از دسته‌های حافظه نهان، استفاده می‌شود. به همین جهت در کد حمله موردنظر، قبل از هربار اجرای سامانه رمزنگاری اقدام به اجرای الگوریتم (۴) می‌شود.

الگوریتم (۴)

$for\ (s = 0; s \leq 18; s+= 2)\ \{$

$for\ (c = 0; c \leq 1;\ c+= 1)\ \{$

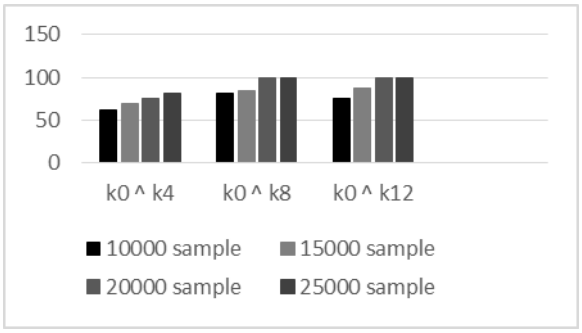
$for\ (d = 0; d \leq 5; d+= 1)\{$

$a^*[s + d]\}\}\}$

$a^*[s + d]$ در الگوریتم (۴) نشان دهنده آدرس مجازی عناصری است که فراخوانی آن‌ها موجب اطلاعات آن‌ها به یک دسته خاص و مشخص می‌شود. جهت خالی کردن لایه اول از حافظه نهان این پردازنده نیز اقدام به فراخوانی ۲۰ عنصر متناسب با هر دسته خواهد شد.

۴-۲. نتایج پیاده‌سازی عملی

شکل (۲) میزان موفقیت در دست‌یابی به چهار بیت بزرگ‌تر از مقادیر $< k^0 \oplus k^4 >, < k^0 \oplus k^8 >, < k^0 \oplus k^{12} >$ به‌ازای تعداد نمونه‌گیری‌های مختلف در حمله صورت گرفته را نشان می‌دهد. سطر افقی این نمودارها تعداد نمونه‌های مختلف و سطر عمودی درصد موفقیت دست‌یابی به مقدار صحیح بر اساس رتبه انتخاب شدن هر مقدار بین ۱۶ مقدار ممکن را نشان می‌دهد. به‌طور مثال ۱۰۰٪ موفقیت یک $< k^i \oplus k^j >$ به‌معنای اختصاص یافتن کم‌ترین میانگین زمان اجرای الگوریتم در اجرای حمله، به مقدار $< p^i \oplus p^j >$ است که مقدار مربوطه به‌طور صد در صد درست استخراج شده است. لازم به ذکر است که با اجرایی کردن این حمله بر روی دور دوم از سامانه رمزنگاری AES می‌توان تمام بیت‌های کلید سامانه رمزنگاری را استخراج نمود.



شکل (۲): رتبه مجموعه کلید صحیح به‌ازای نمونه‌های مختلف

علاوه‌بر شبه‌تصادفی بودن سیاست جایگزینی در لایه آخر حافظه نهان این پردازنده، ناهم‌تراز قرار گرفتن اطلاعات جدول‌های مراجعه‌ای درون این حافظه نیز سبب افزایش تعداد نمونه‌های اندازه‌گیری‌های لازم جهت رسیدن به یک نتیجه مطلوب می‌شود. غیر هم‌تراز قرار گرفتن اطلاعات درون حافظه نهان سبب کاهش تعداد عناصر مناسب برای جستجوی تصادم می‌شود. حالت‌هایی که اولین عناصر جدول مراجعه، دقیقا در اولین آدرس یکی از خطوط حافظه نهان قرار بگیرند، بهترین حالت برای جستجوی تصادم هستند.

۵. نتیجه‌گیری

این مقاله برای اولین بار با استفاده از یک برد صنعتی، قابل اعمال بودن حمله time-driven بر مبنای تصادم روی یکی از پردازنده‌های کاربردی ARM (cortex-A53) را نشان داده است. در این مقاله با بررسی موانع موجود در اجرای این حمله بر پردازنده ARM cortex-A53 که شامل شبه‌تصادفی بودن سیاست جایگزینی لایه آخر، Exclusive بودن این حافظه و عدم پشتیبانی از دستور clflush است، به ارائه چند راه‌کار مناسب پرداخته شده و در نهایت با اجرای این حمله با استفاده از راه‌کارهای اتخاذشده، نتیجه‌بخش بودن آن‌ها بررسی و آشکار شده است. جهت استخراج بیت‌های بیش‌تر از کلید سامانه رمزنگاری AES می‌توان حمله ارائه‌شده را به دورهای دوم و آخر نیز اعمال کرد.

لازم به ذکر است که اخیرا طی تحقیقات زیادی اقدام به ارائه راه‌کارهایی سخت‌افزاری جهت دست‌یابی به آدرس‌های فیزیکی هر برنامه در پردازنده‌های اینتل صورت گرفته است. این راه‌کار نیاز دسترسی مهاجم به کرنل را از بین می‌برد. قابل استفاده‌کردن این راه‌کارها در پردازنده‌های ARM جهت تسهیل در دسترسی به آدرس‌های فیزیکی برای اجرای این حمله می‌تواند از مهم‌ترین اقدامات موثر در اجرای حمله‌های زمانی بر پردازنده‌های ARM باشد.

۶. مراجع

[1] D. J. Bernstein, “Cache-timing attacks onAES,” 2005.

[2] J. Bonneau and I. Mironov, “Cache-collision timing attacks against AES,” In International Workshop on Cryptographic Hardware and Embedded Systems, Springer, pp. 201-215, 2006.

[3] E. Tromer, D. A. Osvik, and A. Shamir, “Efficient cache attacks on AES, and countermeasures,” Journal of Cryptology, vol. 23, no. 1, pp. 37-71, 2010.

[4] Y. Yarom and K. Falkner, “FLUSH+ RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack,” In USENIX Security Symposium, pp. 719-732, 2014.

[5] D. Gruss, R. Spreitzer, and S. Mangard, “Cache Template Attacks: Automating Attacks on Inclusive Last-Level Caches,” In USENIX Security Symposium, pp. 897-912, 2015.

[6] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, “Flush+ Flush: a fast and stealthy cache attack,” In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Springer, pp. 279-299, 2016.

[7] A. Bogdanov, T. Eisenbarth, C. Paar, and M. Wienecke, “Differential cache-collision timing attacks on AES with applications to embedded CPUs,” In Cryptographers’ Track at the RSA Conference, Springer, pp. 235-251, 2010.

[8] M. Weiß, B. Heinz, and F. Stumpf, “A cache timing attack on AES in virtualization environments,” In International Conference on Financial Cryptography and Data Security, Springer, pp. 314-328, 2012.

[9] R. Spreitzer and T. Plos, “On the Applicability of Time-Driven Cache Attacks on Mobile Devices (Extended Version⋆),” 2013.

[10] M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, and S. Mangard, “ARMageddon: Cache Attacks on Mobile Devices,” In USENIX Security Symposium, pp. 549-564, 2016.

[11] D. Gruss, C. Maurice, and S. Mangard, “Rowhammer. js: A remote software-induced fault attack in javascript,” In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Springer, pp. 300-321, 2016.

[12] M. Lipp, M .Schwarz, D.Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, M. Hamburg, Meltdown, arXiv, 2018.

Journal of Electronical & Cyber Defence
Vol. 8, No. 1, 2020, Serial No. 29

Introducing a New Timing Attack on the ARM Processor and its Practical Implementation on the Raspberrypi3 Board

V. Meraji, H. soleimany*

Shahid Beheshti of University
(Received: 10/08/2018, Accepted: 18/06/2019)

ABSTRACT

An important category of the side-channel attacks takes advantage of the fact that cache leads to temporal changes in the execution of encryption algorithms and thus information leakage. Although side-channel attacks based on high cache memory are among the most widely used side-channel attacks, they have been less studied than other side-channel attacks. Accordingly, extensive research has been conducted by the cryptographic community in the area of side-channel attacks based on cache memory.The focus of research has mainly been on the security of encryption algorithms implemented by Intel and Pentium processors, which due to the different cache structure of different processors, cannot be extended to other commonly used processors such as ARM. In response to this challenge, new research is focusing on cache-based side-channel attacks on various mobile processors and other applications including ARM processors. The different cache structure and lack of support for some of the commands needed to execute cache attacks have made it difficult to execute these attacks on ARM processors. In this paper, we first investigate the cache-timing attack using a collision event on one of the ARM processors. In this attack, the attacker only needs to measure the timing of the encryption, and unlike the access-driven attacks, the attacker does not need access to the victim's cache. We also implemented the attack using an industrial automation board called Raspberrypi3, which runs the router operating system, the results of which show the accuracy of the attack.

Keywords: AES, Cortex-A53 Processor, Collision Attack, Exclusive Architecture, Clflush Instruction, Replacement Policy

* Corresponding Author Email: h_soleimany@sbu.ac.ir