به نام خدا

کامیار نصیری – ۹۷۲۲۷۶۲۵۵۳

پروژه درس شبکه های کامپیوتری

فاز صفر

Packet sniffing، به عملیات جمع آوری و گرفتن packet ها در شبکه گفته میشود. این جمع آوری میتواند شامل همه packet های گذرنده از یک لینک یا زیرمجموعه ای از آنها باشد. این جمع آوری عموما مستقل از آدرس مقصد آن packet ها انجام میگیرد. در این حالت، packet های مورد نظر ممکن است برای تحلیل مورد استفاده قرار بگیرند (مثلا برای بدست آوردن ترافیک شبکه یا پهنای باند). یک packet sniffer از دو بخش اصلی تشکیل میشود: یک متصل کننده، که شبکه موجود را به sniffer متصل میکند و یک نرم افزار که داده های بدست آمده را تحلیل یا قابل مشاهده میکند.

هنگامی که یک packet از مبدا به سمت مقصد حرکت میکند، قبل از فرستاده شدن، در مبدا توسط بیت های header مشخص میشود که مقصد این packet کدام end system میباشد. هر node ای که ما بین مبدا و مقصد قرار دارد، صرفا packet ها را به سمت مقصد هدایت میکند و محتوای این داده ها را نادیده میگیرد و کاری به آن ندارد. اما در packet sniffing ، همه یا بخشی از packet ها جمع آوری میشوند و مهم نیست که مقصد آنها دقیقا کدام node بوده است.

این روش از روش های hacker ها نیز است که از شبکه بصورت غیرقانونی داده ها را جمع آوری میکند.

دو نوع اصلی Packet sniffer ها:

۱- Packet sniffer سخت افزاری: برای اتصال فیزیکی به شبکه استفاده میشوند این sniffer یا داده های جمع آوری شده را ذخیره میکند یا آن را به سمت جمع کننده میفرستد. اگر در جایگاه درستی در شبکه قرار بگیرد میتواند تمام packet های مورد نظر را بدون از دست رفتن جمع آوری کند.

۲- Packet sniffer نرم افزاری: این نوع از sniffer تمام ترافیکی که از طریق رابط فیزیکی شبکه جریان میابد را جمع میکند.

Packet analyzing: عملکرد این روش همانند Packet sniffing است با این تفاوت که در packet analyzing، پس از جمع آوری داده ها، تلاش میشود تا درباره محتوای داده ها استنتاج انجام شود. درحالیکه در Packet sniffing، داده هایی که در رابط شبکه مشاهده شده، تنها ضبط و ثبت میشوند.

بنابراین Packet analyzing یا Packet sniffing هر دو یک ابزار تحلیلگر شبکه هستند که ترافیک حرکت داده ها را اندازه میگیرند. به وسیله این روشها میتوان کارایی شبکه و دلایل مشکلات احتمالی شبکه را دریافت کرد.

طبق جستجویی که در اینترنت انجام دادم، در زبان پایتون، چند کتابخانه پایه ای وجود دارد که میتوان از آنها برای packet sniffing استفاده کرد. کتابخانه های socket ، struct و binascii ، کتابخانه هایی هستند که برای این عملیات مورد استفاده قرار میگیرند. بعنوان مثال، کتابخانه struct ، کاربردهایی نظیر unpack کردن را (بعنوان یک function) در خود جای داده است.

بطور خاص، کتابخانه Pcapy نیز بعنوان یک کتابخانه در زبان پایتون مورد استفاده است که این کتابخانه برای گرفتن packet ها برای لینوکس مورد استفاده قرار میگیرد.

من از زبان برنامه نویسی Python بعنوان زبان برنامه نویسی مورد استفاده برای کدنویسی این پروژه استفاده خواهم کرد؛ زیرا استفاده از زبان پایتون برای من نسبت به زبان C راحت تر است و من اخیرا بیشتر درگیر زبان پایتون بوده ام و کار با زبان C برایم سخت تر از گذشته است. از این رو، و همچنین این مورد که همواره داکیومنت های موجود در اینترنت برای زبان پایتون بسیار بیشتر از دیگر زبان ها موجود و معتبر است، تصمیم گرفتم که زبان Python را بعنوان زبان مورد نظر خود انتخاب کنم.

- آدرس repository : https://github.com/KamyarNasiri/Computer_Network_Project_2021

منابع

https://www.kaspersky.com/resource-center/definitions/what-is-a-packet-sniffer

https://www.paessler.com/it-explained/packet-sniffing

https://www.geeksforgeeks.org/what-is-packet-sniffing/

https://osqa-ask.wireshark.org/questions/6737/packet-analyzer-vs-packet-sniffer

https://www.tutorialspoint.com/python_penetration_testing/python_penetration_testing_network_packet_sniffing.htm

https://www.binarytides.com/code-a-packet-sniffer-in-python-with-pcapy-extension/

https://github.com/EONRaider/Packet-Sniffer/blob/master/packet_sniffer.py

https://www.uv.mx/personal/angelperez/files/2018/10/sniffers_texto.pdf

https://github.com/O-Luhishi/Python-Packet-Sniffer

فاز اول – Port Sniffer

در این فاز از پروژه بناست port sniffing انجام شود که در این عملیات، باز یا بسته بودن یک port که بعنوان ورودی دریافت میشود، باید بررسی شود. تابع sniffing در کد ارائه شده، این عملیات را انجام میدهد. ورودی index در این تابع مشخص میکند که قصد داریم کدام یک از سه حالت زیر را بررسی کنیم :

۱- برای تمام پورت های host : پورت های بین ۱ و ۶۵۵۳۵ . که این عملیات با index = 0 انجام میشود. در این حالت، تابع portscan_All صدا زده میشود که ابتدا یک socket ایجاد میکند. در ادامه، با استفاده از socket ایجاد شده و با استفاده از تابع connect_ex ، بین host مورد نظر و بر روی port یک اتصال ایجاد میشود. اگر خروجی این تابع 0 باشد، این port برای host باز است و درغیر اینصورت خیر. در ادامه socket بسته میشود.

برای ایجاد یک socket، در پایتون از کتابخانه socket و تابع socket استفاده میکنیم که بعنوان ورودی این تابع، AF_INET که نشان دهنده خانواده ی آدرسی است که این socket با آنها ارتباط برقرار میکند و ورودی دیگر این تابع، SOCK_STREAM است که مشخص کننده سرویس TCP است. همچنین ذکر این نکته ضروری است که تابع connect_ex، یک socket را به آدرس مورد نظر توسط port اعلام شده، متصل میکند.

۲- برای پورت های مشهور مربوط به host : پورت های بین ۱ و ۱۰۲۳ . که این عملیات با index = 1 انجام میشود. در این حالت از همان تابعی استفاده میشود که برای قسمت ۱ از آن استفاده کردیم یعنی portscan_All. منتها برای تشخیص پورت های بین ۱ تا ۱۰۲۳ لازم است عملیات تنها روی این پورت ها انجام شود. به عبارت بهتر در حالت ۱، عملیات را برای پورت های ۱ تا ۶۵۵۳۵ انجام میدادیم ولی اکنون تنها لازم است این عملیات روی پورت های ۱ تا ۱۰۲۳ انجام شود.

۳- برای پورت های خاص در قالب پرسمان. که این عملیات با index = 2 انجام میشود. شامل پورت های HTTP با شماره پورت ۸۰، پورت TLS با شماره پورت ۴۶۵، پورت SMTP با شماره پورت های ۲۵ و ۴۶۵ و ۵۸۷، پورت FTP با شماره پورت ۲۱، پورت TELNET با شماره پورت ۲۳ و پورت SSH با شماره پورت ۲۲. تابع portscan_Query برای این امر نوشته شده است که یک روش قبلی را بر روی یک port خاص اجرا میکند و در صورت باز بودن پورت، گزارش میدهد و درصورتی که خروجی تابع صفر نباشد نیز عبارت closed را چاپ میکند.

۴- بصورت یک بازه از پورت ها . که این عملیات با index = 3 انجام میشود. در این حالت نیز از همان عملیاتی استفاده میکنیم که در حالت اول استفاده کردیم ولی عملیات ایجاد socket و بررسی باز بودن port را روی بازه مخصوصی از port ها انجام میدهیم.

در این پروژه، کاربر میتواند با دستورات مختلف، پارامتر هایی نظیر timeout را تعیین کند. این پارامتر حداکثر زمانی را مشخص میکند که socket منتظر پاسخ host مورد نظر خواهد بود. این پارامتر بصورت پیش فرض ۰,۳ ثانیه تنظیم شده است. پارامتر دیگری که کاربر میتواند تعیین کند، تعداد نخ هاست که در ادامه به توضیح آن خواهیم پرداخت.

در این پروژه از خاصیت multithreading استفاده میکنیم. تعداد نخ های پیش فرض را برابر ۸ قرار داده ایم. سپس با توجه به تعداد نخ های تعیین شده، آنها را با دستور threading.Thread میسازیم. در هر کدام از حالات چهارگانه ی بالا، پورت هایی را که در نظر داریم آنها را بررسی کنیم به ترتیب در یک queue قرار میدهیم و سپس به ترتیب، پورت های وارد شده در queue را بررسی کرده و در توابعی که در بالا بحث شد مورد آزمایش قرار میدهیم.

در ادامه دو نمونه از ورودی های برنامه را بررسی میکنیم:

1) python Faz_1.py --host 8.8.8.8 -a

در این حالت، مشخص کرده ایم که host، 8.8.8.8 است و با توجه به دستور a– مشخص کرده ایم که تمام port ها لازم است مورد بررسی قرار گیرند.

2) python Faz_1.py --host google.com -q --Service "HTTP"

در اینجا، میزبان google.com است و نوع درخواست از نوع query است. سپس service درخواست شده مشخص شده که از نوع HTTP میباشد.

3) python Faz_1.py --numThreads 15 --timeOut 0.2 --host 8.8.8.8 –r

مثال آخر نیز با مشخص کردن تعداد نخ ها و زمان timeout و سپس میزبان مورد نظر، نوع درخواست را از نوع reserved یا پورت های مشهور انتخاب کرده است.

فاز دوم – Ping

Ping ها برای آزمایش قابل دسترسی بودن یک host بر روی شبکه IP بکار میروند. Ping با فرستادن بسته های درخواست echo از پروتکل ICMP به host مورد نظر شروع بکار میکند. سپس برای پاسخ echo با ICMP ، منتظر میماند.

در این فاز از پروژه، ابتدا یک کلاس با نام Status ایجاد شده است که دارای ویژگی هایی از جمله IP مورد نظر، تعداد packet های ارسال شده و دریافت شده، کمینه و بیشینه زمان انجام یک بار ping، کل زمان انجام فرایند ping ها و loss ایجاد شده می باشد.

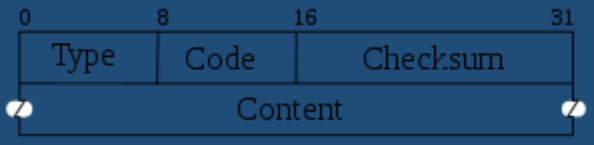
در ابتدای تابع main، با استفاده از تابع signal.signal ، وقوع ctrl+c یا SIGINT را مدیریت میکنیم که این در اینجا تابع signal_handler صدا زده شود. این تابع، با فراخوانی تابع دیگری با نام dump_stats ، برای IP مورد نظر، آمارهایی نظیر Loss (به درصد) ، تعداد packet های دریافتی و ارسالی و کمینه و بیشینه و میانگین زمان (به میلی ثانیه) را چاپ میکند. تابع signal_handler در انتها برنامه را به اتمام میرساند.

براساس ورودی های کاربر که چندین host را مشخص کرده است، تابع verbose_ping، براساس نام host، در یک حلقه به تعداد ۵ مرحله، عملیات ping را انجام میدهد. در این میان، اگر مقدار delay برای انجام هر عمل ping، کمتر از یک مقدار MAX_SLEEP باشد، که ۱ ثانیه تنظیم شده است، سیستم به اندازه (MAX_SLEEP - delay) / 1000 ، صبر میکند (sleep).

اما عملیات اصلی در هنگامی رخ میدهد که عملیات ping، یک بار قرار است انجام شود. برای این کار، تابع do_one فراخوانی میشود. در این تابع، ابتدا یک socket با پارامترهای AF_INET (که خانواده آدرس را مشخص میکند) ، SOCK_RAW (نوع socket را مشخص میکند) و getprotobyname که نام پروتکل اینترنت (icmp) را به یک عدد ثابت ترجمه میکند، ایجاد میشود. پروتکل icmp بخشی از پروتکل اینترنت است که در سیستمهای شبکه برای فرستادن خطاها و دیگر اطلاعات جانبی به کار میرود که موفقیت یا شکست ارتباط را هنگام اتصال به دیگر ip address ها مشخص میکند.

در ادامه تابع do_one، تابع send_one_ping را فراخوانی میکند که در ادامه توضیح داده میشود و یک ping را ارسال میکند. اگر این عملیات موفقیت آمیز باشد، تعداد packet های ارسالی یک عدد افزایش می یابد. پس از آن، تابع receive_one_ping فراخوانی میشود که یک ping را دریافت میکند. اگر زمان عملیات receive، صفر نباشد، میتوان delay را که حاصل تفریق زمان دریافت و زمان ارسال است بدست آورد و براساس آن ممکن است زمان بیشینه و کمینه نیز تغییر کنند. همچنین زمان کل، به همین میزان افزایش می یابد.

در تابع send_one_ping ، لازم است که ابتدا یک بسته برای ارسال آماده کنیم. این بسته شامل header و data است. در پروتکل icmp، هر header باید به فرم زیر باشد:



که در قسمت content ، ip سیستم و شماره عملیات ping را که هرکدام ۱۶ بیتی است قرار میدهیم.

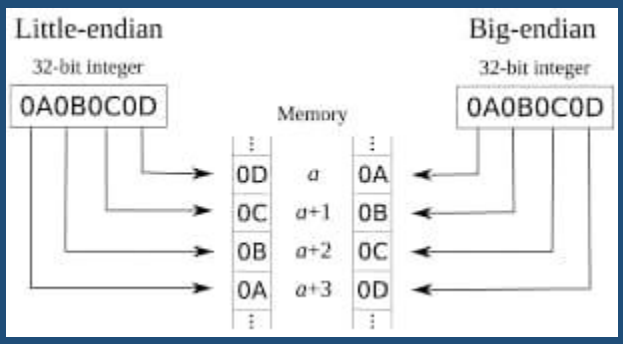
```
typedef struct tagICMPHDR
{
   u_char  Type;       // Type
   u_char  Code;       // Code
   u_short Checksum;   // Checksum
   u_short ID;         // Identification
   u_short Seq;        // Sequence
} ICMPHDR, *PICMPHDR;
```

```
echo_req.icmpHdr.Type          = ICMP_ECHOREQ;
echo_req.icmpHdr.Code          = 0;
echo_req.icmpHdr.Checksum      = 0;
echo_req.icmpHdr.ID            = id++;
echo_req.icmpHdr.Seq           = seq++;
```

در ابتدا "BBHHH" را قرار میدهیم که نوع اعداد ورودی های بعدی را نشان میدهد. B نشان دهنده unsigned char و H نشان دهنده unsigned short است. برای Type از ICMP_ECHO استفاده شده است که مقدار دهی اولیه آن برابر ۸ قرار داده شده و درنهایت مقدار اولیه checksum برابر صفر است. مقدار Code نیز برابر صفر قرار داده میشود. (با توجه به داکیومنت)

به این ترتیب، یک header با ۶۴ بیت برای پروتکل icmp ساخته شد. اکنون باید به ساخت داده ها بپردازیم. اندازه بسته را کاربر مشخص میکند. بنابراین از بیت ۶۶ ام شروع میکنیم و به اندازه ای که کاربر اعلام کرده جلو میرویم و اجازه نمیدهیم مقدار آنها از ۲۵۵ بالاتر برود. پس همه این مقدار ها را در لیست padBytes قرار میدهیم و آن را تبدیل به Byte میکنیم و اکنون داده ها بدست آمده اند. این داده ها را به header وصل میکنیم و با استفاده از عملیاتی در تابع

checksum، نتیجه را ذخیره میکنیم. تابع checksum به این نکته اشاره دارد که داده های شبکه big-endian هستند ولی host ها معمولا little-endian هستند.



اکنون که حاصل این تابع را داریم، یک checksum صحیح وجود دارد ولی در حالت اولیه که header را ساختیم، آن را صفر قرار دادیم. بنابراین اکنون میتوان یک header با checksum صحیح درست کرد و آن را به data وصل کرد(همانند حالت قبل). با استفاده از تابع sendto() در کتابخانه socket، packet ایجاد شده را به مقصد میفرستیم.

در تابع receive_one_ping، ابتدا براساس حداکثر زمانی که کاربر در نظر گرفته است، یعنی timeout، یک حلقه را اجرا میکنیم. در این حلقه، تابع recvfrom صدا زده میشود که یک packet را نیز برمیگرداند که باید این packet را unpack کنیم. در ابتدا، اطلاعات مربوط به ip فرستنده در این packet ذخیره است. ۸ بایت از شماره ۲۰ تا ۲۸ این packet، header مربوط به پروتکل icmp است که به وسیله این پروتکل، ping را فرستاده بودیم. با تابع unpack، میتوانیم به ID سیستمی که این packet به آن ارسال شده، دریابیم که آیا این packet به سیستم ما ارسال شده یا خیر. اگر ارسال انجام شده باشد، میتوانیم محاسبه کنیم که از timeout خارج شده بودیم یا خیر.

در این فاز نیز از multithreading استفاده شده است تا فرایند ping برای host های مختلف بصورت موازی انجام شود. تعداد نخ ها بصورت پیش فرض برابر ۸ تنظیم شده است. سپس بر همان روشی عمل میشود که در فاز اول انجام شد.

در این فاز، میزان timeout و اندازه بسته ارسالی برای انجام ping نیز توسط کاربر قابل تنظیم است که بصورت پیشفرض به ترتیب ۱ ثانیه و ۵۵ بایت تنظیم گشته اند. در انتهای این قسمت، ۱۰ ثانیه برنامه در حالت sleep قرار گرفته است تا درصورتی که کاربر بعد از انجام عملیات لازم باشد، با فشار دادن ctrl+c بتواند آمار مورد نظر را مشاهده کند.

ورودی نمونه برای Ping ها:

```
python Faz_2.py -l  p30download.ir google.com Instagram.com 8.8.8.8 --PacketSize 55 --timeOut 1000
```

همانطور که مشخص است، لیستی از host ها بعنوان ورودی داده شده است و سپس packet size و timeout مقداردهی میشوند.

فاز سوم – Tracerouter

در این فاز، بناست براساس ابزار tracerouting بتوانیم تمام hop ها بین میزبان مبدأ و میزبان مقصد شناسایی شوند. در ابتدا لازم است ورودی های این ابزار را مشخص کنیم. سرور مقصد، تایم اوت (timeout)، اندازه بسته ها، بیشترین تعداد تلاش ها، بیشترین تعداد hop و مقدار اولیه TTL، به عنوان ورودی انتخاب میشوند.

در ادامه لازم است یک iteration را انجام دهیم که آن iteration باید از initial_TTL تا max_TTL صورت بگیرد و در آن، در مرحله عملیات tracerouting انجام میشود. لازم به ذکر است که در مرحله ای که به مقصد دستیابی کنیم، از حلقه for خارج میشویم.

عملیات tracerouting به این صورت است که به اندازه max_tries که بعنوان ورودی داده شده، عملیاتی مشابه عملیات ping انجام میشود. به این صورت که ابتدا یک socket ساخته میشود و سپس مقدار ttl به عنوان ورودی به آن وارد میشود. سپس یک ping را میفرستند و سپس یک ping را دریافت میکند. توضیحات مربوط به فرستادن و دریافت ping در فاز قبلی توضیح داده شد.

در این میان، در تابع total که عملیات tracerouting را انجام میدهد، در لوپ مربوط به فراخوانی ping ها، همه router ها یا hop هایی که دیده شده چاپ خواهد شد و درنتیجه عمل tracerouting صورت میپذیرد.

فاز چهارم – Host discovery

در این فاز لازم است از ابزار ARP استفاده کنیم. به این صورت عمل میکنیم که یک درخواست ARP برای میزبان مورد نظر ارسال میکنیم و با این عملیات، متوجه زنده بودن یا نبودن میزبان میشویم. برای این ابزار، یک بازه از آدرسهای IP را دریافت میکنیم که آدرسهای در این بازه لازم است زنده بودن یا نبودنشان چک شود. همچنین Timeout و nic که مخفف Network interface controller است باید دریافت شود.

در تابع check_errors ابتدا وجود خطا در مقادیر آدرسهای ابتدای بازه و انتهای بازه بررسی میشود. سپس بررسی میشود که مقدار nic صحیح است یا خیر. و درنهایت مقدار IP و mac مربوط به این interface استخراج میشود.

در ادامه در تابع ARP_operation، ابتدا یک شی از کلاس Address ساخته میشود. سپس آدرس ابتدایی و انتهایی داده شده در این کلاس، به چهار section تقسیم میشود و آنها را نگهداری میکنیم. تابع iteration در این کلاس، باعث میشود که آدرس فعلی که در ابتدا آدرس initial به آن انتساب داده شده، یک عدد افزایش یابد. این تابع همچنین تست میکند که آیا اکنون به حالت آدرس نهایی دست یافتیم یا خیر که در این صورت false برمیگرداند. بنابراین در یک while این شرط را چک میکنیم و درصورتی که true برگردانده شود، یک بسته با شرایط مرتبط با ARP ساخته میشود. با استفاده از یک socket این packet را ارسال میکنیم. درصورتی که پاسخی از ARP دریافت شود، این آدرس زنده است. ساختن بسته های ARP نیز در کلاس ARP صورت میپذیرد.

پایان