خلاصه‌ای از مقاله‌ی «تکنیک‌های تهاجمی در بررسی کد باینری[1]»

یکی از راه‌های بررسی آسیب‌پذیری در برنامه‌های کامپیوتری بررسی کد باینری‌ای است که بر روی کامپیوتر اجرا می‌شود. این امر به دو دلیل اهمیت دارد. اول آن که گاهی دسترسی به کد منبع[2] برنامه وجود ندارد و دوم آن که ممکن است برخی از خصوصیاتی که درباره‌ی کد منبع برنامه صادق است برای کد باینری تولید شده به ازای آن صادق نباشد (برای مثال طی فرآیند کامپایل ضعفی امنیتی در باینری حاصله داخل شده باشد). بدین منظور نویسندگان این مقاله ابزاری به نام angr را معرفی کرده‌اند که فرآیند بررسی و تحلیل کد باینری را اتوماتیک کرده و می‌تواند به صورت اتوماتیک اعمال Exploit Generation و Exploit Hardening را نیز انجام دهد. این ابزار از تکنیک‌های مختلف تحلیل پویا و ایستا استفاده می‌کند. از تحلیل‌های ایستا می‌توان به Static Symbolic Execution و از تحلیل‌های پویا می‌توان به Fuzzing اشاره کرد.

نویسندگان مقاله دو دلیل را برای توسعه و ارائه‌ی متن‌باز این پروژه ارائه کرده‌اند. اول این که بدون وجود چنین ابزاری محققین هر بار برای تحقیق درباره‌ی روش‌های بررسی آسیب‌های امنیتی نیاز به پیاده‌سازی ابزار خود دارند و در نتیجه زمان زیادی از ایشان تلف می‌شود. ثانیا یکسان نبودن ابزار محققان منجر می‌شود تا نتوان به درستی نتایج ایشان را تکرار کرد (تکرارپذیری نتایج یکی از اساسی‌ترین نیازمندی‌های تحقیق خوب است).

برای توسعه‌ی angr و تست کردن نتایج حاصله از آن، محققان از داده‌های DARPA که پس از Cyber Grand Challenge منتشر شده است استفاده کرده‌اند. مزیت داده‌های این مسابقه از آن جهت است که طراحان مسابقه برای ساده‌سازی محیط اجرای نرم‌افزار یک سیستم‌عامل بسیار ساده با ۷ فراخوان سیستمی ابتدایی پیاده‌سازی کرده‌اند. پس محیط و تعامل نرم‌افزار با آن بسیار ساده است؛ همچنین نرم‌افزارهای متعددی از Web Application Server تا نرم‌افزارهای پردازش برای این سیستم‌عامل تصویر توسط طراحان مسابقه طراحی شده و مورد ارزیابی شرکت کنندگان این مسابقه قرار گرفته است.

از مصالحه[3]هایی که می‌بایست در طراحی چنین ابزاری مورد توجه قرار گیرد می‌توان به مصالحه میان تکرارپذیری و پوشش کد[4] و نیز مصالحه‌ی میان دید معنایی نسبت به برنامه و مقیاس‌پذیری تکنیک اشاره کرد. تکرارپذیری بدین معناست که باید بتوان مسیری را که منتهی به Crash در نرم‌افزار می‌شود را به درستی تبیین کرد؛ از طرفی برای پوشش کد حدأکثری باید تمامی کد مورد بررسی قرار گیرد؛ یعنی تکنیکی که هم تکرارپذیر و هم دارای پوشش کد حدأکثر باشد باید تمامی مسیرهای ممکن در برنامه را بررسی کند و این امر با توجه و رشد نمایی مسیرها به هنگام پرش در جریان برنامه بسیار پرهزینه است. برای کسب دید معنایی حدأکثری نیز می‌توان اینطور استدلال کرد که باید ابزار تحلیل، توانایی پردازشی‌ای معادل توان لازم برای اجرای نرم‌افزار مورد بررسی در تمام حالات ممکن آن را داشته باشد تا دید معنایی کاملی نسبت به تمامی اجزای آن کسب کند و چنین ابزاری منطقا مقیاس‌پذیر نیست.

برای کشف آسیب‌پذیری در برنامه از تکنیک‌های متعددی استفاده شده است. این تکنیک‌ها شامل تحلیل‌های ایستا از جمله: بازیابی جریان کنترلی، مدل کردن جریان، مدل کردن داده و تحلیل‌های پویا از جمله: اجرای یکپارچه[5]، Fuzzing و اجرای سمبلیک است. به دلیل محدودیت در ارائه‌ی این خلاصه تنها به نتایج حاصله از این ابزار بسنده شده و تحلیل نتایج در ادامه آورده خواهد شد.

---

[1] (State of) The Art of War: Offensive Techniques in Binary Analisys
[2] Source Code
[3] Trade-off
[4] Code coverage
[5] Dynamic Concrete Execution

نویسندگان این مقاله تحلیل‌های ابزار خود را برروی مجموعه داده‌های DARPA اعمال کرده‌اند که نتایج آن در جدول۱ قابل مشاهده است.

جدول ۱ نتایج اعمال تمامی تکنیک‌های ابزار angr بر مجموعه داده‌های DARPA

Technique	Replayable	Semantic Insight	Scalability	Crashes	False Positives
Dynamic Symbolic Execution	Yes	High	Low	16	0
Veritesting	Yes	High	Medium	11	0
Dynamic Symbolic Execution + Veritesting	Yes	High	Medium	23	0
Fuzzing (AFL)	Yes	Low	High	68	0
Symbolic-Assisted Fuzzing	Yes	High	High	77	0
VSA	No	Medium	High	27	130
Under-constrained Symbolic Execution	No	High	High	25	346

همانطور که از نتایج پیداست بهترین نتایج از Symbolic-Assisted Fuzzing حاصل شده است. یکی از مهمترین دلایل بالابودن تعداد آسیب‌های کشف شده توسط Fuzzing آن است که بر خلاف متدهای تحلیل سمبلیک مشکل Path Explosion در آن مطرح نیست زیرا با تغییر دادن وروردی هر بار یک مسیر مشخص را طی می‌کند. بدین ترتیب قادر است بسیار بیشتر از تحلیل‌های سمبلیک در اعماق جستجو کند[۶]. از طرفی یکی از مشکلات Fuzzing آن است که نمی‌داند کدام قسمت از ورودی را باید دستکاری کند تا بتواند مسیرهایی را که تا کنون جستجو نکرده است مورد بررسی قرار دهد؛ برای رفع این مشکل از Symbolic-Assisted Fuzzing استفاده می‌شود که دید معنایی بیشتری نسبت به برنامه داشته و پرش‌ها را مورد بررسی قرار می‌دهد. بدین ترتیب می‌تواند ورودی‌ای تولید کند که مسیری غیر از آن‌هایی که توسط الگوریتم Fuzzing بررسی شده‌اند را فعال کند. سپس این ورودی در اختیار الگوریتم Fuzzing قرار می‌گیرد تا از آن استفاده کرده و سپس با تغییر آن مسیرهای دیگری را نیز فعال کند. همانگونه که مشخص است این تکنیک (Symbolic-Assisted Fuzzing) به دلیل دیدمعنایی بیشتر قادر است کد بیشتری را مورد بررسی قرار داده و آسیب‌پذیری‌های بیشتری را بیابد.

یکی از نتایج جالبی که در این مقاله به چشم می‌خورد، میزان پوشش کد روش Fuzzing است. بدین منظور باید به این نکته اشاره کرد که طراحان ابزار angr از دو ماژول CFGAccurate و CFGFast برای ساخت گراف جریان کنترلی استفاده کرده‌اند. از خصوصیات مورد توجه آنان در بررسی کارکرد این ماژول‌ها Soundness و Completeness گراف حاصله است. گراف جریان کنترلی‌ای Sound است که تمامی مسیرهای قابل طی شدن توسط نرم‌افزار را در خود داشته باشد (یک گراف کامل از بلوک‌های پایه Sound است). گراف جریان کنترلی‌ای Complete است که تمامی یال‌های موجود در آن در واقعیت قابلیت فعال شدن توسط نرم‌افزار را داشته باشند (گراف خالی با این تعریف Complete است)[۷].

آنان پس از بررسی‌های نهایی به این نتیجه رسیدند که اگر با استفاده از مسیرهای طی شده در الگوریتم Fuzzing یک گراف جریان کنترلی تشکیل دهند، گراف به دست آمده پوشش کد بیشتری نسبت به گراف حاصله از CFGAccurate و CFGFast خواهد داشت. از طرفی این گراف طبق تعریف الگوریتم Fuzzing یک گراف Complete است زیرا تمامی مسیرهای آن قابلیت طی شدن داشته و توسط الگوریتم طی شده‌اند.

---

[۶] با افزایش عمق جستجو تعداد مسیرها به صورت نمایی افزایش می‌یابد؛ در اینجا منظور از عمق تعداد دستوراتی است که بررسی شده‌اند.

[۷] در اینجا منظور از Complete بودن گراف کامل بودن به معنی داشتن تمامی یال‌های ممکن نیست. در واقع Completeness را می‌توان معیاری از کم بودن False positive در تشخیص یال‌های گراف در نظر گرفت.

مراجع:

[1] Y. Shoshitaishvili, R. Wang, C. Salls, N. Stephens, M. Polino, A. Dutcher, J. Grosen, S. Feng, C. Hauser, C. Kruegel and G. Vigna, "SOK: (State of) The Art of War: Offensive Techniques in Binary Analysis," in *IEEE Symposium on Security and Privacy (SP)*, San Jose, CA, USA, 2016.