Lær at lave stack pwnyflows

br0ns iDolf Hatler TM
Datalogisk institut, Københavns universitet

Aftenens Program

Aftenens Program

- Lær at bruge gdb

Aftenens Program

- Lær at bruge gdb
- Lær at læse gcc-genereret assembler

Aftenens Program

- Lær at bruge gdb
- Lær at læse gcc-genereret assembler
- Lær at bruge jeres tidligere udviklede shellcode

Stakken og funktionskald

Stakken og funktionskald

- Frames og funktioner

Stakken og funktionskald

- Frames og funktioner
- Argumenter

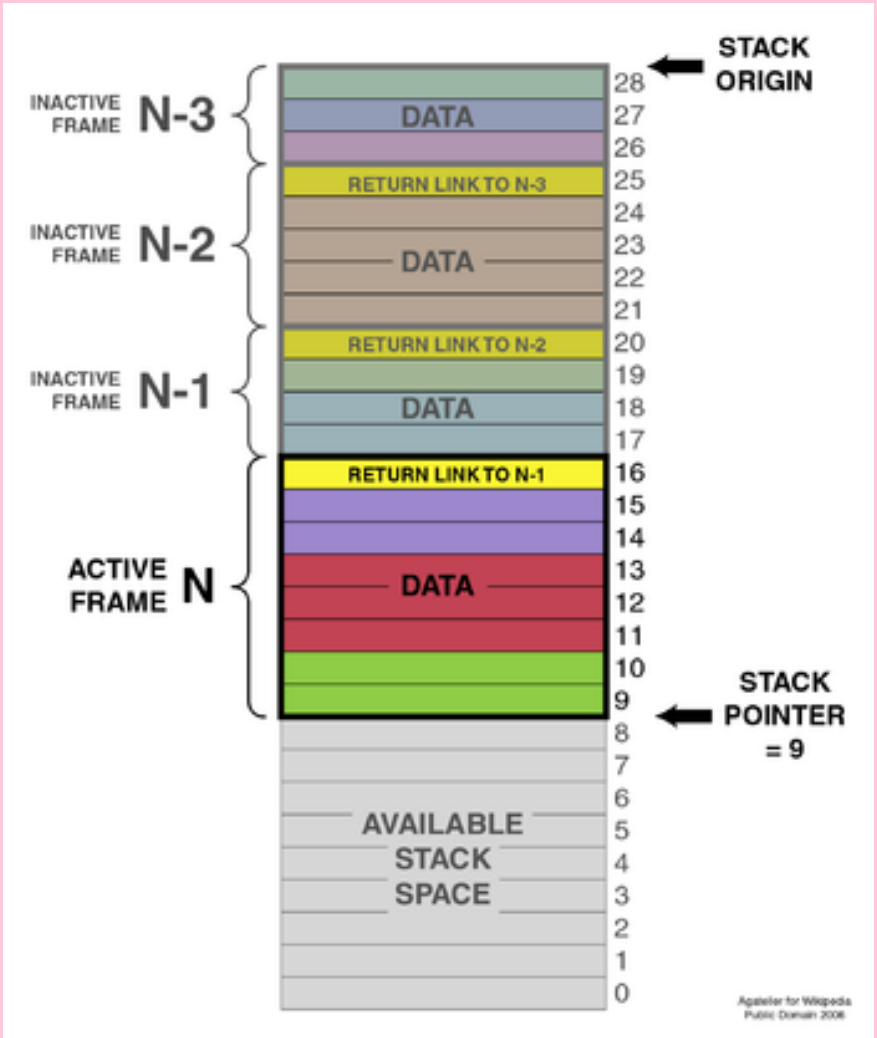
Stakken og funktionskald

- Frames og funktioner
- Argumenter
- Returadresse, gemt basepointer

Stakken og funktionskald

- Frames og funktioner
- Argumenter
- Returadresse, gemt basepointer
- Variable

Stakken med frames, visuelt



Konkret eksempel

Konkret eksempel

```
1 int foo(int a) {
2     int b;
3     b = a;
4     return b;
5 }
```

Konkret eksempel

```
1 int foo(int a) {
2     int b;
3     b = a;
4     return b;
5 }
```

```
1  0x080483b4 <+0>:  push   ebp
2  0x080483b5 <+1>:  mov    ebp,esp
3  0x080483b7 <+3>:  sub    esp,0x10
4  0x080483ba <+6>:  mov    eax,DWORD PTR [ebp+0x8]
5  0x080483bd <+9>:  mov    DWORD PTR [ebp-0x4],eax
6  0x080483c0 <+12>: mov    eax,DWORD PTR [ebp-0x4]
7  0x080483c3 <+15>: leave
8  0x080483c4 <+16>: ret
```

Oversigt over stakken

Dette er sådan stakken (normalt) ser ud. a er antallet af (4-bytes) argumenter, v er antallet af (4-bytes) lokale variable.

Oversigt over stakken

Dette er sådan stakken (normalt) ser ud. a er antallet af (4-bytes) argumenter, v er antallet af (4-bytes) lokale variable.

OBS: v er ikke det egentlig antal lokale variable, idet GCC laver padding og arrays typisk fylder mere end 4 bytes.

Oversigt over stakken

Dette er sådan stakken (normalt) ser ud. a er antallet af (4-bytes) argumenter, v er antallet af (4-bytes) lokale variable.

OBS: v er ikke det egentlig antal lokale variable, idet GCC laver padding og arrays typisk fylder mere end 4 bytes.

`[ebp+0x8+`$4n$`]`	`[esp+`$4v$`+0x8+`$4n$`]`	Argument nummer n (hvor $0 \leq n < a$)
`[ebp+0x4]`	`[esp+`$4v$`+0x4]`	Returadressen
`[ebp]`	`[esp+`$4v$`]`	Den gamle værdi af `ebp`
`[ebp-`$4v + 4n$`]`	`[esp+`$4n$`]`	Lokal variabel position n (hvor $0 \leq n < v$)

Fornuftige gdb-indstillinger

~/.gdbinit (mv dotgdbinit ~/.gdbinit)

```
1 set disassembly-flavor intel
2 set disable-randomization off
3 set pagination off
4 set history filename ~/.gdbhistory
5 set history save
6 set history expansion
```

Start typisk med:

```
1 pwnies@localhost:~ $ gdb --args ./filnavn arg1 arg2
2 start
3 disp/30i $eip
```

Basal GDB

`disas [funktion]`	Disassembler en funktion
`b [funktion]`	Sæt et breakpoint
`c`	Fortsæt indtil næste breakpoint
`run`	Starter kørslen
`x [arg]`	`help x`, eks. `x/20i $esp` Giver 20 instruktioner fra esp
`disp`	Som `x`, men gentages efter hvert step
`i r`	Giver informationer om alle registre
`si`	Træder en instruktion frem
`ni`	Som `si`, men følger ikke funktionskald

Start typisk med:

```
1 pwnies@localhost:~ $ gdb --args ./filnavn arg1 arg2
2 start
3 disp/30i $eip
```

Stackoverflow

Stackoverflow

```
1 int foo() {
2     char buf[100];
3     ...
4 }
```

Legetime resten af aftenen

Legetime resten af aftenen

- Legetime 1: Læs og forstå funktionen `fact` og `fact_helper` - meget gerne ved at steppe igennem dem.
- Legetime 2: Læs og forstå funktionen `my_strncpy`, som gør det samme som `strncpy(3)`.
- Legetime 3: Find et stack overflow i funktionen `log_string` og udnyt det. Det vil være en klar fordel at læse og forstå hvad funktionen gør. Der findes en smart funktion som du gerne vil overskrive returadressen med - du finder den funktion ved at køre kommandoen `readelf -s legetime3 | grep FUNC`.
- Legetime 4: `legetime4` er næsten identisk med `legetime3`. Den smarte funktion fra før er fjernet, til gengæld får du mulighed for at finde din shellcode frem.
- Ekstraopgaver: Løs så mange baner af IO på Smash the stack som du kan: `http://io.smashthestack.org`.