Unleashing MAYHEM on Binary Code

MAYHEM ابزاری برای یافتن آسیب پذیری‌ها در فایل‌های اجرایی یا به عبارتی فایل‌های باینری است.

برای یافتن کد حمله لازم است تا فضای بزرگی جست‌وجو شود. در MAYHAM برای مقابله با این موضوع ۴ اصل وجود دارد:

۱) سیستم باید بتواند همواره پیشرفت کند. بدون اینکه از منابع اختصاص داده شده بیش از حد مجاز استفاده کند.
۲) برای اینکه کارایی بالا رود، نباید یک کار دوباره اجرا شود.
۳) سیستم نباید یک کار یا نتیجه آن را رها کند بلکه در هر اجرا باید از نتایج اجرای گذشته استفاده کند.
۴) سیستم باید بتواند در مورد مقدار نمادین یک حافظه که خواندن و نوشتن در آن بستگی به کاربر دارد، تصمیم گیری کند.

دو نوع تحلیل‌گر کد وجود دارد:

۱) آفلاین: تحلیل‌گرهایی که ابتدا مسیری از کد را به صورت عددی اجرا می‌کنند و سپس با حل مجموعه قیدهای تولید شده به صورت نمادین، سعی می‌کنند ورودی برای مسیر جدید تولید کنند مثل SAGE.
۲) آنلاین: در این نوع تحلیل‌گرها سعی می‌شود در یک اجرا تمام مسیرهای ممکن یک باره اجرا شوند. مثل S2E.

مشکلات تحلیلگرهای آفلاین:

اصل‌های ۱ و ۳ برقرارند. چون هر بار یک اجرا و مستقل از اجراهای دیگر هست. پس از نتایج آن می‌توان دوباره استفاده کرد.

اصل ۲ برقرار نیست. چون هر مسیر باید از اول اجرا شود. پس تعدادی دستور در تمام مسیرها چندباره اجرا می‌شوند.

تحلیلگرهای آنلاین مثل KLEE و S2E در هر نقطه از شروع شاخه جدید از fork استفاده می‌کند. پس دستورات گذشته دوباره اجرا نمی شوند. ولی دستور fork سربار زیادی از نظر حافظه دارد. پس اصل‌های ۱ و ۳ برقرار نیست.

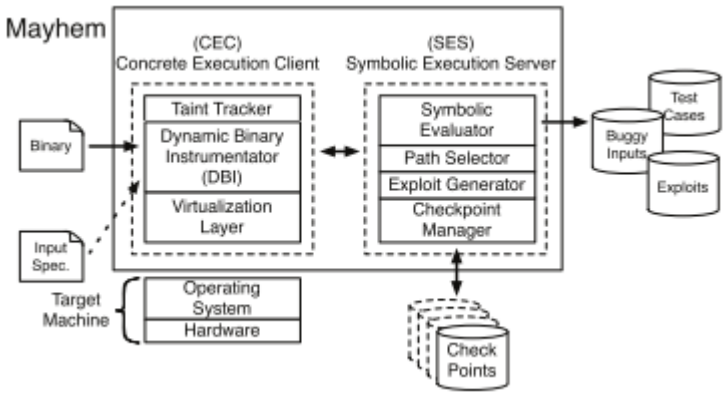
دستاوردهای جدید MAYHEM:

- اجرای ترکیبی آفلاین و آنلاین برای حفظ حافظه و سرعت اجرا
- مدل مبتنی بر ایندکس برای حافظه

- تحلیل فایل‌های اجرایی و باینری

نکات کلیدی در مورد آسیب پذیری‌ها و تحلیلگرها:

۱) نیاز است تا کد را در سطح ماشین کد اجرا شود چون آسیب پذیری‌های زیادی مثل BOF لازم است تا ساختار پشته هم بررسی شود.
۲) تعداد مسیرهای ممکن خیلی زیاد است. هر if یک شاخه درست می‌کند.
۳) هرچه تعداد مسیر بیشتری بررسی شود بهتر است.
۴) بهتر است که کد ابتدا عددی اجرا شود و در صورت نیاز به اجرای نمادین تغییر کند.



شکل ۱: معماری *MAYHEM*

مراحلی که MAYHEM برای تولید اکسپلویت طی می‌کند:

۱) ابزار MAYHEM با تعریف یک پورت کار خود را شروع می‌کند. و کدهای آسیب‌پذیر را از همین طریق دریافت می‌کند. این موضوع باعث می‌شود که ابزار بداند چه کدهایی در اختیار مهاجم است.
۲) واحد CEC برنامه آسیب‌پذیر را دریافت می‌کند. به SES وصل می‌شود تا مقداردهی‌های اولیه صورت پذیرد. سپس کد به صورت عددی اجرا می‌شود و همزمان تحلیل آلایش[۱]پویا نیز روی آن اجرا می‌شود.
۳) اگر CEC با یک بلاک کد آلوده یا یک پرش آلوده رو به رو شود(منظور جایی است که لازم است تا از کاربر ورودی دریافت شود)، CEC موقتاً اجرا نمی شود و شاخه آلوده به SES برای اجرای نمادین ارسال می‌شود. SES مشخص می‌کند که آیا اجرای شاخه ممکن هست یا نه!
۴) واحد SES به صورت موازی با CEC اجرا می‌شود و بلاک‌های کد را دریافت می‌کند. این بلاک‌ها به زبان میانی تبدیل می‌شوند و به صورت concolic اجرا می‌شود. مقادیر عددی مورد نیاز از CEC دریافت می‌شود.

[۱] Taint Analysis

فرمول قابلیت اکسپلویت مشخص می‌کند که:

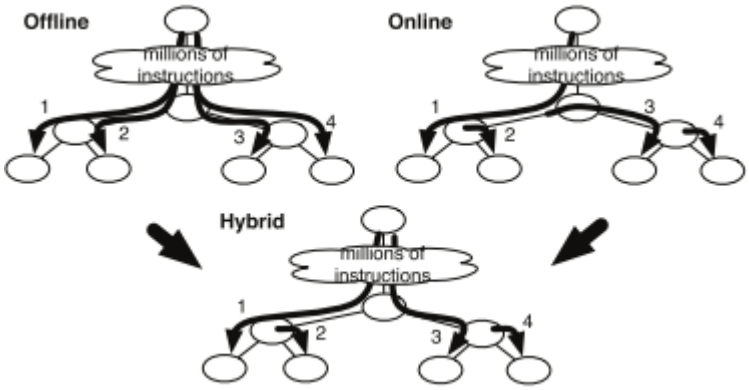
۱. آیا مهاجم می‌تواند کنترل اجرای دستورات یا

۲. اجرای PAYLOAD را بدست آورد یا نه؟

۵) وقتی به یک پرش آلوده می‌رسد SES تصمیم می‌گیرد که آیا FORK لازم هست یا نه. اگر باشد اجراهای جدید اولویت بندی شده و یکی اجرا می‌شود. اگر منابع تمام شوند SES رویه بازگشت را اجرا می‌کند. در نهایت بعد از اتمام اجرای یک پردازه تعدادی موردآزمون تولید می‌شوند.

۶) در پرش‌های آلوده یک فرمول EXPLOIT تولید و به SES داده می‌شود اگر قابل ارضا بود یعنی کد از این مسیر آسیب‌پذیر است.

برای اجرای نمادین لازم است تا کد باینری به کد اسمبلی تبدیل شود. برای این منظور از BAP استفاده می‌شود. سپس کد اسمبلی به شکل نمادین اجرا می‌شود.



شکل ۲: انواع اجراهای CONCOLIC

ابزار MAYHEM از اجرای نمادین هیبرید استفاده می‌کند. ابتدا به صورت آنلاین اجرا می‌کند تا به محدوده منابع برسد، بعد از آن به شکل آفلاین اجرا می‌کند.

ابزار MAYHEM در اجرای هیبرید ۴ مرحله اصلی دارد:

۱) مقداردهی اولیه: در این مرحله پایگاه‌داده CHECKPOINTها، محل نگهداری موردآزمون‌ها و مدیر CHECKPOINTها را مقدار دهی اولیه می‌کند.

۲) جست و جوی آنلاین: کد به صورت آنلاین اجرا می‌شود.

۳) مرحله CHECKPOINTING: در این مرحله مدیر CHECKPOINT بر اجرای آنلاین نظارت می‌کند. اگر حافظه یا تعداد اجراهای حاضر به حد معین‌شده برسد، یک CHECKPOINT تولید می‌شود که شامل اطلاعاتی مثل شرط مسیر، آمارها و غیره است.
۴) مرمت با CHECKPOINT: یکی از CHECKPOINTها با توجه به یک هیوریستیک انتخاب می‌شود. ابزار باید حالت عددی قبلی را از CHECKPOINT بازسازی کند برای این کار از مقادیری استفاده می‌کند که به ازای آنها شرط مسیر قابل ارضا هست. به این وسیله حالت عددی تا جایی که اجرا معلق شده ساخته می‌شود و از این به بعد اجرا به صورت آنلاین ادامه می‌یابد.

معماری و پیاده‌سازی CEC:

قسمت CEC به عنوان ورودی، فایل باینری و یک CHECKPOINT (دلخواه) دریافت می‌کند. ابتدا کد را به صورت عددی اجرا می‌کند همزمان تحلیل آلایش نیز انجام می‌دهد. اگر به بلاک آلوده برسد، بلاک را به SES می‌دهد تا به صورت نمادین تحلیل کند. خروجی آن آدرس پرش به یک بلاک کد یا ایجاد یک CHECKPOINT هست. CEC اجرا را تا زمانی که همه مسیرهای ممکن جست و جو شوند ادامه می‌دهد.

لایه مجازی‌سازی:

این قسمت مسولیت شبیه سازی OS را بر عهده دارد. هر اجراکننده عددی برای خود یک STATE جداگانه دارد که از دید دیگر اجرا کننده‌ها پنهان هست. اگر یک اجراکننده روی فایل می‌نویسد بقیه اجراکننده‌ها یک کپی از این فایل را دارند و می‌توانند همزمان نوشتن روی آن را داشته باشند.

معماری و پیاده‌سازی SES:

قسمت SES وظیفه مدیریت محیط نمادین و انتخاب مسیر بعدی برای CEC را دارد. محیط نمادین شامل اجراکننده نمادین، انتخاب کننده مسیر و مدیر CHECKPOINT هست. SES یک ظرفیت دارد اگر به محدوده آن برسد دیگر به اجرای نمادین مسیر ادامه نمی دهد و یک CHECKPOINT تولید می‌کند. CHECKPOINT شامل حالت‌های از اجراکننده نمادین هست که در اجرای اول به دلیل محدودیت حافظه اجرا نشده اند. هر STATE شامل دو مقدار است اول مقادیر رجیسترهای نمادین و دوم مقادیر حافظه نمادین. در هر CHECKPOINT این حالتها نگهداری می‌شود.

Precondition: کاربر به عنوان ورودی می‌تواند تعدادی قید روی ورودی‌های برنامه تعریف کند مثل طول ورودی، تا فضای جست و جو را کم کند. اگر قیدی تعریف نشود SES تلاش می‌کند تمام فضا را پوشش دهد.

انتخاب مسیر: برای انتخاب مسیر SES از ۳ هیوریستیک استفاده می‌کند:

۱) جست و جوی بلاک جدیدی از کد
۲) اجراکننده‌هایی که از حافظه نمادین استفاده می‌کنند در اولویت هستند.
۳) مسیرهای اجرایی که از دستورها به اشاره‌گرهای نمادین بدست می‌آیند در اولویت هستند.

مدل‌سازی حافظه:

MAYHEM به صورت جزئی حافظه را مدل‌سازی می‌کند. به این معنی که آدرس دستورات read می‌تواند به صورت نمادین یا عددی باشند ولی در دستور write آدرس باید عددی باشد.

برای تعریف حافظه نمادین MAYHEM از مفهوم Memory Object استفاده می‌کند که با M نشان داده می‌شود. هربار که آدرس حافظه به شکل نمادین باشد، یک M تازه ایجاد می‌شود که مجموعه ای از آدرس‌هایی که با عبارت نمادین قابل دسترسی هستند، در آن تعریف می‌شود. این حافظه برخلاف u غیر قابل تغییر است و یک تصویر از u است.

در بدترین حالت M شامل تمام 2^{32} حالت ممکن است که این فضای حالت بسیار بزرگی است. برای کاهش آن از یک جست و جوی دودویی استفاده می‌شود و سعی می‌شود که کران بالا و پایین Memory Object به صورت محافظه کارانه مشخص شود. برای تعیین کران‌ها از SMT استفاده می‌شود. برای مثال اگر نصف اول کل بازه قابل ارضا باشد، کران پایین در نصف اول است.

کش تصفیه:[۲] برای تصفیه و بدست آوردن محدوده لازم است تا تعدادی پرس و جو به SMT زده شود. برای کاهش این گونه درخواست‌ها یک کش قرار داده می‌شود که قدمها و نتیجه تصفیه نهایی را در خود نگهداری می‌کند. برای قدم جدید ابتدا درخواست به این کش ارسال می‌شود در صورت وجود، یک در خواست به SMT ارسال می‌شود تا ارضاپذیری آن و میزان دقت آن برای فورمول جدید بررسی شود.

کش Lemma: با وجود کش تصفیه هر بار لازم است تا درخواست به SMT داده شود و میزان دقت تصفیه موجود بررسی شود. برای بهبود این کار یک کش جدید قرار داده می‌شود که شکل کانونی هر فرمول(F) و نتیجه آن(Q) به شکل F→Q در آن نگهداری می‌شود. با این کار تا ۹۶ درصد می‌توان بهبود داشت.

درخت‌های جست‌وجوی ایندکس:[۳] برای بهبود جست و جو در Memory Object برای یافتن عبارات، MAYHEM از یک درخت دودویی IST(Indexed Search Tree) استفاده می‌کند.

---

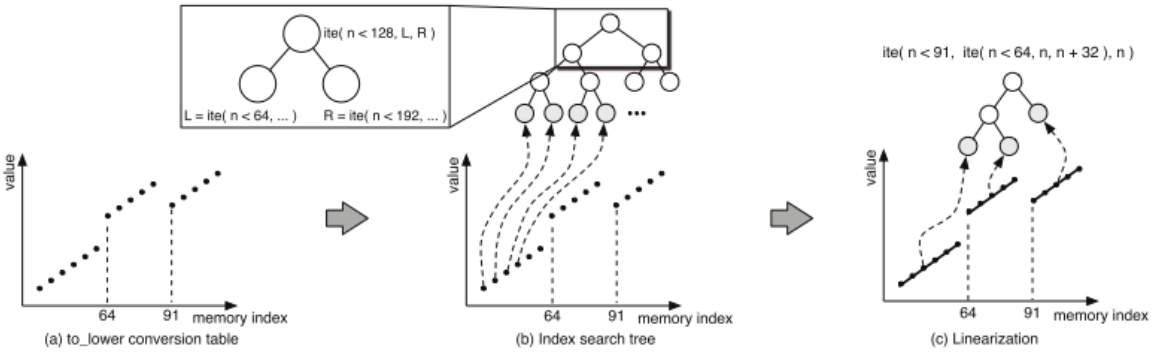
[۲] Refinement Cache

[۳] Index Search Trees

IST(E)= ite(i<addr(E_{right}),E_{left},E_{right}))

E عبارت مورد جست و جو است

و مقدار راست و چپ آن با E_{right} و E_{left} نشان داده می‌شود.



برای اینکه تعداد گره های درخت IST کاهش یابد از ایده خطی سازی پاکتی استفاده می شود. اندکس‌ها و مقادیرشان معمولا در کنار هم در یک خط قرار می گیرند. قبل از تولید درخت با یک پیش پردازش مجموعه پاکت های خطی تولید می شود و سپس با استفاده از آنها درخت تولید می شود با این کار برای مثال تعداد گره ها از ۲۵۶تا به ۳تا کاهش می یابد.

مدل کردن حافظه و استفاده از Memory Object زمانی مناسب است که اندازه M<<U باشد. اگر اندازه M از یک حد آستانه بیشتر شود، ابزار آدرس های نمادین را عددی می کند. البته این عددی کردن دلخواه صورت نمی گیرد و بر اساس سه مورد است:

۱. اگر امکان پذیر است آدرسی تولید شود که برنامه را به قسمت پوشش داده نشده ببرد.

۲. یا آدرسی تولید شود که به حافظه نمادین اشاره کند. ابزار قسمت های حافظه را به تعدادی region نمادین تقسیم می کند که این مناطق بر اساس پیچیدگی قیدهای مرتبط به آنها مرتب شده اند. ابزار سعی می کند اشاره‌گر به منطقه‌ای ساده‌تر را ایجاد کند.

۳. اگر موارد بالا قابل اجرا نبودند آدرسی معتبر به حافظه تولید خواهد شد.

```
1  typedef struct {
2    int value;
3    char * bar;
4  } foo;
5  int vulnerable (char * input)
6  {
7    foo * ptr = init;
8    buffer[100];
9    strcpy(buffer, input);
10   buffer[0] = ptr->bar[0];
11   return 0;
12 }
```