**Какво е сесия**

Протокола HTTP е Stateless т.е. третира всяка нова заявка (Request) като отделна и самостоятелна от предишнота. За това ни трябва техника за да укажем на сървъра, че поредица от заявки идват от един и същи клиент. Такава техника са сесиите - те съхраняват информация за клиента на сървъра и предоставят на клиента псевдопроизволен идентификатор, който той трябва да изпрати на сървъра със всяко запитване за да се идентифицира. Сървъра разпознава идентификатора и го свързва със съответната информация съхранена в него информация за клиента. Този идентификатор се нарича Session ID и обикновено се съхранява като Coockie и се предава в заявките към сървъра в POST информацията или направо в URL-а на зявката. Последния метод е опасен и се избягва с цел сигурност.

**Сисиите в PHP**

Най популярния език за програмиране в интернет PHP предоставя своя собствена система за поддръжка на сесии. Там сесииният идентификатор се казва PHPSESSID. При стартиране на сесия PHP съставя нов PHPSESSID за тази сесия и го предава на клиента. Идентификатора е псевдопроизволен тъй като за съставянето се ползват системни данни като IP адреса на клиента и системното време. Поредицата от слепени един след друг парчета системна информация минават през криптиращ хеширащ алгоритъм за да направят крайния идентификатор. Използваните алгоритми са MD5 и SHA1, като първия се използва по подразбиране. При MD5 идентификатора е дълъг 32 байта и съответно предоставя ентропия (непредвидимост) от 256 бита.

**PHPSESSID**

За първични идентификатор PHPSESSID се слепват IP адреса на клинта, Unix epoch (броя секунди от началото на Unix epoch – 1 Януари 1970г.) и милисекундите спрямо сървъра и псевдопроизволна стойност от LCG (линеен конгруентен генератор).

|  |  |  |  |
| --- | --- | --- | --- |
| IP адрес | Unix epoch | милисекунди | LCG псевдопроизволно число |
| 32 бита | 32 бита | 32 бита | 64 бита |
| 160 бита | | | |

Така представената ни поредица предоставя 160 ентропия. След това тази поредица преминава пред криптиране и резултата от MD5 е 32 байтов низ.

Тези 160 бита ни предоставят огромен брой комбинации - 1.46150164 × 1048. Опит да се приложи метода Brute Force, където трябва да съставим всяка една комбинация т.е. да генерираме всички възможни идентификатори ще се провали. Дори да можем да генерираме 160 трилиона комбинации в секуда все още ще ни отнеме около 900 квадрилио̀на ери (където ера е 500 милиона години).

Една популярна атака е предвиждане на този идентификатор и по този начин друго лице може да се представи за клиента и да ползва неговата сесия. Тези атаки са изключително трудни за изпълнение както ще се види по-нататък.

**Ентропия**

Ако разгледаме всеки елемент на първичният идентификатор можем да намалим първоначалната ентропия.

**IP адрес**

Има много начини да се сдобием с адреса на клиента, който искаме да атакуваме като предвидим PHPSESSID му. Можем да му изпратим адрес на подправен сайт, който да го записва или по-късно да го проверим в логовете на на уеб сървъра като Apache. Ако успеем да си набавим това IP тогава можем да намалим ентропията с 32 бита, защото вече ги знаем.

|  |  |  |  |
| --- | --- | --- | --- |
| IP адрес | Unix epoch | милисекунди | LCG псевдопроизволно число |
| 32 бита | 32 бита | 32 бита | 64 бита |
| - 32 бита |  |  |  |
| 160 бита – 32 бита = 128 бита | | | |

Намирането на IP адреса не е сравнително трудна задача, но пък използването му навреме обаче е. Много хора използват динамични IP адреси предоставени им от интернет доставчиците им и се сменят при всяко свързване в мрежата. Това ограничава много прозореца в който трябва да се изпълни атаката. Също така използването на NAT мрежи, където много клиенти се представят пред мрежата с едно и също публично IP, усложнява обстановката.

**Време на генериране**

Вторият байт в първичния идентификатор е броя секунди от 1 Януари 1970, а третия милисекундите, в който е поискано генерирането. Първото е секундата, в която клиента се логва в системата и му се създава сесия. Много уеб приложения поддържат услуга ифромираща клиента кога друг потребител влиза в системата. Това често става чрез AJAX заявки към сървъра дали даден потребител е online. Лесно може да се направи скрипт, който пуска заявка всяка секунда и следи дали потребителя минава от статус offline във online. По често приложенията имат профилна страница на всеки потребител с индетификация дали потребителя е влязъл в системата или не. Процеса е същия – скрипт който прави заявки до профилната страница на клиента и следи секцията от нея (врез Regular Expressions например) за статуса на потребителя. По това може с точност ±1 сукунда можем да определим кога е поискана нова сесия и можем да намалим ентропията отново с 32 бита. Тук трябва да се обърне внимание, че времето което ни интересува е това на сървъра, защото то участва в генерирането на идентификатора, а не това на клиента. То може да се намери в HTML отговора (response) на нашата заявка.

|  |  |  |  |
| --- | --- | --- | --- |
| IP адрес | Unix epoch | милисекунди | LCG псевдопроизволно число |
| 32 бита | 32 бита | 32 бита | 64 бита |
| - 32 бита | - **32 бита** |  |  |
| 160 бита – 64 бита = 96 бита | | | |

Третият байт е излючително труден за набавяне, дори бихме могли да кажем, че е невъзможно да определим милисекундата, в която PHP генерира идентификатора. Обаче можем да намалим ентропията по логическа разсъждения. Милисекундите са стойност от 0 до 999 999 т.е. стойност побираща се в 20 бита и съответно останалите 12 бита са запълени с статични стойности в случая 0. Това намалява отново ентропията с 12 бита.

|  |  |  |  |
| --- | --- | --- | --- |
| IP адрес | Unix epoch | милисекунди | LCG псевдопроизволно число |
| 32 бита | 32 бита | 32 бита | 64 бита |
| - 32 бита | - **32 бита** | - **12 бита** |  |
| 160 бита – 76 бита = 84 бита | | | |

Останалите 20 бита остават за кандидати за намиране, чрез Brute Force.

**LCG псевдопроизволно число**

Последните 2 байта с първичния идентификатор изглеждат най-трудни за откриване и от една страна е така, но не по очевидните причини.Това 64 битово число е псевдопроизволно, защото се генира от LCG функция. Това е линеен конгруентен генератор, който генерира поредица от числа чрез на seed – число на базата на което се генерира редицата. Ако имаме seed числото можем да генерираме число от тази поредица. В PHP то е съставено от две 32 битови числа: s1 – секундите и милисекундите, в който е поискано първото число в поредицата и s2 - PID на процеса.

Първата част s1 се получава като се изчисли изключващо ИЛИ на секундите или отрицанието на милисекундите:

Тук отново можем да направим същите съждения като при третия байт и да отстраним 12 бита от ентропията. Тук трябва да подчертаем, че това време е когато е поискана първата стойност в редицата. Можем да предположим, че това е скоро след пускането на PHP процеса. Обаче трудно можем да предположим кога е било това. За целта можем да използваме DDOS атака върху сървура и да го накараме да се рестартира. Тъй като няма да знам кога точно се рестартира можем поне да направим предположение, че например е било в последния час.

Втората част s2 е номера на PHP процеса (PID). В \*nix системите това е 15 битово число и съответно можем да намалим ентропията отново с 17 бита. За да пробължим нататък обаче, ще ни са необходими и самия PID. Това е най-трудното в тази атака. Ако по някаква причина можем да извикаме PHP функцията getmypid() или можем да извикаме системна функция като ps на сървъра можем да се снабдим с него. Ако имаме някоя от тези възможности то имаме още по-голям контрол на сървъра и надали е необходимо да се затрудняваме с тази атака. Има обаче и възможноста да попаднем на PHP информациона страница на сървъра, където този процесен номер фигурира.

Тези две числа образуват seed числото използвано за генериране на редицата. Това означава, че сме намалили ентропията в още 12 + 32 бита.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| IP адрес | Unix epoch | милисекунди | LCG псевдопроизволно число | |
| 32 бита | 32 бита | 32 бита | 64 бита | |
| - 32 бита | - **32 бита** | - **12 бита** | - **12 бита** | - **32 бита** |
| 160 бита – 120 бита = 40 бита | | | | |

Успяхме да намалим началната ентропия от 160 бита на едва 40 бита. Това обаче все още е прекалено много за Brute Force атака.

**Атаката**

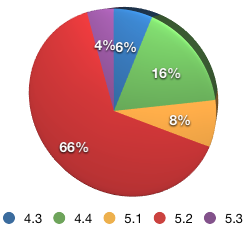
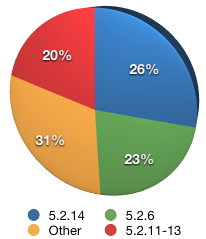
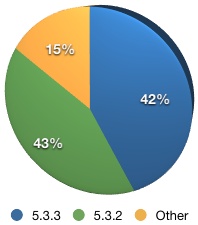
Атаката изисква известно време подготовка. Трябва да получим PHPSESSID от сървъра, който атакуваме и по него след като знаем своято IP, приблизително време на заявката и PID можем да разпознаем използваното за нас LCG числото. За целта използваме Brute Force метода да генерираме съответните първични идентификатори по дадената информация и прилагаме съответни криптиращ алгоритъм, докато не получим даденото ни PHPSESSID. Така получаваме използваното за нас число генерирано от LCG. След това може отново да приложим Brute Force за да намерим seed числто използвано от сървъра на базата на едно от генерираните от него числа получено от горния алгоритъм и s2 (PID на PHP процеса). Тези операции могат да отнемат ден взависимост от това колко бързо намерим seed числото.

След като вече имаме seed числото и ни е известен алгоритъма за генериране на LCG числата вече можем да генерираме таргетираната атака към клиента само с 20 бита ентропия останала от милисекундите когато се генерира неговото PHPSESSID. Това е възможно тъй като поредицата числа генерирани от seed числото са взаимосвързани (линеен конгруентен) и съответно само със самото seed число можем да генерирами всички числа произлизаши от него. Ако следим внимателно числата преди и след като клиент получи PHPSESSID можем да кажем със сигурност диапазона от числа, едно от които е ползвано за генерирането на неговото PHPSESSID.

С тези 20 бита ентропия генерираме около 1 милион възможни първични идентификатора и започваме да пращаме заявки до сървъра докато открием кое е негото. Това само по себе си може да отнеме един ден зависимост от това къде в списъка със генерирани идентификатори седи истинския. За целта може да се използва дистрибутирана атака , защото сървъра може да има ограничение на броя заявки от един клиент.

Защита

Тази уязвимост в генерирането на PHPSESSID беше намерена от Сами Камкар през Март 2010 и е сравнително неизвестна. От тима на PHP ведната направили промяна в кода и добавили допълнително ентропия. Увеличили размера на LCG числто двойно и главното, че по желание вече генерирането на seed числото може да се направи с /dev/urandom – поток от „произволни” числа в \*nix базираните системи. Тези промени са отразени във версии 5.3.2 и 5.2.13 натам.

В света обаче най-използваната версия на PHP е 5.2 както личи от първата графика. Във втората виждаме разбивка на подверсиите на 5.2 и забелязваме, че около половината са застрашени от тази атака. При поколението 5.3 забелязваме, че едва 15% от системи са уязвими.

Проблемът е, че много малко вървъри upgrage-ват тукущата си версия на PHP от страх за съвместимост или бъгове в новите. Това обеснява и извесния брой системи, който все още използват PHP 4 и 3 и дори 1.х.

Почти всички допълнителни системи за менажиране на сесии в PHP в основата си използват вградената подръжка и съответно са уязвими.

За целта най-сигурнония метод за премахването й е или рискования upgrade до последната стабилна версия на PHP където уязвимоста е премахната или собствен алгоритъм за генериране на сесии и по-точно PHPSESSID. Ако използваме свой собствен начин за генерирана тези идентификатори възможността атакуващия да го разбере е от нищожен до никакъв. Съществуват и самостоятелни системи за менажиране на сесиите, който не използват вътрешната имплементация на PHP въобще, но отново, тяхната популярност ги прави потенциална жертва поради факта, че атакуващия може да знае за тях и да им противодейства по друг начин.