|  |
| --- |
| Master in de toegepaste informatica |
| Casper Hacker Wargame |
| [Ondertitel van document] |

|  |
| --- |
| Dries Janse  R0627054  20 december 2019 |

Inhoud

[Overzicht 2](#_Toc25519919)

[Casper4 3](#_Toc25519920)

[Omschrijving 3](#_Toc25519921)

[Vulnerability 3](#_Toc25519922)

[Exploit beschrijving 3](#_Toc25519923)

[Mitigation 3](#_Toc25519924)

[Advanced level: Casper41 3](#_Toc25519925)

[Casper6 4](#_Toc25519926)

[Omschrijving 4](#_Toc25519927)

[Vulnerability 4](#_Toc25519928)

[Exploit beschrijving 4](#_Toc25519929)

[Mitigation 4](#_Toc25519930)

[Advanced level: Casper61 4](#_Toc25519931)

# Overzicht

|  |  |  |
| --- | --- | --- |
| Level | Paswoord | Tijd (uur) |
| casper4 | tnL7L3hY0DVMqEHtrkySQ2M7XwYUNyjE | 12 |
| casper41 | JkZv4uouYkyOP3e1JQwPr4BP6VvcJTWz | 0.5 |
| casper6 | EetVSPHTbn3M1moui3Gg88DbItByyPWz |  |
| casper61 | Sb7ihQ4sGLEQj0O15AG0ke3xgggRsFjF |  |
| Casper8 | q6UYWuXz0iv7b65t1qnHAFBeL4w41YEQ |  |

# Casper4

### Omschrijving

Het casper4 programma vereist één parameter als input. Allereerst wordt er nagekeken of er een parameter meegegeven wordt. Indien dit niet het geval is gaat het een error bericht: “Usage: /casper/casper4 YourName” teruggeve. 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 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 mogelijkheden mogelijk, zo kan de hoeveel NOP operaties en het aantal opvul bytes 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 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 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 functie return 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 aanvaard. 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 functie pointer gïnitialiseerd 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 deze 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 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 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 de return adres.

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 de 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 niet 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

### Vulnerability

### Exploit beschrijving

### Mitigation

## Advanced level: Casper81

# Casper10

### Omschrijving

### Vulnerability

### Exploit beschrijving

### Mitigation