SYN-elárasztás elleni védekezés a RESPIRE algoritmus segítségével

Korn András, Dr. Fehér Gábor, Gyimesi Judit
Budapesti Műszaki és Gazdaságtudományi Egyetem, Távközlési és Médianiformatikai Tanszék, HSNLab

A múltban sok ismert webszervert bénítottak meg rosszindulatú felhasználók hosszabb-rövidebb időre egy SYN-elárasztás (SYN flood) néven ismert támadás segítségével. Ezeknek a támadásoknak a kivédésére több olyan módszert javasoltak, amely bevált és elterjedt. Cikkünkben egy olyan újszerű megoldást ismertetünk, amely lehetőséget biztosít a SYN-áradatok automatikus felismerésére és szűrésére anélkül, hogy számottevő többletterhelést okozna az áldozat számára. Hatékonyságát mind szimulációval, mind numerikus analízissel igazoljuk.

Bevezető

A TCP SYN-elárasztás a TCP-kapcsolatfelépítés (handshake) egy sajátosságát használja ki. A kapcsolatot kezdeményező kliens először olyan csomagot küld a szervernek, amelyen a SYN bit be van állítva, egy kezdeti szekvenciaszámmal. A szerver erre egy SYNACK csomaggal válaszol, a saját szekvenciaszámával. Végül a kapcsolat létrejön, amint a kliens visszaküld egy olyan csomagot, amelyben csak az ACK bit van beállítva, és a szerver kezdeti szekvenciaszámát nyugtázza.

Ahhoz azonban, hogy a szerver el tudja dönteni, hogy egy beérkező ACK üzenet egy kapcsolat-felépítés utolsó üzenete-e, meg kell jegyeznie, melyik kliensnek milyen kezdeti szekvenciaszámú SYNACK csomagot küldött. Ezek az adatok a TCP kapcsolatpuffer (TCP backlog queue) nevű adatstruktúrában tároldónak, amelynek a mérete véges (portonként esetleg csak pár tucat félig nyitott kapcsolat tárolására elegendő).

A SYN-áradat során a támadó nagyszámú SYN csomagot küld, gyakran hamis IP-címről, ám egyik kapcsolat felépítését sem fejezi be ACK üzenettel, így a szerver puffere megtelik: nem képes új kapcsolatokat fogadni. A pufferben tárolt ún. félig nyitott kapcsolatok egy idő után (timeout), ha addig nem érkezik rájuk ACK, törlődnek. Ám ha a támadó csomagok gyorsan jönnek, akkor elárasztják a tárolót, és az áldozat nem lesz képes a legtöbb legitim kliens kérésének feldolgozására.

Létező megoldások

Egyes gyártók, például a Cisco, forgalmaznak a SYN-elárasztás ellen valamekkora védelmet nyújtó routereket. Általában ugyanazt a módszert alkalmazzák, mint az OpenBSD: a TCP-kapcsolatfelépítést ők maguk végzik el, és a védett szervernek csak akkor küldik el a SYN csomagot, ha ők maguk már megkapták a végső ACK-ot. Ahhoz, hogy a kapcsolat működjön, a továbbiakban minden áthaladó csomagon módosítani kell a TCP-szekvenciaszámot (hiszen a router bizonyára más első szekvenciaszámot választott, amikor a SYNACK-ot küldte, mint a szerver). Emellett ezek a routerek hamarabb eldobják a félig nyitott kapcsolatokat, mint a szerverek, így valóban kevésbé érzékenyek a SYN-elárasztásra. Azt azonban látnunk kell, hogy a voltaképpeni problémát nem oldják meg, csupán megnövelik a támadás költségét, mivel továbbra is szükség van erőforrások (memória) allokálására minden félig nyitott kapcsolathoz, ezek az erőforrások pedig végesek.

Egy másik javasolt védelem alapja a SYN csomagok véletlenszerű eldobása (RED jelleggel) [RAD]. Hasonlóan a félig nyitott kapcsolatok élettartamának csökkentéséhez, ez a megoldás sem nyújt valódi védelmet, csupán a támadás költségét növeli meg.

A SYN-elárasztás elleni védekezés egy igen eredeti, és széles körben elterjedt módja az ún. SYN cookie-k használata [SCS]. A módszer lényege az, hogy a szerver a saját SYNACK csomagjában beállított szekvenciaszámba belekódolja azokat az információkat, amelyeket különben a helyi pufferben kellene tárolni; így nincs szükség memória-allokációra, csak néhány számításra. A bejövő ACK csomag által nyugtázott szekvenciaszám segítségével a kapcsolat helyi adatstruktúrája felépíthető anélkül, hogy a SYN és az ACK beérkezése között bármit is tárolnunk kellene. Annak érdekében, hogy ez ne járjon azzal a veszéllyel, hogy az algoritmus ismeretében egy támadó helyesen megválasztott nyugtaszámmal kapcsolatot tudjon hamisítani, a beállított kezdő szekvenciaszámnak kriptográfiai értelemben erősnek kell lennie; így a támadónak csak nagy erőfeszítés (sok próbálkozás) árán sikerülhet érvényes nyugtaszámot generálni.

A SYN-cookie-k hátrányai

Sajnos a SYN cookie-k használata hátrányokkal is jár. Először is, az ilyen módon létrehozott kapcsolatok nem használhatnak nagy ablakméretet, és a maximális szegmensméret sem választható meg szabadon. Másodszor, az erős szekvenciaszám előállítása számításigényes; Bernstein pl. a Rijndael kódoló használatát javasolja minden egyes SYNACK csomag szekvenciaszámának kiszámítására. Harmadszor – és talán ez a legnagyobb probléma – a SYN cookie-kat használó szerver a SYN-áradatra SYNACK-áradattal válaszol: ha a SYN csomagok feladója hamis cím, akkor a hamis címre, vagyis egy mit sem sejtő, ártatlan harmadik félnek (bounce attack).

Így, annak ellenére, hogy a SYN cookie-k még támadás esetén is garantálják a szolgáltatás elérhetőségét, továbbra is van értelme a támadók csomagjait szűrni (vagyis megakadályozni, hogy eljussanak a szerverhez).

Az itt bemutatott RESPIRE algoritmus jó kiegészítése a SYN cookie-knak is; ezek szavatolják a szolgáltatás zavartalanságát, a RESPIRE mechanizmus pedig felderíti a SYN-áradat forrásait, és kiszűri őket. (RESPIRE: Resource Efficient SYN-flood Protection for Internet Routers and End-systems – Erőforráshatékony SYN-áradat elleni védelem az Internet hosztjai és routerei számára). Mivel azonban a RESPIRE reakcióideje igen rövid, a SYN cookie-kra elegendően nagy kapcsolatpuffer megléte esetén voltaképpen nincs szükség.

Az irodalomban több módszert is találunk a folyamatban levő SYN-elárasztás felismerésére; az újabb algoritmusok egyikét az Olvasó a [DSF]-ben találhatja meg. Ennek a módszernek az a hátránya, hogy a támadás elleni védekezéshez csak akkor nyújt hathatós segítséget, ha a támadó közelében lehet elhelyezni azt az eszközt, amely megvalósítja; ez lényegében azt jelenti, hogy minden Internet-szolgáltatónál be kellene vezetni. Amíg erre nem kerül sor, vagy ha az eszközt az áldozat közelében helyezzük el, az algoritmus csak felismerni képes a támadást, azt azonban nem tudja megállapítani, melyik állomás küldi az áradatot.

A RESPIRE rendszer elve

Az itt javasolt módszer nem igényli további adatgyűjtő eszközök elhelyezését. Azokat az adatokat használjuk fel a támadás felismerésére, amelyeket az áldozatnak amúgy is gyűjtenie kell ahhoz, hogy TCP szolgáltatást legyen képes nyújtani.

Az alábbi adatok mind rendelkezésre állnak (vagy könnyen előállíthatók), és alkalmasak a támadás felismerésére, vagy legalábbis valószínűsítésére az alábbi feltételek teljesülése esetén:

- a másodpercenként beérkező SYN csomagok száma meghalad egy küszöbértéket;
- valamely TCP port kapcsolatpuffere (backlog queue) megtelik, SYN cookie-kat kell küldeni a további kapcsolatok fogadásához;
- a félig nyitott kapcsolatok száma meghalad egy küszöbértéket;
- aránytalanul több SYNACK csomag hagyja el a rendszert, mint ahány kapcsolat-felépítést véglegesítő ACK csomag érkezik (a továbbiakban ACK üzeneten mindig ilyen csomagot értünk).

A RESPIRE itt leírt változata az utóbbi heurisztikát alkalmazza, azonban minimális módosításokkal akár az összes módszer kombinációja is használható.

Megjegyezzük, hogy lehetséges lenne a bejövő SYN és bejövő ACK üzenetek számának arányát is vizsgálni. A SYNACK üzenetekben küldött szekvenciaszám ismeretére mindenképpen szükségünk van a számolandó ACK-üzenetek azonosításához, a SYNACK üzenet alapján pedig rekonstruálható a hozzá tartozó SYN, így a SYN-ek számolása redundánsnak tűnhet. Hozzá kell azonban tennünk, hogy ahhoz, hogy a SYNACK-ok számolására alapozhassuk a védelmet, az áldozat képes kell, hogy legyen a bejövő SYN-ek elegendően nagy részére SYNACK-kal válaszolni. Ezt a SYN-cookie-k garantálják, ha azonban nem használunk SYN-cookie-kat, akkor a kapcsolatpuffer méretét kell úgy megválasztanunk, hogy a RESPIRE számára elegendő SYNACK üzenet termelődjön a puffer megtelése előtt. Ha ez sem lehetséges, akkor számolhatjuk a SYN-üzeneteket a SYNACK-ok helyett, ám a SYNACK-okkal ekkor is foglalkoznunk kell a szekvenciaszámok miatt.

Összefoglalva: a SYNACK-üzenetek helyett akkor célszerű a SYN-üzeneteket számolni, ha a védendő szerver nincs felkészítve SYN-cookie-k használatára, és a kapcsolatpuffer méretét sem tudjuk elegendően nagyra beállítani.

Támadáskor egy lehetséges védekezés, ha nem válaszolunk azokra a SYN csomagokra, amelyeket a támadó küld; ezt a legegyszerűbben úgy biztosíthatjuk, hogy tűzfalunkban kiszűrjük őket. Tehát a legfontosabb dolgunk izolálni a támadás forrásait. (Ennél többet csak akkor tehetünk, ha *pushback*kel [PSB] vagy más, hasonló mechanizmussal a támadó csomagjai által használt útvonalon a támadó felé „toljuk” a szűrést.) A támadó kiszűrése az Internet hőskorában gyakorlatilag lehetetlen lett volna, mivel a csomagok forráscímei szabadon hamisíthatóak voltak. Mostanra azonban a legtöbb hálózatból nem engednek ki olyan csomagot, amelynek az állítólagos feladója nem

része a hálózatnak. Emiatt a támadók általában csak saját C osztályú hálózatukon belüli címeket tudnak használni. Globális szolgáltatást nyújtó szerverek esetében ennek az egész tartománynak a szűrése is csak elenyészően kevés legitim klienst érinthet, hiszen nagyon kedvező az arány az összes létező és a támadó által használt hálózatok száma között. Megjegyezzük, hogy a fenti feltételezés alapfeltétele a RESPIRE működésének; ha a támadó tetszőleges forráscímet képes lenne hamisítani, a RESPIRE még ronthatna is a helyzeten, mivel legitim klienseket is kiszűrhet a támadás vélt forrásának kitiltása során. Ennek a veszélye számottevő mértékben csökkenthető, ha a RESPIRE-t csomaghamisítás-érzékelővel (spoof detector), pl. a [HCF]-ben leírt algoritmussal kombináljuk.

A SYN-támadások anatómiája

Napjainkban az ilyen támadások során a támadó általában több tucat olyan számítógépről, ún. „zombiról" küldi a SYN-áradatot, amelyekre korábban betört. Ezeken a gépeken elosztott támadások megvalósítására kifejlesztett programokat helyez el, amelyeket egyszerű utasításokkal képes távvezérelni. Hogy az áradat szűrését megnehezítsék, a zombik általában hamisított forráscímekről küldik a csomagokat; a fent leírtak miatt azonban mindegyik hamisított forráscím ugyanahhoz az IP-alhálózathoz tartozik, mint a zombi tényleges címe.

Megjegyezzük, hogy amennyiben a bejövő SYN-áradat sávszélessége elegendően nagy ahhoz, hogy az áldozat vonalát telítse, már nincs értelme SYN-támadásról beszélni; a támadás olyan általános kapcsolat-telítő támadás, amely történetesen TCP SYN csomagokat használ. Nem célunk ezzel az esettel foglalkozni, noha a *pushback*kel kombinálva a bemutatott RESPIRE algoritmus az ilyen támadások ellen is hatásos lehet.

Ki a támadó?

Korábban már utaltunk arra, hogy SYN-áradat esetén a kimenő SYNACK csomagok és a bejövő, kapcsolat-felépítést véglegesítő ACK csomagok aránya sokkal nagyobb lesz egynél. Mivel a legtöbb válasz nélkül maradó SYNACK csomagot éppen a támadó SYN-csomagjaira adott válaszként küldjük el, a támadót úgy találhatjuk meg, ha megkeressük az(oka)t a hálózato(ka)t, amely(ek)nél nagy az egy érvényes bejövő ACK csomagra eső kimenő SYNACK csomagok száma.

Erre egy lehetséges naiv módszer az lenne, ha egy nagy (2^{24}, azaz 16,7 millió sort tartalmazó) táblázatban számolnánk, hogy hány SYNACK csomagot küldünk az egyes C osztályú hálózatokba, ill. hogy hány ACKot küldenek ezekből nekünk. Jól látható, hogy ez a megoldás nem lenne hatékony: a legtöbb számláló a nullán állna, és a pozitív értékeket mutató számlálópárok többsége is „normális" (1 körüli) arányt tükrözne. A támadó azonosításához mégis az összes sort meg kellene vizsgálnunk (*egy* támadó megtalálásához átlagosan mintegy nyolc és fél millió sort).

A RESPIRE működése

A RESPIRE alapötletét szolgáltató MULTOPS[MPS] ezt a problémát úgy oldja meg, hogy a számlálókat dinamikusan bővíthető hierarchikus adatstruktúrában, egy 256-odrendű fában tárolja, kihasználva az IP-címek hierarchikus jellegét.

A RESPIRE-ben a fa gyökere két, kezdetben nulla értékű számlálót, és 256 darab kezdetben NULL mutatót tartalmaz. Az egyik számláló – a neve `Synack_Out` – a rendszert elhagyó SYNACK üzeneteket számolja, a másik – `Ack_In` nevű – pedig a beérkező hiteles ACK csomagokat (azokat, amelyek egy TCP-kapcsolat felépítését véglegesítik).

Miután legalább `Synack_Min` darab SYNACK csomagot küldtünk ki, minden további `Synack_Period` darab csomag után inkrementáljuk a megfelelő számlálókat és megvizsgáljuk a tárolt érékek arányait. Mint azt később látni fogjuk, a fastruktúra miatt ez aránylag olcsó művelet; így azt javasoljuk, hogy `Synack_Period` értéke legyen 1. Olyan szervereken, amelyeken különösen nagy forgalomra számítunk, a paraméter értéke növelhető a többletterhelés csökkentése érdekében; ez azonban rontja az algoritmus pontosságát és reakcióidejét. A determinisztikus mintavételezés helyett természetesen alkalmazhatunk valamilyen sztochasztikus módszert is, vagy változtathatjuk a paraméter értékét dinamikusan (pl. a forgalomtól függően), ez azonban a lényeget nem érinti.

Amint a fastruktúra gyökérelemében `Synack_Out` és `Ack_In` aránya meghaladja a választott, 1-nél nagyobb R_{max} paraméter értékét (1.5 körül javasoljuk megválasztani; az alacsonyabb értékekhez jobb reakcióidő tartozik, ám növelik a tévedés valószínűségét), feltételezzük, hogy SYN-elárasztás áldozatai vagyunk, és megkezdjük a kiküldött SYNACK csomagok célcímeinek figyelését a támadók felismerése érdekében. Ezekhez tartozó levelekkel bővítjük a fát, a következő módon:

Minden további `Synack_Period` darab kimenő SYNACK vagy bejövő ACK csomag esetén megjegyezzük a távoli IP-címet: legyen ez A.B.C.D. Amennyiben a gyökér A-adik mutatója NULL, létrehozunk egy új csúcsot, és

befűzzük a gyökér alá (root→A). A továbbiakban minden, az A.0.0.0/8 hálózattal összefüggő SYN/ACK forgalmat két helyen számolunk: a gyökérben és az imént létrehozott új csúcsban.

Ha root→A már létezik, megvizsgáljuk, igaz-e, hogy

$$\text{root}\rightarrow A\rightarrow \text{Synack\_Out} \geq \text{Synack\_Min}_{A,} \qquad \text{és}$$

$$\frac{\text{root}\rightarrow A\rightarrow \texttt{Synack\_Out}}{\text{root}\rightarrow A\rightarrow \texttt{Ack\_In}} > R_{\max}$$

Amennyiben mindez teljesül, valószínűsíthető, hogy az A.0.0.0/8 hálózat rejti a(z egyik) támadót. A fát a fenti algoritmussal tovább bővítjük, amíg az A→B→C levél létre nem jön.

A Synack_Min paraméter értéke különbözhet a fa egyes szintjein. Ha a fában lefelé haladva csökkentjük a paraméter értékét, a támadások felismerése gyorsul, a pontosság azonban romlik; ezt ellensúlyozandó növelhetjük pl. R_{max} értékét.

Amennyiben az A→B→C levél létezik, már legalább $\texttt{Synack\_Min}_C$ SYNACK csomagot gyűjtött, és számlálóinak aránya meghaladja R_{max}-ot, feltételezzük, hogy az A.B.C.0/24 egy támadó ellenőrzése alatt áll, és forgalmát a továbbiakban kiszűrjük (pl. az operációs rendszer beépített csomagszűrője segítségével).

Célszerű ezeket a szűréseket egy idő (Block_Timeout) után megszüntetni. Az általunk megfigyelt SYN-támadások nem tartottak 15 percnél tovább, így ezt az értéket javasoljuk. A támadás megszűntének érzékeléséhez lehetne ugyan néhány drága heurisztikával próbálkozni, mint például az újraküldött SYN-ek felismerése, ennél azonban sokkal gazdaságosabb a szűrés átmeneti megszüntetésével kipróbálni, véget ért-e a támadás. Amennyiben a támadás folytatódik, természetesen újra felismerjük és újabb 15 percre kiszűrjük (itt is lehetséges lenne valamilyen adaptív viselkedés bevezetése).

Ha találtunk és kiszűrtünk egy támadó alhálózatot, a hozzá tartozó levelet törölhetjük a fából, mivel már nem fogunk innen SYN csomagot kapni; számlálóinak értékét levonjuk a fában fölötte levő csúcsok számlálóiból. Ha a gyökérben még ezután sem áll helyre az arány, további támadók is létezhetnek. További optimalizációs lehetőség az imént eltávolított csúcsok újbóli létrehozása a szűrés feloldásakor, hogy gyorsabban tudjunk reagálni, ha a támadás még nem ért volna véget.

Prune_Interval másodpercenként (2-nél alacsonyabb érték nem javasolt) megvizsgáljuk, van-e „gyanús” csúcs a fában (tehát olyan, ahol a számlálók hányadosa nagyobb R_{max}-nál, de ahol Synack_Min-t még nem értük el). Az összes nemgyanús csúcsot töröljük, a gyanúsaknak pedig nullázzuk a számlálóit. Ezzel memóriát és későbbi feldolgozási időt takarítunk meg. Megjegyezzük, hogy a törlendő csúcsok kiválasztására összetettebb algoritmust is használhatunk, amely pl. figyelembe veszi, hogy az adott csúcsban hogyan változott a számlálók aránya az előző Prune_Interval alatt; a RESPIRE alapötletének megértéséhez azonban elegendő az itt bemutatott naiv módszer. Egy összetettebb algoritmus a „lassú áradatok” lejjebb ismertetett problémájának megoldásában segíthet.

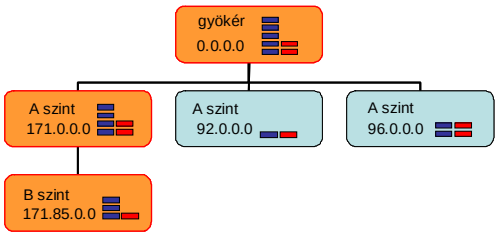
Azáltal, hogy a gyanús csúcsoknak csak nullázzuk a számlálóit, de a csúcsokat nem szüntetjük meg, az adott alhálózatból érkező támadót a következő Prune_Interval alatt hamarabb megtaláljuk, mivel nem kell megvárnunk, amíg a szülő-csúcsokban összegyűlik Synack_Min csomag: az alacsonyabb szintű csúcs eleve létezik. Az összetettebb algoritmusra vonatkozó fenti megjegyzések itt is megállják a helyüket.

A számlálók nullázására azért van szükség, mert csak a ténylegesen folyamatban levő támadásokra akarunk reagálni. Sajnos azonban így a támadó „lassú áradatok” („slow flood”) segítségével elkerülheti az észlelést. Ha rendkívül sok különböző C-osztályú hálózatból küld másodpercenként kevesebb, mint $\texttt{Synack\_Min}_C$/Prune_Interval csomagot, akkor a fa gyökerében ugyan látjuk, hogy támadás alatt vagyunk, de a támadó folyamok egyenként nem okoznak akkora forgalmat, hogy a fa alsóbb szintjein is elérjék Synack_Min-t a számlálók a nullázás előtt; együttes hatásukra mégis betelik a kapcsolatpuffer.

Ebben az esetben egy lehetséges reakció a következő: a gyanús csúcsok számlálóit nem töröljük Prune_Interval lejártakor; továbbá finomítjuk a gyanússág megítélését. Minden gyanú felett áll az olyan csúcs, amelyben a számlálók aránya R_{legit}-nél alacsonyabb (javasolt érték pl. 1,1 vagy 1,05). Az ilyeneket töröljük a fából az intervallumhatárokon. Enyhén gyanúsak azok a csúcsok, amelyekben az arány R_{legit} és R_{max} közötti: ezeket nem töröljük, csak nullázzuk, és további megfigyelés alatt tartjuk. Azon csúcsok számlálóit pedig, amelyeknél az arány meghaladja R_{max} értékét, még csak nem is nullázzuk. Arra számítunk, hogy így idővel lesz olyan C szintű csúcs, amely a szűrés feltételeit teljesíti.

Az 1. ábrán egy háromszintű RESPIRE-fát láthatunk. Az egyes csúcsokban levő oszlopok a SYNACK és ACK csomagok relatív számát mutatják (nem pontos számértékeket). A 92.0.0.0/8-as és a 96.0.0.0/8-as csomópont közel ugyanannyi ACK csomagot küldött, amennyi SYNACK-ot kapott, tehát valószínűsíthetően legitim. A bal oldalon

látható sötétebb hátterű csúcsok viszont nagyon is gyanúsak. Vegyük észre, hogy a gyökér is gyanús – ebből következtethetünk a támadás tényére.



1. ábra: példa egy RESPIRE-fára

A RESPIRE memóriaigénye

Mivel dinamikus adatstruktúrákkal dolgozunk, el kell kerülnünk, hogy maga a RESPIRE rendszer DoS-támadás eszköze lehessen: korlátoznunk kell a memóriahasználatát. Egy csúcs memóriaigénye 32 bites architektúrán (2+256)×32 bit (a két számláló plusz a 256 mutató). Ez összesen 1032 byte. Ha nem korlátoznánk a létrehozható csúcsok számát, összesen legfeljebb 16777216 + 65536 + 256 darab csúcsunk lehetne (ez a fenntartott IP-tartományok elhanyagolása miatt nem túl pontos felső becslés); így a teljes RESPIRE adatstruktúra mérete elérhetné a 16,25 gigabájtot, aminek a kezelése már nehézkes.

A létrehozandó csomópontok száma a gyanús (támadó) alhálózatok számától függ. Nem valószínű, hogy egyetlen támadó kétszáznál több C-osztályú hálózatot ellenőrizne. A legkedvezőtlenebb eset az, ha ez a kétszáz C-hálózat mind különböző A-hálózatban található, ekkor ugyanis mindegyik áradatforráshoz három csomópontot kell létrehoznunk, összesen tehát hatszáz darabot (mintegy 600 kilobyte). Általában elegendőnek tűnik ötszázra korlátozni a létrehozható csúcsok számát; értelemszerűen, ha rendkívül elosztott támadásokra számítunk, ez a korlát növelhető.

Ha elértük a korlátot, de új csúcsot kellene létrehoznunk, egy lehetséges eljárás az, hogy megkeressük a gyökér legkevésbé gyanús gyermekét, és töröljük a hozzá tartozó részfával együtt. Ha csak egyetlen A-csúcsunk van, folytassuk a keresést az A-szinten; az egyetlen A-csúcsnak bizonyosan egynél több gyermeke lesz, mivel különben nem érhettük volna el az ötszázas korlátot. Idővel feltehetően izolálunk egy támadót, és csökken majd a csúcsok száma.

Analízis – a RESPIRE reakcióideje

Először általános közelítést adunk az algoritmus reakcióidejére, majd pedig felső határértéket. Látni fogjuk, hogy a RESPIRE még a legrosszabb esetben is gyorsan reagál (így a SYN cookie-k használata nem feltétlenül szükséges), ráadásul az észlelés ideje szempontjából a kis intenzitású áradatok jelentik a legrosszabb esetet: minél nagyobb az áradat intenzitása, annál gyorsabban ki tudjuk szűrni.

Jelen analízis arra az esetre vonatkozik, amikor a támadók – ha többen vannak –, azonos C alhálózatban vannak. A számítások általánosíthatók összetettebb támadásra is, kivéve a „lassú áradat” esetét, amit már tárgyaltunk.

Feltesszük, hogy a rosszindulatú SYN csomagok egyenletesen érkeznek, ψ csomag/sec intenzitással. Ha több támadó van, akkor hatásukat összegződve tartalmazza ez az érték. A legitim kliensek SYN forgalma Poisson-folyamattal közelíthető, mivel a támadáson kívül időegységenként érkező csomagok száma független egymástól. Az egyszerűség kedvéért feltesszük, hogy a SYNACK válaszcsomagot a szerver a SYN kézhezvételével egy időben kiküldi, így a kimenő SYNACK csomagok is Poisson-eloszlást mutatnak. Ugyanez érvényes a beérkező ACK csomagokra is, mivel bár az egy szakaszban bejövő ACK-ok nem az abban a szakaszban kimenő SYNACK-okra adott válaszok, de a darabszámuk várható értéke megegyezik, hiszen Poisson-folyamatnál csak a vizsgált időintervallum hossza, és nem a kezdőideje számít. Az RTT (Round Trip Time) figyelembevétele ugyan bonyolíthatná a helyzetet, de ha nincsen komoly burst-ösödés, akkor nem okoz nagy hibát a közelítés.

A támadás kezdetét attól az időponttól számoljuk, amikor a szerverhez ér az első rosszindulatú SYN csomag. Ezzel az adattal kapcsolatban csak az számít, hogy mennyi idővel van egy `Prune_Interval` kezdete után: legyen ez Δt_1. A továbbiakban Δt idő múlva már $\psi \cdot \Delta t$ támadó SYN csomag érkezett. Hogy eközben mennyi legitim kapcsolatkezdeményezés történt, azt a következő módon határozhatjuk meg: az egy időintervallumban beérkező legitim csomagok számának várható értéke $\lambda_l \cdot \Delta t$. Ugyanennyi SYNACK-ot küld ki a szerver, és közel ennyi ACK-nak is kell visszaérkeznie.

A RESPIRE mechanizmusa szerint két feltételnek kell teljesülnie, hogy felismerje a támadás tényét:

$$\frac{l_l \cdot \Delta t + y_t \cdot \Delta t}{l_l \cdot \Delta t} \geq R_{\max} \quad \text{és}$$

$$l_l \cdot \Delta t + y_t \cdot \Delta t \geq synack\_min_{root}$$

Látszik, hogy jelen feltételezések mellett az első egyenlőtlenségben az arány nem függ az időtől, támadás esetén ennek mindig teljesülnie kell. Tehát ha egy támadás detektálható, akkor felismerjük, amint a minimális SYNACK értéket elérjük az adott `Prune_Interval`-ban (Δt_p).

A detektálhatóság feltétele pedig a megfelelő arány, amiből a támadás intenzitására a következő adódik: $y_t \geq (R_{\max} - 1) \cdot l_l$

Minden szintre hasonló gondolatmenet követhető, mint a gyökér esetén, azzal a különbséggel, hogy az egyes szintekre a legitim forgalomnak csak valamekkora hányada jut el. Ha feltételezzük, hogy minden alhálózatból azonos mennyiségű csomag jön, az egy A alhálózatból érkezők csak kb. az 1/256-odát teszik ki az egész beérkező forgalomnak, B-nél ennek négyzetét, C-nél köbét. Ez természetesen nem lesz igaz, mivel a kliensek IP-címeinek eloszlása nem egyenletes. Az A szinten még közelítőleg sem az, mivel a teljes címtér jelentős része speciális célokra van fenntartva. Sajnos azonban még a B szinten sem tételezhetünk fel egyenletes eloszlást: Magyarországon például a 195-tel kezdődő IP-címek második oktetje nagy valószínűséggel 228, stb. Bevezethetnénk a $\phi(x)$ paramétert úgy, hogy $\phi(x)$ a bejövő összes legitim SYN csomagnak az x alhálózatra eső részét jelenti. Közelítésnek azonban elfogadhatjuk azt a megoldást, hogy A szinten valamilyen "a" paraméterrel számolunk, a B és C szinten pedig egyenletesnek vesszük az eloszlást.

Ha szintenként másképp választottuk meg `Synack_Min`-t, akkor ezt is figyelembe kell venni.

Könnyen belátható, hogy hosszabb idő szükséges, ha a csomagszámlálás közben átlépünk egy `Prune_Interval`-határt, és törlődnek a számláló-értékek. Ekkor csak az adott szint számlálása kezdődik újból, hiszen magukat a csomópontokat nem töröljük. Szintenként csak egy határátlépés lehetséges, ugyanis ha a csomópont gyanúságának eldöntéséhez több, mint egy intervallumnyi időre volna szükség, akkor a vizsgálat soha nem fejeződne be.

A szintenként szükséges időre a második felismerési feltétel alapján a következő képlet adható, ha határátlépés sehol nem történik:

$$\Delta t'_{root} \geq \frac{synack\_min_{root}}{l_l + y_t}$$

$$\Delta t'_A \geq \frac{synack\_min_A}{\frac{1}{a} \cdot l_l + y_t}$$

$$\Delta t'_B \geq \frac{synack\_min_B}{\frac{1}{a \cdot 256} \cdot l_l + y_t}$$

$$\Delta t'_C \geq \frac{synack\_min_C}{\frac{1}{a \cdot 256^2} \cdot l_l + y_t}$$

Ha az intervallumhatár-átlépéseket is figyelembe vesszük, akkor a fenti közelítésekkel élve az alábbi módon fejezhető ki az algoritmus reakcióideje a fa egyes szintjein (I{A} az A esemény indikátora):

$$\Delta t_{root} = I\left\{\left\lfloor \frac{\Delta t_t}{\Delta t_p} \right\rfloor < \left\lfloor \frac{\Delta t_t + \Delta t'_{root}}{\Delta t_p} \right\rfloor\right\} \cdot \left(\left\lceil \frac{\Delta t_t}{\Delta t_p} \right\rceil \cdot \Delta t_p - \Delta t_t \right) + \Delta t'_{root}$$

$$\Delta t_A = I\left\{\left\lfloor \frac{\Delta t_t + \Delta t_{root}}{\Delta t_p} \right\rfloor < \left\lfloor \frac{\Delta t_t + \Delta t_{root} + \Delta t'_A}{\Delta t_p} \right\rfloor\right\} \cdot \left(\left\lceil \frac{\Delta t_t + \Delta t_{root}}{\Delta t_p} \right\rceil \cdot \Delta t_p - \Delta t_t - \Delta t_{root} \right) + \Delta t'_A$$

$$\Delta t_B = I\left\{\left\lfloor \frac{\Delta t_t + \Delta t_{root} + \Delta t_A}{\Delta t_p} \right\rfloor < \left\lfloor \frac{\Delta t_t + \Delta t_{root} + \Delta t_A + \Delta t'_B}{\Delta t_p} \right\rfloor\right\} \cdot \left(\left\lceil \frac{\Delta t_t + \Delta t_{root} + \Delta t_A}{\Delta t_p} \right\rceil \cdot \Delta t_p - \Delta t_t - \Delta t_{root} - \Delta t_A \right) + \Delta t'_B$$

$$\Delta t_C = I\left\{\left\lfloor \frac{\Delta t_t + \Delta t_{root} + \Delta t_A + \Delta t_B}{\Delta t_p} \right\rfloor < \left\lfloor \frac{\Delta t_t + \Delta t_{root} + \Delta t_A + \Delta t'_C}{\Delta t_p} \right\rfloor\right\} \cdot \left(\left\lceil \frac{\Delta t_t + \Delta t_{root} + \Delta t_A + \Delta t_B}{\Delta t_p} \right\rceil \cdot \Delta t_p - \Delta t_t - \Delta t_{root} - \Delta t_A - \Delta t_B \right) + \Delta t'_C$$

Ha az előző szint vizsgálatának vége és az aktuális szint vizsgálatának vége külön intervallumba esik, akkor a szükséges idő a teljes, ehhez a szinthez szükséges, intervallumátlépés nélküli idő, plusz az előző szint vizsgálatának végétől az intervallumhatárig eltelő idő.

A teljes reakcióidő a fa egyes szintjein eltöltött idő összege:

$$\Delta T = \Delta t_{root} + \Delta t_A + \Delta t_B + \Delta t_C$$

Felső becslésként nézzük a reakcióidő szempontjából legrosszabb esetet, amely – mivel a reakcióidő detektálható támadás esetén csak a `Synack_Min` elérésétől függ – akkor következik be, amikor minden legális SYN csomag más A alhálózatból jön, mint a támadó csomagok. Ebben az esetben nem kell semmiféle feltételezéssel élnünk a forgalom eloszlásával kapcsolatban. Az értékek a következők szerint változnak:

$$\Delta t'_{szint} = \frac{synack\_min_{szint}}{y_t}$$

Az egyes szinteken töltött idő felülről becsülhető az átlépés nélküli eset idejének kétszeresével, hiszen ha a vizsgálat nem fejeződik be az intervallum-határig, akkor az indikátorfüggvény utáni szorzó kisebb, mint $\Delta t'_{szint}$, ellenkező esetben a vizsgálat befejeződhetett volna abban az intervallumban, amelyben kezdődött.

$$\Delta t_{szint} \le 2 \cdot \Delta t'_{szint}$$

Tehát az algoritmus teljes reakcióideje nem haladhatja meg a legtöbb időt igénylő szint átlépés nélküli idejének négyszeresét:

$$\Delta T \le 8 \cdot \max_{szint}\left\{\frac{synack\_min_{szint}}{y_t}\right\}$$

A formulából jól látszik, hogy minél nagyobb intenzitású a támadás, annál gyorsabban reagál a RESPIRE. Mint már utaltunk rá, ezt a felső becslést csak a különösen kis intenzitású támadások tudják megközelíteni. ψ egyre nagyobb értékeinél egyre kisebb a valószínűsége, hogy kettőnél több intervallum kellene a támadó hálózat azonosításához, ugyanis a nagyon gyors támadások felismerési ideje 0-hoz tart. A második intervallum csak akkor szükséges, ha a támadás egy intervallum végéhez közel kezdődik.

Szimuláció

Az algoritmus teljesítőképességét szimulációval is vizsgáltuk [RSP]. A kép teljessé tétele érdekében röviden e helyütt is összefoglaljuk a szimuláció eredményeit.

Egy nagy forgalmú SMTP-szervert szimuláltunk, átlagosan 62,8 folyamatban levő legitim kapcsolattal és 12,6 új kapcsolatkéréssel másodpercenként. A legitim kliensek IP-címe teljesen véletlenszerű volt. A rendszerben elhelyeztünk nyolc támadót, akik véletlen időpontban véletlen intenzitású SYN-áradattal támadták meg a szervert; a rendelkezésükre álló alhálózat mérete szintén véletlenszerű volt, /24-es alhálózati maszktól /16-osig. Az alhálózatukon belül véletlenszerűen választottak forráscímet minden egyes támadó SYN csomaguknak, egy támadáson belül is váltogatva azt.

A szimuláció néhány számszerű eredményét az alábbi táblázat foglalja össze:

Támadás sorszáma	Első csomag (s)	Alhálózat mérete	Csomagráta (csomag/s)	Elfogadott csomagok	Reakcióidő (s)
1	80,780	32768	41274	22191	0,629
2	239,046	512	91938	505	0,011
3	764,754	4096	58563	1920	0,045
4	890,803	512	82013	505	0,011
5	1229,573	16384	39586	6766	0,244
6	2039,080	8192	40932	3535	0,101
*5**	*2129,699*	*16384*	*39586*	*6767*	*0,165*
*6**	*2939,157*	*8192*	*40932*	*3535*	*0,113*
7	4060,253	32768	88895	13231	0,193
8	4729,277	512	31267	505	0,021

Láthatjuk, hogy a táblázatban az ötös és a hatos támadás kétszer is szerepel; ennek az a magyarázata, hogy a tűzfalszabály maximális élettartamát 900 másodpercre (15 perc) állítottuk be, ezek a támadások pedig ennél hosszabb ideig tartottak. A szabály elévülése után ismét fel kellett őket ismerni és kiszűrni. Az „elfogadott csomagok” oszlopban található értékek azt adják meg, hány – az adott támadótól származó – hamis SYN csomag érte el a szervert a támadás kiszűrésének pillanatáig; a többi oszlop jelentése remélhetőleg egyértelmű. A reakcióidő tekintetében megállapíthatjuk, hogy az azonos mértékben elosztott, de nagyobb intenzitású támadások kiszűréséhez szükséges idő általában kisebb volt; a közel azonos csomagrátájú, ám elosztottabb támadás kiszűréséhez szükséges idő pedig általában nagyobb.

Elvégeztünk néhány próbaszámítást is, hogy összevessük az analízis eredményeit a szimulációéival. Vegyük a 3. támadót. Ő 4096 címből álló tartományt ellenőriz; ez 16 darab szomszédos C osztályú hálózatot jelent. Így egyszerre 16 gyanús C szintű csúcsunk lesz a fában, mind azonos B csúcs alatt. Egyenletes címkiválasztást feltételezve minden C csúcsba a teljes intenzitás 1/16-od része jut, vagyis 3660,19 csomag/s. A B szintig tehát alkalmazhatjuk az egy támadó alosztályt feltételező analízis eredményét:

$$\Delta T_{root} + \Delta T_A + \Delta T_B \le 6 \cdot \max_{szint=root,A,B} \left\{ \frac{synack\_min_{szint}}{y_t} \right\} = 0,01$$

A C csúcsok felismerésénél pedig egyenként 1830,09 csomag/s támadó intenzitással számolva:

$$\Delta T_C \le 2 \cdot \frac{synack\_min_C}{y_C} = 0.055$$

Az összes időre tehát az analízis alapján a fenti két eredmény összegét, 0,065s-ot kaptunk, ami jó felső becslése a mért értéknek. Nagyságrendileg tehát megegyezik a két módszer által mutatott eredmény.

Hasonlóképpen az 1. támadás által kiváltott reakció idejének felső becslésére 0,635s-ot, a 2.-éra 0,011s-ot, a 4.-ére 0,012s-ot, az 5.-ére 0,338s-ot, a 6.-éra 0,17s-ot, a 7.-ére 0,293s-ot, a 8.-éra pedig 0,032s-ot kapunk.

Természetesen előfordulhat, hogy nem azonos a választott hamis IP címek eloszlása, és esetleg egy C alhálózatot hamarabb felismerünk, mint egy másikat. Ha vannak olyan alhálózatok, amikben lényegesen kisebb az intenzitás, akkor az összidő növekedhet. Vegyük észre azonban, hogy ez lényegét tekintve az az eset, amikor a lassú, ennélfogva veszélytelenebb támadásokat később ismerjük föl, hiszen a tartomány többi részét már tiltottuk, így az áldozatot ténylegesen elérő támadó forgalom kisebb intenzitású lesz.

Értékelés

A fent ismertetett RESPIRE algoritmus nem az egyetlen, amely a SYN-áradatok elleni védelmet szolgálja, ám kétségkívül az egyik legkisebb járulékos számításigényű módszer, ami megbízhatóan, gyorsan szűri ki a támadást; ezenkívül részben védelmet nyújt az ellen is, ha a szervert használják kisebb sávszélességű áldozatok kapcsolat-telítésére („bounce attack”). A SYN-támadások elleni védelem többi eszközének, elsősorban a SYN cookie-k kiegészítéseként is hasznos. Memóriaigénye is csak támadás esetén nő meg néhány kilobájtnyinál nagyobbra.

A fejlesztés korai fázisában levő linuxos referenciaimplementáció elkészülte után életszerű körülmények között is kipróbálható lesz az algoritmus.

Irodalomjegyzék

[ACC] Ratul Mahajan, Steven M. Bellovin, Sally Floyd, John Ioannidis, Vern Paxson, Scott Shenker, *"Controlling High Bandwidth Aggregates in the Network"*, Computer Communications Review 32:3, July 2002, pp. 62-73.
[HCF] Cheng Jin, Haining Wang, Kang G. Shin, *"Hop-Count Filtering: An Effective Defense Against Spoofed DDoS Traffic"*, Proceedings of the 10th ACM conference on Computer and communication security, 2003, pp. 30-41.
[SCS] Daniel J. Bernstein, *"SYN cookies"*, http://cr.yp.to/syncookies.html, 1997.
[DSF] Haining Wang, Danlu Zhang, Kang G. Shin, *"Detecting SYN Flooding Attacks"*, Proceedings of IEEE InfoCom, 2002
[PSB] John Ioannidis, Steven M. Bellovin, *"Implementing Pushback: Router-Based Defense Against DDoS Attacks"*, Network and Distributed System Security Symposium, February 2002.
[RAD] Livio Ricciulli, Patrick Lincoln, and Pankaj Kakkar, *"TCP SYN Flooding Defense"*, Comm. Net. and Dist. Systems Modeling and Simulation Conf. (CNDS'99), 1999.
[MPS] Thomer M. Gil, Massimiliano Poletto, *"MULTOPS: a data-structure for bandwidth attack detection"*, Proceedings of the 10th Usenix Security Symposium, August 2001.
[RSP] Gábor Fehér, András Korn, *"RESPIRE – a Novel Approach to Automatically Blocking SYN Flooding Attacks*", Proceedings of EUNICE 2004, pp. 181-187