Master in de Toegepaste Informatica

Casper Hacker Wargame

Dries Janse
r0627054
20 december 2019

Inhoud

Overzicht

Level	Paswoord	Tijd (uur)
casper4	tnL7L3hY0DVMqEHtrkySQ2M7XwYUNyjE	12
casper41	JkZv4uouYkyOP3e1JQwPr4BP6VvcJTWz	1
casper6	EetVSPHTbn3M1moui3Gg88DbItByyPWz	2
casper61	Sb7ihQ4sGLEQj0O15AG0ke3xgggRsFjF	1
casper8	q6UYWuXz0iv7b65t1qnHAFBeL4w41YEQ	16
casper80	swaysxx5L11ycLAOP3urZ1nLfFlocrqY	1
casper10	Ztbqs7CPzvM49D5PXyW5VvJSc1MvTODU	4

Casper4

Omschrijving

Het casper4 programma vereist één parameter als input. Eerst wordt er nagekeken of er een parameter meegegeven wordt. Indien dit niet het geval is gaat het een error bericht: “Usage: /casper/casper4 YourName” teruggeven. Vervolgens stopt het programma met een errorcode 1.

Indien er wel een parameter meegegeven wordt gaat de “greetUser” functie opgeroepen worden met de meegegeven input. Deze functie maakt een buffer aan van 999 karakters en gaat vervolgens de meegegeven input kopiëren naar deze buffer. Tot slot print het “Hello” gevolgd door de inhoud van de buffer.

Vulnerability

Dit level bevat een stack-based buffer overflow. In de “greetUser” functie wordt er gebruik gemaakt van de “strcpy” functie. Deze kopieer functie gaat de input kopiëren naar de buffer. Het is een onveilige functie omdat er niet wordt gekeken of de input langer is dan 999 karakters, hierdoor kan de aanvaller meer karakters geven als input en zo het return adres overschrijven (of zelfs verder). Deze aanval is mogelijk omdat het geen gebruik maakt van stack canaries en de stack is uitvoerbaar.

Exploit beschrijving

Om deze exploit uit te voeren, moeten we eerst weten hoeveel bytes er overschreven moeten worden om zo het return adres te overschrijven. Als allereerst beginnen we met de totale lengte van de buffer te berekenen, deze is niet exact gelijk aan 999 bytes. We kunnen dit op de volgende manier berekenen met gdb:

- Door het commando “disassemble greetUser” uit te voeren in gdb. Dit geeft de range van memory als machine instructies terug. We zien hierin “0x3ef” staan, in het decimaal stelsel wil dit zeggen dat er 1007 bytes zijn.
- Een alternatieve methode is om het programma uit te voeren met een breakpoint in de greetUser methode en hierin het adres van de buffer (p &buf) en het adres van base pointer (p $ebp) op te vragen. Deze zijn, bij een invoer van 1000 karakters, respectievelijk: 0xbfffee59 en 0xbffff248. Als we deze van elkaar aftrekken geeft dit in het decimaal stelsel: 1007.

Uit bovenstaande berekening weten we dat de buffer een grootte heeft van 1007 bytes. Hierbij moeten we nog eens 8 bytes optellen. Deze 8 bytes bestaat uit 4 bytes voor de base pointer en 4 bytes voor het eigenlijke return adres. In totaal moeten er dus 1015 bytes overschreven worden.

In deze aanval gaat de buffer overschreven worden met uitvoerbare code en het nieuwe return adres gaat hiernaar verwijzen.

De 1015 bytes invoer ziet er als volgt uit:

- 900 NOP bytes (“x90”)
- 21 bytes shellcode
- 90 bytes voor opvulling tot het return adres
- 4 bytes voor het eigenlijke adres

De eerste 900 bytes bestaan uit NOP operaties, dit staat voor: null operation. Het maakt deze aanval makkelijker omdat mijn eigen overschreven return adres niet moet verwijzen naar de exacte locatie van de shell code. Het kan nu verwijzen naar een locatie in het begin van de buffer (in de NOP range). Deze NOP instructies worden gevolgd door de 21 bytes shellcode die /bin/xh gaan uitvoeren. Deze shell code is een variant van de 21 bytes shellcode door kernel_panik, beschreven in het extra materiaal op de casper hacker wargame website. Het enige verschil met deze code is dat het gebruik maakt van /bin/sh in plaats van /bin/xh. Vervolgens gaan er 90 bytes geschreven worden aan opvulling om tot aan het return adres te komen. Tot slot wordt er een adres van 4 bytes geschreven die verwijst naar een locatie in het begin van de buffer (in de NOP range).

Er zijn meerdere oplossingen mogelijk, zo kan de hoeveel NOP operaties en het aantal opvulbytes aangepast worden. Dit blijf correct zolang de totale invoer 1015 bytes bevat, de 21 bytes shellcode hetzelfde blijft en het adres verwijst naar een plaats in het geheugen van de geschreven NOP operaties voor de shellcode.

Mitigation

Deze aanval kan op verschillende manieren worden gemitigate. De implementatie van de code kan aangepast worden door gebruik te maken van de “strcpy“ functie met 3 parameters. Deze extra parameter geeft het maximum aantal karakters dat gekopieerd kan worden naar de buffer. Specifiek in dit programma is het maximum gelijk aan 999 karakters.

Een tweede manier is door gebruik te maken van een niet uitvoerbare stack. Hierdoor kan het returnadres nog steeds overschreven worden maar de code kan niet uitgevoerd worden in de stack.

Een derde manier is door gebruik te maken van stack canaries. Deze canary is een geheim die geplaatst is op de stack en dewelke verandert telkens het programma uitgevoerd wordt. Als de stack overschreven wordt gaat de canary ook overschreven worden. Bij elke functiereturn gaat gekeken worden of de canary aangepast werd, als dit het geval is stop het programma onmiddellijk.

Een vierde mitigatie is door gebruik te maken van ASLR, Address Space Layout Randomization. Dit maakt het moeilijker om het correcte adres te specifiëren van de shell code.

Advanced level: Casper41

Dit geavanceerde level heeft dezelfde structuur als casper4. Het enige verschil is dat de het gebruik maakt van de “clearEnvironment” functie om de environment variabelen te verwijderen.

Dezelfde exploit als deze van casper4 kan gebruikt worden omdat deze oplossing niet gebruik maakt van environment variabelen. Het bevat nog steeds dezelfde vulnerabilities.

Casper6

Omschrijving

Het programma in casper6 is gelijkaardig aan casper4. Het verschil is het gebruik van de struct. De struct slaagt een buffer van 999 karakters op gevolgd met een functie pointer die als input parameter een pointer naar karakters aanvaardt. Deze struct wordt opgeslagen in het datasegment van de memory. De “greetUser” functie gaat in dit programma enkel “Hello” printen gevolgd door de karakters dewelke voorzien worden als argument van de functie.

Bij de start van het programma gaat de somedata.fp functiepointer geïnitialiseerd worden met het adres van de “greetUser” functie. Vervolgens wordt er nagekeken of er een parameter meegegeven wordt, net zoals in casper4. Hierna wordt de input gekopieerd naar de somedata.buf variabele. Dewelke, tot slot, gebruikt wordt bij het aanroepen van de somedata.fp functie.

Wat opvalt zijn de variabelen in de struct en het onveilig kopiëren met behulp van de “strcpy” functie.

Vulnerability

Dit programma bevat een heap-based buffer overflow vulnerability. Het is mogelijk om de functie pointer van de struct (somedata.fp) te overschrijven. Dit is mogelijk omdat de pointer boven de buffer variabele is opgeslagen en er niet gekeken wordt hoeveel karakters er gekopieerd worden in de buffer. Dit laat de aanvaller toe om genoeg data als input te geven om de functie pointer te overschrijven en deze te laten wijzen naar eigen geschreven code.

Exploit beschrijving

Als allereerste zoek ik de lengte van de buffer. Dit kan gedaan worden in gdb, zoals uitgelegd bij de casper4 exploit. De adressen van somedata.fp en somedata.buf zijn respectievelijk 0x8049bc8 en 0x80497e0. Beide adressen wijzen naar de start van de variabele. Door deze van elkaar af te trekken bekomen we de lengte van de buffer, dit is 1000 bytes. Hierbij moeten nog eens 4 bytes opgeteld worden om de functie pointer te overschrijven. In totaal moet de attacker 1004 bytes schrijven. In mijn specifieke oplossing ziet dit er als volgt uit:

- 900 NOP bytes (“x90”)
- 21 bytes shellcode
- 79 bytes voor opvulling tot de functie pointer
- 4 bytes voor het eigenlijke adres

De overschreven functie pointer wijst nu naar een locatie waar een NOP operaties geschreven staat. Het programma gaat verder deze operaties overlopen en uiteindelijk de shellcode uitvoeren.

Mitigation

Deze aanval kan op verschillende manieren worden gemitigate. De implementatie van de code kan aangepast worden door gebruik te maken van de “strcpy“ functie met 3 parameters. Deze extra parameter geeft het maximum aantal karakters dat gekopieerd kan worden naar de buffer. Specifiek in dit programma is het maximum gelijk aan 999 karakters.

Een tweede manier is door gebruik te maken van een niet uitvoerbare stack. Dit omdat de eigenlijke geïnjecteerde (input) shellcode wordt opgeslagen in de stack.

Stack canaries kunnen in dit geval niet gebruikt worden omdat er in deze aanval enkel een functie pointer wordt overschreven en niet helemaal tot aan het returnadres.

Een derde mitigatie is door gebruik te maken van ASLR, Address Space Layout Randomization. Dit maakt het moeilijker om het correcte adres te specifiëren van de shell code.

Advanced level: Casper61

Dit geavanceerde level heeft dezelfde structuur als casper6. Het enige verschil is dat het gebruik maakt van de “clearEnvironment” functie om de environment variabelen te verwijderen.

Dezelfde exploit als deze van casper6 kan gebruikt worden omdat deze oplossing geen gebruik maakt van environment variabelen. Het bevat nog steeds dezelfde vulnerabilities.

Enkel het adres in het script moet aangepast worden, dit zijn de laatste 4 bytes. De reden hiervan is dat het adres van de buffer anders is dan het adres van de buffer van casper6.

Casper8

Omschrijving

Het programma van casper8 is gelijkaardig aan dit van casper4. Allereerst wordt er nagekeken of er een parameter meegegeven wordt. Indien dit niet het geval is gaat het een error bericht: “Usage: /casper/casper8 YourName” teruggeven. Vervolgens stopt het programma met een errorcode 1.

Indien er wel een parameter meegegeven wordt gaat de “greetUser” functie opgeroepen worden met de meegegeven input. Deze functie maakt een buffer aan van 999 karakters en gaat vervolgens de meegegeven input kopiëren naar deze buffer. Tot slot print het “Hello” gevolgd door de inhoud van de buffer.

Vulnerability

Dit level bevat een stack-based buffer overflow. In de “greetUser” functie wordt er gebruik gemaakt van de “strcpy” functie. Deze kopieer functie gaat de input kopiëren naar de buffer. Het is een onveilige functie omdat er niet wordt gekeken of de input langer is dan 999 karakters, hierdoor kan de aanvaller meer karakters geven als input en zo het return adres en verder te overschrijven Het maakt geen gebruik van stack canaries. Het belangrijkste verschil met casper4 is dat de stack niet uitvoerbaar is.

Exploit beschrijving

Voor dit programma heb ik gebruik gemaakt van een “return-to-libc” attack. Als allereerste ben ik begonnen met de grootte van de buffer op te vragen, gebruik makend van gdb. De buffer heeft een grootte van 1007 bytes.

De exploit is als volgt opgebouwd:

1. Een variabele; genaamd PAAXX, wordt aangemaakt en geëxporteerd als environment variabele. Deze variabele is opgebouwd uit spaties (in plaats van NOP) en vervolgens de string “/bin/xh”
2. Voer het programma uit met de volgende invoer:
    a. 1011 keer de letter “D”. De gehele buffer (1007 bytes) moet overschreven worden samen met de frame pointer (4 bytes).
    b. Vervolgens wordt het return adres overschreven met ”\xa0\x0d\xe5\xb7”. Dit is het adres dat verwijst naar de “system” functie in de C library. Gevolgd door een fake stack, voor deze functie.
    c. Dit wordt gevolgd door “\xd0\x49\xe4\xb7”, dewelke verwijst naar het adres van de “exit” functie in de C library. Het laat het programma normaal stoppen zonder enige errors.
    d. Tot slot wordt “\xc0\xfa\xff\xbf” geprint. Dit is het adres dat verwijst naar de environment variabele PAAXX die “/bin/xh” bevat.

Met deze oplossing moet er dus geen code van de stack uitgevoerd worden. De system functie wordt opgeroepen die “/bin/xh” als invoer neemt.

Mitigation

Deze aanval kan op verschillende manieren worden gemitigate. De implementatie van de code kan aangepast worden door gebruik maakt van de “strcpy“ functie met 3 parameters. Deze extra parameter geeft het maximum aantal karakters dat gekopieerd kan worden naar de buffer. Specifiek in dit programma is dat het maximum gelijk is aan 999 karakters.

Een tweede manier is door gebruik te maken van stack canaries. Deze aanval vereist het volledig overschrijven tot na het return adres. Hierdoor kunnen stack canaries gebruikt worden.

Een derde mitigatie is door gebruik te maken van ASLR, Address Space Layout Randomization. Dit maakt het moeilijker om het correcte adres te specifiëren van de shell code.

Een vierde manier is door de environment variabelen te cleanen in het programma. Maar dit werkt niet in alle gevallen van een return to libc attack. De aanvaller kan “/bin/xh” in de buffer schrijven en hiernaar verwijzen in plaats van naar een environment variabele.

Advanced level: Casper80

Casper80 gaat bijkomend de data in de buffer nakijken naar NOP operaties. Aangezien ik geen van deze operaties gebruik in de buffer kan de vorige exploit gebruikt worden. De enigste aanpassing die dient gemaakt te worden is het aantal karakters die geschreven moeten worden. Er moeten 4 bytes meer geschreven worden in de buffer om het return adres correct te overschrijven. Dit heb ik gedaan door 4 keer het karakter “D” vooraan toe te voegen.

Casper10

Omschrijving

Dit programma is verschillend van alle voorgaande casper programmas. Allereerst wordt er nagekeken of er een parameter meegegeven wordt. Indien dit niet het geval is gaat het een error bericht: “Usage: /casper/casper8 YourName” teruggeven. Vervolgens stopt het programma met een errorcode 1.

Indien er wel een parameter meegegeven wordt gaat de “greetUser” functie opgeroepen worden met de meegegeven input. Deze functie maakt een buffer aan van 999 karakters en gaat vervolgens gebruik maken van een string format functie “sprintf”. Deze functie print in een string. Het gaat de meegegeven string s printen in de “Hello %s!\n” format naar de buf variabele. Vervolgens wordt er onveilig gebruik gemaakt van de printf functie.

Vulnerability

Casper10 heeft een format-string vulnerability dat een data-only attack kan veroorzaken. Zowel stack canaries als een niet uitvoerbare stack zijn ingeschakeld. Het doel van deze exploit is om de waarde van de isAdmin variabele te veranderen naar een positieve waarde.

Exploit beschrijving

Om deze exploit te vinden ben ik begonnen door het programma te laten crashen. Dit heb ik gedaan door middel van de volgende code: /casper/casper10 AAAA$(python -c 'print ".%08x"*15')

Deze code gaat 4 maal een “A” printen gevolgd door 15 maal “.%08x”.Dit geeft de volgende uitvoer: *HelloAAAA.080486d0.bffff7d1.bffff7d1.001b023c.6c654808.41206f6c.2e414141.78383025.3830252e.30252e78.252e7838.2e783830.78383025.3830252e.30252e78!*

Deze code laat een deel van de stack memory zien. Het werkt omdat de printf-functie 15 keer 4 bytes van de stack opvraagt en deze elk toont als een hexadecimaal getal dat bestaat uit 8 cijfers. Dit geeft de bovenstaande output. Door dit te doen kunnen we een deel van de stack reconstrueren.

We zien waar de “AAAA” string zich in memory bevindt, zie naar de onderlijnde bytes. Het eerste probleem hierbij is dat deze string is opgedeeld over 2 hexadecimale waarden. Maar dit is makkelijk op te lossen door voor deze string nog een dummie letter voor te voegen, bijvoorbeeld “D”.

Vervolgens vraag ik via gdb het adres op van de globale variabele isAdmin. Nu heb ik alle informatie om de exploit code op te bouwen. Deze code bestaat uit:

- 1 dummie letter (“D”)
- Adres van de isAdmin variabele
- 6 keer de “%x” variabele om naar het begin van de isAdmin adres te springen
- “%n” om te schrijven naar de variabele

Bovenstaande exploit gaat een waarde schrijven naar de isAdmin variabele. Elke positieve waarde gaat er voor zorgen om in de if statement te komen en vervolgens “/bin/xh” uit te voeren.

Mitigation

Deze aanval kan op verschillende manieren worden gemitigate. De implementatie van de code kan aangepast worden door gebruik maakt van de “printf“ functie met 2 parameters. Deze extra parameter “%s” gaat er voor zorgen dat de string niet geïnterpreteerd worden als een format string.

Een tweede manier is om te checken of de invoer gebruik maakt van speciale karakters zoals “%n” en “%x”. Deze karakters kunnen eruit gefilterd worden.

Een derde manier is om gebruik te maken van ASLR, Address Space Layout Randomization. Dit maakt het moeilijker om het correcte adres te specifiëren van de isAdmin variabele.