بسمه تعالی

گزارش پروژه نهایی درس سیستم های تشخیص نفوذ – مرداد ۹۲

پیاده‌سازی سیستم تشخیص DDoS در ترافیک بالا

مقدمه

در این پروژه یک سیستم تشخیص حمله DDoS در شرایطی که حجم درخواست از یک سرور زیاد است را بر اساس روشی که در مقاله [1] ذکر شده است، پیاده‌سازی کرده ایم. جزئیات روش در تمرین قبلی گزارش شده است ولی بطور مختصر ایده آن را در بخش «ایده مقاله» ذکر کرده ایم. سپس در بخش «پیاده سازی» به توضیحات مختصری در پیاده‌سازی می پردازیم و سپس در بخش «نتایج» به گزارش نتایج بدست آمده از اجرای سیستم اشاره می‌کنیم.

ایده مقاله

هدف ما این است که اگر در زمان پیک ترافیک یک سرور، از طریق یک Botnet به سرور حمله شود، بتوانیم درخواست های کاربران و درخواست‌های نامشروع را تفکیک کنیم و ترافیک کاربران بدخواه را مسدود کنیم تا سرور از سرویس‌دهی بازنماند.
در این مقاله برای رسیدن به این هدف از این فرض استفاده شده است که کاربران Botnet چون بصورت خودکار و توسط یک برنامه درخواست ارسال می‌کنند، در اکثر مواقع، سرعت دریافت بسته‌ها الگوی قابل پیش‌بینی دارد. برای مثال در برنامه ای که بسته ها را تولید می کند ممکن است یک حلقه وجود داشته باشد و دراین حلقه درخواست هایی بسمت سرور ارسال می شود. در این حالت مشخصات سخت افزاری یا پهنای باند رایانه مهاجم عامل تعیین کننده سرعت ارسال بسته ها است. پس در سمت سرور سرعت دریافت بسته های این کاربر یکنواخت است.

پیاده‌سازی

این سیستم با توجه به توضیحات موجود در مقاله [1] پیاده سازی شده است. از توضیح جزئیات روش صرف‌نظر می‌کنیم و در ادامه به توضیحات جزئیات سورس پیاده‌سازی شده می‌پردازیم:
این سیستم بصورت یک scrip به زبان Ruby پیاده‌سازی شده است. علت انتخاب این زبان سادگی کدنویسی و تسلط بنده به آن است. سورس این سیستم بصورت متن باز در آدرس https://github.com/mabdi/Fifa98DS قابل دسترس است.

کلیات پیاده سازی:

در این سیستم یک فایل دیتاست موجود است که اطلاعات آن خوانده و به سیستم داده خواهد شد. علاوه بر آن یک Botnet شبیه سازی شده است تا راحت تر بتوان درباره نتایج کار قضاوت کرد.
برنامه از زمان شروع ترافیک دیتاست شروع می‌کند و در هر واحد زمانی تمام بسته های دیتاست و شبیه‌ساز که مربوط به آن

واحد زمانی هستند را گرفته و به ids ارسال خواهد کرد. سیستم تشخیص نیز بدون داشتن دانشی از اینکه کدام بسته ها به دیتاست و کدام بسته ها به شبیه ساز هستند، بسته ها را پردازش کرده و کاربران با ترافیک غیرعادی را بلاک خواهد کرد.
در این پروژه فقط شبیه‌ساز بات‌نت با سرعت ثابت را پیاده سازی کرده ایم. همچنین فقط یکی از دو متد معرفی شده در مقاله را پیاده سازی کرده ایم. همچنین در این سیستم ویژگی دیگری نیز اضافه شده است — برای سرعت بخشیدن به پردازش بسته ها - که اگر کاربری برای یک مدت خاص درخواستی ارسال نکند، آنرا کاربر را کاربر با ترافیک سالم در نظر گرفته ایم.

توضیحات کد:

بخش ثابت ها: در این بخش بعضی ثابت ها تعریف شده اند. برای مثال نام فایل Dataset یا اندازه شبیه‌ساز Botnet یا ثابت های مربوط به روش مانند آلفاها یا K ها و ...
بخش متغییرها: در این بخش متغیرهای سراسری کل برنامه تعریف می شوند. این متغیرها عبارتند از:

- لیست کاربرانی که سالم تشخیص داده شده اند
- لیست کاربرانی که بدخواه تشخیص داده شده اند
- لیست همه کاربران و آبجکت پردازش بسته های آن

بخش کلاس ها: در این بخش کلاس‌هایی تعریف شده اند که مختصرا توضیحاتی درباره آنها داده می‌شود:

- کلاس Worldcup98: این کلاس بعنوان ساختار داده ای است که بسته ها را نشان میدهد.
- کلاس FlashCrowd: این کلاس فایل دیتاست را خوانده و ترافیکی که از آن قرار است دریافت شود را مدیریت می‌کند
- کلاس BotNet: این کلاس شبیه ساز یک بات نت است و ترافیکی که قرار است از بات نت بدست بیاید را مدیریت می‌کند.
- کلاس DSRequest: این کلاس محلی برای پردازش بسته ها دریافتی از یک کاربر است. الگوریتم در این کلاس پیاده سازی شده است.

بخش توابع: در این بخش توابعی تعریف شده اند که به بعضی توابع مهم در ادامه اشاره می‌کنیم:

- تابع process: این تابع ورودی ids پیاده سازی شده است. در این تابع یک درخواست رسیده است و ما باید تشخیص دهیم که با آن چه کنیم. اگر مبدا این درخواست از کاربران با ترافیک سالم بود که به آن اجازه می دهیم. اگر از کاربران با ترافیک ناسالم بود، بسته را دراپ می کنیم. و اگر جز این دو حالت نبود، با توجه به این بسته به ادامه بررسی ترافیک این کاربر می پردازیم. بررسی ترافیک درون نمونه ای از DSRequest که برای هر کاربری ساخته شده است انجام می شود.
- تابع simulate: این تابع نمونه هایی از کلاس FlashCrowd (به‌نام flash) و از کلاس BotNet (به‌نام botnet) می سازد و در یک حلقه از زمان ابتدایی ترافیک flash تا آخر آن بسته ها را از هر دو منبع درخواست جمع آوری می کند و به کلاس process ارسال می کند. در پایان حلقه گزارشی از عملکرد ids چاپ می کند.

بخش main: در این بخش فقط تابع simulate فراخوانی می شود و ادامه اجرا درون آن تابع انجام می شود.

نتایج

در تست سیستم پیاده سازی شده، ما از فایل wc_day58_3 دیتاست (یعنی ترافیک روز ۵۸ ام بخش سوم) استفاده کردیم — این انتخاب تصادفی بوده و می توان از روزهای دیگر نیز استفاده کرد.

مقدار k1 و k2 مقاله را برابر ۵ و مقدار ALPHA1 را برابـر ۰.۸۵ و مقـدار ALPHA2 را برابـر ۰.۹۵ گرفته‌ایـم. از دیتاسـت انتخاب شده فقط ۱۰۰ ثانیه (FLASHDUR) نمونه برداشته ایم تا پردازش کنیم. و بازه های زمـانی را ۱ ثـانیه (PERIOD) در نظر گرفته ایم. شبیه ساز نیز طوری ساخته شده است که تعـداد ۱۰۰ کـاربر (BOTSIZE) از ثـانیه ۲۰ ام تـا ۲۵ شـروع بـه ارسال بسته خواهند کرد (BOTACTIVEFROM=20 و BOTACTIVERAND=5)

زمان اجرا تقریبا ۸۰ ثانیه به طول انجامید. در این ۸۰ ثانیه 49028 بسته پردازش شد که به 10234 بسـته اجـازه داده شـده و 24964 بسته اجازه داده نشده است. همچنین ۲۳۰۰ کاربر در این مدت بسته ارسال کـرده اسـت. در کـل ۳۰۴ کـاربر بعنـوان بدخواه شناخته شد که ۱۰۰ تای آن مربوط به شبیه ساز و بقیه مربوط به کاربران دیتاست است. همچنین Ids توانسته کـه تمـام کاربران شبیه ساز را شناسایی و آنها را بلاک کند.

Packets Analysied:	49028
Packets Allowed:	10234
Packets Denied:	24964
Clients Analysied:	2300
Clients Determined Safe:	1806
Clients Determined Attacker:	304
Botnet clients not Detected:	0
Dataset clients detected as attacker:	203

[1] T. Thapngam, S. Yu, W. Zhou, and G. Beliakov, "Discriminating DDoS Attack Traffic from Flash Crowd through Packet Arrival Patterns," in the 30th Annual IEEE International Conference on Computer Communications (IEEE INFOCOM 2011), Shanghai, China, 2011, pp. 969-974.