Syddansk Universitet

DM829

Kaminsky DNS exploit

Jan Christensen - 241189
Anders Knudsen 150885

12. maj 2012

Indhold

1 Indledning

Denne rapport beskriver det sikkerhedshul som Dan Kaminsky fandt i den måde de fleste navneservere opererer på. For at kunne forstå de tekniske detaljer bag, er det nødvendigt at kende den basale opbygning bag DNS. Derfor vil der først blive givet kort introduktion til DNS. Dernæst vil "cache poisoning", som er en centrel del af angrebet, blive forklaret og der vises hvordan Dan Kaminsky udnyttede det til at kunne overtage domænenavne.

2 Introduktion til DNS

DNS står for **Domain Name System** og er et navne system som bruges til at oversætte fra domænenavne til IP-adresser. Denne "mapping" mellem domænenavne og IP-adresser ændrer sig løbende, og derfor kender en navneserver ikke mappingen for alle domæner. Hver navneserver har kun informationer om forholdsvis få domæner, men ved så til gengæld hvilke andre servere der skal spørges hvis der skal indhentes oplysninger om et domæne den ikke kender noget til. Basalt set kan en navneserver give følgende 3 slags svar:

1. Et **autoritativt** svar. Dvs. klienten får direkte svar på den forespørgsel der er lavet.
2. Uddelegering. Navneserveren kender ikke svaret, men fortæller hvor man kan finde svaret.
3. Afvisning. En navneserver kan vælge helt at ignorere en forespørgsel.

Inden det beskrives hvordan DNS fungerer, defineres der en række begreber som der bliver benyttet i forbindelse med DNS.

Top Level Domain (TLD): Eksempelvis .dk .com og .net

Navneserver: Serveren som svarer på DNS forespørgsler fra DNS klienter. Hvis en navneserver kender alle informationer om et domæne, kaldes den en **autoritativ** navneserver for det domæne.

Rekursiv forespørgsel: Modtager navneserveren en rekursiv forespørgsel, skal den sende et komplet svar tilbage til klienten. Hvis der forespørges på et

domæne den ikke er autoritativ for, spørger den (rekursivt) andre navneservere indtil den har det komplette svar. Bemærk at en navneserver godt kan nægte at understøtte rekursive forespørgsler.

Ikke rekursiv forespørgsel: Hvis serveren er autoriativ for domænet sendes det komplette svar tilbage. Hvis ikke, sendes et ukomplet svar tilbage hvor der henvises til hvilken navneserver der ved mere om domænet.

Rod servere: DNS er opbygget hierakisk. Øverst oppe er der 13 såkaldte rod-servere. De ved hvilke navneservere der bestyrerer de enkelte TLD'er, og er således de første navneservere der bliver spurgt når der laves en forespørgsel.

A record: Angiver for et givent domænenavn hvilken IP det peger på

NS record: Dette er et record som en navneserver returnerer hvis den ikke er autoriativ for det domæne der spørges om. Recorden peger på den navneserver som ved mere om domænet.

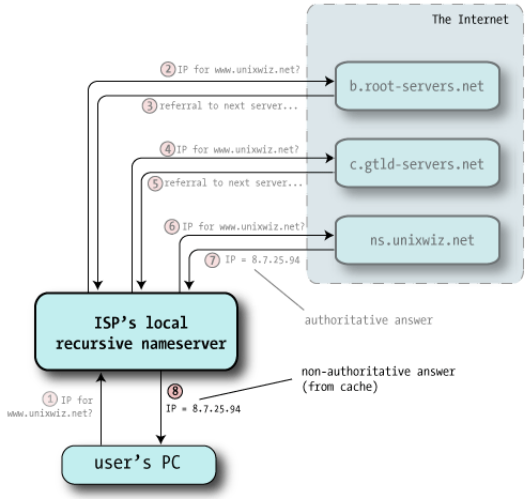
2.1 Cache

Pga. den distribuerede opbygning af DNS, skal der potentielt sendes mange forespørgsler til mange forskellige navneservere før der opnås et svar. Da man naturligvis ønsker at DNS er hurtigt, er der introduceret en cache til hvert trin i processen. Hver gang en navneserver får et svar, cacher den resultatet inden det sendes videre. Ved fremtidige forespørgsler på et domæne skal der ikke laves nogle rekursive forespørgsler, men i stedet kan svaret returneres fra cachen med det samme. Til hvert element i cachen er der så tilknyttet en **Time-to-live** (TTL) som angiver hvor lang tid elementet er gyldigt.

Det er samtidig cachen som Dan Kaminsky udnytter i sit DNS angreb. "Cache poisoning" foregår ved at man får indsat korrupt data i cachen. Ved at få indsat korrupt data i cachen med en høj TTL, kan en hacker forhindre navneserveren i at foretage yderligere rekursive forespørgsler, og på den måde vil navneserveren blive ved med at give korrupte svar.

2.2 Et eksempel

For at illustrere hvordan DNS virker vises her et eksempel på en DNS-forespørgsel. Det første der sker er at klienten sender en rekursiv forespørgsel til dennes internetudbyders navneserver. Denne navneserver tillader rekursive forespørgsler og da den ikke er autoritativ for www.unixwiz.net, sender den forespørgslen videre til en af de 13 rod-servere. Rod-serveren modtager forespørgslen og eftersom den ikke tillader rekursivitet, sender den en reference til navneserveren som fortæller hvor den skal spørge næste gang (et NS record). Det fortsættes indtil ns.unixwiz.net spørges og denne sender et autoritativt svar tilbage (trin 7). Dette svar caches og videresendes til klienten.



Figur 1: Eksempel på en DNS forespørgsel. Billedet er taget fra: http://unixwiz.net/techtips/iguide-kaminsky-dns-vuln.html

3 Cache poisoning

En navneserver sender og modtager hele tiden mange DNS forespørgsler, og skal derfor have en måde hvorpå den kan matche en forespørgsel med et svar. Til det bruges der et forespørgsels id (QID), som er et id der bliver sendt med hver DNS pakke. Når serveren modtager et svar, kigger den på QID'et og finder hvilken forespørgsel den hører til. Modtager den en pakke med et ugyldigt QID (dvs. den matcher ikke nogen forespørgsler) ignoreres

pakken. Modtages der flere gyldige pakker med samme QID, er det det senest modtagne resultat der bruges - *first good answer wins*.

Det er denne opbygning som gør cache poisoning muligt i forbindelse med DNS. Som det ses i eksemplet i figur 1 modtages det autoritative svar i trin 7. Men i virkeligheden udfører navneserveren ikke noget tjek for hvor pakken rent faktisk er sendt fra. Med andre ord, hvis man kan sende et gyldigt svar (men med korrupt data) til navneserveren inden den modtager det rigtige svar, er det muligt at få navneserveren til at cache det forkerte resultat. I forhold til figur 1 svarer det til hvis en angriber sender en pakke med det rigtige QID til navneserveren, inden ns.unixwiz.net når at sende det rigtige svar i trin 7.

3.1 Finde det rigtige QID

Udfordringen er således at få finde/gætte det rigtige QID og få det sendt hurtigt nok. Nedenfor er et eksempel på hvordan et dns cache poisoning angreb kan se ud:

1. Angriberen sender en forespørgsel til navneserveren på det domæne (fx. sdu.dk) han vil overtage.
2. Angriberen ved nu at serveren inden længe vil spørge ns.sdu.dk efter en IP. Han flooder derfor serveren med forfalskede A records
3. Serveren forespørger ns.sdu.dk om ip'en til sdu.dk
4. Angriberen rammer rigtigt og giver serveren en falsk IP.
5. ns.sdu.dk svarer serveren, men dette svar ignoreres.
6. Serveren gemmer den falske IP i cache og sender den videre som svar.
7. Fremtidige forespørgsler på sdu.dk vil få det samme (forkerte) svar.

3.1.1 Problemer med dette angreb

Der er dog en række problemer med ovenstående angreb som gør at det i praksis ikke er muligt

- Navneserveren der angribes har allerede cachet sdu.dk. I det tilfælde vil navneserveren returnere svaret med det samme, og ingen rekursive forespørgsler laves.

- Den rigtige navneserver og den navneserver der angribes står topologisk nærtliggende (i forhold til netværket). I det tilfælde er det ikke muligt at nå at sende pakken hurtigt nok.
- QIDs er randomiseret. QID feltet er på 16 bit, så der er $2^{16} = 65536$ forskellige muligheder. Det er ikke sandsynligt at der gættes rigtigt inden den rigtige navneserver svarer.

3.2 Dan Kaminskys angreb

Dan Kaminskys angrebs angrib bygger på det simple cache poisoning angreb. Men i stedet for bare at angribe/overtage et enkelt domæne, overtager angriberen de autoritative records for domænet i stedet. På den måde styrer man ikke kun selve domænet, men også samtlige underdomæner.

Angrebet foregår ved at angriberen opsætter en sin egen navneserver og gør den til en autoritativ navneserver for domænet han vil overtage. Som udgangspunkt udgør det ikke nogen fare, da der alligevel ikke er nogen navneservere der vil spørge angriberens falske navneserver. Udfordringen er nu at få navneserveren der skal angribes til at spørge angriberens navneserver.

Angrebet herefter er næsten helt analogt til det oprindelige angreb. Ændringen er i trin 2 i angrebet. I stedet for at sende A-records sendes der nu NS-records som henviser til angriberens egen navneserver. Hvis det på et tidspunkt lykkedes for ham at gætte det rigtige QID, vil den angrebne navneserver så foretage en ny rekursiv forespørgsel mod den korrupte navneserver. Samtidig vil den cache resultatet, og alle fremtidige forespørgsler mod det domæne vil havne ved den korrupte navneserver.

I forhold til svaghederne ved det oprindelige angreb, er fordelen at der ikke er problemer med allerede cachede domænet. Hvis man ønsker at overtage eksempelvis sdu.dk, laves der bare forespørgsler på underdomæner som angriberen ved ikke eksisterer (blah.sdu.dk, blah2.sdu.dk, osv.). Disse forespørgsler tvinger navneserveren til at lave en række rekursive forespørgsler. I tilfælde af at den rigtige navneserver "vinder" racet og svarer først (og dermed får cachet et gyldigt resultat), benyttes bare et nyt underdomæne som ikke er cachet.

Med denne forbedring kan man automatisere hele processen, og så bliver det at "bruteforce" 2^{16} muligheder ikke helt så umuligt. Der er lavet proof-

of-concept programmer som bruger omkring 10-15 minutter på at angreb.

3.3 Fiks til problemet

Der er to måder hvorved Kaminskys angreb kan undgås. For det første kan en navneserver sættes op til at detektere at den er under angreb. Hvis den opdager at den modtager en række ugyldige pakker fra den samme server, kan den uddele et midlertidigt ban eller lignende og derved gøre chancen for at korrupt data bliver indsat i cachen mindre.

Den anden måde er at gøre chancen for at QID'et gættes markant mindre. Man kunne forestille sig at dette kunne gøres ved at gøre antallet af QID's større, for eksempel 2^{32}. Men da det, på grund af den distribuerede opbygning af DNS, vil være nødvendigt at rulle ud på samtlige DNS-servere og klienter, er det ikke en holdbar læsning. I stedet lå løsningen i den port som navneserverne sendte og modtog pakker på. Oprindeligt brugte en navneserver den samme port til alle dens pakker og derved kunne angriberen altid vide hvilken port han skulle sende de falske pakker til.

I samarbejde med udviklerne af alle de mest udbredte DNS-servere fandt Kaminsky på at randomisere denne port. Således blev chancen for at en angriber gættede det rigtige QID, samt gættede den rigtige port meget mindre. For eksempel blev Microsofts servere sat til at tage en tilfældig port ud af 2^{11} mulige. Dette fik så sandsynligheden for at en angriber gættede rigtigt ned på $1/(2^{16} \cdot 2^{11}) = 1/134.000.000$.