



Financira
Evropska unija
NextGenerationEU



THE RECOVERY
AND RESILIENCE
PLAN

VARNOST PROGRAMOV



Predavanja #8
Matevž Pesek

Teme!

- Vprašanja?
- Ne pozabite na rok! (27. april – prva različica)



Od prejšnjič

- Kaj je TOCTOU
- Kaj je problematika hkratnega dostopa?
- Kaj je t.i. “side channel”?





DANAŠNJE TEME

- Defenzivno programiranje
- Dobre prakse



DEFENZIVNO PROGRAMIRANJE

Principi

Različne perspektive

✂ Napad

- Najdemo ranljivost
 - Ponesreči, namenoma
- Izkoristimo ranljivost
 - Pridobimo/spremenimo podatke (prekoračitev pomnilnika, ROP, TOCTOU, ...)
- Dvignemo privilegije
 - ali pridobimo dostop do sistema (`win()`, dostop do konzole, dostop do admin vmesnika, ...)

🛡 Obramba

- Izogibanje ranljivostim
 - Preverba kode, uporaba specifičnih jezikov, izogibanje vzorcem, rokovanje z izjemami, ...
- Preventiva
 - Uporaba IDE (vzorci), monitoriranje procesov, požarni zid (in vzorci), ...
- Omejevanje privilegijev
 - Virtualizacija, omejitve pri dostopu (človek), omejitve na nivoju procesov, ...

Kako pristopiti?

- Najdemo ranljivost 🧑
 - Ponesreči, namenoma
- Izkoristimo ranljivost 🧑
 - Pridobimo/spremenimo podatke
(prekoračitev pomnilnika, ROP, TOCTOU, ...)
- Dvignemo privilegije 🧑
 - ali pridobimo dostop do sistema
(`win()`, dostop do konzole, dostop do admin vmesnika, ...)

- Zadnjih 8 predavanj ✓
 - Poznamo osnovne principe
 - Poznamo "hipotetične" primere
- Kako omejiti takšne napade?
 1. Okolje (programsko, delovno)
 2. Dobra praksa in izogibanje začasnim rešitvam
 3. **Konsistenca in disciplina**

1. Programsko okolje

- V teoriji:
 - Kvaliteta kode
 - Berljivost kode
 - Predvidljivost delovnega toka

```
-[--->+<]>-. [---->+++++<]>-. -----, ++++++  
+,.---, -[++>---<]>+,.--[->++++<]>+,.-----, ++++++, -  
[---->+<]>++++, ++[->++++<]>,.+++++ +++++, +++[->++  
++++<]>++++, --[->++++<]>--. [----->+<]>,.+++++  
+,.-----, ++++++ +++++, -----, +++++, -----, -[---  
>+<]>--. ---[->++++<]>+,.--. +[->+++++<]>-. ., +++++[-  
>++<]>-. [->+<]>+++. +. -----, --[->+<]>-. --[->+++  
+<]>+,.-----, ++++++, -[---->+<]>++++, +[->++++<]>+  
+,.+++++ +++++, ----, +++++, ++++++ +++++, --[->++  
+<]>-. ., -[--->+<]>-. ---[->++++<]>,.-----, ---, --[---  
>+<]>-. [->+++++<]>-. ., +++++, +. [---->+<]>++++, ---[-  
>++<]>+. ++[->+<]>+.
```

- V praksi:
 - Piši lepo kodo!
 - Piši kodo, ki jo razumeš!
 - Piši dokumentacijo!
 - Predvidi vsa stanja programa!

2. Dobra praksa in vzorci

- Ponovna uporaba
 - Ponovno uporabi kodo, ki je že dobro spisana in pregledana
- Preverba vrednosti
 - Npr. t.i. "null check"
- Preverba dosegljivosti
 - Npr. brisanje datoteke med pošiljanjem
- Avtomat (state machine)
 - Katera stanja obstajajo?
 - Kaj se zgodi, če izpademo iz predvidenih stanj?
- Dokumentacija
 - Napiši kaj je bil namen kode, ne kopiraj kode!
 - Redno posodablaj!

Zakaj je vzdrževanje takšnega načina težko?



- Na istih predpostavkah
 1. Okolje (programsko, delovno)
 2. Dobra praksa in izogibanje začasnim rešitvam
 3. Konsistenca in disciplina

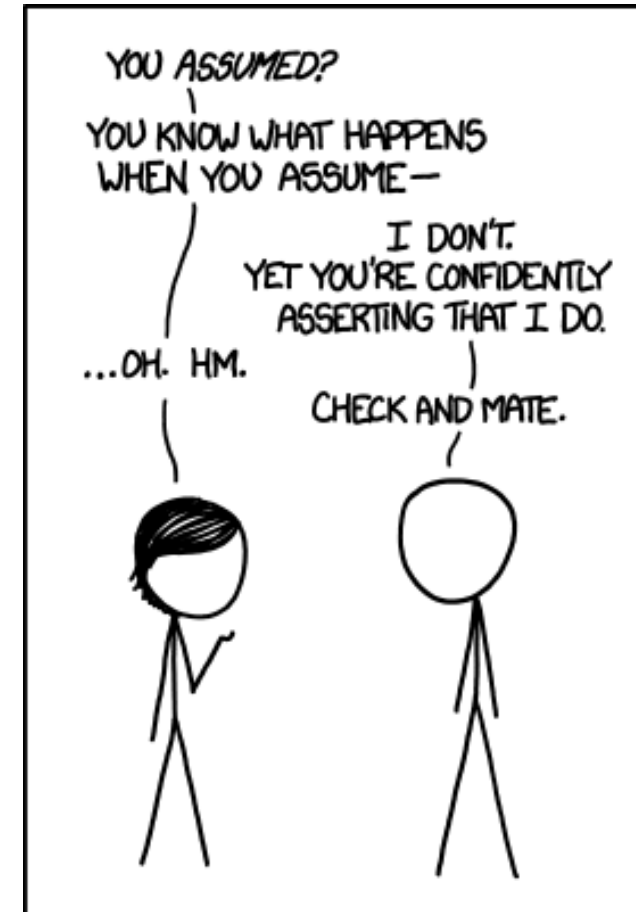
1a. Problemi okolja

- Legacy koda
 - Ni nujno dobra
 - ponovna uporaba je vprašljiva
 - C problem – `gets()`
- Problematika naslavljanja pomnilnika
 - C problem – kako alocirati pomnilnik?
- Problem sanitizacije vhodnih podatkov
 - `mysql_real_escape_string`



1b. Problemi okolja

- Predvidevanja
 - Povprečen uporabnik ne bo izkoristil ranljivosti
 - Slabi vmesniki
 - Slabo preverjanje na zalednem delu
 - Uporabniku lahko povemo kaj je narobe s sistemom
 - `debug=true`
 - Zanašanje na QA
 - Vse začasne rešitve so trajne
 - In začasne rešitve potrebujemo na dnevni ravni!





KAKO POSTOPATI?

Ranljivosti so neizogibne ...

Dobre prakse – v teoriji

- zelo strog QA (a ne zanašaj se!)
- zelo temeljiti testi
- zelo temeljito spremljanje časa delovanja
- zelo strogi standardi kodiranja in razvoja (npr. prepoved določenih vzorcev programiranja)
- dobra splošna kakovost programske opreme
- izvorna kodo, ki je lahko razumljiva
- programska oprema, ki se obnaša predvidljivo

```
function foo(nonEmptyString, naturalInteger) {  
  if (  
    typeof nonEmptyString !== 'string' || // if it's not a  
    string  
    nonEmptyString === '' || // if it's the empty string  
    !Number.isInteger(naturalInteger) || // if it's not an  
    integer  
    naturalInteger < 1 // if it's not a natural integer (1  
    or more)  
  ) {  
    // crash the program  
    // or handle the error here  
    // or throw an exception so some code higher up handles  
    the error  
    // or do anything else your error recovery  
    implementation requires  
  }  
  // code for normal function execution  
}
```


Dobre prakse – v praksi

- **Varno** rokovanje s **števili**
- **Varna hramba** in uporaba tipiziranih spremenljivk
- **Varnost pomnilnika**
- **Varna** interpretacija in **sanitizacija** podatkov
- **Varno** izvajanje programa



Varno rokovanje s števili

- Uporaba **pravih tipov**
 - `size_t` za indekse in dolžino
 - `uintptr_t` za pretvorbo kazalcev
- Pozornost pri **pretvorbi** med celimi števili in celimi nenegativnimi števili (`signed` in `unsigned`)
- Varna uporaba pri **aritmetičnih operacijah**
 - Shranjevanje v tip z večjo velikostjo
 - Shranjevanje v tip z manjšo velikostjo

```
int x = 0x7fffffff;  
long y = x + 1;  
long z = (long)x + 1L;
```

```
long x = 0xffffffffffffffff;  
if (x > INT_MAX) ...  
if (x < INT_MIN) ...  
int y = (int)x;
```

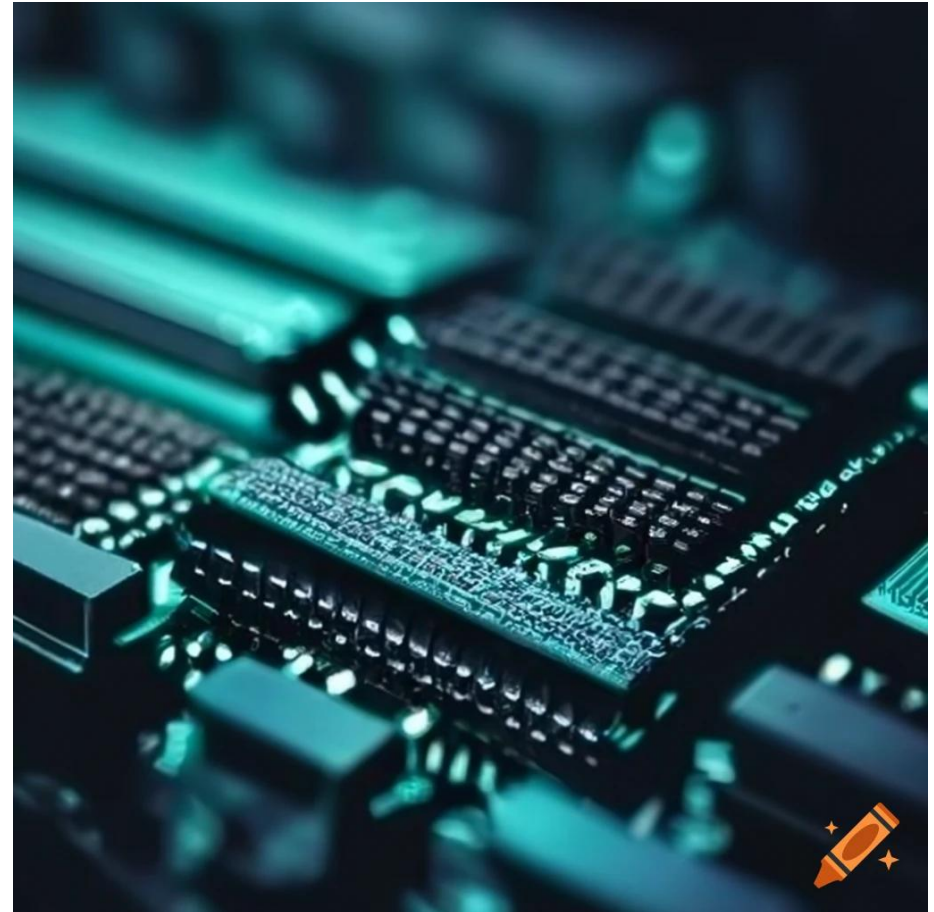
Varna uporaba tipov

- Preverjanje tipov pri dinamično tipiziranih jezikih
- **Nearna pretvorba** tipov

```
float Q_rsqrt(float number) {  
    long i;  
    float x2 = number * 0.5F;  
    float y = number;  
    i = *(long*) &y;  
    i = 0x5f3759df - (i >> 1);  
    y = *(float*) &i;  
    y = y * (1.5F - (x2*y*y));  
    // y = y * (1.5F - (x2*y*y));  
    return y;  
}
```

Varnost pomnilnika

- Preverjanje **prekoračitev**
 - `array[-12]`
 - `array[123456]`
 - `foo.at(x)` namesto `foo[x]`
- **Štetje referenc**, da ne sprostimo kazalcev prekmalu/prepozno
 - `free(ptr); strcpy(ptr, "use after free");`
- **Varna arhitektura** spomina
 - En proizvajalec več porabnikov (SPMC)
 - Npr. rust



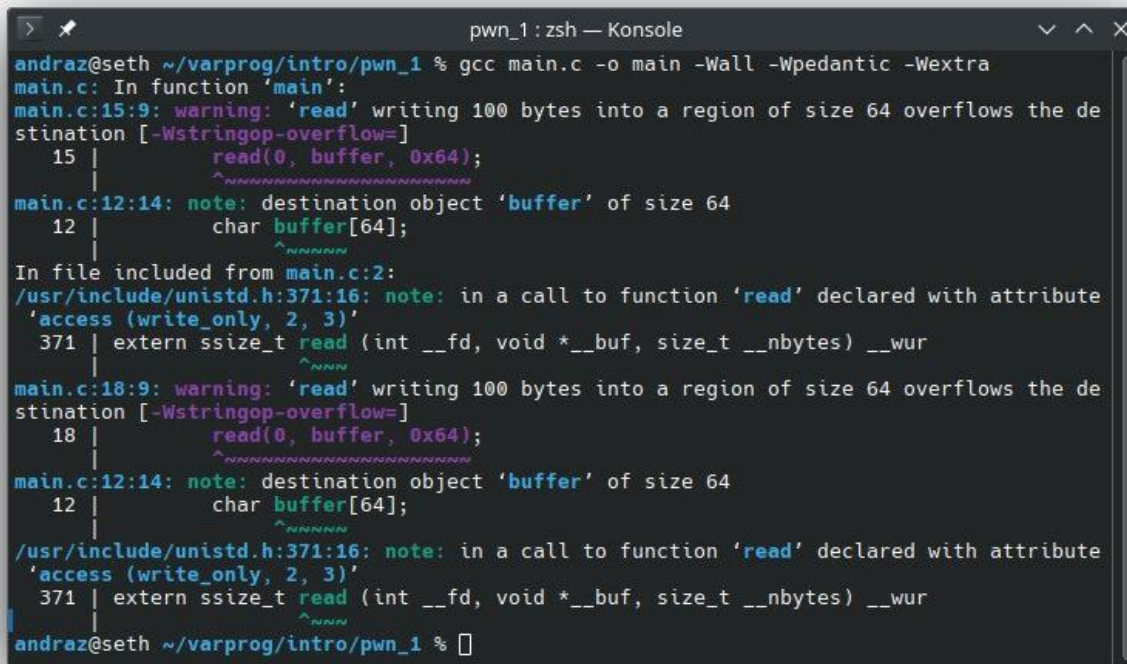
Varni uporabniški vnosi

- Interpretacija vnosa kot koda (injection)
 - Sanitizacija uporabniškega vnosa
 - Zaznavanje in zavračanje vnosa pred izvajanjem
- Kanonizacija poti pred uporabo
 - `/app/data/../../etc/passwd`
 - C#: `Path.GetFullPath()`
 - Java: `File.getPath()`
- Uporaba obstoječih rešitev!

```
@app.route("/uploads/<path:f>")
def uploaded_file(f: str):
    base_path: Path = Path("/app/data")
    f = (base_path / f)
    f = f.resolve().as_posix()
    isadmin = session.get("admin") == 1
    if not f.startswith("/app/data"):
        return "Invalid path", 403
    if "flag.txt" in f and not admin:
        return "Invalid path", 403
    return send_file(f)
```

Varno izvajanje programa

- **Preverjanje napak**
 - Preverjamo vsaj kjer gre lahko kaj narobe ob uporabi
 - Vsako možno napako obdelamo posebej
- Predvidevamo, da bo naša **koda ponovno uporabljena** (pogosto s strani drugih)
- **Pregledamo opozorila in napake**
 - `gcc -Wall -Wextra -Wpedantic`
 - `gcc -fsanitize`
 - Statična analiza kode



```
pwn_1: zsh — Konsole
andraz@seth ~/varprog/intro/pwn_1 % gcc main.c -o main -Wall -Wpedantic -Wextra
main.c: In function 'main':
main.c:15:9: warning: 'read' writing 100 bytes into a region of size 64 overflows the de
stination [-Wstringop-overflow=]
   15 |         read(0, buffer, 0x64);
      |         ^~
main.c:12:14: note: destination object 'buffer' of size 64
   12 |         char buffer[64];
      |         ^~
In file included from main.c:2:
/usr/include/unistd.h:371:16: note: in a call to function 'read' declared with attribute
'access(write_only, 2, 3)'
   371 | extern ssize_t read (int __fd, void *__buf, size_t __nbytes) __wur
      | 
main.c:18:9: warning: 'read' writing 100 bytes into a region of size 64 overflows the de
stination [-Wstringop-overflow=]
   18 |         read(0, buffer, 0x64);
      |         ^~
main.c:12:14: note: destination object 'buffer' of size 64
   12 |         char buffer[64];
      |         ^~
/usr/include/unistd.h:371:16: note: in a call to function 'read' declared with attribute
'access(write_only, 2, 3)'
   371 | extern ssize_t read (int __fd, void *__buf, size_t __nbytes) __wur
      | 
andraz@seth ~/varprog/intro/pwn_1 %
```


PRIMERI



Nevarna hramba tipov

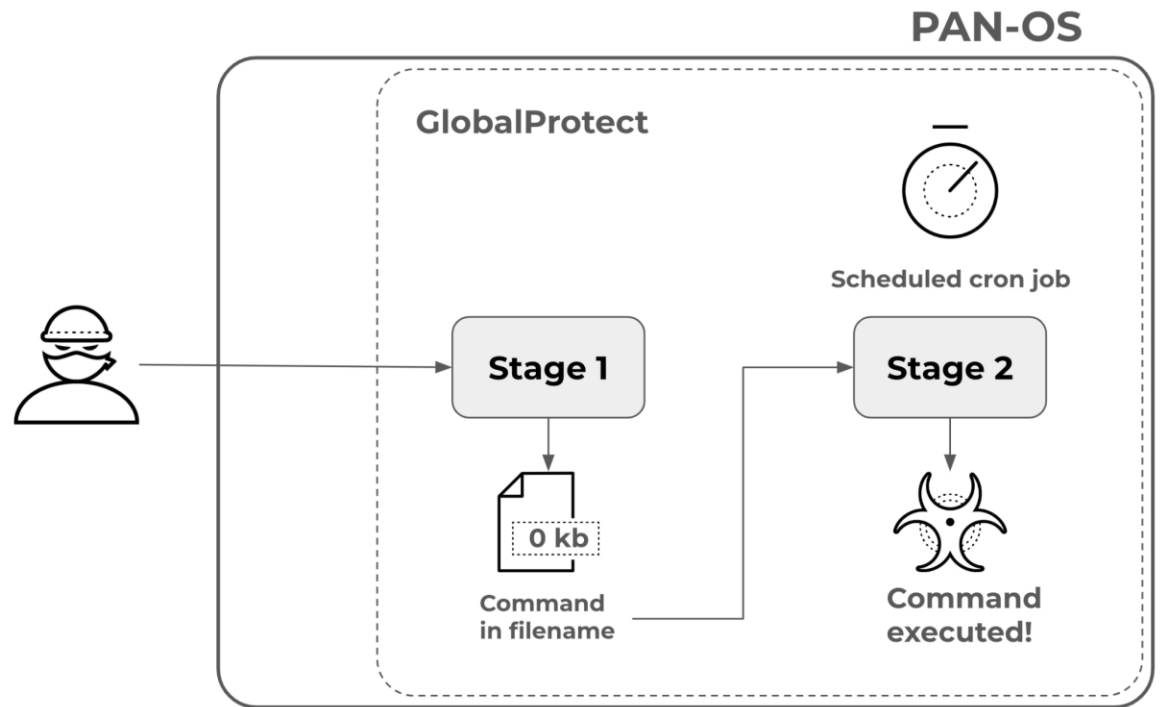
- SAP (CVE-2022-41203)
- Pogosto hranimo podatke med uporabo
- Serializiramo lahko v različnih formatih
 - Pri interpretiranih jezikih pogosto obstajajo možnosti za serializacijo neposredno v spremenljivko ali razred jezika
 - Med podatke lahko nato tudi skrijemo funkcije

```
import dill
file = open('test.pkl', 'wb')
func = lambda: print('Hello')
dill.dump(func, file)
```

```
import dill
file = open('test.pkl', 'rb')
func = dill.load(file)
func() # -> Hello
```

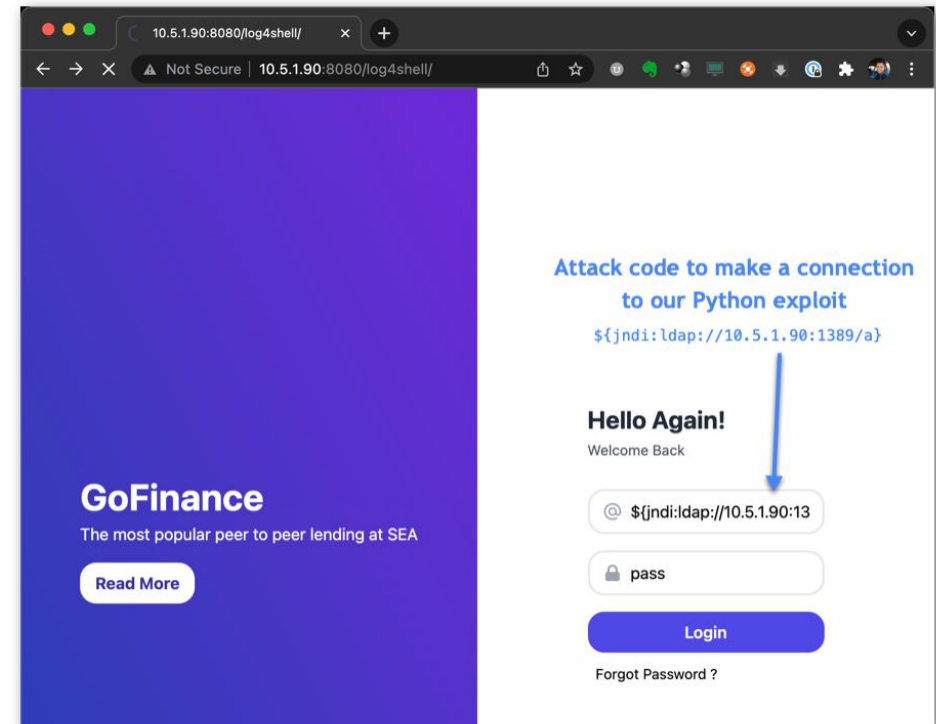
Sanitizacija vnosa

- Palo Alto (CVE-2024-3400)
- Nikoli ne zaupaj uporabniškem vnosu!
 - Branje in razbiranje dnevniških zapisov
 - Med branjem se zlonamerna koda tudi izvede!



Sanitizacija vnosa

- Log4J
 - CVE-2021-44228
- Koraki:
 - Shranjevanje uporabniških vnosov v dnevniške zapise
 - Med shranjevanjem razbiranje dnevniških zapisov (parsing)
 - Izkoristimo ponovno razbiranje, da podtaknemo JNDI povezavo na zunanjo datoteko
 - Datoteka se prenese in izvede na strežniku



+



o



•



HVALA

Vaje

- Primeri defenzivnega programiranja