Casper Hacker Wargame: Solutions

Seppe Duwé : r0349304

19 december 2016

1 Overview

Level	Password	Time
4	`LBc1dJGMRUDGGdWCnmo2E3F0QYVMmBmv`	6 uren
40	`B6OyaozZDg3ysFmEaYeGAzaUP8GbN4An`	3 uren
6	`qCeyA3Rb1Xuw4tpDXoaNFftxSU9DwjHJ`	3 uren
60	`dUPC70LY2jqnpog0cBpZtoSTDSnLD1UG`	1 uren
8	`HEpkEkasRhTGxvMxOIHV9L11XPCgw1p6`	6 uren
80	`5GOVGN27jm8yoPhAD8IeFO1EBBbQB1OK`	1 uren
10	`oMX51Zm1PJAJPp25uagP7RUOHRjHizgN`	4 uren
Verslag		5 uren
Total		31 uren

2 Casper4 solution

2.1 Description

Casper4 is een programma waar als argument een string (naam) wordt aan meegegeven. De `greetUser` functie wordt aangeroepen en de meegegeven string wordt in een char array gekopieerd. Nadien wordt de tekst "Hello"samen met de ingegeven string uitgeprint.

Er is 1 belangrijke variabele in dit programma, toegewezen op de stack:

- Een buffer `buf` met lengte 666 die de ingegeven string bevat.

2.2 Vulnerability

De ingevoerde string kan langer zijn dan de voorzien lengte van 666 karakters. Waardoor de return adres van de `greetUser` functie kan worden overschreven. En verwijzen naar een locatie in de buffer met onze eigen ingevoerde instructies. Er wordt gebruik gemaakt van een stack-based buffer overflow.

2.3 Exploit description

We gebruiken de stack-based buffer overflow om dit niveau te exploiteren. Omdat we het return adres van de functie/stack frame kunnen wijzigen naar onze eigen ingevoerde instructies, is het mogelijk om een shell (met de rechten van het uitvoerende programma) voor ons te openen. Doordat voor de shellcode een grote hoeveelheid NOP-bytes zijn, moet de door ons gekozen return adres niet

precies in het begin van de shellcode wijzen maar is ergens in het deel ervoor ook goed om de aanval te laten slagen.

Startadres buf = `0xbffff1d6`

Adres base pointer= `0xbffff478`

Adres return adres = adres base pointer + 4 bytes = `0xbffff47c`

Aantal te schrijven bytes = Adres return adres - Startadres buf + 4 return adres = 682 bytes = NOPs (645 bytes) + Shellcode (21 bytes) + Random karakters (12 bytes) + Return adres (4 bytes verwijzen ergens in het NOPs gebied) = (645 + 21) + 12 + 4)

2.4 Exploit usage

Een werkende exploit is inbegrepen in de tar-ball en kan worden gestart met `sh exploit4.sh`.

2.5 Mitigation

Het programma zou maximum 666 karakters mogen kopiëren, en zou gebruik kunnen maken van een lengte specifier voor strncpy. "Stack Canaries"zou deze aanval detecteren of bij een non-executable stack zou deze aanval niet werken.

3 Casper40 solution

3.1 Description

Gelijkaardig aan Casper4 maar nu gaat men eerst de input nakijken op `x90` (NOP) waarden vooraleer men de input kopieert in de buffer.

3.2 Vulnerability

Dezelfde vulnerability als bij Casper4 maar met extra security checks (geen NOPs in de opgegeven string).

3.3 Exploit description

We gebruiken terug de stack-based buffer overflow om dit niveau te exploiteren Mijn eerste idee was om in het begin van de string een null karakter te plaatsen zodat de for-loop zou worden beëindigt. Deze truc werkte niet aangezien ook de string eindigde bij de `strcpy` functie. Een andere optie was om de NOP `x90` instructie te vervangen door een alternatief. Als alternatief heb ik gekozen voor `push eax; pop eax`. Deze instructies geven tezamen hetzelfde resultaat als een NOP. Maar aangezien er nu twee operaties zijn, zou de kans wel kleiner zijn dat we met de door ons gekozen return adres op een `push eax` terechtkomen (indien de adressen dynamisch zijn).

Net als bij Casper4 worden er 682 bytes als argument meegegeven. Maar het aantal NOP instructies wordt nu gedeeld door twee en vervangen door de `push eax; pop eax` instructies. Aangezien ik bij Casper4 645 NOP bytes nodig had, heb ik er nu 324 `push eax; pop eax` bytes plus één random byte.

3.4 Exploit usage

Een werkende exploit is inbegrepen in de tar-ball en kan worden gestart met `sh exploit40.sh`.

3.5 Mitigation

Net als bij Casper4, zou het programma niet buiten de array mogen kopiëren. Dit kan men bereiken door een veiligere copy functie te gebruiken waar men de maximum lengte van de string meegeeft. De "Stack Canaries"techniek zou deze aanval detecteren waarbij we een waarde in de stackframe plaatst juist voor de base pointer/return adres. Bij het terugkeren van een functie wordt er nagekeken of deze waarde niet gewijzigd is. Een andere methode zou zijn om de stack non-executable te maken, waardoor deze aanval niet zou werken.

4 Casper6 solution

4.1 Description

Casper6 is een programma waar als argument een string (naam) wordt meegegeven. Het programma bestaat uit een struct `data_t` met als naam `somedata`, wat uit een char-array `buf` met lengte 666 bytes en een functionpointer bestaat. Het adres van `greetUser` functie wordt in deze functionpointer opgeslagen en in de char-array worden de meegegeven bytes gekopieerd. Hierna wordt de functiepointer uitgevoerd met als argument de buffer. Waarna de tekst "Hello"samen met de ingegeven string wordt uitgeprint.

Er zijn 2 belangrijke variabele in dit programma, toegewezen op de heap:

- Een buffer `buf` met lengte 666 die de ingegeven string bevat.
- Een functiepointer `*fp` die verwijst naar de greetUser functie.

4.2 Vulnerability

De ingevoerde string kan langer zijn dan de voorzien lengte van 666 karakters. Waardoor de functionpointer van de struct kan worden overschreven. En deze kan laten verwijzen naar een locatie in de buffer met onze eigen ingevoerde instructies. Er wordt gebruik gemaakt van een heap-based buffer overflow.

4.3 Exploit description

Er wordt een heap-based buffer overflow gebruikt om dit level te voltooien. Dit door een langere string mee te geven dan voorzien (666 bytes), kan men de functiepointer in de struct overschrijven.

Startadres buf = `0x8049800`

Adres (*fp) = `0x8049a9c`

Aantal te schrijven bytes = Adres return adres - Startadres buf + 4 bytes return adres = 672 bytes = NOPs (647 bytes) + Shellcode (21 bytes) + return adres (4 bytes verwijzen ergens in het NOPs gebied) = (647 + 21) + 4)

4.4 Exploit usage

Een werkende exploit is inbegrepen in de tar-ball en kan worden gestart met `sh exploit6.sh`.

4.5 Mitigation

De kwetsbaarheid zit hem in de onveilige copy functie, die over de grenzen van de array kopieert. Een oplossing is een veiligere functie te gebruiken waar men de maximum lengte van de string meegeeft. Een andere optie is om de heap als non-executable te markeren.

5 Casper60 solution

5.1 Description

Programma is gelijkaardig aan Casper6, de enige aanpassing is dat de `greetUser` functie char-array gaat nakijken op `x90` (NOP) waarden.

5.2 Vulnerability

Gelijkaardig aan Casper6, maar met een extra beveiliging check. Er kan geen gebruik meer gemaakt worden van de `x90` (NOP) waarden in de string. Er wordt terug een heap-based buffer overflow toegepast.

5.3 Exploit description

We passen dezelfde techniek toe als bij Casper6 enkel wijzigt men de `NOP` instructies naar `push eax; pop eax` instructies. Wat uiteindelijk hetzelfde resultaat heeft.

5.4 Exploit usage

Een werkende exploit is inbegrepen in de tar-ball en kan worden gestart met `sh exploit60.sh`.

5.5 Mitigation

Alsook hier zouden dezelfde beveiligingstechnieken als bij Casper6 werken.

6 Casper8 solution

6.1 Description

Identiek programma aan Casper4 met enkel het verschil dat de stack nu non-executable is.

- Een buffer `buf` met lengte 666 die de ingegeven string bevat.

6.2 Vulnerability

Aangezien de stack non-executable is, kan er geen gebruik meer gemaakt worden van de vorige stack-based buffer overflow techniek. Waardoor er nu gebruik wordt gemaakt van de Return-to-libc techniek. Bij deze techniek wordt de stack zodanig aangepast dat men de uitvoering van het programma wijzigt. En het mogelijk maakt om code die al aanwezig is in het geheugen (libc) te gebruiken. Het uiteindelijk doel is om het commando `system(``/bin/xh'')` uit te voeren.

6.3 Exploit description

Door een buffer-overflow is het mogelijk om de stack aan te passen. Deze passen we zodanig aan dat de return waarde van de `greetUser` functie naar de `system()` functie verwijst. Aangezien we precies weten hoe een functie aanroep wordt afgehandeld. Kunnen we nu manueel de stack in zo een toestand zetten dat we aan `system()` functie een parameter kunnen meegeven. Deze parameter is de pointer naar de string `/bin/xh` die kan worden opgeslagen in een environment variabele of meegeven in de buffer zelf. Aangezien het niet evident is om een null termination karakter (`\0`) mee te geven, heb ik voor de eerste mogelijkheid gekozen.

Na het terugkeren van `greetUser` functie, verwijst de stack pointer naar de vorige argumentenlijst en de base pointer naar de voorgaande frame pointer. Begin van een elke functie-aanroep:

```
<+0>:  push %ebp
<+1>:  mov %esp,%ebp
<+2>:  subl $8, %esp
<+3>:  ...
```

Elke functie (alsook de `system()` functie) pushed men de oude base pointer op de stack en wijzigt men de stack pointer naar de base pointer. Om de argumenten te lezen telt men 8 bytes bij de base pointer op. En voor het return adres te kennen worden er 4 bytes bij de base pointer opgeteld. Om geen log gegevens van het foutief afsluiten te krijgen, zetten we de return waarde van de `system()` functie op de exit functie.

System() pointer: `0xb7e6b0b0`

Exit() pointer: `0xb7e5ebf0`

Environment variabele pointer (variabel): `0xbffff91e`

Doordat de locatie van de environment variabele wijzigt, afhankelijk van de bestandsnaam, heb ik een korte for-loop geschreven die alle adressen rond `0xbffff91e` afgaat.

6.4 Exploit usage

Een werkende exploit is inbegrepen in de tar-ball en kan worden gestart met `sh exploit8.sh`. Dit script zal een environment variabele aanmaken met als string `/bin/xh` en een for-loop zal enkele verschillende adressen afgaan om de environment variabele te zoeken.

6.5 Mitigation

Mogelijke technieken om deze aanval te stoppen is "Layout randomizationën Control-flow integrity. Bij de eerste techniek gaan we variaties in de run time memory adressen steken. De tweede techniek gaat de compiler extra checks inbouwen, om na te gaan dat de volgorde van uitvoering van het programma gelijk is zoals gecodeerd wordt in het bronprogramma.

7 Casper80 solution

7.1 Description

Programma is gelijkaardig aan Casper8, met het verschil dat de input string, vooraleer te kopiëren naar de buffer, wordt gecontroleerd op `NOP` instructies.

7.2 Vulnerability

Identiek aan Casper8

7.3 Exploit description

De techniek toegepast bij Casper8 werkt hier ook. Aangezien we geen gebruik maken van `NOP` instructies. Hierdoor geen aanpassingen nodig.

7.4 Exploit usage

Een werkende exploit is inbegrepen in de tar-ball en kan worden gestart met `sh exploit80.sh`. Dit script zal een environment variabele aanmaken met als string `/bin/xh` en een for-loop zal enkele verschillende adressen afgaan om de environment variabele te zoeken.

7.5 Mitigation

Dezelfde besproken technieken als bij Casper8 zullen deze soort aanvallen tegenhouden.

8 Casper10 solution

8.1 Description

Casper10 is een programma waar als argument een string (naam) wordt aan meegegeven. De `greetUser` functie wordt aangeroepen en de meegegeven string wordt via de `sprintf` functie in geformatteerde output opgeslagen in de buffer en uitgeprint. Waarna de functie terugkeert en indien de globale variabele `isAdmin` op true staat wordt de shell uitgevoerd.

- Een buffer `buf` met lengte 666 die de ingegeven string bevat.
- Een boolean `isAdmin` die bepaalt of een shell wordt geopend.

8.2 Vulnerability

Aangezien stack canaries en de non-executable stack zijn geactiveerd alsook maakt men geen gebruik meer van de `strcpy` functie, zijn voorgaande technieken niet meer van toepassing. Aangezien we enkel een variabele dient aan te passen om de shell te openen, maken we gebruik van een data-only attack, toegepast op de format string van de `printf` functie.

8.3 Exploit description

Aangezien we de format string van de `printf` functie beheersen, is het mogelijk om waarden uit het geheugen te lezen en te schrijven. Waardoor het mogelijk wordt om de boolean `isAdmin` op een niet null getal (true) te zetten.

Adres isAdmin: `0x8049940`

Adres eerste opgegeven karakter in buffer array: `0xbffff478`

Eerste adres locatie ingelezen met %p = `0xbffff454` = locatie return adres

Verschil geeft 36+4 bytes = 40 bytes gedeeld door 4bytes per pointer = 10 locaties opschuiven. We weten nu dat we een waarde, in ons geval het adres van de `isAdmin` variable kunnen lezen als we de 10de pointer uitlezen.

`./casper10 'AAAA %10$p' -> Hello AAAA 0x41414141!`

We kunnen nu ook een waarde wegschrijven naar het opgegeven adres. Aangezien men de boolean als true willen zien, moeten we juist een niet null getal op dit adres wegschrijven.

8.4 Exploit usage

Een werkende exploit is inbegrepen in de tar-ball en kan worden gestart met `sh exploit10.sh`.

8.5 Mitigation

Een mogelijk tegenmaatregel is om Adres randomization te gebruiken. Waardoor het moeilijker wordt om uit te zoeken welk adres moet worden aangepast. Een andere mogelijkheid is om ervoor te zorgen dat de format string niet afhankelijk is van de gebruikersinvoer.