هک کورکورانه

چکیده

نشان می‌دهیم که نوشتن از راه دور برای سرریز بافر بدون داشتن یک کپی از هدف باینری یا کد منبع، در برابر خدماتی که پس از شکست مجدد راه‌اندازی می‌شوند ممکن است. این مسئله امکان هک خدمات اختصاصی باینری، یا سرورهای منبع باز گردآوری شده به‌صورت دستی و نصب از منبع را فراهم می‌کند. تکنیک سنتی معمولا در یک فایل باینری خاص و توزیع‌شده، به‌صورت یکسان عمل می‌کند که در آن هکر، محل ابزار مفید برای برنامه‌نویسی بازگشت‌گرا (ROP) را می‌داند. ROP کورکورانه (BROP) که در این مقاله ارائه شده است به جای حمله از راه دور، ابزارهای ROP کافی برای انجام یک سیستم فراخوانی Write و انتقال آسیب‌پذیر باینری بر روی شبکه را می‌یابد و پس از بهره‌برداری، می‌تواند با استفاده از تکنیک شناخته شده‌ای تکمیل شود. بنابراین با نفوذ در اطلاعات یک بیت براساس اینکه آیا یک فرایند شکست خورده است یا نه، عملیات را شروع می‌کند. BROP نیاز به آسیب‌پذیری پشته و یک سرویس دارد که پس از شکست شروع به اجرا کند. در این مقاله Braille پیاده‌سازی شده است، بهره‌برداری کاملا خودکار، که کمتر از 4000 درخواست (20 دقیقه) در برابر آسیب‌پذیری در nginx، yaSSL + MySQL و سرور اختصاصی را متحمل است. حمله در لینوکس 64 بیتی با فضای آدرس‌دهی تصادفی (ASLR)، بدون محافظت از اجرای صفحه (NX) و پشته صورت می‌گیرد.

1. مقدمه

مهاجمان در سوءاستفاده بر روی هدف، با درجه‌ای از اطلاعات مختلف بسیار موفق عمل کرده‌اند. نرم‌افزار متن باز دردسترس است زیرا مهاجمان می‌توانند کد را برای یافتن آسیب‌پذیری پویش کنند. نرم‌افزار منبع بسته هک ممکن است برای ایجاد انگیزه در مهاجمان از طریق استفاده از تست fuzz و مهندسی معکوس به کار گرفته شود. تلاش برای درک محدودیت‌های مهاجم، این سؤال را مطرح می‌کند که: آیا مهاجمان ممکن است تلاش خود برای رسیدن به هدف و سوء استفاده از خدمات اختصاصی را نه تنها برای منبع بلکه برای کد باینری گسترش دهند؟ در نگاه اول، ممکن است چنین به نظر برسد که دست نیافتنی است زیرا سوء استفاده به داشتن یک کپی از هدف دودویی برای استفاده در برنامه‌نویسی بازگشت‌گرا (ROP) نیاز دارد [1]. ROP لازم است، زیرا در سیستم‌های مدرن، حفاظت غیراجرایی (NX) از حافظه تا حد زیادی مانع از تزریق کد حملات می‌شود. برای پاسخ به این سوال، با ساده‌ترین آسیب‌پذیری ممکن شروع می‌کنیم: پشته سرریز می‌شود. متاسفانه این مسئله هنوز هم در نرم‌افزارهای محبوب (به‌عنوان مثال، در nginx CVE-2013- 2028 [2]) قابل اعمال است. بنابراین تنها می‌توان چنین حدس زد که باگ‌ها از نرم‌افزار اختصاصی دور بماند، که در آن منبع (و دودویی) تحت بررسی عمومی و دقیق متخصصان امنیت نیست. بااین‌حال، این امکان برای یک مهاجم فراهم است که از تست fuzz برای یافتن باگ‌ها از طریق رابط‌های سرویس شناخته شده یا طراحی معکوس استفاده کند. در روش دیگر، مهاجمان می‌توانند آسیب‌پذیری‌های شناخته شده در کتابخانه‌های محبوب را (به‌عنوان مثال، SSL یا تجزیه‌کننده PNG) مورد هدف قرار دهند که ممکن است در خدمات اختصاصی مورد استفاده قرار گیرند. مهم‌ترین چالش این مسئله، توسعه‌ی یک روش برای بهره‌برداری از این آسیب‌پذیری‌ها است زمانی‌که اطلاعات در مورد هدف دودویی محدود است.

یکی از مزیت‌هایی که اغلب حملات دارند این است که بسیاری از سرورها فرآیندهای خود را پس از یک شکست در جهت موفقیت دوباره مجددا راه‌اندازی می‌کنند. نمونه‌های قابل توجه عبارتند از، آپاچی، وب سرور nginx، سامبا و OpenSSH. اسکریپت Wrapper مانند mysqld_safe.sh یا daemon ها مانند systemd، این قابلیت را دارند. متعادل‌کننده‌های بار به‌طور فزاینده‌ای شایع هستند و اغلب اتصالات را به تعداد زیادی از میزبان‌هابا

پیکربندی‌های یکسان برای اجرای فایل‌های باینری برنامه توزیع می‌کنند. بدین ترتیب، موقعیت‌های بسیاری وجود دارد که در آن یک مهاجم به‌طور بالقوه برای ایجاد بهره‌برداری (تا تشخیص) بی‌نهایت تلاش می‌کند.

در این مقاله یک حمله جدید ارائه می‌کنیم، برنامه‌نویسی بازگشت‌گرا کور (BROP)، که از مزیت‌های این موارد، سوء استفاده کورکورانه از خدمات اختصاصی باینری و منابع ناشناخته است. در حمله BROP، نرم‌افزار سرور با یک آسیب‌پذیری پشته فرض می‌شود و فرآیندی که پس از خرابی دوباره راه‌اندازی می‌شود. حمله در لینوکس 64 بیتی با ASLR (فضای آدرس‌دهی تصادفی)، حافظهnonexecutable (NX) و پشته فعالیت می‌کند. با این حال تعداد زیادی از سرورها را پوشش می‌دهد که نمی‌توانیم در حال حاضر در سیستم ویندوز مورد هدف قرار دهیم چون باید حمله به ویندوز ABI منطبق شود. این حمله توسط دو تکنیک جدید فعال می‌شود:

1) خواندن از پشته تعمیم‌یافته: این تعمیم شناخته شده برای پشته مورد استفاده قرار می‌گیرد و موجب ذخیره آدرس بازگشت به‌منظور ASLR پیش‌فرض در سیستم 64 بیتی می‌گردد، حتی زمانی که مکان اجرای مستقل (PIE) استفاده می‌شود.

2) ROP کور: این روش ابزار ROP را از راه دور در مکان قرار می‌دهد.

هر دو روش، ایده‌ی استفاده از آسیب‌پذیری پشته را برای نشت اطلاعات براساس اینکه آیا یک سرور با شکست مواجه می‌شود یا نه، به اشتراک می‌گذارند. روش خواندن از پشته، بایت‌های پشته را بایت به بایت با حدس دوباره نویسی می‌کند، تا زمانی که عملیات درست یافت شود و سرور دچار شکست نشود و عملیات خواندن (با نوشتن) پشته به طور موثر انجام گیرد. حمله کورکورانه ROP ابزارهای کافی برای انجام سیستم پاسخ write را پس از انتقال باینری سرور از حافظه به سوکت مهاجم می‌یابد. در این ح، ASLR و NX شکست خورده‌اند و بهره‌برداری می‌تواند با استفاده از تکنیک‌های شناخته شده انجام گیرد.

حمله BROP سوء استفاده قوی برای اهداف عمومی را نوسط سه سناریو جدید انجام می‌دهد:

1) خدمات بسته باینری و اختصاصی هک. ممکن است متوجه یک تصادف در هنگام استفاده از یک سرویس از راه دور و یا کشف از طریق آزمایش fuzz از راه دور شویم.

2) هک یک آسیب‌پذیری در یک کتابخانه منبع باز تصور می‌شود که در یک سرویس اختصاصی باینری استفاده شود. برای مثال کتابخانه محبوب SSL ممکن است آسیب‌پذیری پشته داشته باشد و ممکن است حدس بزنیم که توسط یک سرویس اختصاصی استفاده می‌شود.

3) هک یک سرور منبع باز برای منابع باینری ناشناخته است. این امر به صورت نصب دستی و راه‌اندازی یا توزیع‌هایی مانند Gentoo انجام می‌شود.

هر سه سناریو را مورد ارزیابی قرار دادیم. در حالت ایده‌آل، برای سناریوی اول تکنیک‌های ما در برابر خدمات تولید که هیچ اطلاعاتی در مورد نرم‌افزار نگه نمی‌دارد مورد تست قرار گرفت، اما به دلایل قانونی محدودیت‌هایی وجود دارد. برای شبیه‌سازی یک سناریو، روش خودمان را در برابر یک سرویس اختصاصی مورد آزمون قرار دادیم که هیچ اطلاعاتی در مورد منبع آن، باینری یا عملکرد نداشتیم. برای سناریوی دوم، یک آسیب‌پذیری واقعی در کتابخانه yaSSL را هدف قرار دادیم [3]. این کتابخانه در گذشته توسط MySQL استفاده شده بود و ما آن را به‌عنوان برنامه میزبان در نظر می‌گیریم. برای سناریوی سوم، یک (2013) آسیب‌پذیری اخیر در nginx [2] را مورد هدف قرار دادیم و به‌صورت عمومی بهره بردیم که به یک فایل باینری خاص بستگی نداشت. این امر به‌ویژه در هر توزیع و نسخه آسیب‌پذیر وب سرور nginx بدون نیاز به حمله برای بهره‌برداری از هر توزیع و نسخه ترکیبی (همان‌گونه که امروزه انجام می‌شود) مفید است و به کار می‌رود.

در این مقاله یک ابزار امنیتی جدید، Braille، پیاده‌سازی کردیم که باعث می‌شود حملات BROP کاملا خودکار عمل کنند. Braille می‌تواند بر روی عملکرد سرور آسیب‌پذیر در حدود 4000 درخواست، یک فرایند که در کمتر از 20 دقیقه کامل می‌شود و در برخی شرایط، فقط در چند دقیقه بهترین کارآیی را داشته باشد. یک مهاجم تنها نیاز به ارائه یک تابع دارد که یک درخواست با حداقل طول برای شکست سرور ایجاد کند و یک رشته توسط Braille اضافه کند. همچنین کارکرد این روش به عملکرد یک بیت براساس اینکه آیا سرور شکست می‌خورد یا نه، بستگی دارد.

کارهای انجام شده عبارتند از:

1) یک روش برای شکست ASLR بر روی سرورها (خواندن عمومی از پشته).

2) روش برای یافتن از راه دور ابزارهای ROP(BROP) به‌طوری‌که نرم‌افزار، زمانی که باینری ناشناخته است مورد حمله قرار گیرد.

3) Braille: ابزاری برای بهره‌برداری خودکار از ورودی داده شده در مورد اینکه چگونه یک پشته در سرور سرریز می‌شود.

4) اولین بهره‌برداری عمومی (بنابه دانش ما) برای آسیب‌پذیری اخیر وب سرور nginx، در حالت عمومی، 64 بیتی و شکستASLR (کامل / PIE) ، canaries و NX است.

5) پیشنهادات برای دفاع در برابر حملات BROP. به‌طورخلاصه، ASLR باید به تمام بخش‌های اجرایی (PIE) اعمال شود و پس از هر شکست (در تقابل با سرورهای fork) مجددا به حالت تصادفی برگردد. حفاظت باینری در برابر مهاجم یا تغییر هدفمند ممکن است یک اقدام متقابل امنیتی موثر نباشد.

2. تاریخچه مختصری از سرریزهای بافر

سرریزهای بافر یک آسیب‌پذیری کلاسیک با تاریخچه طولانی در سوء استفاده‌ها هستند [4]. از لحاظ مفهومی، حمله به آنها نسبتا آسان است. به‌عنوان‌مثال، یک برنامه آسیب‌پذیر ممکن است داده‌ها را از شبکه خوانده و در یک بافر جای دهد. سپس، با فرض اینکه برنامه فاقد مرزهای بررسی کافی برای محدود کردن اندازه ورودی داده است، مهاجم می‌تواند در پایان بافر حافظه را مجددا بازنویسی کند. در نتیجه، حالت بحرانی جریان کنترل، مانند بازگشت آدرس یا اشاره‌گر تابع، می‌تواند دستکاری کرد. سرریزهای پشته بافر تمایل ویژه‌ای به بازگشت آدرس‌ها به طور ضمنی در نزدیکی حافظه به‌دلیل عملکرد فراخوانی دارند. با این‌حال، حملاتی که بافر را مورد هدف قرار می‌دهند عملی می‌باشند [5].

در روزهای اولیه سرریز بافر پشته، برای یک مهاجم بسیاررایج بود که شامل کدهای مخرب به‌عنوان بخشی مورد استفاده در سرریز بافر باشد. در نتیجه، مهاجم می‌تواند به سادگی آدرس برگشت را به مکانی شناخته شده در مجموعه پشته نگاشت کند و دستورالعمل‌هایی را که در بافر ارائه شده‌اند اجرا کند. چنین حملاتی نظیر "تزریق کد" دیگر در ماشین‌های معاصر امکان‌پذیر نیست چرا که پردازنده‌های مدرن و سیستم عامل‌ها در حال حاضر توانایی علامت زدن

صفحات حافظه داده‌ها به‌عنوان غیر قابل اجرا (به‌عنوان مثال، NX در x86) را دارند. در نتیجه، اگر یک مهاجم سعی در اجرای کد بر روی پشته داشته باشد، تنها یک استثنا اتفاق خواهد افتاد.

روش های نوآورانه، که با نام برنامه‌نویسی بازگشت‌گرا (ROP) شناخته شده‌اند [1]، برای دفاع در برابر شکست‌ها بر روی حافظه‌های غیر قابل اجرا توسعه داده شده‌اند. که از طریق ایجاد ارتباط کوتاه با هم کار می‌کنند که در حال حاضر در فضای آدرس برنامه وجود دارند. چنین قطعه کدهایی، با نام gadgets شناخته می‌شوند و می‌توانند به شکل محاسبات دلخواه ترکیب شوند. در نتیجه، مهاجمان می‌توانند از ROP برای به‌دست آوردن کنترل برنامه بدون هیچ وابستگی به کد تزریق شده استفاده کنند. گاهی اوقات تغییرات ساده در ROP امکان‌پذیر است. به‌عنوان مثال، با حملات بازگشت به libc، عملکرد یک کتابخانه سطح بالا را می‌توان به‌عنوان آدرس برگشت استفاده کرد. به‌طورخاص، تابع ()system برای مهاجمان مفید است زیرا می‌توانند کد دلخواه خود را تنها با یک آرگومان اجرا کنند [6]. این حملات در سیستم‌های 32 بیتی که در آن آرگومان‌ها بر روی پشته عمل می‌کند بسیار مؤثر بودند و در حال حاضر تحت کنترل مهاجم هستند. بر روی سیستم‌های 64 بیتی، که در آن آرگومان‌ها بر روی رجیسترها عمل می‌کند، gadget های اضافی برای ثبت جمعیت مورد نیاز است.

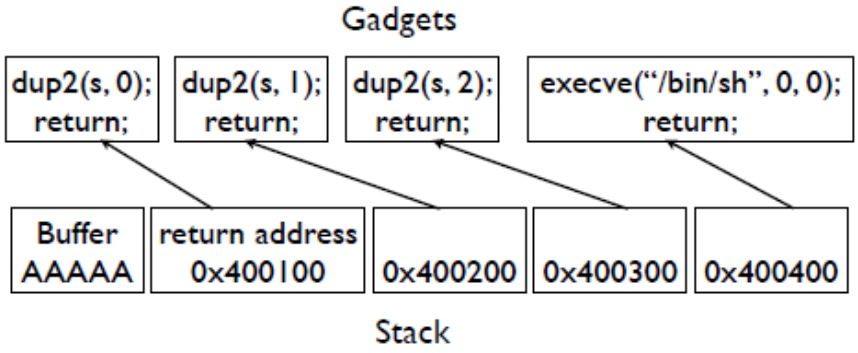
چیدمان تصادفی فضای آدرس‌دهی(ASLR)، [7] و [8] به‌عنوان یک دفاع در برابر حملات سرریز بافر معرفی می‌شود و با تصادفی‌سازی محل کد و بخش‌های داده در فضای آدرس فرآیند کار می‌کند. در بسیاری از پیاده‌سازی‌های تصادفی قطعه کد که تنها به کتابخانه اعمال می‌شوند، تصادفی‌سازی پر فضای آدرس نیز ممکن است. ASLR یک چالش عمده برای مهاجمان ایجاد می‌کند چراکه مکان‌های آدرس کد (و یا حتی پشته) برای پیش‌بینی غیرممکن هستند. متاسفانه، بر روی سیستم عامل 32 بیتی، ASLR توسط تعدادی از بیت‌های موجود (معمولا 16) برای محدود تصادفی ایجاد می‌شود. در نتیجه، حملات brute-force کاملا موثر است [9]. با این‌حال، بر روی سیستم عامل 64 بیتی به‌طور معمول بیت تصادفی بسیاری وجود دارد که باید امکان‌پذیر باشد. در چنین مواردی، ASLR می‌تواند استفاده نشود، اما تنها زمانی که همراه با یک آسیب‌پذیری ناشی از نشت اطلاعات در مورد فضا آدرس‌دهی باشد، مانند فرمت رشته [10].

علاوه بر فضای آدرس بزرگتر برای ASLR و نیاز به ابزارهای اضافی برای پر کردن آرگومان‌ها، سیستم‌های 64بیتی یک عارضه سوم برای حملات هستند. زیرا معماری موجب محدودیت آدرس‌های مجازی به 48 بیت می‌شود، اشاره‌گر حافظه در سطح کاربر برای بایت‌های با ارزش صفر مورد نیاز است. این صفرها موجب ختم اولیه سرریزها با تکیه بر عملیات رشته مانند strcpy می‌شوند.

Canaries [11] دفاع مشترک در برابر حملات سرریز بافر است. Canaries نمی‌توانند از سرریزهای بافر جلوگیری کنند، اما می‌توانند سابقه و پایان برنامه را قبل از این که یک مهاجم بر جریان کنترل تاثیر بگذارد شناسایی کنند. مثلا، با پشته canaries، یک مقدار پنهان و نامعلوم، پیش از هر اشاره‌گر ذخیره شده و آدرس بازگشت تعیین می‌شود. سپس، هنگامی که یک تابع مجددا فراخوانی می‌شود، ارزش آن بررسی می‌شود تا از عدم تغییر آن اطمینان حاصل کند.

```
dup2(s, 0);
dup2(s, 1);
dup2(s, 2);
execve("/bin/sh", 0, 0);
```

شکل 1. سوکت، shellcode را مجدا استفاده می‌کند و مسیر ورودی استاندارد، stdout و stderr را به سوکت و اجرا یک shell تغییر می‌دهد.



شکل 2. نسخه ROP برای استفاده مجدد در سوکت shellcode. ابزار با قرار دادن آدرس خود در پشته وارد زنجیره می‌شوند.

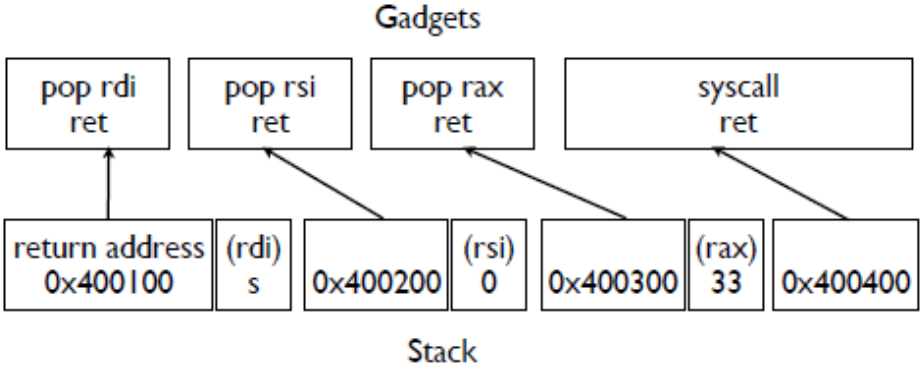
این امر می‌تواند از سرریز بافر پشته به دلیل سوء استفاده جلوگیری کند، زیرا یک مهاجم باید به‌درستی مقادیر پنهان را به منظور استفاده دقیق برنامه از آدرس برگشت بازنویسی کند. بااین‌حال، فقط به‌عنوان ASLR، canaries می‌تواند از شکست در برابر آسیب‌پذیری‌های اضافی که از نشت اطلاعات در مورد مقادیر پنهان نشات می‌گیرد مقابله کند. طرح پشته حافظه می‌تواند یک ایده مهم برای پیاده‌سازی canaries باشد. یکی از رویکردهای شایع، قرار دادن تمام بافرها در بالای فریم است، بنابراین اگر سرریز اتفاق بیافتد بازنویسی مجدد متغیرها امکان‌پذیر نخواهد بود [12]. انگیزه‌ی این عمل محافظت از اشاره‌گر است زیرا گاهی اوقات می‌توان آنها را مجددا و به دلخواه بازنویسی استفاده کرد [13]. متاسفانه، canaries یک راه حل مناسب نیست، حتی با اقدامات احتیاطی، گاهی اوقات ساختار یک سرریز بافر می‌تواند به مهاجم این اجازه را بدهد که به حالت بحرانی به طور مستقیم دسترسی یابد، به‌عنوان مثال، اشاره‌گر ناامنی که در yaSSL اتفاق افتاد [3].

3. آموزش ROP

قبل از بحث در مورد روش کورکورانه ROP، ابتدا با ROP آشنا می‌شویم. سوء استفاده مدرن به شدت به ROP متکی است. هدف از ROP ساخت یک توالی است که معمولا براساس کد موجود عمل می‌کند. هنگامی که یک پوسته اجرا می‌شود، مهاجم می‌تواند دستورات بیشتری را برای ادامه حمله اجرا کند. شکل 1 shellcode معمولی را نشان می‌دهد که سوکت مهاجم را به ورودی استاندارد، خروجی، خطا و اجرای یک shell خط لوله می‌کند.

البته تزریق shellcode دیگر امکان‌پذیر نیست چرا که این روزها حافظه قابل نوشتن (به‌عنوان مثال، پشته) غیرقابل اجرا است، به همین ترتیب ROP بجای آن استفاده می‌شود. شکل 2 نشان می‌دهد که چگونه ROP می‌تواند برای ایجاد shellcode که قبلا در شکل 1 نشان داده شده است استفاده شود. پشته سرریز می‌کند به‌طوری‌که آدرس تمام ابزارهای موجود در دنباله هستند. هر ابزار با بازگشت به‌طوری‌که ابزار بعدی بتواند اجرا شود به پایان می‌رسد.

در عمل، هر ابزار ROP یک دنباله کوتاه از دستورالعمل‌های خاتمه دستگاه با بازگشت خواهد بود.



شکل 3. زنجیره ROP برای (s،0) dup2. تعداد پاسخ سیستم نیاز به RAX و dup2 در سیستم پاسخ # 33 دارد.

آرگومان‌های RDI و RSI به تصویب رسیده‌اند چراکه مهاجم در حال حاضر پشته را کنترل می‌کند، pop برای بارگذاری رجیسترها از مقادیر بر روی پشته استفاده می‌شود.

اجرای یک پاسخ سیستم ساده مانند dup2 به چندین ابزار نیاز دارد چرا که آرگومان‌ها از ریجسترها عبود خواهند کرد. شکل 3 ابزارهای مورد نیاز برای dup2 را نشان می‌دهد. رجیستر RDI و RSI دو آرگومان سیستم فراخوانی، و RAX تعداد تماس سیستم را کنترل می‌کند. ریجسترها را می‌توان با استفاده از ابزارهای pop و قرار دادن مقادیر برای بارگذاری در پشته کنترل کرد. با وصل زنجیروار این ابزارها، shellcode کامل در نهایت ساخته می‌شود.

4. سرریزهای بافر

در اکثر سیستم عامل‌های جدید، که در آن‌ها NX و ASLR شایع هستند، یک مهاجم باید حداقل دو نیازمندی به‌منظور به‌دست آوردن کنترل کامل برنامه برای اجرای از راه دور را انجام دهد:

1) برای شکست NX، مهاجم باید بدانید که ابزار در داخل برنامه اجرایی کجا قرار دارد.

2) برای شکست ASLR، مهاجم باید مکانی را که بخش متن اجرایی در ان واقع است در حافظه لود کند.

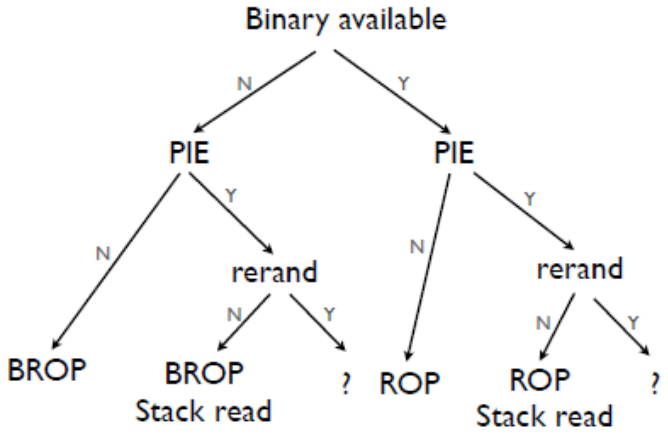
این الزامات به‌راحتی در سیستم 32 بیتی [9]، [14] از طریق حدس زدن ساده عملی هستند. این مورد برای سیستم‌های 64 بیتی عملی نیست. در واقع، بسیاری از سوء استفاده‌های عمومی سیستم‌های 32 بیتی را مورد هدف قرار می‌دهند. هدف از حمله BROP

دور زدن این شرایط در سیستم‌های 64 بیتی است. از این رو، بقیه این بحث منحصرا به حملات 64 بیتی اختصاص می‌یابد.

نخستین نیازمندی در عمل بدان معنی است که مهاجم باید یک نسخه از باینری آسیب‌پذیر برای پیاده‌سازی و یافتن ابزار داشته باشد. بنا به سطح دانش ما، حمله BROP پیشنهادی ما اولین روش عمومی است که می‌تواند برای شکست NX زمانی که کد باینری به‌طورکامل در دسترس نیست مورد استفاده قرار گیرد.

شکست ASLR نیز یک چالش قابل توجه و بدون BROP است، اما برخی از استراتژی‌های ممکن وجود دارد. در مرحله اول، نشت اطلاعات ممکن است موقعیت مکانی آدرس یک کد باشد. در مرحله دوم، ممکن است بهره‌برداری از هر کد باقی‌مانده در سراسر اجرا ممکن باشد. مثلا، در لینوکس، معمولا کد اجرایی به یک آدرس ثابت نگاشت می‌شود حتی اگر کتابخانه‌های پویا و مناطق دیگر حافظه داده‌ها با ASLR تصادفی‌سازی شده باشند. در نتیجه، مهاجم به‌سادگی می‌تواند ROP را به طور مستقیم به بخش متنی برنامه اعمال کند. علاوه‌براین، در برخی از سیستم عامل‌ها، مانند ویندوز، کتابخانه‌هایی که با هم ناسازگار هستند و با ASLR به اشتراک گذاشته می‌شوند وجود دارد، در نتیجه چنین کتابخانه‌ای به مکانی ثابت نگاشت می‌شود.

در لینوکس، امکان به کار بردن ASLR در کل فضای آدرس‌دهی، از جمله بخش متنی برنامه، با فعال کردن PIE وجود دارد.



شکل 4. تکنیک‌های موردنیاز برای حمله به سناریوهای مختلف 64 بیتی. BROP و خواندن از پشته به ما کمک می‌کنند.

با استفاده از GCC، این مورد از طریق نشانه‌ی -pie قابل دستیابی است. استفاده از PIE در مطالعات قبلی [15] توصیه شده است، اما متاسفانه، به طورگسترده‌ای در کارهای گذشته بیان نشده است. زمانی که PIE فعال باشد، علیرغم پیشنهاد حمله ما مبنی بر تعمیم خواندن از پشته، که می‌تواند برای شکست ASLR مورد استفاده قرار گیرد، تکنیک عمومی 64 بیتی و شناخته شده‌ای وجود ندارد.

شکل 4 نشان می‌دهد که چگونه حمله BROP حالت تکنیک 64 بیتی را بهبود می‌بخشد. امروزه تکنیک‌های (ROP) زیادی برای حمله به سرور 64 بیتی وجود دارد ولی تنها زمانی که باینری دقیق برای مهاجم دردسترس باشد و PIE استفاده نکند به کار می‌روند. روش پیشنهادی ما مبنی بر خواندن از پشته، امکان حمله به سرور PIE را فراهم می‌سازد که پس از یک شکست مجدد تصادفی‌سازی نشده است (به‌عنوان مثال، fork بدون execve). علاوه براین حمله BROP امکان هک سیستم را که در آن باینری ناشناخته است فراهم می‌کند. در تمام موارد، حمله BROP نمی‌تواند سرور PIE را که تصادفی‌سازی نشده، مورد هدف قرار دهد (به‌عنوان مثال، execve) پس از شکست.

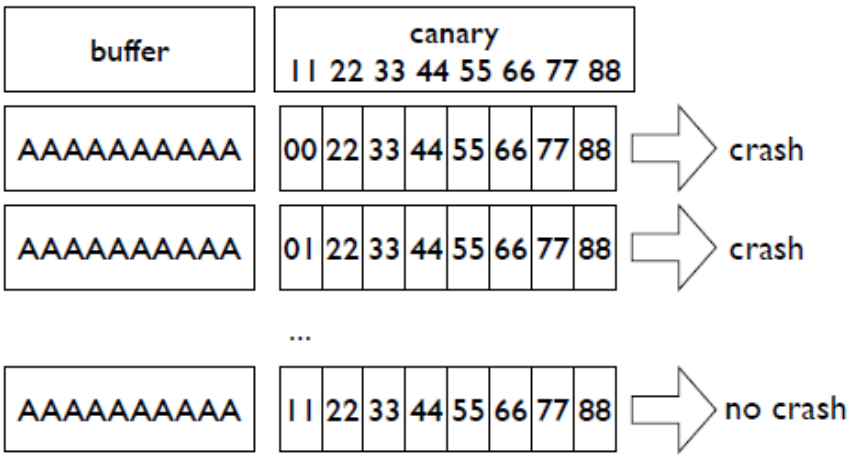
هک بدون دانش باینری حتی در مواردی که به‌طور کامل کورکورانه نیست مفید است (به‌عنوان مثال، منبع باز) زیرا باعث نوشتن عمومی و سوء استفاده قوی می‌شود و در تمام توزیع‌ها قابل اعمال است. امروزه، مهاجم‌ها نیاز به جمع‌آوری اطلاعات دقیق (به‌عنوان مثال، فایل‌های باینری) برای تمام ترکیبات ممکن از نسخه‌های توزیع و نسخه‌های نرم‌افزارهای آسیب پذیر برای عملکرد بهتر در هر مورد هستند. ممکن است تصور شود که مهاجمان بیشتر با ترکیب‌های محبوب سروکار دارند. مفهوم کار ما این است که بیشتر توزیع‌ها، با حفاظت کمی (از طریق ابهام) در سرریز بافر روبه‌رو هستند.

5. محیط BROP

حمله برنامه‌نویسی از راه دور کورکورانه (BROP) از مفروضات زیر پیروی می‌کند و به محیط زیر نیاز دارد:

- آسیب‌پذیری پشته و دانش کنترل و راه‌اندازی آن.
- برنامه سرویس‌دهنده یا سرور که پس از شکست مجدد شروع به کار می‌کند.

مدل تهدید برای حمله BROP، یک مهاجم است که رشته ورودی را با توجه به شکست سرور و اشکال در سرریز می‌داند. مهاجم باید قادر به بازنویسی بایت‌های یک متغیر از جمله اشاره‌گر دستور بازگشت باشد. مهاجم نیاز به دانستن منبع یا دودویی سرور ندارد. مهاجم باید قادر به شکست چندین باره سرور به هنگام انجام حمله باشد و سرور باید مجدد راه‌اندازی شود.



شکل 5. خواندن از پشته. یک بایت تنها در پشته با حدس X رونویسی می‌شود. اگر سرویس دچار شکست شود، مقدار اشتباه حدس زده شده است. درغیراین صورت، پشته با همان مقدار رونویسی و بدون شکست رخ می‌دهد. پس از حداکثر 256 تلاش، مقدار صحیح حدس زده می‌شود. این فرایند برای بایت بعدی در پشته تکرار می‌شود.

اگر سرور با پرچم PIE کامپایل شده باشد، سرور باید یک انشعاب بدون استفاده از execve برای راه اندازی مجدد باشد. این روش برای سرریز نیز درست است چرا که در آن canaries باید اصلاح شود. همچنین مهاجم قادر به تشخیص شکست سرور در موعد مقرر است، به‌عنوان مثال، با توجه به این که سوکت بدون دریافت پاسخ بسته می‌شود.

6. طرح کلی حمله

حمله BROP شامل مراحل زیر است:

1) خواندن از پشته: خواندن از پشته برای نشت canaries و برگرداندن آدرس برای شکست ASLR.

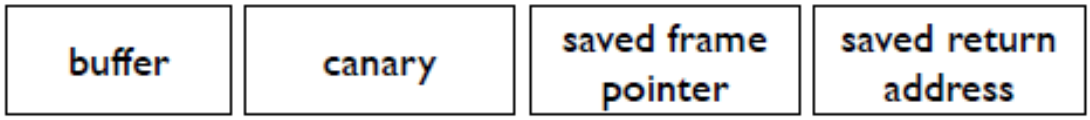
2) ROP کورکورانه: یافتن ابزار کافی برای فراخوانی write و کنترل آرگومان‌های آن.

3) ایجاد بهره‌برداری: هدر رفت کافی باینری برای یافتن ابزار کافی جهت ایجاد یک کد و راه‌اندازی نهایی بهره‌برداری. مرحله اول، همانند آدرس نقطه شروع برای اسکن ابزارهای پیدا شده ضروری است. سپس ابزار برای دستور write پیدا می‌شوند. پس از آن، باینری بر روی شبکه‌ای از حافظه منتقل می‌شود و قادر به اعمال تکنیک‌های شناخته شده جهت بهره‌برداری نهایی است.

7. خواندن از پشته: تصادفی‌سازی مجدد ASLR

سوء استفاده باید براساس یک روش شکست ASLR برای تنظیمات بنا شده باشد که در آن PIE استفاده شده است. در حال حاضر ما یک روش جدید خواندن از پشته ارائه می‌کنیم که تعمیم یافته یک تکنیک شناخته شده مورد استفاده برای نشت canaries است. این روش در مواردی که باینری شناخته شده است و حمله کامل BROP الزامی نیست بسیار مفید است. ایده اصلی در canaries سرریز یک بایت، بازنویسی بایت canaries با مقدار x است. اگر x درست بود، سرور شکست نمی‌خورد. این الگوریتم برای 256 مقدار بایت (به طور متوسط128) امکان‌پذیر است. حمله برای بایت بعدی تا زمانی که همه 8 بایت (در 64 بیتی) به بیرون درز کنند ادامه می‌یابد. شکل 5 حمله را نشان می‌دهد. پس از canary، عموما اشاره‌گر فریم ذخیره می‌شود و پس از آن آدرس بازگشت ذخیره می‌شود، به‌طوری‌که سه کلمه باید خوانده شوند. شکل 6 طرح پشته را نشان می‌دهد.

چند پیچیدگی خاص وجود دارد که به خواندن از پشته اعمال می‌شود اما برای خواندن canary مناسب نیست.



شکل 6. نمونه طرح پشته. Canary از رجیسترهای ذخیره شده محافظت می‌کند. اگر بازنویسی (توسط یک سرریز) صورت گیرد برنامه قبل از بازگشت به آدرس برگشت ذخیره شده از کار می‌افتد.

جدول1. میانگین درخواست لازم برای ASLR در مقابل روش پیشنهادی ما برای خواندن از پشته.

Platform	Entropy	Brute Force	Stack Reading
32-bit Linux	16-bits	2^{15}	512
64-bit Linux	28-bits	2^{27}	640
64-bit Mac OS X	16-bits	2^{15}	640

به‌طورکلی هر چند، خواندن از پشته لزوما دستورالعمل دقیق اشاره‌گر ذخیره شده در پشته را برنمی‌گرداند. این امکان وجود دارد که مقدار متفاوتی بسته به اینکه آیا مقدار دیگری برنامه را بدون ایجاد شکست اجرا می‌کند برگرداند. به‌عنوان مثال، فرض کنید که x400010 مقدار ذخیره شده در پشته و x400007 مقدار فعلی باشد. ممکن است که برنامه اجرای بدون شکست را نگه داشته باشد و x400007 از خواندن پشته بدست آمده باشد. بنابراین بهتر است که مهاجم مقدار معتبر در محدوده قطعه text را نه برای یک مقدار خاص جستجو کند.

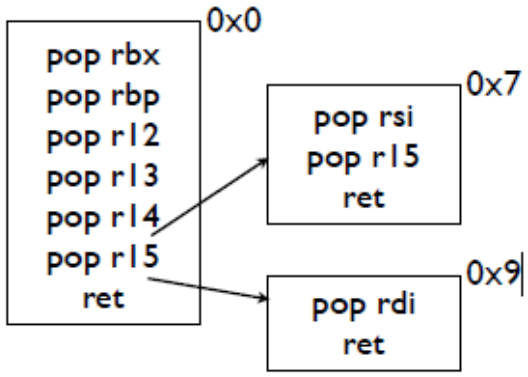
ممکن است که خواندن از پشته، آدرسی در قطعه text برنامه را برنگرداند، بلکه آدرس کتابخانه را برگرداند. این مسئله زمانی می‌تواند اتفاق بیافتد، که آسیب‌پذیری نهفته در کتابخانه یا مخاطبین وجود داشته باشد. این مسئله خوب است چرا که ابزارها می‌توانند در کتابخانه یافت شوند. بنابراین پشته می‌تواند برای یافتن آدرس‌های برگشت، اگر مورد نیاز باشد خوانده شود.

در سیستمهای 64 بیتی X86تنها بخشی از فضای آدرس به سیستم عامل در دسترس اختصاص می‌یابد (فرم استاندارد آدرس). این مسئله این امکان را فراهم می‌کند که به جستجوی چند بایت به هنگام خواندن اشاره‌گر باشیم. برای فرآیندهای موجود در فضای کاربر دو بایت بالا همواره صفر است. در واقع، بر روی لینوکس بایت سوم 0x7f برای کتابخانه‌ها و پشته است. باینری اصلی و پشته معمولا در مقدار 0x00 برای اجرای بدون پرچم PIE ذخیره می‌شود. بنابراین می‌توانیم به‌طور متوسط از سه بایت (384 درخواست) به هنگام خواندن آدرس صرفه نظر کنیم.

جدول 1 پیچیدگی استفاده از خواندن از پشته را در مقابل حملات brute-force استاندارد نشان می‌دهد. سیستم‌عاملهای 32 بیتی و 64 بیتی را مقایسه می‌کنیم. واضح است که حملات Bruteforce در لینوکس به 64 بیتی عملی نیستند و مهاجم برای دور زدن ASLR باید به روش‌های دیگر متوسل شود. بسیاری از حملات به

تصادفی‌سازی (بدون PIE) فایل‌های باینری وابسته اند و در لینوکس مشترک هستند. به‌طور مشابه، سوء استفاده از ویندوز، فایل‌های باینری مورد حمله را مجدد تصادفی‌سازی می‌کند. دیگر حملات اشاره‌گر برای نشات اشاره‌گرها که نیاز به آسیب‌پذیری دیگری دارند استفاده می‌شود.

واقعیت این است که خواندن از پشته موجب می‌شود که مهاجمِ محیط BROP وجود داشته باشد و سرریز پشته، علیرغم باگ‌های تصادفی، در حال وقوع باشد. باگ همانند یک اشاه‌گر تهی ممکن است منجر به شکست برای همه بایت‌های ممکن شود، یا همراه با عدم شکست برای چندین مقدار ممکن باشد.



شکل 7. ابزارBROP . اگر در 0X7 عبور کند، به ابزار pop rsi دست می‌یابد و افست 0x9 می‌شود ودر نتیجه ابزار pop rdi به دست می‌آید. این دو ابزار دو آرگومان اول را کنترل می‌کنند. با یافتن یک ابزار واحد (ابزارBROP) در واقع می‌توان دو ابزار مفید یافت.

کلمات بازگردانده شده توسط خواندن از پشته، شواهد بیشتری در مورد حمله BROP ارائه می‌کنند چرا که مقادیر را می‌توان تا حدودی به خوبی بررسی کرد: به‌عنوان مثال، یک canary تصادفی (که همیشه با صفر در لینوکس شروع می‌شود)، یک اشاره‌گر قاب و آدرس بازگشت با بیت بالا شناخته شده (0x40 برای غیر PIE و یا 0x7f).

8. حمله BROP

حمله BROP اجازه‌ی نوشتن به‌منظور سوء استفاده و بدون داشتن هدف باینری را می‌دهد. این روش تکنیک‌هایی برای یافتن ابزار از راه دورROP و بهینه‌سازی برای حمله کاربردی را معرفی می‌کند.

A. قطعاتی از پازل

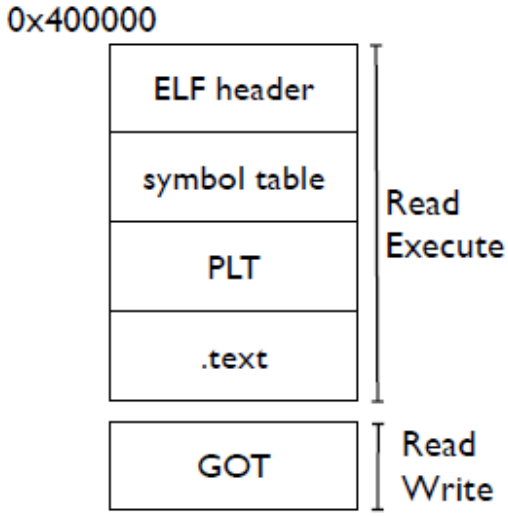
هدف، یافتن ابزار کافی با استناد به write است. بعد از این، باینری را می‌توان از حافظه به شبکه برای یافتن ابزارهای بیشتر انتقال داد. سیستم فراخوانی write سه آرگومان را در پی دارد: سوکت، بافر و طول. آرگومان‌ها از رجیسترهای RDI، RSI و RDX عبور می‌کنند و تعداد پاسخ سیستم در ثبات RAX ذخیره می‌شود. بنابراین ابزارهای زیر مورد نیاز است:

```
1)  pop rdi; ret (socket)
2)  pop rsi; ret (buffer)
3)  pop rdx; ret (length)
4)  pop rax; ret (write syscall number)
5)  syscall
```

درحالی‌که حمله‌ای که همه این ابزارهای ممکن را بیابد (به بخش هشتم مراجعه کنید) برای اولین بار یک نسخه بهینه‌سازی توصیف می‌کند که باعث حمله عملی‌تر می‌شود.

اولین بهینه‌سازی، ابزار BROP است. همانطور که در شکل 7 نشان داده شده است، ابزار BROP برای بازیابی تمام رجیسترهای ذخیره شده بسیار معمول است. تجزیه ناصحیح آن، عملکر pop RDI و pop rsi را تحت تاثیر قرار می‌دهد. بنابراین با یافتن یک ابزار واحد، دو دستگاه برای کنترل دو آرگومان اول می‌یابیم.

بهینه سازی دوم، یافتت یک call write است. به جای پیدا کردن دو ابزار مورد نظر (pop RAX؛ ret و فراخوانی سیستمی) می‌توانیم تنها یک دستور call write بیابیم. پیدا یک مکان مناسب برای call write موجب یافتن جدول لینک فرآیندهای برنامه (PLT) می‌گردد. PLT یک جدول پرش استفاده شده برای اتصال پویا شامل همه تماس‌های کتابخانه‌های خارجی توسط نرم‌افزار است. شکل 8 ساختار یک ELF باینری را نشان می‌دهد؛ PLT اولین منطقه‌ی شامل کدهای اجرایی معتبر است.



شکل 8. ELF در حافظه لود می‌شود. PLT شامل یک میز پرش به توابع خارجی است (به‌عنوان مثال، فراخوانی‌های libc).

در حال حاضر مشکل اصلی، کاهش یافتن ابزار BROP است، ورود نوشتن در PLT و یک روش برای کنترل RDX برای طول نوشتن. هر چه طول نوشتن بزرگتر از صفر باشد می‌توان باینری را در تکه های کوچک متعدد با زنجیر کردن نوشت. توجه داشته باشید که در زمان بهره‌برداری، RDX ممکن است یک مقدار سالم (بزرگتر از صفر) داشته بشاد بنابراین نیاز به کنترل RDX ممکن است غیر ضروری باشد، اما در حالت کلی ضروری است. متاسفانه pop RDX و ابزار ret بسیار نادر هستند، بنابراین بهینه‌سازی برای پیدا کردن یک پاسخ به جای strcmp (دوباره در PLT) باید انجام گیرد که RDX را به طول رشته‌ی در حال مقایسه نگاشت می‌کند. حمله‌ی بهینه شده نیاز به موارد زیر دارد:

1) پیدا کردن ابزار BROP.

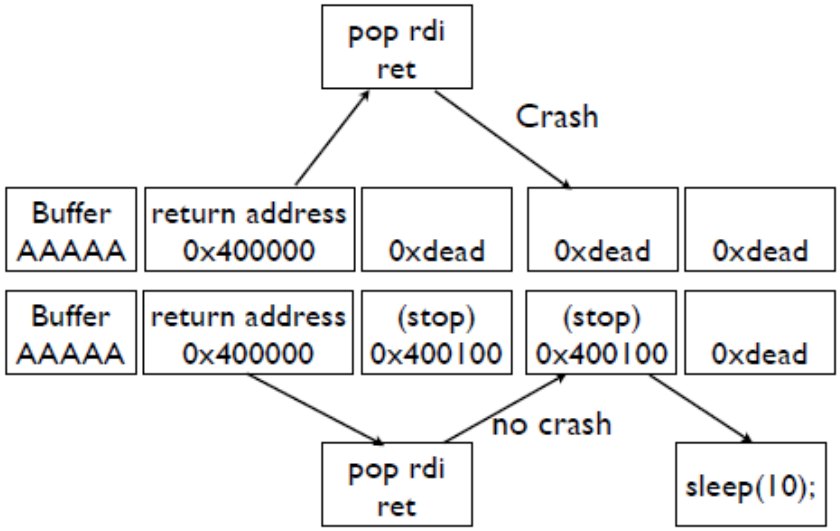
2) پیدا کردن PLT.

- پیدا کردن ورودی برای نوشتن.
- پیدا کردن ورودی برای strcmp.

B. پیدا کردن ابزار و توقف ابزار

ایده اصلی در پیدا کردن ابزارهای از راه دور، اسکن بخش متنی برنامه به جای نوشتن آدرس برگشت ذخیره شده با یک آدرس اشاره به متن و بررسی رفتار برنامه است. که آدرس شروع را می‌توان از فاز اولیه خواندن از پشته یافت یا 0x400000 را می‌توان به‌طور پیش فرض در لینوکس غیر PIE استفاده کرد. به‌طورکلی دو حالت رخ خواهد داد: برنامه شکست خواهد خورد یا متوقف خواهد شد، و در اتصال مجدد بسته خواهد شد و یا باز خواهد ماند. بیشتر زمان‌ها، برنامه شکست می‌خورد، اما هنگامی که این اتفاق نمی‌افتد، یک ابزار پیدا می‌شود. به‌عنوان مثال، 0x400000 ممکن است به کد با اشاره‌گر تهی اشاره کند و باعث شکست شود. آدرس بعدی، 0x400001، ممکن است به کدی اشاره کند که باعث یک حلقه بی‌نهایت و اتصال باز گردد. این ابزار، اجرای برنامه را برای یافتن ابزار دیگر متوقف می‌کنند: این‌ها را ابزارهای توقف می‌نامیم.

مشکل استفاده از این تکنیک، در یافتن ابزار است که اگر آدرس برگشت با آدرس یک ابزار مفید مانند pop RDI بازنویسی شود، نرم‌افزار به احتمال زیاد با شکست مواجه می‌شود چراکه تلاش برای بازگشت به کلمه بعدی در پشته، به احتمال زیاد به آدرس نامعتبر برمی‌گردد. شکست موجب عدم موفقیت در طبقه‌بندی ابزار می‌گردد. شکل 9 این مسئله را نشان می‌دهد. برای یافتن ابزار نیاز به متوقف کردن اجرای زنجیره وار ROP داریم. ابزار توقف چیزی است که، مانند یک حلقه بی‌نهایت یا مسدود کردن سیستم پاسخ (مانند sleep) موجب بلاکه شدن نرم‌افزار می‌گردد. برای یافتن ابزارهای مفید، یکی از آدرس مکان مورد استفاده، آدرس برگشت است.



شکل 9. پویش برای یافتن ابزار و استفاده از ابزارهای توقف. برای اسکن ابزارهای بازنویسی آدرس برگشت با کاندید آدرس text. (به‌عنوان مثال، 0x400000) روبه‌رو هستیم. اگر یک ابزار پیدا شود، آن را باز می‌گرداند، بنابراین باید "ابزارهای توقف" به پشته برای جلوگیری از اجرای زنجیره وار ROP اضافه شود و این ارم در یافتن ابزرا امکان‌پذیر است.

توجه داشته باشید که pop RDI یا ابزار ret، آیتم بعدی را از پشته به pop RDI انتقال می‌دهند و دو ابزار توقف مورد نظر در این مورد نیاز خواهیم داشت. هر زمان که یک ابزار مفید منجر به تصادفدر برنامه نشود، ابزار توقف اجرا خواهد شد، برنامه مسدود و سوکت باز ترک خواهد شد (به‌جای شکست). هم‌اکنون می‌توانیم کل قطعه text را برای کامپایل یک لیست از ابزارها پویش کنیم. بخش بعدی شرح می‌دهدکه مهاجم چگونه می‌تواند دستورالعمل را شناسایی کند. به‌عنوان مثال، فرق بین pop rdi و ret؛ ret و pop RSI؛

ابزار توقف ضرورتا نباید ابزاری برای "توقف" برنامه باشد. آنها صرفا یک مکانیزم سیگنال هستند. به‌عنوان مثال، ابزار توقف می‌تواند یکی از ابزارهایی باشد که موجب نوشتن‌های خاصی در شبکه گردد بنابراین مهاجم زمان توقف را به ابزار منتقل می‌کند. یکی دیگر از سناریوها که حضور ابزار توقف در قاب پشته است. پشته در واقع آدرس‌های برگشت متعددی دارد و یکی از آنها به‌عنوان یک ابزار توقف عمل می‌کند. به‌عنوان مثال یک سرور ممکن است مسئولیت رسیدگی به درخواست‌ها در یک حلقه while-true را برعهده داشته باشد، بنابراین بازگشت به آن حلقه ممکن است

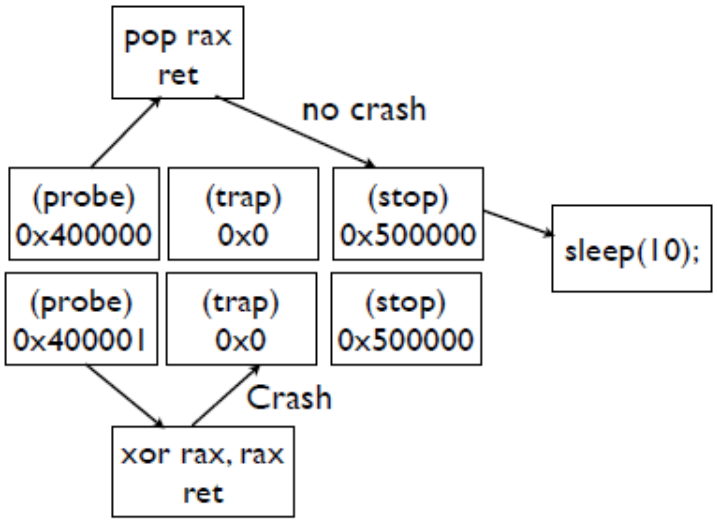
اجرای برنامه را از جایی که متوقف شده ادامه دهد و درخواست دیگری می‌تواند گرفته شده است. این مسئله می‌تواند به‌عنوان سیگنال مورد استفاده قرار گیرد که آیا یک برنامه شکست خورده یا هنوز در حال اجرا است (به‌عنوان مثال، ابزار توقف اجرا شده است). در این مورد مهاجم، پشته را دستورالعمل‌های کافی ret پوشش می‌دهد تا کلمه بعدی در پشته آدرس بازگشت، به‌عنوان ابزار توقف (به‌عنوان مثال، بازگشت به حلقه اصلی برنامه) عمل کند. این بهینه‌سازی خاص برای جلوگیری از شرایطی که در آن فرایندها محدود هستند و بی‌نهایت حلقه از نوع ابزار توقف وجود دارد، مفید است. در بخش هشتم جزئیات بیشتری در مورد چگونگی حمله به سیستم با چند فرآیند توصیف شده است.

C. شناسایی ابزارها

در حال حاضر در مورد چگونگی طبقه‌بندی ابزار بحث می‌کنیم. این همل می‌تواند توسط طرح پشته و بازرسی رفتار برنامه کنترل شود. سه مقدار تعریف می‌کنیم که مهاجم بتواند در پشته جای گیرد:

کاوشگر آدرس از ابزار در حال اسکن.

توقف آدرس ابزار توقف که با مشکل مواجه نشود.



شکل 10. اسکن ابزارهای pop. با تغییر طرح پشته، می‌توانید ابزارها را از pop کلمات از پشته جدا کنید. برای مثال، اگر یک "ابزار تله" اجرا شود، برنامه شکست خواهد خورد.

تله آدرس حافظه غیراجرایی است که موجب شکست خواهد شد (به‌عنوان مثال، خانه 0X0).

ایده این چنین است که با تغییر موقعیت از توقف و تله در پشته، می‌توان نتیجه گرفت که دستورالعمل توسط ابزار اجرا می‌شود، یا به‌دلیل اینکه تله یا توقف اجرا خواهد شد، باعث یک شکست و یا عدم شکست خواهد شد. در اینجا چند مثال و طرح‌بندی پشته ممکن بیان شده است:

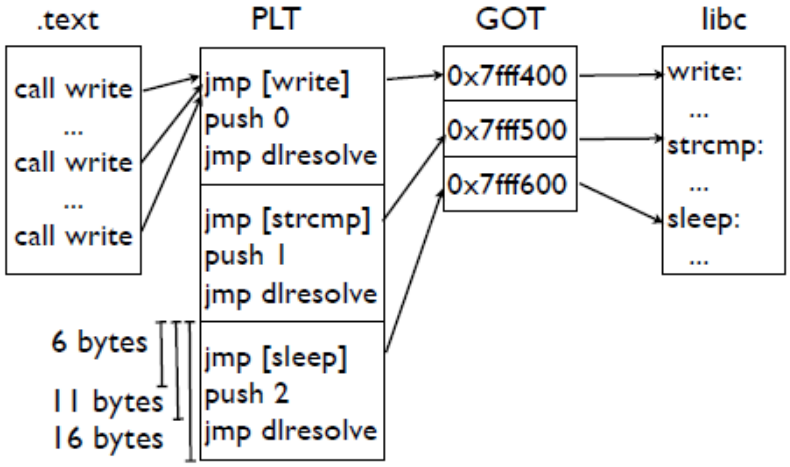
• بررسی، توقف، تله (دام، تله،...). ابزارهایی را پیدا خواهد کرد که مانند ret یا xor rax، rax، ret پشته را pop نمی‌کنند.

• بررسی، تله، توقف، تله‌ها. ابزارهایی را خواهند یافت که دقیقا یک کلمه از پشته را pop می‌کنند مانند pop RAX؛ قثف و یا pop RDI؛ ret. شکل 10 یک تصویر از این مورد را نشان می‌دهد.

• بررسی، توقف، توقف، توقف، توقف، توقف، توقف، تله. ابزارهایی را خواهد یافت که تا شش کلمه را pop می‌کنند (به‌عنوان مثال، ابزار BROP).

تله در پایان هر دنباله اطمینان حاصل می‌کند که اگر یک ابزار بیش از ابزارهای توقف عمل کرده باشد، شکست رخ خواهد داد. در عمل تنها چند تله (در صورت وجود) لازم خواهد بود چون پشته به احتمال زیاد حاوی مقادیری (به‌عنوان مثال، رشته‌ها، اعداد صحیح) است که به هنگام تفسیر به عنوان آدرس‌های برگشت موجب شکست می‌شوند.

با استفاده از طرح دوم پشته می‌توانید یک لیست از ابزارهای پاپ x ایجاد کنید. هنوز هم نمی‌دانیم که آیا یک pop RDI یا pop RSI پیدا شده است. در این مرحله حمله چنین اتفاق می‌افتد: هم می‌تواند "اصول اولیه" حمله را برای شناسایی ابزار pop براساس رفتار سیستم تماس یا یک نسخه بهینه‌سازی از حمله با تکیه بر ابزار BROP انجام دهد. ابزار BROP دارای امضای منحصر به فردی هستند. که شش آیتم از پشته را pop می‌کنند و در دیگر بخش‌های آن قرار می‌دهند، حذف کمتر آیتم‌ها از پشته را می‌توان یک کاندید برای بررسی تله‌ها و ابزار توقف در ترکیب‌های مختلف و بررسی رفتار تلقی کرد. ابزار دارای طولی برابر با 11 بایت است بنابراین می‌توان 7 بایت به هنگام اسکن متنی برای پیدا کردن موثرترین آنها نادیده گرفت. اگر بیش از 7 بایت نادیده گرفته شود در pop RSP خطایی رخ می‌دهد، در نتیجه یک کپی از ابزار پیدا نمی‌شود. بعد از یافتن ابزار BROP، مهاجم می‌تواند دو آرگومان اول (RDI و RSI) را برای هر فراخوانی کنترل کند.



شکل 11. ساختار PLT و عملیات. همه تماس‌های خارجی در باینری، از طریق PLT صورت می‌گیرد. PLT، GOT را برای پیدا کردن آدرس نهایی برای استفاده در یک کتابخانه ارجاع می‌دهد.

D. پیدا کردن جدول روش لینک کردن

برای کنترل آرگومان سوم (RDX) نیاز به پیدا کردن یک پاسخ به strcmp داریم، که RDX را به طول رشته مقایسه شده نگاشت می‌دهد. PLT یک جدول پرش در آغاز اجرا برای تمامی تماس‌های خارجی (به عنوان مثال libc) است. مثلا، یک تماس به strcmp در واقع یک تماس به PLT خواهد بود. PLT به جدول جهانی افست (GOT) ارجاع داده خواهد شد و به آدرس‌های ذخیره شده در آن پرش خواهد کرد. GOT به بارکننده‌ی پویا با آدرس توابع کتابخانه وابسته است. GOT با کسالت همراه است، به‌طوری‌که اولین بار که هر ورودی PLT فراخوانی می‌شود، یک مسیر آهسته از طریق dlresolve برای حل موقعیت نمادین و ورود GOT برای دفعه بعد انتخاب می‌کند. ساختار PLT در شکل 11 نشان داده شده است. که بسیار منحصر به فرد است: هر ورودی از 16 بایت جدا از هم (16 بایت تراز شده) تشکیل شده و مسیر آهسته برای هر ورودی را می‌توان در افست 6 بایتی اجرا کرد.

بسیاری از ورودی‌های PLT یک شکست منتج از آرگومان نخواهد بود چرا که آنها فراخوانی‌های سیستم هستند و EFAULT در پارامترهای نامعتبر را برمی‌گردانند. بنابراین می‌توانید یک PLT با اعتماد به نفس بیشتر بیابید اگر یک

جفت از آدرس‌های 16 بایت جدا از هم منجر به شکست نشوند می‌توانند بررسی کنند که آدرس‌های به‌علاوه شش منجر به شکست نمی‌شوند. این آدرس‌ها اولین موقعیت برای داشتن کد معتبر در فضای آدرس‌دهی اجرایی هستند.

بنابراینPLT می‌تواند با اسکن منشاء برنامه (0x400000) یا به عقب راندن آدرس از طریق خواندن از پشته یافت شود. هر آدرس باید 16 بایت تراز شده باشد و 16 بایت می‌تواند نادیده گرفته شود. توجه داشته باشید که PLTها اغلب بزرگ (200 ورودی) هستند بنابراین می‌توان بایت بیشتری را (با نادیده گرفتن ورودی PLT) نادیده گرفت.زمانی که به دنبال بهینه‌سازی سرعت هستیم، امیدواریم که تابعی که با مشکل مواجه نشده است مورد شکست قرا نگیرد.

طرح پشته برای پیدا کردن یک ورودی PLT این چنین خواهد بود: کاوش یا جستجو، توقف، تله. PLT می‌تواند با مشاهده‌ی ورودی‌های همسایه که شکست نخورده‌اند تأیید شود، بنابراین اگر آفست شش بایت باشد (PLT slowpath) هنوز شکست رخ نداده است.

E. کنترل RDX از طریق strcmp

اولین بار که مهاجم PLT را پیدا می‌کند، این سوال را می‌پرسد که چه تابع فراخوانی مربوط به ورودی‌های مختلف انجام خواهد شد؟ یکی از آنها strcmp خواهد بود، مگر اینکه برنامه یا یکی از کتابخانه‌های آن، از آن تابع در مواردی که مهاجم می‌تواند "اصول اولیه" حمله را که قبلا توضیح داده شده است برای یافتن pop rdx به کار ببرد، استفاده نکند. توجه داشته باشید که هیچ چیز خاصی برای strcmp جدا از هم وجود ندارد بنابراین معمولا برای تنظیم RDX به بزرگتر از مقدار صفر استفاده می‌شود. هر تابع دیگری نیز همین کار را انجام می‌دهد.

مهاجم می‌تواند ورودی‌های PLT مربوط به هر ورودی را با آرگومان‌های مختلف و مشاهده عملکرد تشخیص دهد. دو آرگومان اول می‌توانند ابزار BROP را کنترل کنند. strcmp رفتاری به شرح زیر دارد، که در آن "بد" محل نامعتبر حافظه (به‌عنوان مثال، 0X0) و "قابل خواندن" یک اشاره‌گر قابل خواندن است (به‌عنوان مثال، یک آدرس درtext.):

- strcmp (بد، بد): شکست
- strcmp (بد، قابل خواندن): شکست

• strcmp (قابل خواندن ، بد): شکست

• strcmp (قابل خواندن، قابل خواندن): عدم شکست

مهاجم، strcmp را با پیدا کردن یک ورودی در پاسخ به امضایی که قبلا ذکر شد، می‌یابد. PLT می‌تواند به دو روش مورد ارزیابی قرار گیرد. مهاجم آدرس یک ورودی PLT معتبر را که قبلا پیدا شده است خواهد داشت. روش ساده، آدرس پروب +_ 0x10 بایت است. روش موثرتر اجتناب از اجرای دو انتهای PLT با استفاده از مسیر آهستهPLT است. مسیر شماره PLT را در پشته ذخیره می‌کند و پس از آن dlresolve را فراخوانی می‌کند. این فراخوانی در هر ورودی PLT در افست 0xb حضور دارد. بنابراین می‌توان آدرس برگشت ذخیره شده را با یافتن ورودی PLT + 0xb بازنویسی کرد و به دنبال تعدادی از ورودها برای بررسی (با شروع از صفر) سیستماتیک تمام ورودی‌های PLT بود.
هنگامی‌که strcmp پیدا شد، مهاجم می‌تواند RDX را به مقدار غیرصفر وتنها با یک اشاره‌گر به هر PLT ورودی (کد غیر صفر) و یا شروع هدر ELF (0x400000) که دارای هفت بایت غیرصفر است تنظیم کند.
مثبت کاذب می‌تواند به هنگام جستجو strcmp پیدا شود. جالب توجه است که در تمام تست‌ها متوجه شدیم که strncmp و strcasecmp به جای یکدیگر پیدا شدند و از تنظیم RDX به یک مقدار بزرگتر از صفر تاثیر گرفتند.

F. پیدا کردن نوشتن

اکنون مهاجم می‌تواند سه آرگومان اول را برای هر فراخوانی کنترل کند: دو آرگومان اول از طریق ابزار BROP، و سوم به‌طورغیرمستقیم از طریق strcmp. اکنون write را می‌توان با اسکن هر ورودی PLT و نوشتن بالاجبار در سوکت و چک کردن اینکه آیا نوشتن رخ داده است یافت. تنها مشکل، تعداد فایل توصیفی برای سوکت است. بنابراین دو روش وجود دارد: زنجیروار هر کدام را با شماره متفاوت فایل توصیفی در یک زنجیره ROP واحد می‌نویسد و باز کردن اتصالات متعدد و استفاده از یک توصیفگر فایل نسبتا بالا، تعداد اتصالات را مطابقت می‌دهد. ما از ترکیب هر دو روش استفاده می‌کنیم.

به طور پیش فرض لینوکس، فرآیندها را به حداکثر 1024 فایل همزمان باز محدود می‌کند بنابراین فضای جستجو کوچک است. علاوه‌براین، POSIX مستلزم استفاده از فایل‌های جدید برای کمترین تعداد موجود و سپس جستجو در چند فایل توصیفی است.

G. مرحله پایانی حمله

در این مرحله مهاجم می‌تواند کل قطعه text را از حافظه به سوکت مهاجم ارسال کند. مهاجم همچنین می‌تواند جدول نماد را نادیده گرفته و توابع مفید در PLT را مانند dup2 و execve بیابد. به‌طورکلی مهاجم نیاز به:

1) تغییر مسیر سوکت به استاندارد ورودی/ خروجی دارد. مهاجم می‌تواند از dup2 یا close استفاده کند و fcntl DUP یا (F_DUPFD) را دنبال کند. که اغلب در PLT هستند.

2) یافتن "/bin / sh " در حافظه است. یک روش موثر برای پیدا کردن یک منطقه قابل نوشتن در حافظه مانند محیط، environ، از جدول نماد، و read "/ bin / sh " از سوکت مهاجم به آن آدرس.

3) execve. اگر execve در PLT نباشد، مهاجم نیاز به انتقال بیشتر از باینری برای یافتن یک pop RAX؛ ret و ابزار syscall دارد.

نادیده گرفتن جدول نماد به این ساده‌گی‌ها ممکن نیست. درحالی‌که هدر ELF در حافظه بارگذاری می‌شود، بخش جدول (در پایان باینری) نیست. بخش هدر شامل اطلاعاتی در مورد آغاز جدول نماد است. به‌منظور پیدا کردن جدول نماد بدون این اطلاعات، مهاجم باید شروع به آزادسازی باینری از آغاز تا رشته ASCII (نام تابع) یافت شده کند (بخش dynstr). براساس بخش dynstr، دیگر بخش‌های مجاور حاوی اطلاعات جدول نماد را می‌توان یافت.

H. خلاصه حمله

حمله بهینه‌سازی شده‌ی BROP به شرح زیر است:

1) پیدا کردن مکانی برای اجرای تا لود شود. هر دو موردِ 0x400000 برای اجرای غیر PIE (پیش‌فرض) و یا خواندن از پشته، آدرس برگشت را ذخیره می‌کنند.

2) یافتن یک ابزار توقف. که معمولا یک سیستم پاسخ مسدود کردن (مانند خواب و یا خواندن) در PLT است. مهاجم PLT را در این مرحله پیدا می‌کند.

3) یافتن ابزار BROP. مهاجم هم‌اکنون می‌تواند دو آرگومان اول را برای فراخوانی کنترل کند.

4) پیدا کردن strcmp در PLT. مهاجم هم‌اکنون می‌تواند سه آرگومان اول را برای فراخوانی کنترل کند.

5) یافتن write در PLT. مهاجم هم‌اکنون می‌توانید کل باینری را برای یافتن ابزارهای بیشتر جستجو کند.

6) ساخت یک shellcode و بهره‌برداری از سرور.

حمله نیاز به یک اسکن (جزئی) برای قابلیت اجرا دارد. PLT بخش اول از یک فایل اجرایی است و همچنین اولین موردی است که مهاجم نیاز به راه‌اندازی آن دارد (به‌عنوان مثال، پیدا کردن یک ابزار توقف). ابزار BROP که در قطعه text. پیدا می‌شود، بعد از PLT، موجب حمله می‌گردد. پس از ابزار BROP، حمله بسیار موثر است چون که ابزاری برای اسکن PLT برای دو عملکرد است. پیچیدگی حمله به چگالی ابزارهای BROP و مدت زمان لازم برای یافتن PLT بستگی دارد.

I. اصول اولیه حمله

ممکن است فکر کنیم که از بین بردن ابزارهای BROP از حالت اجرایی و یا ساخت PLT برای توقف حملات BROPدشوار است. در حالی که چنین نیست، در حال حاضر یک روش با کارایی پایین برای این حمله ارائه می‌کنیم که نه تنها متکی بر ابزار BROP نیست بلکه به PLT نیز وابسته نیست. حمله تمام ابزارهای ذکر شده در بخش هشتم-الف را می‌یابد. طرح کلی حمله به صورت زیر است:

1) یافتن تمام pop x؛ ابزارهای Ret

2) یافتن یک ابزار syscall.

3) شناسایی ابزارهای pop که قبلا پیدا شده‌اند.

مهاجم با پیدا کردن یک ابزار توقف و تمام pop x ها؛ دستورالعمل ret شروع می‌کند. در حال حاضر مشکل این مسئله، شناسایی دستورالعمل‌های pop و پیدا کردن یک ابزار syscall است. ایده این مسئله، شناسایی دستورالعمل pop براساس رفتار سیستم فراخوانی پس از افزایش سرعت آرگومان سیستم فراخوانی است، در روشی مشابه به چگونگی یافتن strcmp در حمله بهینه‌سازی شده می‌پردازد. تنها مشکل موجود bootstarp است، بااین حال، به دلیل یافتن یک syscall باید تعداد پاسخ سیستم (RAX) کنترل شود، به‌طوری‌که باید شناسایی از پیش تعریف شده‌ای برای pop rax؛ ret صورت گیرد.

راه‌حل زنجیروار کردن تمام دستورالعمل‌های pop که توسط مهاجم پیدا می‌شوند، تعدادی سیستم پاسخ مورد نظر را اضافه می‌کند که یکی از آنها به احتمال زیاد RAX خواهد شد. سیستم فراخوانی برای استفاده از ()pouse که هیچ آرگومانی نمی‌گیرد مانند سایر رجیسترها می‌توان نادیده گرفت. همچنین اجرای برنامه تا زمانی که یک سیگنال افزایش یابد متوقف می‌شود و پس از آن به عنوان یک ابزار توقف عمل می‌کند که قابل شناسایی است. مهاجم می‌تواند آدرس probe را برای syscall و برای اضافه کردن زنجیروار pop برای پیدا کردن یک ابزار فراخوانی سیستمی در نظر بگیرد. هنگامی‌که یک آدرس، برنامه pause را یافت، مهاجم می‌تواند pop برای یافتن کنترل rax را نادیده بگیرد. در این مرحله مهاجم آدرس ابزار syscall و pop rax را دارد. همچنین مهاجم دارای یک لیست از pop های ناشناس است. که با استفاده از فراخوان‌های سیستمی زیر قابل دستیابی هستند:

1) اولین آرگومان (pop rdi): nanosleep (len، rem). این مسئله منجر به خواب چند نانو ثانیه‌ای (بدون شکست) می‌شود. اگر خواب قطع شده باشد، rem جای می‌گیرد و می‌تواند یک آدرس نامعتبر شناخته شود که پس از خواب بررسی می‌شود.

2) دومین آرگومان (pop rsi): kill(SIG ،PID) . اگر Sig صفر باشد، هیچ سیگنالی فرستاده نمی‌شود، درغیراین‌صورت یک فرستاده می‌شود (که منجر به شکست می‌گردد). PID نیازی به شناخته شدن ندارد: زمانی‌که سیگنال را در گروه

به تمام فرآیندها می‌فرستد می‌تواند صفر باشد. برای بررسی اینکه آیا سیگنال فرستاده شده است، مهاجم می‌تواند برای مشاهده‌ی اتصالات قطع شده (رفتن به فرآیندهای کارگر مختلف) اتصالات چندگانه برقرار کند.

3) آرگومان سوم (pop rdx): (rem، len، flags، clock)clock_nanosleep . مشابه nanosleep، اما دو استدلال اضافی برای کنترل آرگومان سوم در طول خواب می‌طلبد.

هم اکنون می‌توانیم write را ارسال کنیم و حمله را با آزادسازی قطعه text و پیدا کردن ابزار بیشتر ادامه دهیم. درحالی‌که این حمله کلی است، اجرای آن پیچیده خاصی می‌طلبد چرا که نیاز به دو اسکن از قطعه text دارد: یکی برای پیدا کردن یک لیست از ابزارهای پاپ و یکی برای پیدا کردن یک ابزار فراخوانی سیستمی.

بهینه‌سازی قابل توجه این چنین است که تمام pop rax ها و ابزارهای ret که یافتیم برای تجزیه add rsp، 0x58 نادیده گرفته شوند. این اطلاعات می‌تواند برای طبقه‌بندی ابزار pop rax مستقل از ابزارهای فراخوانی سیستمی و با توجه به افزایش قابل توجه سرعت حمله مورد استفاده قرار گیرد- بنابراین نیازی به اسکن کل قطعه text برای دو بار نیست. در واقع می‌توان برای add rsp، ابزار 0x58 را با راه‌اندازی پشته با 11 تله با ابزار توقف بررسی کرد. به‌منظور بررسی ابزار، مهاجم، تجزیه را برای دستیابی به pop rax نادیده می‌گیرد

ل. دیگر جزئیات سطح پایین

در این بخش تعدادی از جزئیات حمله سطح پایین را که مشخص نیستند لیست می‌کنیم، که بسیاری از آنها به ثبات حمله اضافه شده‌اند.

الف) خواندن از پشته با صفر: یک راه موثر برای خواندن از پشته با قرار دادن صفر در کلماتی مانند اشاره‌گر فریم ذخیره شده است. بنابراین این احتمال وجود دارد که اشاره‌گر دستورالعمل بدون در نظر گرفتن اشاره‌گر فریم یافت شود. همچنین باعث می‌شود که خواندن از پشته قوی‌تر اتفاق بیافتد اگر فرآیندهای متفاوت استفاده شود. بنابراین ممکن است به پایان رساندن خواندن به‌عنوان خواندن از اشاره‌گر فریم هنگامی که به یک فرآیند مختلف فرستاده می‌شود

امکان‌پذیر نباشد چرا که هر مقدار منجر به شکست خواهد شد. وارد کردن یک کلمه صفر در این مورد موجب رفع مشکل می‌شود.

ب) تایید بیشتر strcmp: به‌منظور بررسی بیشتر strcmp، آن را در برابر آخرین بایت از صفحه vsyscall اجرا می‌کنیم، که به یک مکان ثابت نگاشت می‌شود. strcmp قبل از رسیدن به پایان vsyscall خاتمه خواهد یافت و موجب شکست خواهد شد. اکثر توابع برای خواندن صفحه vsyscall که موجب شکست می‌شود تلاش خواهند کرد. این مسئله توابعی را که به هنگام به کار بردن دو آرگومان منجر به شکست می‌شوند جدا خواهد کرد.

ج) برخورد با بافر کوچک: گاهی اوقات حمله باید طول زنجیره ROP را به حداقل برساند و قادر به بهره برداری از بافر کوچک باشد. این وضعیت به هنگام خواندن کوتاه و یا نیاز به دست نخورده نگه داشتن برخی از حافظه اتفاق می‌افتد، که موجب محدودیت در طول سرریز و فضای بافر دردسترس می‌شود. MySQL + yaSSL نیاز به این بهینه‌سازی را به‌منظور جلوگیری از canary بهره‌برداری می‌کند. این مسئله یک چک لیست برای انجام BROP با زنجیرهای کوتاه ROP در 8 کلمه (64 بایت) فراهم می‌کند:

• یافتن ورودی‌های PLT واقعی براساس آدرس آنها، نه براساس تعداد فشار آنها و مسیر آهسته. این مسئله منجر به فراخوانی PLT با زنجیره ROP کوتاه خواهد شد.

• نادیده گرفتن باینری با حداقل زنجیره ROP: آدرس strcmp برای نادیده گرفتن، RSI را مجدد تنظیم نمی‌کند (در حال حاضر به strcmp تنظیم شده است، و پاسخ ارسال می‌کند. اگر صفر خوانده شود، آدرس نادیده گرفته شده شامل صفر می‌شود. درغیراین صورت مقدار کمی از باینری (تا صفر) قابل خواندن خواهد بود. این عمل را تا زمانی که pop rdx پیدا شود ادامه می‌دهیم. پس از آن از pop rdx برای کنترل طول به‌جای strcmp (کوتاه‌تر از زنجیر ROP) استفاده می‌کنیم.

• ایجاد محیط shellcode در مراحل متعدد: یک اتصال برای dup سوکت مهاجم، یکی برای "read "/bin / sh در حافظه، و یکی برای execve. همه این اتصالات (به غیر از execve) باید زنجیره ROP با یک ابزار توقف برای جلوگیری از شکست خاتمه می‌یابد چرا که فرایند کارگر در حال آماده شدن تدریجی است.

د) برخورد با worker مبتنی بر رویداد: شرایطی وجود دارد که در آن یک برنامه کاربردی با تعداد بسیار کمی کارگر مبتنی بر رویداد پیکربندی می‌شود و می‌تواند در طول حمله BROP، به‌هنگام اجرای ابزار توقف (به‌عنوان مثال، یک حلقه بی‌نهایت) بدون پاسخ باشد و ادامه حمله را غیرممکن سازد. nginx یک مثال است و با چهار کارگر به‌صورت پیش‌فرض پیکربندی می‌شود. این مورد یک مشکل با برنامه‌های کاربردی که هر اتصال را توزیع می‌کنند نیست. در مورد دوم، می‌توانید اتصال ایجاد کنید و توسط یک نخ جدید آن را کنترل کنید. راه‌حل کلی ما سعی در استفاده از یک ابزار توقف برای برگردان یک فریم فراخوانی بزرگتر است (ابزار توقف مبتنی بر پشته) به‌جای این که یک راه حل مبتنی بر حلقه بینهایت و امکان‌پذیر داشته باشیم. پیاده‌سازی این مقاله به کمک الگوریتم زیر ارائه شده است که حداقل با سه فرآیند کارگر عمل می‌کند:

1) پیدا کردن یک ابزار توقف مبتنی بر PLT (به‌عنوان مثال، خواب).

2) یافتن یک ورودی PLT که برمی‌گرداند. که می‌تواند هر تابع PLT بدون شکست و نیاز به ایجاد حلقه بی‌نهایت باشد (به‌عنوان مثال، به عنوان یک ابزار توقف عمل نمی‌کند). این تابع بسیار مفید است چرا که شبیه یک دستور ret است و این اجازه را می‌دهد که کلمات فردی از پشته pop شوند (به عنوان یک NOP در ROP به‌طورموثر عمل می‌کند). این مورد را "ابزار بازگشت" معرفی خواهیم کرد.

3) دو مرحله اول این الگوریتم باعث به کارافتادن کارگر می‌شود. بنابراین می‌توان از این مورد اجتناب کرد و هدف یافتن یک ابزار توقف مبتنی بر پشته با افزایش تدریجی پشته با ابزار بازگشت است. بنابراین: تلاش در راستای بهره‌برداری از یک ابزار بازگشت سپس با دو، سپس سه تکرار و الی آخر است. در نهایت، اگر یک ابزار توقف مبتنی بر پشته ارائه شود، برنامه شکست نخواهد خورد. الگوریتم باید طوری باشد که اگر ابزار یافت نشد ابزار توقف مبتنی بر PLT استفاده شود. اگر ابزار پیدا شد، با این که راه‌اندازی پشته برای ادامه حمله زنجیره ROP خواهد بود ولی تعدادی از ابزارهای بازگشت جستجو می‌شود که پس از آن توسط ابزار توقف مبتنی بر پشته (بازنویسی توسط مهاجم) عمل می‌کند.

اجرای حفاظت شده، کار با یک کارگر فرآیند است، پشته فریم‌های پشته بالاتر را برای دستیابی به آدرس‌های برگشت می‌خواند و تلاش برای بازگشت در ادامه از سرگیری برنامه رخ می‌دهد. این روش با این که چندین پشته خوانده می‌شود ناکارآمد است.

9. پیاده‌سازی

ما حملات BROP را در یک ابزار به نام "Braille" که به‌طور خودکار ناشی از یک شکست از راه دور است اجرا می‌کنیم. این برنامه در 2000 خط کد روبی نوشته شده است. Braille اساسا بهره‌برداری است که منجر به شکست سرور و تمام اطلاعات مورد نیاز برای بهره‌برداری می‌شود. رابط کاربری به‌صورت زیر است:

```
try_exp(data) -> CRASH, NO_CRASH, INF
```

بنابراین نیاز به پیاده‌سازی تابع try_exp () وجود دارد و تضمین می‌کند که بایت‌های به اندازه کافی "داده" برای نوشتن آدرس بازگشت ذخیره شده بر روی پشته به پایان خواهد رسید. اگر سوکت پس از ارسال داده بسته شود تابع شکست را برمی‌گرداند، اگر برنامه به نظر طبیعی عمل کند، یا اگر سوکت برای بیش از زمان مورد نیاز باز باقی بماند، شکست رخ نخواهد داد. ایست به‌طور خودکار توسط چارچوب و براساس سرعت شناسایی شکست تعیین می‌شود.

```
def try_exp(data)
  s    = TCPSocket.new($victim, 80)

  req = "GET / HTTP/1.1\r\n"
  req << "Host: site.com\r\n"
  req << "Transfer-Encoding: Chunked\r\n"
  req << "Connection: Keep-Alive\r\n"
  req << "\r\n"
  req << "#{0xdeadbeefdead.to_s(16)}\r\n"
  req << "#{data}"

  s.write(req)

  if s.read() == nil
    return RC_CRASH
  else
    return RC_NOCRASH
  end
end
```

شکل 12. nginx کد را به کار می‌برد. که صرفا رشته ورودی ارائه شده توسط Braille را به یک درخواست HTTP و یک سیگنال شکست اگر اتصال بسته شود منتقل می‌کند.

کد بازگشت بدون شکست برای تحقیق در مورد طول بافر در حال سرریز پشته و یا زمانی که شروع به خواندن پشته می‌کند مفید است. این کد بازگشت هنگامی که هیچ سرریزی رخ نمی‌دهد و یا همان مقدار در پشته بازنویسی می‌شود مورد انتظار اسا. کد برگشتی INF به هنگام جستجوی ابزارها مانند ابزار توقف ("حلقه بی نهایت") مورد انتظار هستند. در مراحل بعد از حمله که در آن داده از سوکت انتظار می‌رود، برای مثال پس از آن که write پیدا شد و باینری نادیده گرفته شد، یک نشانه را می‌توان برای بازگشت به سوکت واقعی به‌جای کد نتیجه CRASH / INF عبور داد.

کد اجرایی می‌تواند به‌سادگی باز کردن یک سوکت به سرور، و ارسال اطلاعات از طریق سوکت باشد. با این حال، سرویس‌ها انتظار دارند تا داده در قالب خاصی باشد. به عنوان مثال، هدرهای HTTP ممکن است نیاز به ارائه داشته باشند. کد اجرایی مسئول این قالب‌بندی است. ما سه کد اجرایی، که همگی100 خط کد هستند می‌نویسیم. یکی داده خام را عبور می‌دهد، یکی از بسته SSL ساخته شده است و یکی دیگر تقطیع‌کننده درخواستHTTP است. شکل 12 کد یک نسخه اولیه در nginx را نشان می‌دهد.

همچنین روتر قطعه‌بندی با IP عمومی را پیاده‌سازی می‌کنیم که در مواردی مفید است که در آن یک TCP برای خواندن و برای سرریز بافر مورد نیاز است. برای مثال پرکردن بافر 4096 بایت تنها با خواندن غیر مسدود ممکن است با MTU از 1500 غیر ممکن باشد، بنابراین ممکن است به اندازه کافی داده در صف برای بهره برداری قابل اعتماد وجود نداشته باشد. روتر ما به‌جای فرستادن بخش‌های بزرگ TCP به‌عنوان قطعات متعدد IP به‌طوری‌که خواندن برای برگرداندن یک بسته بزرگ تضمین شده است، موجب سرریز قابل اعتماد می‌گردد. این روش را در 300 خط کد C پیاده‌سازی می‌کنیم. که یک رابط tun مجازی ایجاد می‌کند که در آن هیچ تقسیم‌بندی TCP رخ نمی‌دهد، write به قطعات متعدد IP به‌عنوان یک بسته TCP ارسال می‌شود. این روتر برای nginx ضروری است.

ارسال بخش‌های TCP خارج از سفارش یک رویکرد جایگزین خواهد بود که ممکن است کار کند و برای فایروال قوی عمل کند.

```
ClientHello ch;

for (uint16 i = 0; i < ch.suite_len_; i += 3) {
  input.read(&ch.cipher_suites_[j], SUITE_LEN);
  j += SUITE_LEN;
}

input.read(ch.session_id_, ch.id_len_);

if (randomLen < RAN_LEN)
  memset(ch.random_, 0, RAN_LEN - randomLen);

input.read(&ch.random_[RAN_LEN - randomLen],
           randomLen);
```

شکل 13. کد آسیب‌پذیر در yaSSL. مهاجم suite_len _ ، id_len_ و randomLen را کنترل می‌کند.

بافرهای ClientHello دارای اندازه ثابت، در پشته هستند. بااستفاده از randomLen به‌عنوان یک بردار حمله اجازه بازنویسی را به مهاجم، از طریق اشاره‌گر و بازگشت آدرس بدون نیاز به canary می‌دهد. randomLen نمی‌تواند بیش از حد بزرگ باشد و تنها بافر کوچک برای بهره‌برداری در این شرایط موجود است.

10. ارزیابی

ما حمله BROP را در سه سناریو مورد آزمایش قرار می‌دهیم:

1) یک کتابخانه باز SSL با آسیب‌پذیری شناخته شده پشته (yaSSL). این سناریو،حمله به یک سرویس اختصاصی است که اعتقاد بر استفاده از یک منبع باز آسیب‌پذیر دارد. به‌عنوان یک هدف نمونه، ما از نسخه قدیمی MySQL که از yaSSL استفاده می‌کند کمک می‌گیرد.

2) یک نرم‌افزار منبع باز با یک پشته آسیب‌پذیر شناخته شده (در nginx)، به‌صورت دستی از منبع شروع به اجرا می‌کند. در این سناریوی مهاجم منبع کل سرور را می‌داند، اما باینری را نگه نمی‌دارد.

3) یک سرویس باینری با یک پشته آسیب‌پذیر. که توسط یکی از همکاران نوشته شده بود و هر دو منبع باینری و منبع به‌صورت مخفی نگه داشته شده است. در حالت ایده‌آل این مورد را در برابر دنیای واقعی مورد آزمایش قرار می‌دهیم اما از نظر قانونی بسیار دشوار است.

این آسیب‌پذیری‌ها به شرح زیر هستند.

الف) yaSSL: شکل 13 کد آسیب‌پذیر در yaSSL را نشان می‌دهد. بسته SSL دارای طول متغیر برای رمز-مجموعه، شناسه جلسه، و مشتری تصادفی است. طول در بسته مشخص شده است و yaSSL این مقادیر را به بافرهای با اندازه ثابت در پشته کپی می‌کند. این مسئله باعث می‌شود تا امکان بهره‌برداری از برنامه به سه روش پر کردن بافر صورت گیرد. جالب توجه است، کپی مشتری تصادفی شامل حساب اشاره‌گر است که امکان ارسال بر روی پشته با شروع از آدرس بازگشت ذخیره شده به جای اجبار به بازنویسی canary را فراهم می‌کند. این مسئله برای MySQL مهم است و سرور پس از یک شکست و خواندن از پشته برای canary مجدد اجرا می‌شود. خطر استفاده از این روش این است که اندازه سرریز در عمل بسیار کوچک است: اگر یک بافر بزرگ استفاده شده باشد (randomLen بزرگ) پس از آن canary شروع به بازنویسی مجدد خواهد کرد. بنابراین مجبور به استفاده از بافر کوتاه است. Braille با دقت برای پشتیبانی از بافر کوتاه با استفاده از زنجیره کوتاه ROP پیاده‌سازی شده است.

ب) nginx: درخواست‌های HTTP می‌توانند به عنوان تکه‌هایی ارسال شوند، که در آن ارسال هر قطعه دارای طول و اطلاعات تکه است. اگر مقدار بزرگی عرضه شده باشد، در یک متغیر با مقدار منفی ذخیره خواهد شد.

```
typedef struct {
  ...
  off_t content_length_n;
} ngx_http_headers_in_t;

u_char buffer[NGX_HTTP_DISCARD_BUFFER_SIZE];

size = (size_t) ngx_min(
            r->headers_in.content_length_n,
            NGX_HTTP_DISCARD_BUFFER_SIZE);

n = r->connection->recv(r->connection, buffer, size);
```

شکل 14. کد آسیب‌پذیر در nginx. مهاجم content_length_n را کنترل می‌کند و می‌تواند یک مقدار منفی را فراهم کند. اندازه در تعداد زیاد بدون علامت خواهد بود اگر content_length_n منفی باشد. بافر 4096 بایت، در پشته است.

جدول 2. تعداد تجمعی درخواست در هر فاز حمله BROP.

فاز حمله	سرور اختصاصی	yaSSL+MySQL	Nginx
خواندن از پشته	1028	406	846
یافتن PLT	1394	1454	1548
یافتن ابزار BROP	1565	1481	2017
یافتن strcmp	1614	1545	2079
یافتن write	1624	1602	2179
Dump bin &exploit	1950	3851	2401
زمان به دقیقه	5	20	1

اگر اندازه (در حال حاضر منفی) کوچکتر از اندازه بافر باشد بررسی انجام خواهد شد. پس از آن اندازه به یک مقدار بدون علامت به عنوان پارامتر طول نسبت داده خواهد شد و ممکن است منجر به خواندن یک تکه بزرگ در یک بافر 4096- بایت در پشته گردد. شکل 14 کد آسیب پذیر در nginx را نشان می‌دهد.

ج) خدمات اختصاصی: هنگام ارسال یک رشته کوتاه، "OK" از سرور خوانده می‌شود. در یک رشته بسیار طولانی، ارتباط بسته شده است.

ما Baaille را در برابر هر سه سناریوی حمله، بدون هر نرم‌افزار بهینه‌سازی خاصی، اجرا کردیم و حمله در همه موارد موفق عمل کرد. جنبه‌های زیر را ارزیابی می‌کنیم:

1) عملکرد: تعداد درخواست و زمان.

2) ثبات: حمله چقدر قوی است.

3) حمله همراه با دانش منبع کد: آیا داشتن دسترسی به کد منبع (اما نه باینری) می‌توانید حمله را بهتر کند.

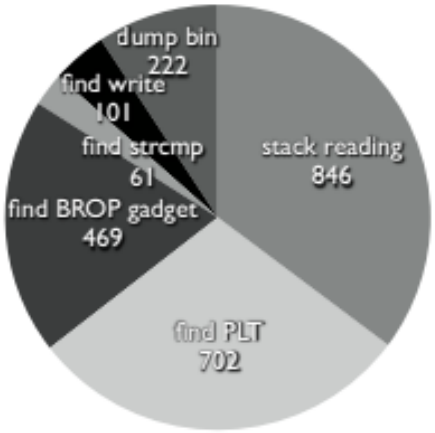
A. عملکرد

جدول 2 تعداد تجمعی درخواست مورد نیاز را برای هر فاز حمله نشان می‌دهد. این حمله می‌تواند در کمتر از 4000 درخواست، و یا 20 دقیقه تکمیل شود. بنابراین با توجه به اینکه در حملات گذشته در ASLR 32 بیتی به طور متوسط 32768 درخواست مورد نیاز بود قابل قبول است [9]. در تمام موارد، در 1500 مورد درخواست حمله قادر به شکست دادن canary، ASLR و پیدا کردن یک ابزار توقف است. چه مدت طول می‌کشد تا از دانش صفر به اجرای یک قطعه کد مفید برسد. پس از آن، پیدا کردن ابزار BROP ، بسته به محبوبیت خود، می‌تواند بین 27-467 درخواست قبول کند.

جدول 3 نشان می‌دهد که ابزار محبوب BROP چگونه است و یافتن چه تعداد probe در یک مجموعه از فایل‌های باینری انتظار می‌رود.

جدول 3. ابزار BROP

باینری	تعداد BROP	طول اسکن مورد انتظار
سرویس اختصاصی	194	154
MySQL	639	501
Nginx	130	566
Apache	65	860
openSSH	78	972



شکل 15. پیچیدگی حمله برای nginx. تعداد درخواست مورد نیاز برای هر فاز نمایش داده شده است. به‌طورکلی، پیچیدگی حمله به چهار بخش تقسیم می‌شود: خواندن از پشته، پیدا کردن PLT، پیدا کردن ابزار BROP، و آزادسازی باینری برای به پایان رساندن حمله.

داده‌ها تعداد ابزارهای حاضرBROP و چگالی آنها را نشان می‌دهند: $\frac{\text{.text size}}{7\times\text{BROP count}}$ (به یاد داشته باشید که 7 بایت می‌تواند در هر پروب با توجه به اندازه ابزار نادیده گرفته شود). به نظر می‌رسد ابزارهای BROP بسیار محبوب باشند و می‌توانند در کمتر از 1000 آدرس پیدا شوند. توجه داشته باشید که در عمل بیشتر درخواست‌های مورد نیاز برای بررسی ابزار و نادیده گرفتن مثبت کاذب صرف خواهند شد.

پس از پیدا شدن ابزار BROP، پیدا کردن write تنها چند درخواست اضافی می‌طلبد و معمولا می‌تواند در حدود 2000 درخواست یافت می‌شود. در این مرحله، حمله تقریبا کامل است. یکی ممکن است write دستی را برای قطعات بسیار خاص از باینری و برای به حداقل رساندن تعداد درخواست براساس اطلاعات به دست آمده انتخاب کند. در غیر این صورت، ابزار Braille شروع به آزادسازی باینری از آغاز آن می‌کند، تا زمانی‌که کل جدول نماد به هم بریزد تا یک کد ساخته شود. حمله به‌طور معمول در 500 درخواست اضافی (حدود 2500 در کل) کامل می‌شود. yaSSL، درخواست‌های بسیاری برای آزادسازی باینری می‌طلبد چرا که بافر درحال سرریز بسیار کوتاه است و به همین ترتیب Braille در توان زنجیره ROP محدود شده است. Braille مجبور به آزادسازی باینری در تکه‌های کوچک برای پیدا کردن یک pop RDX؛ ret (ابزار کمیاب) قبل از دانلود باینری در تکه‌های بزرگتر است.

شکل 15 پیچیدگی حمله برای nginx را نشان می‌دهد. سربار حمله را می‌توان به چهار بخش تقسیم کرد: خواندن از پشته (35٪)، پیدا کردن PLT (29٪)، پیدا کردن ابزار BROP (20 درصد) و اتمام (16٪). توجه داشته باشید که اگر canary استفاده نشود، پرچم PIE نیز استفاده نمی‌شود (به‌طور پیش فرض) پس از آن خواندن از پشته قابل اجتناب خواهد بود. پیدا کردن PLT تا حد زیادی به اندازه اجرا و اینکه چگونه بسیاری از PLTها در طول یک اسکن از قلم می‌افتند بستگی دارد. همانطور که قبلا ذکر شد، ابزار اسکن BROP به فرکانس آن بستگی دارد.

این حمله می‌تواند در عرض 20 دقیقه کامل شود. MySQL مدت زمان طولانی را می‌طلبد زیرا بعد از هر شکست مجدد راه‌اندازی می‌شود. در این میان nginx سریع‌ترین (تنها یک دقیقه) بود چرا که در ابزار توقف براساس غیر زمان استفاده می‌شد. یک خطای HTTP برای زنده نگه داشتن ارتباط مورد استفاده قرار می‌گیرد و به همین ترتیب بعد از درخواست بهره‌برداری، یک درخواست عادی فرستاده می‌شود تا بررسی کند که آیا اتصال هنوز زنده است. در سرور اختصاصی به جای آن، یک ایست برای تعیین اینکه آیا سرور هنوز زنده است استفاده شود، حمله آهسته‌تر می‌گردد.

حمله به وضوح پر سر و صدا است، اما ما معتقدیم اگر آن به اندازه کافی سریع اجرا شود، مهاجم ممکن است قادر به انجام هر فعالیتی قبل از گرفتار شدن باشد. به عنوان مثال لاگ مربوط به nginx در هر شکست، در یک فایل متعلق به ریشه ذخیره می‌گردد. سرور طوری اجرا می‌شود که مهاجم قادر به پاک کردن لاگ‌های مربوطه نیست. ما متوجه می‌شویم، که فرایندهای کارگر توصیفگرهای فایل را در لاگ‌های مربوط نگه می‌دارند و ممکن است shellcode برای فراخوانی ftruncate برای پاک کردن آثار حمله نوشته شود.

B. ثبات

سه سرور از فرآیندهای کارگر بسیار متفاوت استفاده می‌کنند، تمرین BROP به روش‌های مختلف صورت می‌گیرد. در تمام موارد حمله BROP قابل اعتماد بود و بدون کمک تکمیل می‌شد.

MySQL (معمولا) تک فرآیندی، چندنخی است. در شکست، یک اسکریپت (mysqld امن) سرور را مجدد راه‌اندازی می‌کند. حمله BROP تحت تنظیمات پیش‌فرض (بدون PIE، اما canary) علیرغم اجرای مجدد عمل می‌کند زیرا

canary هرگز به چگونگی عمل باگ آسیب نمی‌رساند. در صورت اجرا با پرچم PIE، حمله نمی‌تواند به‌عنوان ابزاری که نمی‌تواند (تغییر) آدرس برگشت از پشته را برای شکست ASLR بخواند. این مورد در nginx اعمال نمی‌شود و خدمات اختصاصی آن با توجه به ماهیت fork و حمله موفق حتی زمانی که PIE استفاده شده باشد اتفاق می‌افتد.

nginx چندین فرآیندهای کارگر متعدد و یک معماری مبتنی بر رویداد و تک نخی دارد. اکثر توزیعهای آن، چهار پردازش کارگر را به‌طور پیش‌فرض پیکربندی می‌کنند. این عمل باعث ایجاد سناریو حیله می‌شودس زیرا یک حلقه نامتناهی براساس ابزار توقف به طور کامل عمل می‌کند. در اینجا ابزار توقف به یک فریم بالاتر پشته بازگردانده می‌شود، که از هر گونه مسدود کردن اجتناب شده است. با بهره‌برداری تخصصی، ما قادر به بهره‌برداری از nginx حتی به هنگام پیکربندی آن برای استفاده به عنوان یک کارگر هستیم.

سرور اختصاصی در هر اتصال fork می‌شود. این باعث می‌شود هنگامی که تعدادی فرآیند نامتناهی کارگر دردسترس هستند، حملات بسیار قابل اعتماد باشد. ما در مورد پیشینه و جزئیات سرور اطلاعاتی نداریم اما شامل اندک چیز منحصر به فردی است. سرریز پشته یک فریم پشته با باگ واقعی به‌عنوان یک تابع است که خواندن پاسخ سیستم را دچار مشکل می‌کند. همچنین سرور شامل یک حلقه تنها است که، وابسته به یک متغیر استفاده شده برای خروج از حلقه برای بستن خدمات است. این مسئله چالش اضافی ایجاد می‌کند که به‌راحتی به عنوان یک ابزار حلقه بی‌نهایت قابل استفاده نیست.

ابزار توقف برای yaSSL + MySQL، nginx و سرور اختصاصی به ترتیب زیر است: futex، بازگشت به بالاترین فریم فراخوانی و sleep.

سناریوی yaSSL + MySQL یک سرریز بافر بسیار کوچک ارائه می‌کند و نشان می‌دهد که BROP می‌تواند حتی با بافر کوچک کار کند (64 بایت کافی است).

کلید موفقیت و ثبات حمله BROP این که مهاجم نیاز به اسکن برای یک آیتم در هر زمان دارد. هیچ وابستگی بین موارد مورد نیاز برای حمله وجود دارد. همچنین، مواردی که جستجو می‌شوند (به‌عنوان مثال، ابزارPLT و BROP) و هیچ مثبت کاذبی در طول حمله پیدا نمی‌شود. همچنین حمله با توجه به پیشرفت صورت گرفته قوی‌تر می‌شود.

C. کمک سورس کد

اگر نفوذگر اطلاعاتی از سرور (به عنوان مثال، کد منبع) داشته باشد تعداد درخواست ممکن است پایین بیاید. برای مثال در nginx، مهاجم نیازی به پیدا کردن ابزار توقف ندارد زیرا در حال حاضر در یک فریم بالاتر پشته وجود دارد. همچنین مهاجم ایده‌ای در مورد بزرگی باینری و تعداد ورودی‌های PLT برای اسکن سریع‌ترPLT ، به‌عنوان مثال، با پرش از ورودی‌های بیشتر و نزدیک‌تر شده به PLT دارد. در مورد وب سرور nginx، درست پس از تماس سرریز رخ می‌دهد بنابراین RDI (آرگومان اول) دارای سوکت عدد است. مهاجم می‌تواند این حقیقت را با فراخوانی write بدون تنظیم RDI یا فراخوانی dup2 برای کپی کردن توصیفگر فایل با یک عدد ثابت انجام دهد. آگاهی از این که خواندن‌ها غیر قابل مسدود شدن هستند مفید است به‌طوری‌که روتر قطعه‌بندی IP برای غلبه بر سرریز استفاده شده باشد. با همه موارد ذکر شده، نسخه بهینه‌سازی شده از وب سرور nginx برای حمله BROP که 1000 درخواست در زمان نیاز دارد یک پیش‌فرض در نصب Debian (بدون canary یا PIE) است. این نسخه باید بر روی هر توزیع کار کند و مختص باینری نباشد.

دانش منبع در مورد yaSSL کمک کرده است. باگ‌ها را می‌توان به سه روش مورد بهره برداری قرار داد، اما تنها مجاز به دور زدن canar هستیم. هر سه آسیب‌پذیری باید از طریق آزمایش‌های fuzz از راه دور و به طور مستقل آشکار شوند. BROP تنها بر روی آسیب‌پذیری که در آن خواندن از پشته اتفاق می‌افتد موفق عمل می‌کند: که در آن canary قابل لمس نیست.

براساس اطلاعات منبع، یک مهاجم ممکن است تعیین کند که آیا RDX در زمان بهره‌برداری دارای ارزش است. اگر چنین باشد، مهاجم ممکن است نیاز به پیدا کردن strcmp برای کنترل RDX داتشته باشد و با ادامه حمله بهینه‌تر گردد.

برخی از اکتشافات درحالی اتفاق می‌افتند که خدمات اختصاصی هک کورکورانه می‌توانست آشکارا در مقابل کد منبع و ساده شده حمله قرار گیرد. در طول فاز خواندن از پشته، متوجه شدیم که عدد 4 در پشته حاضر بود. در واقع تعداد سوکت و حمله می‌توانست یک توصیفگر فایل brute-force باشد که بعدا از آن اجتناب می‌شود. همچنین، به هنگام

تلاش برای خواندن آدرس بازگشت ذخیره شده، آدرس برگشت خاصی پیدا شد که نوشتن "OK" در شبکه را اجبار می‌کرد. این مسئله می‌تواند به‌عنوان یک ابزار توقف استفاده شده است تا از نیاز به اسکن اجتناب گردد.

خواندن از پشته یک ابزار بسیار مفید برای هک کور است. که نشان می‌دهد که آیا canary استفاده شده است، اشاره‌گر فریم فعال شده است. این مسئله به انگشت‌نگاری توزیع (براساس پیش فرض) یا محیط هک شده کمک می‌کند.

خواندن از پشته در yaSSL نیز نشان داد که سرریز در "خلاف" جهت و با توجه به اشاره‌گر محاسباتی اتفاق می‌افتد- اولین بایت از اطلاعات ارسال شده استکه در شکست برنامه بسیار موثر بود.

11. محدودیت ها

حمله BROP محدودیت‌های خاص خود را دارد. ما آن را تنها به سرریز ساده پشته اعمال می‌کنیم. درحالی‌که آن یک نقطه شروع خوب در بسیاری از آسیب‌پذیری‌های پیچیده و مبتنی بر heap است.

خواندن از پشته فرض می‌کند که مهاجم می‌تواند در یک بایت سرریز کند و آخرین بایت سرریز شده را کنترل کند (به‌عنوان مثال، یک صفر توسط سرور اضافه نشده است).

حمله فرض می‌کند که دستگاه و روند یکسانی بعد از هر تلاش می‌تواند اتفاق بیافتد. توازن بار می‌تواند باعث شکست حمله گردد به خصوص زمانی که PIE استفاده شده است و canary نمی‌تواند برخورد کند.

همچنین این حمله به تعدادی کارگران دردسترس متکی است و در وضعیتی که همه کارگران در یک حلقه بی نهایت گرفتار شده‌اند به پایان نمی‌رسد. این مسئله باعث می‌شود انتخاب ابزار توقف بسیار مهم باشد. بازگشت به فریم بالاتر پشته، بهینه‌سازی کلیدی به حساب می‌آید، که در آن کارگر به جای توقف"از سر گرفته" می‌شود. اگر این مسئله انجام نشود و تعداد محدودی از فرآیندهای کارگر وجود داشته باشند ابزار توقف آنها به‌طور نامحدود از کار می‌افتد، این حمله ممکن است کامل نباشد. nginx یک مثال است که به‌عنوان یک پیکربندی با یک کارگر و مبتنی بر رویداد اتفاق می‌افتد. بااین حال، هنوز هم BROP موفق عمل می‌کند چرا که برگرداندن یک فریم بالاتر پشته امکان‌پذیر است.

12. بحث

A. BROP در سیستم عامل‌های مختلف

ویندوز فاقد یک API شبیه fork (تنها CreateProcess) است بنابراین canary و بخش متنی آدرس پایه برای تصادفی‌سازی پس از شکست تضمین شده‌اند، بنابراین ساخت سیستم قوی در برابر حملاتی مانندBROP بسیار ضروری به نظر می‌رسد. ویندوز ABI آرگومان‌ها را در ابتدای رجیستر (به‌عنوان مثال، RCX، RDX) برای ساختن ابزار و وسایل pop عبور می‌دهد. این ابزارها بسیار کمیاب هستند چرا که در طول فراخوانی تابع حفظ نمی‌شوند، بنابراین کامپایلر نیازی به ذخیره آنها در پشته ندارد. این ابزارها به احتمال زیاد وجود دارند ولی از آنها کمتر استفاده می‌شود.

پیاده‌سازی ASLR توسط سیستم‌عامل متفاوت است. ویندوز 8.1 و Mac OS X همه‌چیز را به‌طور پیش فرض تصادفی‌سازی می‌کند. متاسفانه، هر دو سیستم، کتابخانه‌های سیستم را تنها در زمان راه‌اندازی مجدد تصادفی‌سازی می‌کنند. این مسئله می‌تواند یک محیطBROP مانند اشاره‌گر به ایجاد کتابخانه‌های سیستم ایجاد کند. ری بسیار کمیاب است، به‌خصوص در مشتریان و سیستم‌های لپ‌تاپ که در آن کاربران ممکن است راه‌اندازی مجدد را به تعویق بیاندازند. Mac OS X هم تنها از آنتروپی16 بیتی برای ASLR پشتیبانی می‌کند، که به مراتب بالاتر از دیگر سیستم عامل‌های 64 بیتی است. در لینوکس، اثربخشی ASLR به توزیع و پیکربندی PIE آن بستگی دارد. برای مثال، اوبونتو، PIE را به‌طور پیش فرض فعال نمی‌کند، اما آن را براساس کاربرد آن در خطرات فعال می‌کند[16].

B. BROP فراتر از سرریز پشته

حمله BROP بر حملات پشته، ساده‌ترین سناریو ممکن تمرکز دارد. ما سوء استفاده از پشته را در نظر نمی‌گیریم، هر چند ممکن است اتفاق بیافتد. به عنوان مثال، Bootstrap، به‌عنوان یک ابزار پشته محور پس از ابزار توقف متفاوت خواهد بود. مهاجم هم اکنون می‌تواند یک پشته و زنجیر ROP در بافر راه‌اندازی کند.

جدول 4. تنوع کد زمانی که نسخه‌های مشابه Nginx (1.4.0) با نسخه‌های مختلف Debian و لینوکس کامپایل می‌شوند.

	Text Size	Text Start	# of Gadgets
Squeeze	0x5fc58	0x4031e0	206
Wheezy	0x61f0c	0x4032f0	255
Jessie (testing)	0x5fbd2	0x402ee0	323

C. سمت سرویس گیرنده در مقابل سمت سرور

امکان راه اندازی حمله BROP مانند در مشتریان وجود دارد. برای مثال،مرورگرهایی مانند کروم، پلاگین‌ها را در یک فرایند جداگانه برای نیرومندی بیشتر راه‌اندازی می‌کند. جاوا اسکریپت می‌تواند برای ایجاد اشیاء آسیب‌پذیر با چندین پلاگین، تلاش برای بهره‌برداری و تشخیص اینکه آیا آنها بدون کاربران شکست می‌خورند یا نه، مورد استفاده قرار گیرد. توجه داشته باشید، که به‌طور معمول شکست کمتر در سمت سرویس گیرنده اتفاق می‌افتد. داشتن قدرت اجرای جاوا اسکریپت دردسترس می‌تواند مکانیسم‌های سیگنالینگ بیشتری در مقایسه با شکست/ عدم شکست در BROP سمت سرور به مهاجم ارائه دهد.

تمایز جالب بین سمت سرویس‌گیرنده و سمت سرویس‌دهنده این است که اغلب حملات سمت سرویس‌گیرنده کمتر مورد هدف قرار می‌گیرند. به‌عنوان مثال، یک مهاجم ممکن است هر تعداد از مشتریان خود را برای سرقت اطلاعات و یا ساخت یک botnet وارد عمل کند. این عمل باعث سوء استفاده برای اهداف بزرگتر با حمایت‌های کمتری (به‌عنوان مثال، ویندوز XP) می‌شود که هنوز هم باارزش هستند و مردم در حال اجرای پیکربندی آن هستند. حملات سمت سرور اغلب به‌عنوان حمله به یک سایت خاص مورد هدف قرار می‌گیرند. تکیه بر اهداف 32 بیتی و یا نصب باینری خاص یا حرکت به سوی قربانی بعدی ممکن است یک گزینه نباشد. این باعث می‌شود که BROP در سمت سرور به عنوان یک مهاجم در زمان مورد نیاز بسیار با ارزش باشد.

D. واریانس در فایل‌های باینری

برخلاف سایر سیستم‌ها، سیستم‌های منبع بسته (در openbinary) در نوشتن سوء استفاده ساده‌تر عمل می‌کنند. بسیاری از سوء استفاده‌هایی که ویندوز را مورد هدف قرار می‌دهند بسیار قوی هستند و توانایی ساخت زنجیره‌های ROP در DLL را برای تغییر ناگهانی دارند به‌طوری‌که تنها چند نسخه وجود دارد. در یک محیط منبع باز، چندین نوع باینری وجود دارد و مهاجم باید یک زنجیره ROP مختلف برای هر یک بسازد. جدول 4 اندازه و آدرس شروع توزیع‌های مختلف با نسخه یکسان در nginx را نشان می‌دهد. همانطور که مشاهده کردیم تنوع بسیاری براساس محیط ساخته شده وجود دارد، نسخه کتابخانه‌ها باهم در ارتباط هستند و نسخه کامپایلر در توزیع یکسان لینوکس استفاده می‌شود. حتی تفاوت یک بایتی و یا آفست موجب شکست استاتیک از پیش محاسبه در زنجیره ROP می‌گردد.

بدترین مورد برای مهاجم این است که، یک سیستم به صورت دستی توسط کاربر نهایی کامپایل شود، این مسئله حمله برای ایجاد یک زنجیره آنلاین ROP در باینری ناشناخته را غیر ممکن می‌سازد. در اینگونه موارد BROP یک ضرورت است. حتی اگر یک سرور از باینری از پیش تالیف شده استفاده کند، می‌تواند در تعیین یکی موارد خاص مورد استفاده قرار گرفته مشکل ایجاد کند: سیستم عامل انگشت‌نگاری از راه دور نسخه هسته‌ی یک توزیع را به صورت تقریبی نشان می‌دهد. در واقعBROP می‌تواند برای توزیع اثر انگشت و برنامه‌های کاربردی مورد استفاده قرار گیرد.

E. تست fuzz از راه دور

حمله BROP می‌تواند یک ابزار قدرتمند برای هک خدمات اختصاصی بسته‌های باینری به هنگام کنترل از راه دور همراه با تست fuzz باشد. توجه داشته باشید که در دو تا از برنامه‌های کاربردی که هدف قرار دادیم، سرریز رخ داده است چرا که طول در بسته مشخص بود اما یک مقدار بزرگتر فرستاده شد. مطمئنا امکان ارسال یک تست کننده fuzz وجود دارد که درباره‌ی یک پروتکل و تلاش برای سرریز تهیه شده توسط طول نادرست اطلاعاتی دارد [17]. جالب توجه است، تقریبا همان تقطیع آسیب‌پذیر که در nginx قابل رویت است در گذشته در آپاچی [18] ارائه شده بود. امکان نوشتن تست fuzz برای شرایط خاص پروتکل که به‌عنوان پیاده‌سازی سخت شناخته شده وجود دارد.

13. پیشگیری BROP

در زیر در مورد مکانیسم‌های دفاعی بحث شده است که حمله BROP نیز جزء آنها است، برای مثال دو اقدام احتیاطی برای استفاده‌ی توسعه‌دهندگان سرور پیشنهاد شده است. بسیاری از تحقیقات گذشته در مورد مکانیسم‌های دفاعی حمله ROP است و بسیاری از این تکنیک‌ها قابل اجرا در برابر BROP هستند. بنابراین، این لیست به هیچ وجه جامع نیست.

A. تصادفی‌سازی

حفاظت اساسی در برابر حمله BROP تصادفی‌سازی canary و ASLR به‌عنوان امکان پذیر است. این مکانیسم‌های حافظت موثر هستند، اما توسعه‌دهندگان سرور آنها را با عدم تصادفی‌سازی تضعیف می‌کنند. ساده‌ترین روش فرآیند fork و exec در یک شکست است، که canary و ASLR را تصادفی‌سازی می‌کند. بنابراین مهم است که هر پروسه‌ی فرزند به‌طور مستقل تصادفی شود به‌طوری‌که هر گونه اطلاعات به دست آمده از فرزند نتواند در برابر دیگری مورد استفاده قرار گیرد. تحقیقات زیادی در زمینه تصادفی‌سازی باینری در زمان اجرا وجود دارد. یکی از روش‌هایی که توسط Giuffrida و همکارانش بیان شده، استفاده از یک کامپایلر اصلاح شده برای انتقال حالت اجرا بین دو نمونه (با روش تصادفی‌ساز مختلف ی ASLR) است [19]. همچنین نمونه‌سازی مجدد یک تکنیک برای انتقال بخش متن باینری به یک مکان جدید با استفاده از mmap / munmap در این مقاله اعمال شده است که از یک کنترلر خطای صفحه برای تعیین اینکه آیا اشاره‌گر باید به عنوان خطا بازنویسی شود، استفاده می‌کند.

بهبودی که در این مقاله بیان شده است، تصادفی‌سازی canary در هر کاربر یا هر درخواست است. ما پیشنهاد می‌کنیم که سرور یک canary جدید قبل از ورود به عملکرد perrequest بنویسد. در بازگشت به سوی آن تابع، canary قدیمی باید ترمیم شود به‌طوری‌که اجرا را ادامه دهد. درحالی‌که این محافظت در برابر باگ‌ها در nginx و سرور اختصاصی صورت می‌گیرد، حمله خاص علیه yaSSL از canary جلوگیری می‌کند.

B. خواب در شکست

سیستم‌هایی مانند NetBSD’s segvguard [20] و grsec’s deter_bruteforce برای لینوکس [21] به تاخیر انداختن fork بعد از تقسیم‌بندی خطا را پیشنهاد می‌کنند. این تکنیک موجب کاهش سرعت حملات می‌گردد به‌طوری‌که یک مدیر می‌تواند متوجه اشتباه شده و به مشکل رسیدگی کند. حرکت نزولی این رویکرد این است که باگ‌ها می‌توانند حملات را به آسانی انکار کنند. همچنین معلوم نیست چه مقداری برای تاخیر خوب است. grsec یک تاخیر 30 ثانیه‌ای پیشنهاد می‌کند. درحالی‌که برای اکثر تنظیمات کافی است: حمله بهینه‌سازی شده‌ی BROP برای nginx را می‌توانید در 1،000 درخواست، برای حدود 8 ساعت حمله کامل کنید.

درحالی‌که حملات انکار سرویس جدی هستند و ازدست دادن داده‌های خصوصی می‌تواند بدتر باشد. در برخی شرایط، سرورها نباید مجددا عمل کنند، اما کاربران و توسعه‌دهندگان در عمل این راه حل قابل قبول نمی‌یابند. دسکتاپ‌های مدرن لینوکس از systemd برای نظارت بر خدمات و راه اندازی مجدد خدمات در شکست استفاده می‌کنند. توسعه‌دهندگان باید در مورد استفاده از خدمات از راه دور که واقعا نیاز به راه‌اندازی مجدد به صورت خودکار برای کاهش سطح حمله دارند محتاط باشند.

C. حفاظت ROP

دسته‌ی دیگری از مکانیسم‌های دفاعی وجود دارند که در برابر حملات ROP دفاع می‌کنند. در مرحله اول، تمامیت کنترل جریان (CFI) [22] مانع از برنامه‌نویسی بازگشتی با اجرای نمودار کنترل جریان می‌گردد. چندین تکنیک مشابه دیگر نیز وجود دارد.

رویکرد دیگر توسعه یافته توسط Pappas و همکارانش، به اجرا درآوردن کنترل جریان در داخل کنترل سیستم فراخوانی با مقایسه پشته در برابر آخرین رکورد (LBR) موجود در پردازنده های اینتل است [23]. این را می‌توان به منظور بررسی پشته‌ی دستکاری نشده است استفاده کرد. محدودیت اصلی عمق پشته است که می‌تواند به‌عنوان چهار ورودی بسته به مدل پردازنده بررسی گردد.

راه‌حل‌هایی وجود دارد که اضافه کردن اتفاقی به فایل‌های باینری را پیشنهاد می‌دهند [24]. درحالی‌که این مسئله در برابر ROP در یک محیط BROP نیست موثر است. تکنیک‌های اضافی در جهت تلاش برای تصادفی‌سازی مکان ابزار در هر نمونه قابل اجرا وجود دارد، اما هیچ دفاعی در برابر BROP ارائه نمی‌دهد مگر اینکه باینری به طور کامل مجدد راه اندازی (fork و exec) شود [25]، [26]. همچنین تکنیک‌هایی برای حذف و یا کاهش تعداد ابزارهای دردسترس وجود دارد [27] که می‌تواند به‌طور موثر در برابر حملات ROP محافظت شود.

D. تکنیک‌های کامپایلر

بسیاری از کامپایلرهای مدرن از قرار دادن مرزهای زمان اجرا برای بررسی بافر پشتیبانی می‌کنند. این مسئله مانع از کلاس باگ‌ها می‌شود. LLVM شامل AddressSanitizer است، ابزاری برای بررسی مرزها و استفاده از باگ‌های آزاد [28]. چارچوب SafeCode بر روی LLVM ساخته شده است و همچنین به بررسی مرزهای میان چیزهای دیگر می‌پردازد [29]. همچنین کامپایلر اینتل از بررسی مرزهای زمان اجرا پشتیبانی می‌کند.

مشکل اصلی تمام این راه‌حل‌ها این است که ممکن است آنها سرعت عملکرد را مانند تست استفاده شده کاهش دهند. یک نقطه روشن برای این راه‌حل‌های عملی این است که اینتل مجموعه‌ای از دستورالعمل‌ها را به منظور کاهش هزینه بررسی مرزهای متغیرها اعلام کرده است [30].

14. کارهای مرتبط

کارهای مرتبط زیادی برای اسکن یک ابزار ROP وجود دارد. حمله‌ی نیم کور Goodspeed علیه میکروکنترلرها [31] متکی بر دانش (عمومی) کد bootloader است و یک ابزار را در یک بخش ناشناخته از حافظه برای ایجاد حمله مورد استفاده برای استخراج سیستم عامل مهیا می‌کند. حمله BROP کلی‌تر است زیرا به‌طورکامل کور است و تکنیک‌هایی برای پیدا کردن و زنجیروارد کردن ابزار مختلف ارائه می‌دهد.

در زمینه‌ی مجموعه دستورالعمل‌های تصادفی‌سازی [32] که از تکنیک‌های مشابه استفاده می‌کنند حمله BROP وجود دارد. در این مورد، سیگنال برای حدس درست / نادرست از شکست یا عدم شکست کمک میگیرد. با این حال، هدف، از دست دادن یک کلید رمزنگاری و روش فرضی است که تزریق کد مخرب را می‌توان در وهله اول انجام داد: به‌عنوان مثال، هیچ ASLR و هیچ NX یکجا اتفاق نمی‌افتد.

خواندن از پشته برای تعیین canary به‌خوبی شناخته شده است [33]. محققان همانند مهاجمان نشان داده‌اند که چگونه روش brute-force 32 بیتی ASLR [9] عمل می‌کند، اما این رویکرد بر روی ماشین های 64 بیتی عملی نیست. این مقاله خواندن از پشته را برای ذخیره‌ی آدرس برگشت و اشاره‌گر فریم برای شکستن 64 بیتی ASLR تعمیم می‌دهد.

بسیاری از محققان برایم مورد اشاره دارند که سوء استفاده چند مرحله‌ای است و نیاز به ازدست دادن اطلاعات دارد. در nginx 32 بیتی Kingcope محل نوشتن در PLT برای نشت باینری جهت یافتن ابزار است [14]، [34]. این تکنیک می‌افتد بر روی سیستم‌های 64 بیتی به‌صورت کوتاه اتفاق می‌افتد چون آیتم‌های متعددی نیاز به اجبار دارند: محل نوشتن و تمام ابزار مورد نیاز برای استدلال نوشتن (دومی درسیستم 32 بیتی مورد نیاز نیست) ضروری است. این مسئله باعث می‌شود که حمله BROP مانند یک ضرورت در سیستم 64 بیتی عمل کند. حتی Kingcope اذعان می‌کند که مشکل اصلی در تعمیم رویکرد به این سیستم عامل است. نویسنده همچنین اذعان می‌کند که بهره برداری بر روی شبکه‌های گسترده به دلیل خواندن غیر مسدود شده وب سرور nginx خوب عمل نمی‌کند. nginx یک مطالعه موردی برای نشان دادن مشکلات درگیر در هنگام نوشتن در سمت سرور است.

سمت سرویس‌گیرنده از نویسندگانی که شانس بیشتری در 64 بیتی و ASLR دارند استفاده می‌کند [35]. Pwn2Own از آسیب‌پذیری جاوا اسکریپت برای نشت اشاره‌گر استفاده می‌کند و سپس از همان آسیب‌پذیری برای نشت تمام محتویات کتابخانه chrome.dll برای ایجاد یک زنجیره ROP استفاده می‌کند. این مسئله نشان می‌دهد که چگونه سوءاستفاده‌ها تمایل به باینری مستقل از استحکام دارند. این مسئله موجب سافت سیستم‌های منبع بسته

متفاوتی می‌گردد که در آن نسخه نسبتا کمتری از باینری وجود دارد، چرا که کروم نسخه جدیدی از DLL را که مشابه نسخه‌ی قبلی است نمایش می‌دهد.

15. نتیجه‌گیری

ما نشان دادیم که، تحت شرایط مناسبی، نوشتن سوء استفاده بدون هیچ گونه دانش باینری و یا کد منبعی ممکن است. این روش برای آسیب‌پذیری‌های پشته که در آن فرآیند سرور پس از شکست مجدد راه‌اندازی می‌شود بهرت عمل می‌کند. حمله ارائه دهش در این مقاله قادر به شکست ASLR، NX و پشته canary در سرور مدرن لینوکس 64 بیتی است. در این مقاله دو روش جدید ارائه شده است: خواندن از پشته تعمیم‌یافته، که موجب شکست کامل ASLR بر روی سیستم‌های 64 بیتی می‌شود و حمله BROP، که قادر به یافتن از راه دور ابزارهای ROP است. ابزار کاملا خودکار ارائه شده در این مقاله، Braille، می‌تواند 4000 درخواست را در کمتر از 20 دقیقه، در برابر نسخه‌های واقعی yaSSL + MySQL و nginx با آسیب‌پذیری‌های شناخته شده و ابزار خدمات اختصاصی در حال اجرای یک باینری ناشناخته، مورد آزمایش قرار دهد.

ما نشان دادیم که الگوهای طراحی مانند fork سرور با فرآیندهای متعدد کارگر می‌تواند در تقابل با ASLR باشد و ASLR فقط هنگامی مؤثر است که همه کد به بخش باینری (از جمله PIE) اعمال شود. علاوه‌براین، امنیت از طریق ابهام، که در آن باینری ناشناخته و یا تصادفی است، تنها می‌تواند کاهش سرعت گردد اما از حملات سرریز بافر جلوگیری نمی‌کند. برای دفاع در برابر حمله ما، پیشنهاد می‌کنیم که سیستم‌ها باید ASLR و canary را بعد از هر شکست تصادفی‌سازی کنند و هیچ کتابخانه یا قابلیت اجرایی نباید از ASLR معاف باشد.

Braille در لینک زیر قابل دسترسی است: http://www.scs.stanford.edu/brop

REFERENCES

[1] R. Roemer, E. Buchanan, H. Shacham, and S. Savage, "Return-oriented programming: Systems, languages, and applications," ACM Trans. Inf. Syst. Secur., vol. 15, no. 1, pp. 2:1–2:34, Mar. 2012. [Online]. Available: http://doi.acm.org/10.1145/2133375.2133377

[2] mitre. Cve-2013-2028. [Online]. Available: http://cve.mitre.org/cgi-bin/ cvename.cgi?name=CVE-2013-2028

[3] ——. Cve-2008-0226. [Online]. Available: http://cve.mitre.org/cgi-bin/ cvename.cgi?name=CVE-2008-0226

[4] A. One, "Smashing The Stack For Fun And Profit," Phrack, vol. 7, no. 49, Nov. 1996. [Online]. Available: http://phrack.com/issues.html? issue=49&id=14#article

[5] M. Kaempf. Vudo malloc tricks by maxx. [Online]. Available: http://www.phrack.org/issues.html?issue=57&id=8&mode=txt

[6] S. Designer. Getting around non-executable stack (and fix). [Online]. Available: http://seclists.org/bugtraq/1997/Aug/63

[7] P. Team. Pax address space layout randomization (aslr). [Online]. Available: http://pax.grsecurity.net/docs/aslr.txt

[8] S. Bhatkar, D. C. DuVarney, and R. Sekar, "Address obfuscation: an efficient approach to combat a board range of memory error exploits," in Proceedings of the 12th conference on USENIX Security Symposium - Volume 12, ser. SSYM'03. Berkeley, CA, USA: USENIX Association, 2003, pp. 8–8. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251353.1251361

[9] H. Shacham, M. Page, B. Pfaff, E.-J. Goh, N. Modadugu, and D. Boneh, "On the effectiveness of address-space randomization," in Proceedings of the 11th ACM conference on Computer and communications security, ser. CCS '04. New York, NY, USA: ACM, 2004, pp. 298–307. [Online]. Available: http://doi.acm.org/10.1145/1030083.1030124

[10] gera and riq. Advances in format string exploitation. [Online]. Available: http://www.phrack.org/archives/59/p59 0x07 Advances% 20in%20format%20string%20exploitation by riq%20&%20gera.txt

[11] C. Cowan, C. Pu, D. Maier, H. Hintony, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, and Q. Zhang, "Stackguard: automatic adaptive detection and prevention of buffer-overflow attacks," in Proceedings of the 7th conference on USENIX Security Symposium - Volume 7, ser. SSYM'98. Berkeley, CA, USA: USENIX Association, 1998, pp. 5–5. [Online]. Available: http://dl.acm.org/citation.cfm?id=1267549.1267554

[12] H. Etoh, "GCC extension for protecting applications from stacksmashing attacks (ProPolice)," 2003, http://www.trl.ibm.com/projects/ security/ssp/. [Online]. Available: http://www.trl.ibm.com/projects/ security/ssp/

[13] Bulba and Kil3r, "Bypassing stackguard and stackshield," Phrack Magazine, May 2000. [Online]. Available: http://phrack.org/issues. html?issue=56&id=5#article

[14] Kingcope. About a generic way to exploit linux targets. [Online]. Available: http://www.exploit-db.com/wp-content/themes/exploit/docs/ 27074.pdf

[15] G. F. Roglia, L. Martignoni, R. Paleari, and D. Bruschi, "Surgically returning to randomized lib(c)," in Proceedings of the 2009 Annual Computer Security Applications Conference, ser. ACSAC '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 60–69. [Online]. Available: http://dx.doi.org/10.1109/ACSAC.2009.16

[16] Ubuntu security features. [Online]. Available: https://wiki.ubuntu.com/ Security/Features

[17] Peach fuzzer. [Online]. Available: http://peachfuzzer.com/

[18] mitre. Cve-2002-0392. [Online]. Available: http://cve.mitre.org/cgi-bin/ cvename.cgi?name=CVE-2002-0392

[19] C. Giuffrida, A. Kuijsten, and A. S. Tanenbaum, "Enhanced operating system security through efficient and fine-grained address space randomization," in Proceedings of the 21st USENIX conference on Security

symposium, ser. Security'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 40–40. [Online]. Available: http: //dl.acm.org/citation.cfm?id=2362793.2362833

[20] E. Efrat. Segvguard. [Online]. Available: http://www.netbsd.org/~elad/ recent/man/security.8.html

[21] grsecurity. Deter exploit bruteforcing. [Online]. Available: http://en.wikibooks.org/wiki/Grsecurity/Appendix/Grsecurity and PaX Configuration Options#Deter exploit bruteforcing

[22] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti, "Controlflow integrity," in Proceedings of the 12th ACM Conference on Computer and Communications Security, ser. CCS '05. New York, NY, USA: ACM, 2005, pp. 340–353. [Online]. Available: http://doi.acm.org/10.1145/1102120.1102165

[23] V. Pappas, M. Polychronakis, and A. D. Keromytis, "Transparent ROP exploit mitigation using indirect branch tracing," in Proceedings of the 22nd USENIX conference on Security, ser. SEC'13. Berkeley, CA, USA: USENIX Association, 2013, pp. 447–462. [Online]. Available: http://dl.acm.org/citation.cfm?id=2534766.2534805

[24] R. Wartell, V. Mohan, K. W. Hamlen, and Z. Lin, "Binary stirring: Self-randomizing instruction addresses of legacy x86 binary code," in Proceedings of the 2012 ACM Conference on Computer and Communications Security, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 157–168. [Online]. Available: http://doi.acm.org/10.1145/ 2382196.2382216

[25] J. Hiser, A. Nguyen-Tuong, M. Co, M. Hall, and J. W. Davidson, "Ilr: Where'd my gadgets go?" in Proceedings of the 2012 IEEE Symposium on Security and Privacy, ser. SP '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 571–585. [Online]. Available: http://dx.doi.org/10.1109/SP.2012.39

[26] V. Pappas, M. Polychronakis, and A. D. Keromytis, "Smashing the gadgets: Hindering return-oriented programming using in-place code randomization," in Proceedings of the 2012 IEEE Symposium on Security and Privacy, ser. SP '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 601–615. [Online]. Available: http://dx.doi.org/10.1109/SP.2012.41

[27] K. Onarlioglu, L. Bilge, A. Lanzi, D. Balzarotti, and E. Kirda, "G-free: defeating return-oriented programming through gadget-less binaries," in Proceedings of the 26th Annual Computer Security Applications Conference. ACM, 2010, pp. 49–58.

[28] T. C. Team. Addresssanitizer - clang 3.4 documentation. [Online]. Available: http://clang.llvm.org/docs/AddressSanitizer.html

[29] D. Dhurjati, S. Kowshik, and V. Adve, "SAFECode: Enforcing alias analysis for weakly typed languages," in Proceedings of the 2006 ACM SIGPLAN Conference on Programming Language Design and Implementation, ser. PLDI '06. New York, NY, USA: ACM, 2006, pp. 144–157. [Online]. Available: http://doi.acm.org/10.1145/1133981. 1133999

[30] Intel. Introduction to intel memory protection extensions. [Online]. Available: http://software.intel.com/en-us/articles/ introduction-to-intel-memory-protection-extensions

[31] T. Goodspeed and A. Francillon, "Half-Blind Attacks: Mask ROM Bootloaders are Dangerous," in WOOT, 2009.

[32] A. N. Sovarel, D. Evans, and N. Paul, "Where's the feeb?: The effectiveness of instruction set randomization," in Usenix Security, 2005.

[33] A. Zabrocki. Scraps of notes on remote stack overflow exploitation. [Online]. Available: http://www.phrack.org/issues.html?issue=67&id= 13#article

[34] Kingcope. nginx 1.3.9/1.4.0 x86 brute force remote exploit. [Online]. Available: http://www.exploit-db.com/exploits/26737/

[35] M. Labes. Mwr labs pwn2own 2013 write-up - webkit exploit. [Online]. Available: https://labs.mwrinfosecurity.com/blog/2013/04/19/ mwr-labs-pwn2own-2013-write-up---webkit-exploit/