پوریا غفوری ۴۰۱۱۰۶۲۷۷

سوال اول:

الف)

در صورتی که مهاجم امضای یک پیام را داشته باشد، بر اساس الگوریتم می‌تواند یک بار دیگه تابع p را بر روی آن امضا اعمال کند و سپس امضای جمله قبلی تولید می‌شود و همین سناریو را تا انتها می‌تواند اجرا کند. مثلا اگر امضای جمله آخر را بداند، صرفا کافی‌است تا تابع p را بر روی آن امضا یک‌بار دیگر اعمال کند و آن‌گاه امضای جمله یکی مانده به آخر را بدون اینکه کلید خصوصی را بداند، خواهد داشت.

به طور کلی در صورتی که یک امضا از یک جمله را داشته باشیم، می‌توانیم تمامی امضاهای جملات قبل از آن را جعل کنیم.

ب)

بدیهی‌ترین روشی که می‌توان پیشنهاد داد این است که به ازای هر پیام، یک کلید خصوصی و یک کلید عمومی تعریف کنیم. به طور مثال اگر n پیام داشته باشیم، باید a n و b n نیز داشته باشیم که:

$$p^n(a_i) = b_i$$

با توجه به اینکه a عددی n بیتی است، این راه ممکن است.

حال برای ارسال پیام i می‌توانیم امضای آن را به روش زیر درست کنیم:

$$p^{n-i}(a_i) = \sigma_i$$

و طرف مقابل پس از دریافت آن، باید با روش زیر آن را وارسی کند:

$$p^i(\sigma_i) = b_i$$

در این حالت با توجه به اینکه کلید هر پیام با پیام دیگر متفاوت است، فرد حمله‌کننده نمی‌تواند امضای یک پیام را پیام دیگر بسازد و آسیب‌پذیری ابتدایی بر آن وارد نیست.

سوال دوم:

- **امکان اجرای حمله تکرار:** این پروتکل از این لحاظ آسیب‌پذیر است. اگر فرض کنیم که یک k قدیمی که یک کلید میان X و Y است لو رفته باشد، حمله‌کننده می‌توانند N_Y را از پیام دوم از بالا دریافت کند (با توجه به اینکه رمز نشده) و آن را با استفاده از همان k قدیمی رمز کند و سپس از $\{X, k\}_{K_{YT}}$ که در معرفی همان k قدیمی استفاده شده بود،

استفاده کند و حمله را انجام دهد و پیام چهارم از بالا را ارسال کند. پس دیدیم که حمله‌کننده به راحتی می‌تواند N_Y که جدید است را دریافت کند (چون رمز نشده) و آن را با همان کلید قدیمی رمز کند و سپس قسمت اول را نیز طبق اطلاعاتی که از قبل داشته ارسال کند و نیازی به K_{YT} یا K_{XT} پیدا نمی‌کند و حمله به درستی انجام می‌شود.

- **احراز تازگی کلید توسط هر یک از طرفین:** همان‌طور که در قسمت قبل بیان شد، ما می‌توانیم به راحتی با استفاده از یک کلید قدیمی وانمود کنیم که از یک کلید جدید استفاده کرده‌ایم و در نتیجه نمی‌توان احراز تازگی کلید را سمت Y انجام داد. اما برای X می‌توان تازگی کلید را تایید کرد. در ابتدا N_x به صورت عادی به Y منتقل می‌شود اما در ادامه با استفاده از K_{YT} رمز شده و در ادامه توسط TTP با استفاده از K_{XT} رمز می‌شود و X می‌تواند آن را مشاهده کنید و در این میان نمی‌توان X را تغییر داد و اگر این کار را انجام دهیم X متوجه خواهد شد و در نتیجه نمی‌توان یک بسته دیگر از $\{N_X, k\}_{K_{XY}}$ برای X ارسال کرد چون آن‌گاه N_X تغییر می‌کند و X متوجه می‌شود که کلیدی که در این بسته است، معتبر نیست.
- **احراز اصالت طرفین و تایید کلید:** اصالت X به راحتی احراز می‌شود و کسی نمی‌تواند خود را جای آن بگذارد چون که در ابتدا شناسه آن مستقیما برای Y فرستاده می‌شود و در ادامه Y آن را با استفاده از کلید K_{YT} رمز می‌کند و برای TTP ارسال می‌کند سپس TTP از روی شناسه X که با استفاده از K_{YT} رمز شده، آن پیام را برای X ارسال می‌کند و حمله‌کننده کاری نمی‌تواند انجام دهد. اگر فرض کنیم حمله‌کننده در همان ابتدا خود را به جای X جای بزند، مجبور است شناسه خود را جایگزین X کند که در این صورت Y متوجه می‌شود X در این مکالمه نیست اگر هم شناسه را عوض نکند، TTP پیام را به گونه‌ای رمز می‌کند که فقط X می‌تواند آن را مشاهده کند و باز هم حمله‌کننده کاری نمی‌تواند انجام دهد.
برای Y اما حمله‌کننده می‌تواند خود را جای Y جای بزند و X, N_X را از X دریافت کند و سپس شناسه و نانس خود را برای TTP ارسال کند. چون TTP هیچ شناسه‌ی قابل خواندنی از Y برای X ارسال نمی‌کند پس X نمی‌تواند متوجه شود که طرف مقابلش واقعا Y است. تنها چیزی که مرتبط با Y است که سمت X دریافت می‌شود، نانس Y است که راهی برای شناسایی طرف مقابل نیست.
برای تایید کلید، X به هیچ عنوان نمی‌تواند متوجه شود که واقعا کلید توسط Y دریافت شده است یا خیر چون هیچ تاییدیه‌ای شامل k از سمت Y به سمت X ارسال نمی‌شود. همچنین Y نیز مطمئن نمی‌شود که کلید به درستی سمت X رسیده یا خیر چون همان‌طور که در بخش اول اشاره شد، امکان بروز حمله تکرار وجود دارد و این قضیه باعث می‌شود که مطمئن نشویم کلید سمت X به درستی مشخص‌شده. چون به راحتی یک حمله‌کننده می‌تواند بسته سوم از بالا را خودش بگیرد و نگذارد به X برسد و سپس با حمله تکرار، بسته چهارم از بالا را برای Y ارسال کند.

برای رفع مشکلات بیان‌شده در ادامه بحث خواهیم کرد.

برای رفع حمله تکرار، کافی است کاری انجام دهیم تا حمله‌کننده متوجه N_Y نشود و برای این کار صرفا کافی است تا N_Y را با استفاده از K_{YT} رمز کرده و برای TTP ارسال کنیم. در این حالت حمله‌کننده متوجه N_Y نخواهد شد. در ادامه نیز TTP می‌تواند

N_Y را در کنار کلید قرار داده و در بسته $\{X, k, N_Y\}_{K_{YT}}$ قرار داده و برای X ارسال می‌کند تا X نیز آن را برای Y ارسال کند. در این حالت اگر حمله تکرار صورت گیرد، با توجه به اینکه کلید و N_Y با هم در یک بسته هستند پس هر دو تغییر کرده و در نتیجه Y می‌فهمد که N_Y رسیده به آن قدیمی است و در نتیجه کلید معتبر نیست. با اعمال این بهبود در پروتکل، آسیب‌پذیری دوم نیز رفع می‌شود چون در اثر حمله تکرار به وجود آمده بود.

برای احراز اصالت Y توسط X نیز لازم است تا TTP شناسه Y (یا همان فرستنده پیام) را در $\{N_X, k, Y\}_{K\,XT}$ قرار دهیم تا X متوجه شود که فرستنده برای TTP چه شخصی بوده است و مطمئن شود که آن شخص Y بوده و اصالت آن احراز شود.

برای تایید کلید دو طرفه لازم است در ابتدا X، N_Y را با کلید جدید برای Y ارسال کند که قبلا به دلیل حمله تکرار، به درستی صورت نمی‌گرفت اما در این حالت با قرار دادن N_Y از Y به TTP در $\{X, N_Y, N_X\}_{K_{YT}}$، دیگر برای حمله‌کننده قابل مشاهده نبوده و همچنین در ارسال آن از TTP به سمت X نیز باید آن را در بسته‌های $\{X, k, N_Y\}_{K_{YT}}$ قرار دهیم تا به صورت مستقیم از X به Y فرستاده شود همچنین باید آن را در بسته $\{N_X, k, Y, N_Y\}_{K_{XT}}$ نیز قرار دهیم تا X بتواند آن را باز کند و مشاهده کند تا در ادامه بتواند با ارسال $\{N_Y\}_K$ به سوی Y، به Y اطمینان دهد که کلید درستی را دریافت کرده. دقت کنید که چون $\{X, k, N_Y\}_{K_{YT}}$ را نیز برای Y ارسال می‌کنیم، Y می‌تواند متوجه شود که کلید هم جدید است هم در X شناخته شده و X هم زنده است. در ادامه نیز برای تایید کلید از سمت Y برای X نیاز است تا N_X دریافت‌شده را با استفاده از k رمز کنیم و برای X ارسال کنیم تا X نیز مطمئن شود که کلید جدید و معتبری که سمت خودش شناخته است، در سمت Y نیز شناخته شده و همچنین Y هم زنده است.

به طور کلی روند ارسال پیام‌ها به صورت زیر می‌شود:

$$X \xrightarrow{X, N_X} Y$$

$$TTP \xleftarrow{\{X, N_X, N_Y\}_{K_{YT}}, Y} Y$$

$$X \xleftarrow{\{X, N_Y, k\}_{K_{YT}}, \{N_X, k, N_Y, Y\}_{K_{XT}}} TTP$$

$$X \xrightarrow{\{X, N_Y, k\}_{K_{YT}}, \{N_Y\}_k} Y$$

$$X \xleftarrow{\{N_X\}_k} Y$$

سوال سوم:

الف)

با توجه به اینکه ترافیک میان S و C توسط KDC قابل تغییر دادن نیست، به راحتی می‌توانیم با استفاده از دیفی-هیلمن که الگوریتم به اشتراک گذاری کلید است، یک کلید مشترک را میان این دو به اشتراک بگذاریم.

ب)

در این صورت حمله مرد میانی ممکن است صورت بگیرد و KDC خود را جای S و C جا بزند و میان KDC و C و همچنین میان KDC و S یک کلید مشترک تعریف شود و آن‌گاه S و C فکر می‌کنند با یکدیگر ارتباط دارند در صورتی که با یک واسطه (KDC) به یکدیگر متصل‌اند و KDC به راحتی می‌تواند پیام‌هایی که میان S و C منتقل می‌شوند را مشاهده کند. به این صورت که پیامی که از S دریافت کرده است را با کلید مشترک KDC و S کدگشایی کرده و آن را مشاهده می‌کند و سپس با کلید مشترک KDC و C آن را رمز کرده و برای C ارسال می‌کند تا برای S و C همه چیز عادی به نظر برسد.

ج)

با توجه به اینکه امکان تبانی میان دو *KDC* وجود ندارد، می‌توانیم از Authenticated Diffie-Hellman استفاده کنیم. در ابتدای کار یک LTK را با استفاده از $K_{C,S,1}$ و $K_{C,S,2}$ رمز کرده و میان S و C به اشتراک بگذاریم. با توجه به اینکه KDCها نمی‌توانند تبانی کنند پس هیچکدام به تنهایی نمی‌تواند به LTK دسترسی داشته باشد. در ادامه با استفاده از این LTK می‌توانیم α^{X_S} و α^{X_B} را به اشتراک بگذاریم و از صحت آن‌ها مطمئن شویم تا حمله مرد میانی صورت نگیرد.

سوال چهارم:

الف)

- Volumetric Attacks: این نوع از حمله‌ها توسط حمله‌کننده‌هایی اتفاق می‌افتد که کنترل گسترده‌ای بر روی botهایی که در شبکه هستند، دارند که به اصطلاح به این botها، botnet گفته می‌شود. این botها می‌توانند شامل کامپیوترها، دستگاه‌های اینترنت اشیا و یا دیگر دستگاه‌هایی که توسط یک بدافزار آلوده شده‌اند تا حمله‌کننده بتواند بر روی آن‌ها کنترل داشته باشد، می‌شوند. در این نوع حمله، حمله‌کننده با کنترلی که بر روی این دستگاه‌ها دارد، آن‌ها را مجبور می‌کند تا به طور همزمان یک درخواست مستقیم به یک سرور به خصوص بفرستند تا ترافیک آن را افزایش دهند و کارایی آن از بین برود.
- Protocol Attacks: این نوع حمله از ضعف‌ها و یا نحوه پیاده‌سازی پروتکل‌های استفاده‌شده در لایه‌های سوم و چهارم شبکه (در مدل OSI) بهره می‌برد تا بتواند سرویس‌ها را مختل کند. معمولا سناریو این نوع حمله‌ها به این صورت است که سرور یک بسته یا درخواست از سمت یک کامپیوتر دریافت کرده و توقع دارد تا ارتباطی را با آن برقرار کند و منابعی را به آن اختصاص می‌دهد و این کار باعث هدر رفت منابع شده و در نتیجه سرعت و ظرفیت پاسخگویی سرور کاهش می‌یابد.
- Application Layer Attacks: این نوع حمله در لایه هفت شبکه (لایه کاربرد) اتفاق می‌افتد و آسیب‌پذیری‌های خاصی را در برنامه‌ها هدف گرفته تا بتواند در عملکرد آن اختلال ایجاد کند. این نوع حملات معمولا پروتکل‌های ارتباطی

که در تبادل داده‌ها بین دو برنامه از طریق اینترنت استفاده می‌شوند را هدف قرار می‌دهد (مانند HTTP) و با ارسال حجم زیادی درخواست به لایه کاربرد، آن وب‌سایت یا سرویس اینترنتی را مختل می‌کند و آن سرور مجبور است تا به همه آن درخواست‌ها پاسخ دهد. تفاوت این حمله با *Volumetric* این است که در حمله *Volumetric* هدف اشغال پهنای باند و یا منابع شبکه است اما در این نوع حمله هدف اصلی اشباع منابع پردازشی و نرم‌افزاری است.

ب)

i. بر اساس مقاله، ما باید بسته‌هایی که مربوط به حمله هستند را شناسایی کنیم و جلوی آن‌ها را بگیریم. پس در ابتدای کار باید این بسته‌ها را شناسایی کنیم. در این روش، بسته‌هایی که توسط rate limiter دراپ می‌شوند، وارد pushbackd شده تا شناسایی شوند. مکانی در pushbackd که این بسته‌ها را ذخیره می‌کند، با نام drop set شناخته می‌شود. حال ابتدا باید بررسی کنیم که آیا نرخ دراپ شدن از یک بازه‌ای فراتر رفته که نیاز باشد فرایند را آغاز کنیم یا خیر. این محدوده به صورت زیر بررسی می‌شود که ω_i پهنای باند ورودی و ω_0 پهنای باند لینک خروجی است.

$$\omega_i > 1.2 \times \omega_0$$

که به این معنا است که پهنای باند در لینک ورودی، بیش از ۲۰ درصد از پهنای باند لینک خروجی است و در این حالت ما باید فرایند خود را آغاز کنیم. نحوه کار به این صورت است که بسته‌هایی که در drop set هستند را باید بر اساس IP مقصدشان تقسیم‌بندی کنیم. در این حالت باید بسته‌ها را بر اساس routing table تقسیم‌بندی کرده و بلندترین پیش‌وند را با آن‌ها match کنیم. در ادامه در میان این گروه‌ها باید گروهی که بیشترین تعداد بسته را شامل می‌شود پیدا کنیم. پس از اینکه گروه با بیشتری تعداد بسته پیدا شد، حال باید بررسی کنیم که آیا بسته‌های درون این گروه، پیش‌وندهای مشترکی بزرگ‌تر از مقادیر موجود در routing table را شامل می‌شوند یا خیر تا با دقت بیشتری بتوانیم بسته‌های مربوط به حمله را شناسایی کنیم. این گروه نهایی به دست آمده، به عنوان بسته‌های حمله شناخته می‌شوند و پیش‌وند آن‌ها را به عنوان امضای حمله در نظر می‌گیریم و بسته‌هایی که با این پیش‌وند می‌آیند را حذف می‌کنیم. البته برای حذف باید شرایط زیر را بررسی کنیم که ω_b مقداری پهنای باندی است که قصد داریم حذف کنیم:

$$\omega_l = \omega_i - 1.2 \times \omega_0$$

در اینجا داریم بررسی می‌کنیم که چقدر پهنای باند مورد نظر ما با چیزی که الان داریم تفاوت دارد. حال بررسی می‌کنیم که آیا $\omega_b > \omega_l$ هست یا خیر. اگر این شرط برقرار بود، می‌توان حدس زد که یک حمله در جریان است و مقدار آن گروهی که امضا شده است را تا ω_l پایین می‌آوریم و شرایط درست می‌شود. اما اگر $\omega_b < \omega_l$، آن‌گاه می‌توان حدس زد که ممکن است چندین حمله در جریان باشند و با حذف بسته‌هایی که در این مرحله تشخیص داده‌ایم، به شرایط موردنظر خود نمی‌رسیم پس تمامی این ω_b را حذف می‌کنیم.

بعد از این مراحل دوباره شرایط را بررسی می‌کنیم و اگر شرطی که در بالا گفتیم برقرار بود، لازم است تا دوباره این فرایند انجام گیرد و می‌توان فهمید که ممکن است بیش از یک حمله در جریان باشد. اگر شرط بالا برقرار نبود، شرایط عادی است و با حذف این ترافیک مخرب، به پایداری رسیده‌ایم.

در تمامی این مراحل، pushnackd به rate-limiter می‌گوید که چه بسته‌هایی را دراپ کند تا شرایط برقرار شود و پس از اینکه حمله پایان یافت، آن قوانین را عوض کرده و قوانین جدیدی به rate-limiter می‌دهد که به طور مثال دیگر این امضای خاص را نباید دراپ کند.

ii. در این مقاله اشاره شده که روش‌های قبلی مانند RED، با اتکا به کنترل ازدحام مبتنی بر جریان‌ها کار می‌کرده‌اند و بسته‌هایی که از جریان end-to-end در پروتکل TCP پیروی نمی‌کردند را به عنوان بسته‌های مخرب که حاصل حمله بودند در نظر می‌گرفتند و دراپ می‌کردند اما همان‌طور که در مقاله اشاره شد، بسته‌هایی که مربوط به حمله DDOS هستند، یک جریان قابل‌تشخیص و مشخصی ندارند که بتوان آن را به عنوان امضای آن‌ها در نظر گرفت و همه بسته‌های مرتبط با آن را دراپ کرد. به همین دلیل از روش کنترل ازدحام مبتنی بر تجمیع در این مقاله استفاده شده است تا یک ویژگی مشترکی را میان بسته‌هایی که حاصل یک حمله هستند پیدا کند و بر آن اساس، آن‌ها را دراپ کند.

iii. بسته‌های درخواست شامل RLS-ID (Rate-Limiting Session ID) هستند که برای match کردن جواب‌ها و درخواست‌ها استفاده می‌شوند. Depth برای این استفاده می‌شود که مشخص کنیم پیام درخواست ارسالی از سمت pushbackd، باید تا کجا پیش برود. این فیلد در مبدا برابر صفر است و هرچه جلوتر می‌رویم، به مقدار آن یکی اضافه می‌شود تا زمانی که مقدار آن به فیلد maximum depth برسد که توسط مبدا مقداردهی شده است. همچنین یک فیلد bandwidth limit نیز در این پیام برای این است که مشخص کنیم گروهی که به عنوان بسته حاصل از حمله شناسایی شده، حداکثر جه مقدار پهنای باند می‌تواند داشته باشد. همچنین یک فیلد expiration time نیز داریم که مشخص می‌کند تا چه زمانی این درخواست باید برقرار باشد و آن بسته‌ها را دراپ کنیم و اگر پیام Refresh تا قبل از این زمان به ما نرسد، داده‌های مرتبط با این پیام از بین رفته و دیگر آن بسته‌ها دراپ نمی‌شوند. در انتها نیز فیلد Congestion Signature در این پیام است که مشخص می‌کند بر روی چه پیشوندی از IPهای مقصد باید محدودیت پهنای باند اعمال شود. همچنین یک نوع پیام درخواست با نام کنسل داریم که به روتر بالادستی می‌گوید تا یک‌سری قوانین را حذف کند. کاربرد آن برای زمانی است که یک expiration time طولانی برای یک درخواست ست شده است. پیام وضعیت شامل فیلد depth است که تا زمانی که مقدار آن به صفر نرسیده، میان مسیریاب‌های پایین‌دستی جابه‌جا می‌شود و با هر گذر، مقدار آن یک واحد کاهش می‌یابد.

پیام جواب یک فیلد depth دارد که اگر زمانی که به یک سرویس رسید و مقدار عمق آن صفر بود، آن‌گاه از اطلاعات درون این پیام استفاده می‌شود تا pushback را ادامه دهیم یا بر روی آن تغییراتی اعمال کنیم.

سوال پنجم:

الف)

Flag	Protocol	DST Port	SRC Port	DST IP	SRC IP	Act	Int#
Any	HTTP	80	Any	Any	10.0.0.0/16	Accept	2

Any	HTTPS	443	Any	Any	10.0.0.0/16	Accept	2

ب)

Flag	Protocol	DST Port	SRC Port	DST IP	SRC IP	Act	Int#
Any	Any	Any	Any	Any	NOT 10.0.0.0/16	Block	2
Any	Any	Any	Any	Any	NOT 10.1.0.0/16	Block	3
Any	Any	Any	Any	Any	NOT 192.168.1.0/24	Block	4
Any	Any	Any	Any	Any	10.0.0.0/16	Block	1
Any	Any	Any	Any	Any	10.1.0.0/16	Block	1
Any	Any	Any	Any	Any	192.168.1.0/24	Block	1

ج)

Flag	Protocol	DST Port	SRC Port	DST IP	SRC IP	Act	Int#
Any	HTTPS	443	Any	192.168.1.0/24	Any	Accept	1
Any	HTTPS	443	Any	10.1.0.0/16	Any	Block	1
Any	HTTPS	443	Any	10.0.0.0/16	Any	Block	1

د)

Flag	Protocol	DST Port	SRC Port	DST IP	SRC IP	Act	Int#
Any	HTTPS	443	Any	192.168.1.1	Any	Accept	1
Any	HTTP	80	Any	192.168.1.1	Any	Accept	1
Any	SMTP	25	Any	192.168.1.2	Any	Accept	1
Any	SMTPS	587, 465	Any	192.168.1.2	Any	Accept	1
Any	Any	Any	Any	192.168.1.0/24	Any	Block	1

ه)

Flag	Protocol	DST Port	SRC Port	DST IP	SRC IP	Act	Int#
Any	SSH	22	Any	192.168.1.1	10.0.0.0/16	Accept	2
Any	SSH	Any	22	10.0.0.0/16	192.168.1.1	Accept	2
Any	SSH	22	Any	192.168.1.1	10.0.0.0/16	Accept	4
Any	SSH	Any	22	10.0.0.0/16	192.168.1.1	Accept	4

و)

Flag	Protocol	DST Port	SRC Port	DST IP	SRC IP	Act	Int#
Any	IMAPS	Any	993	10.0.0.0/16	192.168.1.2	Accept	2

Any	IMAPS	Any	993	10.1.0.0/16	192.168.1.2	Accept	3
Any	IMAPS	Any	993	10.0.0.0/16	192.168.1.2	Accept	4
Any	IMAPS	Any	993	10.1.0.0/16	192.168.1.2	Accept	4

سوال ششم:

الف)

$$n = p \times q = 23 \times 11 = 253$$

$$\phi(n) = (p-1)(q-1) = 22 \times 10 = 220$$

$$\gcd(220, 3) = 1$$

$$3d = 1 \ [mod\ 220] \Rightarrow d = 147$$

$$PU = \{3, 253\}$$

$$PR = \{147, 253\}$$

ب)

فرض می‌کنیم تابع Hash ما همانی است:

$$H(x) = x$$

$$\sigma_1 = E\big(PR, H(m_1)\big) = H(4)^{147}\%\ 253 = 4^{147}\%\ 253 = 49$$

ج)

$$D(PU, \sigma_2) = 48^3\ \%\ 253 = 31$$

$$H(m_2) = H(31) = 31 = D(PU, \sigma_2)$$

پس می‌توان نتیجه گرفت که $\sigma_2 = 48$ یک امضای معتبر برای پیام $m_2 = 31$ است.

د)

$$\sigma_1 \times \sigma_2 = 49 \times 48 = 2352$$

$$D(PU, \sigma_1 \times \sigma_2) = 2352^3\ \%\ 253 = 124$$

$$H(m_3) = H(124) = 124 = D(PU, \sigma_1 \times \sigma_2)$$

پس می‌توان نتیجه گرفت که $\sigma_1 \times \sigma_2$ یک امضای معتبر برای $m_3 = 124$ است.