Binarna eksploatacija

Sadržaj

- ELF struktura
- Jednostavniji zadaci
  - Var overwrite
  - Integer overflow
  - Ret2system
- Stack
  - Ret2dlresolve
  - Ret2sigreturn/SROP
- Heap
  - Struktura
  - Funkcionalnost
  - Ranjivosti

Var overwrite

- Zadatak “Tajne korneta” (s Hacknite 2020.) - probajte sami

Integer overflow

- Zadatak “Jednadžba prijevoza”
- Zadatak “Tajni chat”

Ret2win

- Zadatak “Utrka 1/Utrka 2”
  - Probajte Utrka 1 bez da dobijete shell
- Zadatak “Tajne čokolade”

Ret2win - greške

- Alignment → ponekad jednostavno treba dodati još jedan “ret”
- Skok na krivu instrukciju/krivi offset
- PatriotCTF – “bookshelf” – moj krivi exploit

Format string

- Zadatak “printshop” (PatriotCtf) – (format string write – zajedno proučiti rješenje)

“Ret2plt”

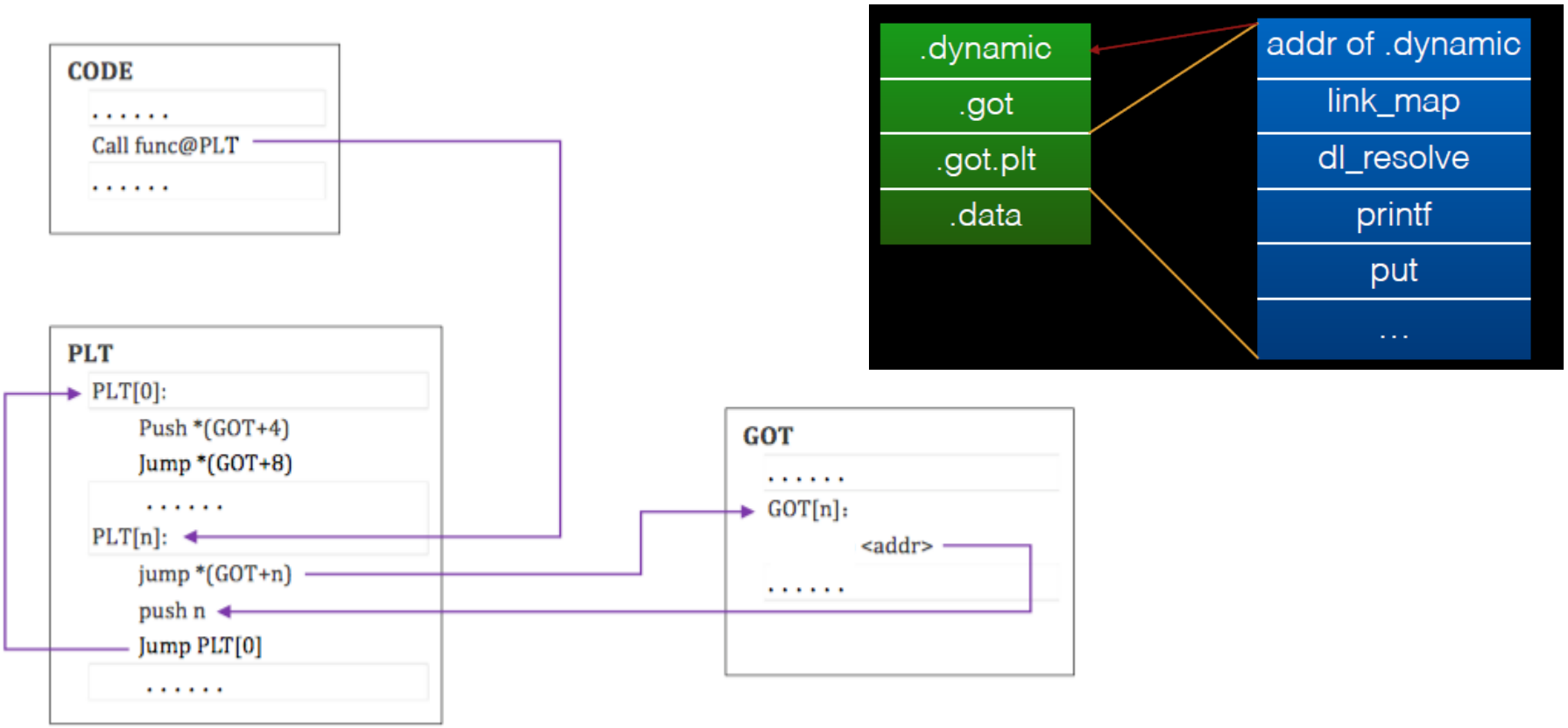
- ECSC 2023 “knife party”

Stack

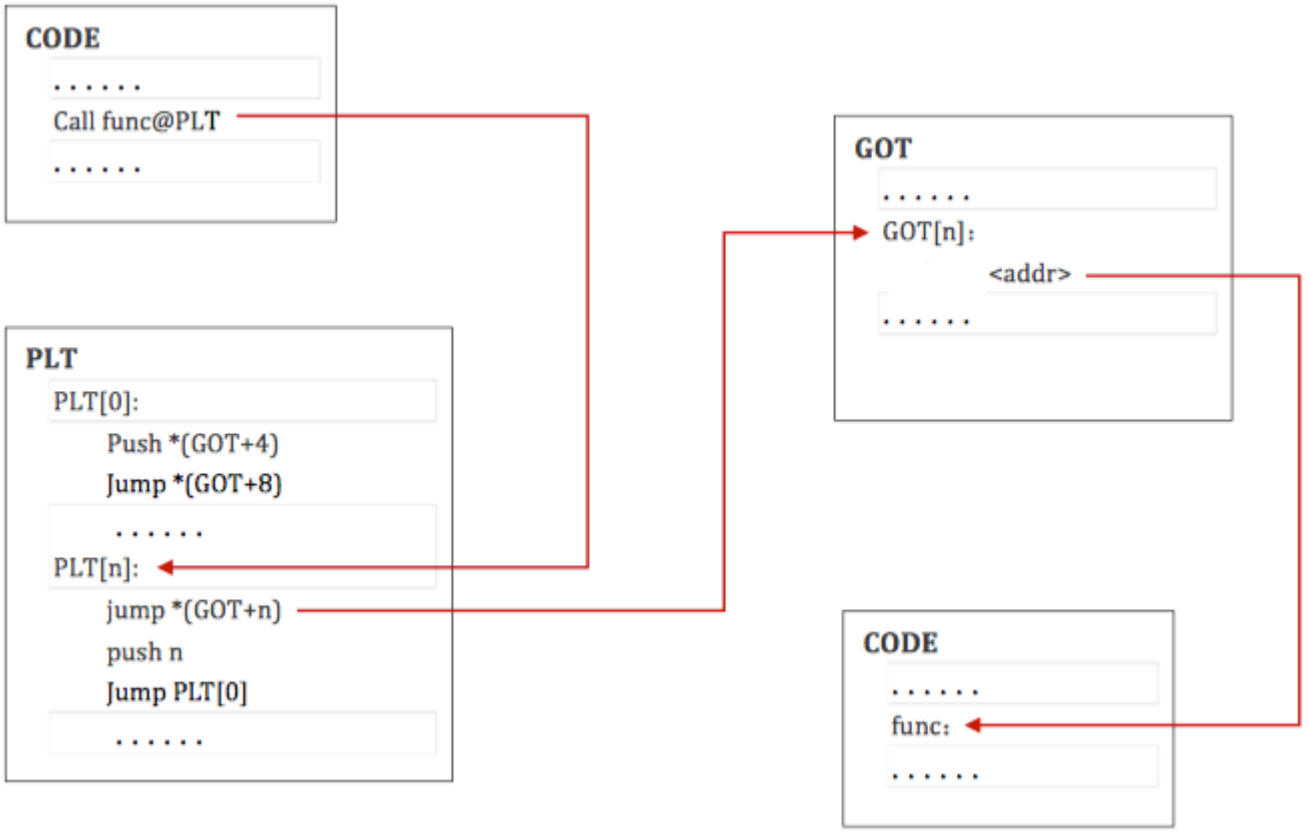
Ret2dlresolve

- Ideja je iskoristiti funkciju za symbol resolving
  - samo kada se koristi lazy binding (adresa funkcije se pronađe tek kada se pozove)
  - dl_runtime_resolve je funkcija koja pronalazi adresu funkcije dinamički loadanih shared objecta (npr. libc)
  - krivotvore se različiti argumenti (sekcije, objekti dinamičkog linkera itd...)
- Prednost jest ta što nije potreban libc leak
  - resolver sam pronađe adresu
  - koristi se u slučajevima kada program ne sadrži funkcije s outputom (printf, puts...) kako bi se dobio leak
- Nedostatak jest potreba za velikim prostorom za pisanje (potreban veliki overflow ili .bss itd...)
- Rezultat jest resolvanje i pozivanje proizvoljne funkcije (u većini slučajeva) sa zadanim argumentima

Normalan flow poziva plt funkcije 1.



Normalan flow poziva plt funkcije 2.



Ključne interne funkcije

- _dl_runtime_resolve(link_map_obj, reloc_index)
  - funkcija za pripremanje konteksta
  - interno poziva _dl_fixup
- _dl_fixup
  - iterira po svim shared objectima (link_mapi) koji su uključeni u proces te na pronalasku prve funkcije s ispravnim imenom i verzijom vraća tu adresu (tj. base adresa SO (shared object) + offset funkcije) te ju poziva

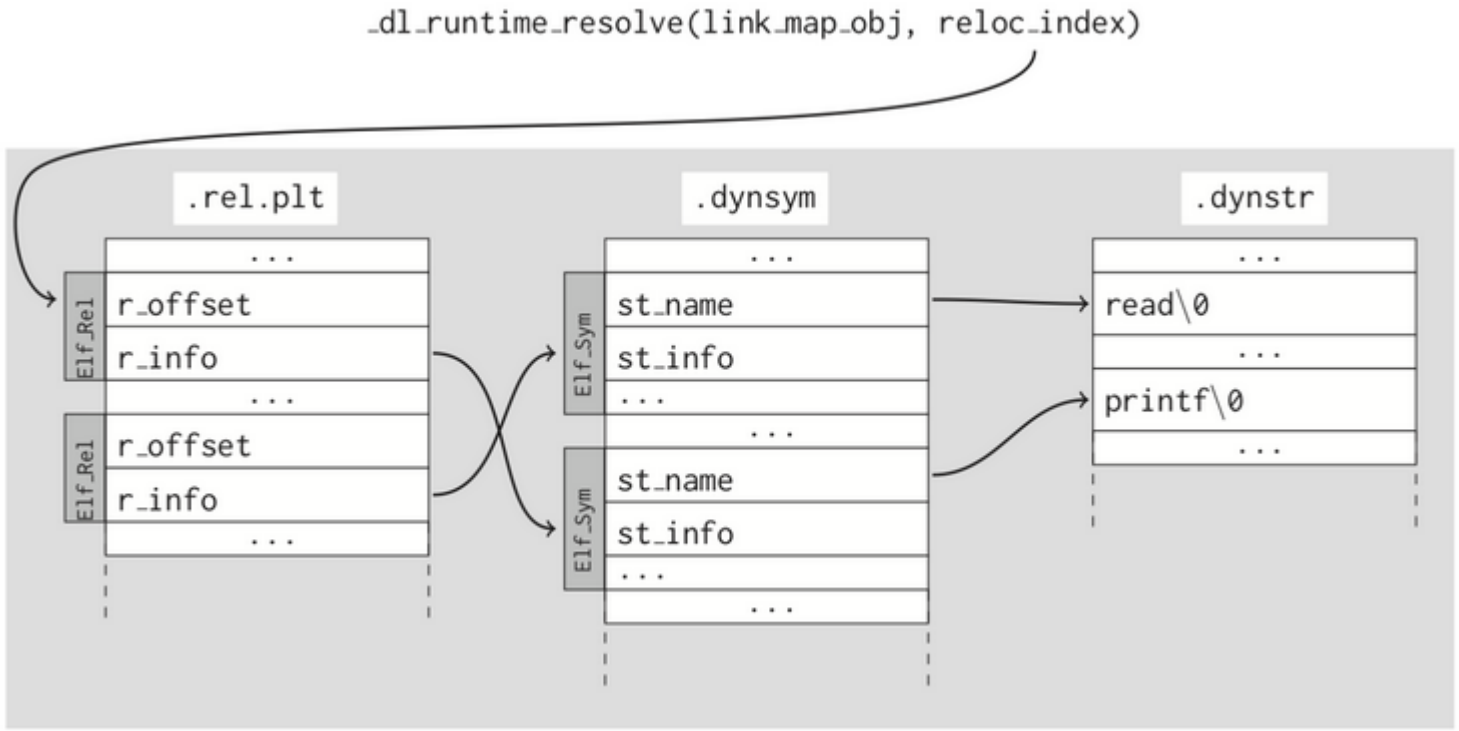
Link map

- Objekt stvoren tokom runtime-a
  - Lista s čvorovima
  - Svaki čvor predstavlja jednu datoteku
- Sadrži informacije o svim datotekama uključenim u proces
  - Offseti sekcija, base adresa itd...
  - struktura link_mape

Reloc_index

- Index unutar rel.plt/rela.plt sekcije
- Preko entrija unutar te sekcije dobiva se adresa koju je potrebno izmijeniti te naziv i verzija funkcije koje je potrebno resolvati (ne direktno već kroz strukture drugih sekcija)
  - konkretno ostale bitne sekcije su .dynsym, .gnu.version i .dynstr
  - Struktura relokacijskih tablica

```
Relocation section '.rela.plt' at offset 0x4e8 contains 1 entry:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000404018  000200000007 R_X86_64_JUMP_SLO 0000000000000000 printf@GLIBC_2.2.5 + 0
```


_dl_runtime_resolve(link_map_obj, reloc_index)
.rel.plt
.dynsym
.dynstr
Elf_Rel
r_offset
r_info
Elf_Rel
r_offset
r_info
Elf_Sym
st_name
st_info
Elf_Sym
st_name
st_info
read\0
printf\0

Vrste

- 3 tipa ret2dlresolve napada:

1) Overwrite na .dynamic sekciji
   - U slučaju kada je NO RELRO moguće je promijeniti adresu .dynstr tablice unutar .dynamic sekcije na krivotvoreni .dynstr koji na ispravnom offsetu ima izmijenjen string u naziv željene funkcije (npr. “printf\0” se izmijeni u “system\0”)
2) Krivotvorenje sekcije
   - krivotvorenje entrija unutar svih potrebnih sekcija (.rela.plt, .dynsym, .dynstr)
3) Krivotvorenje link mape
   - koristi se kada .gnu.version offset nije unutar readable memorije ili ne sadrži vrijednosti 0/1 za halfword (short)
   - teži za izvest zbog potrebe za velikim prostorom za pisanje
   - Proučiti više na https://web.archive.org/web/20240617230726/https://ctf-wiki.org/pwn/linux/user-mode/stackoverflow/x86/advanced-rop/ret2dlresolve/ (uz neku ekstenziju za prevođenje)

2. tip

- Ideja jest krivotvoriti retke sekcija koji se međusobno referiraju (koristi se originalni link_map)
- Koraci:

  1) Zapišemo lažne strukture (lažni rel entry, lažni dynsim entry i lažni dynstr entry)

  2) Pushamo rel index (lažiramo onaj n iz PLT-a) tako da pointa na našu krivotvoreni rel entry

  3) Skočimo na dio plt stuba gdje se pusha link mapa i poziva _dl_runtime_resolve (s ispravnom, nekrivotvorenom link_mapom)

Napomena

- Kako bi _dl_fixup normalno funkcionirao .gnu.version redak mora sadržavati vrijednost 0 ili 1 za short (halfword)
  - Ponekad to neće biti moguće, zbog čega je jedina preostala opcija 3. tip ret2dlresolve napada

Automatizacija kroz pwntools

- Pwntools automatizira proces stvaranja payloada za 2. tip ret2dlresolve napada s funkcijom Ret2dlresolvePayload, te postavljanja konteksta na stacku kroz rop.ret2dlresolve(payload)

```
>>> context.binary = elf = ELF(pwnlib.data.elf.ret2dlresolve.get('amd64'))
>>> rop = ROP(elf)
>>> dlresolve = Ret2dlresolvePayload(elf, symbol="system", args=["echo pwned"])
>>> rop.read(0, dlresolve.data_addr) # do not forget this step, but use whatever function you like
>>> rop.ret2dlresolve(dlresolve)
>>> raw_rop = rop.chain()
>>> print(rop.dump())
0x0000:         0x400593 pop rdi; ret
0x0008:              0x0 [arg0] rdi = 0
0x0010:         0x400591 pop rsi; pop r15; ret
0x0018:         0x601e00 [arg1] rsi = 6299136
0x0020:      b'iaaajaaa' <pad r15>
0x0028:         0x4003f0 read
0x0030:         0x400593 pop rdi; ret
0x0038:         0x601e48 [arg0] rdi = 6299208
0x0040:         0x4003e0 [plt_init] system
0x0048:          0x15670 [dlresolve index]
>>> p = elf.process()
>>> p.sendline(fit({64+context.bytes: raw_rop, 200: dlresolve.payload}))
>>> if dlresolve.unreliable:
...     p.poll(True) == -signal.SIGSEGV
... else:
...     p.recvline() == b'pwned\n'
True
```

Unos payloada na adresu kroz stdin

Pripremanje argumenta za system(“echo pwned”)

_dl_resolve prima argumente na stacku (kao i na x86), plt_init pusha adresu link_mape, a dlresolve_index je index krivotvorenog retka .rel.plt ili .rela.plt sekcija

Zadaci/primjeri

Ret2sigreturn

- Ret2sigreturn ili još zvan SROP
- Ideja je iskoristiti instrukciju syscall/funkciju sigreturn koja služi za vraćanje konteksta sa stacka
- Prednosti
  - Koristi se kada program ima manjak ROP gadgeta
- Nedostatci
  - Potreban veliki prostor za pisanje
  - Potrebno znati adresu sigreturna/instrukcije syscall (ako imamo samo syscall instrikciju, rax mora biti postavljan na vrijednost 15 - pogledati ovdje)

Sigreturn - legitimno korištenje

- Signal → poruka prema procesu (npr. “kill”)
- Kad program primi signal, izvršavanje se signal-handler
- Ako signal handler nije ubio sam proces, mora moći nastaviti program
- To je implementirano tako da se stanje svih registra spremi na stack
- Funkcija “sigreturn” (to je zapravo syscall oznake 15) vraća vrijednosti sa stacka u registar

0x00	rt_sigreturn	uc_flags
0x11	&uc	uc_stack.ss_sp
0x20	uc_stack.ss_flags	uc_stack.ss_size
0x30	r8	r9
0x40	r10	r11
0x50	r12	r13
0x60	r14	r15
0x70	rdi = &“/bin/sh”	rsi
0x80	rbp	rbx
0x90	rdx	rax = 59 (execve)
0xA0	rcx	rsp
0xB0	rip = &syscall	eflags
0xC0	cs / gs / fs	err
0xD0	trapno	oldmask (unused)
0xE0	cr2 (segfault addr)	&fpstate
0xF0	__reserved	sigmask

Primjer Signal Frame-a prije poziva syscall s execve(“/bin/sh”)

Rezultat je vraćanje konteksta registra, međuostalom rip s adresom syscall-a te rax i rdi s vrijednostima 59 i “/bin/sh”

Zadatak

- SIG but not INT – infobip CTF 2022.
- CSAW19 SROP

Heap

Predgovor

- Postoje različite libc implementacije
  - Glibc, musl...
- Time postoje i različite implementacije heap funkcija
  - ptmalloc2, mallocng, phkmalloc...
- Na radionici radimo s glibc implementacijom koja koristi ptmalloc2
- Neki implementacijski detalji u ovoj prezentaciji možda pojednostavljeni (nisu spomenuti neki edge-caseovi)
- Za ozbiljno bavljenje heap exploitationom – proučavati izvorni kod ptmalloc2 i igrati se, čitati o novim tehnikama, writeupe itd.

Heap

- Ako nema mjesta na stacku ili želimo da neki objekt preživi return funkcije
- Funkcije malloc/free
  - I calloc, realloc ...

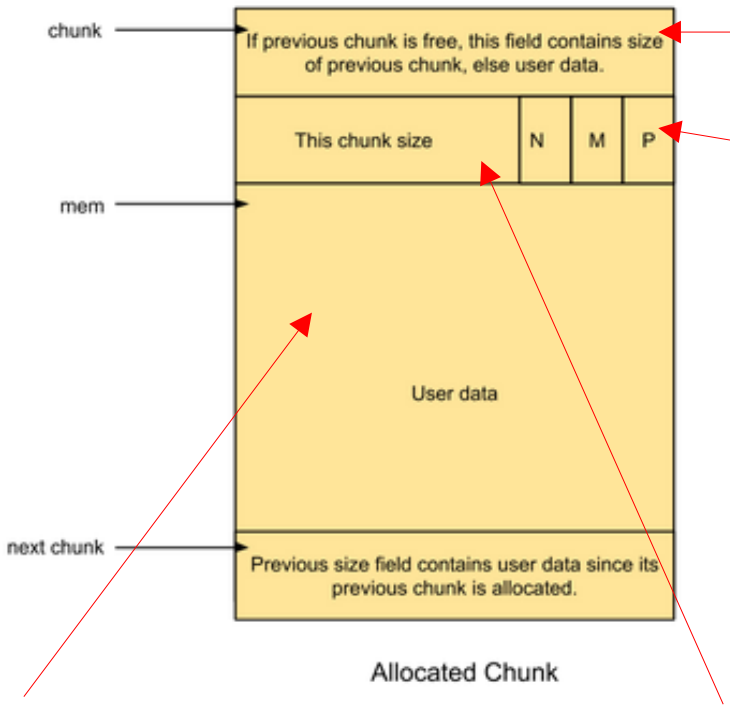
Arena

- Kad program tek krene – nema heapa
- Prvi put kad se pozove malloc, stvori se heap i main arena
- Main arena – struktura u libcu, koja pointa na heap i sadrži neke metapodatke heapa i pointere na razne slobodne dijelove heapa
- Može postojati više arena (u drugim threadovima) – van scopea predavanja

Struktura chunkova

- Chunk jest naziv za dio heap-a koji je dodijeljen korisniku (rezultat npr. malloc(10))
- Chunk (odnosno taj dio memorije) ima dvije strukture ovisno o stanju u kojemu se nalazi (allocated/freed)
- U oba stanja chunk sadrži headere prev_size i size nakon čega slijedi user data ili pointeri na chunkove u binovima (više o tome kasnije)

Allocated stanje



Veličina prijašnjeg chunka (4 bajta na x86, 8 bajta na x64)

In-use bit koji pokazuje u kojem stanju jest prijašnji chunk (allocated/freed)
Sva 3 bita (N,M,P - N i M manje bitni) se mogu iskoristiti zbog alignmenta (veličina chunka je uvijek višekratnik word_size*2, 8 na x32 i 16 na x64) -> niži bitovi slobodni

Koristi ga program/korisnik (alignment 2*word_size)

Veličina chunka, user data + 2*word_size

Freed stanje

Kada je chunk oslobođen,
dio prostora koji su zauzimali podaci (“user data”)
sad ima neke chunk metapodatke
(pointere na sljedeći slobodni chunk i slično) -
jako puno napada proizlazi iz te činjenice

Veličina chunka

- Veličina je uvijek višekratnik 2*word_size jer chunk uvijek ima prev_size i size (dakle word_size + word_size), a user data je aligned s 2*word_size
  - Ako program zatraži veličinu 10 s malloc(10) vratit će mu se chunk s veličinom 32 gdje je 16 veličina user data, a 16 headera prev_size i size
- Minimalna vrijednost za size chunka jest 32 (0x20)

Binovi

- Optimizacijski mehanizmi koji omogućavaju alokatoru brži rad i manju segmentiranost heapa
- Bin jest skup listi koji sadrže chunkove u freed stanju
- Razlikujemo:
  - Tcache (od glibc 2.26)
  - Fastbin
  - Unsorted bin
  - Small bin
  - Large bin
- Pointer na Tcache bin je u području “TLS” (Thread-local storage), pointeri na ostale su u areni (u libc-u)

Cilj binova

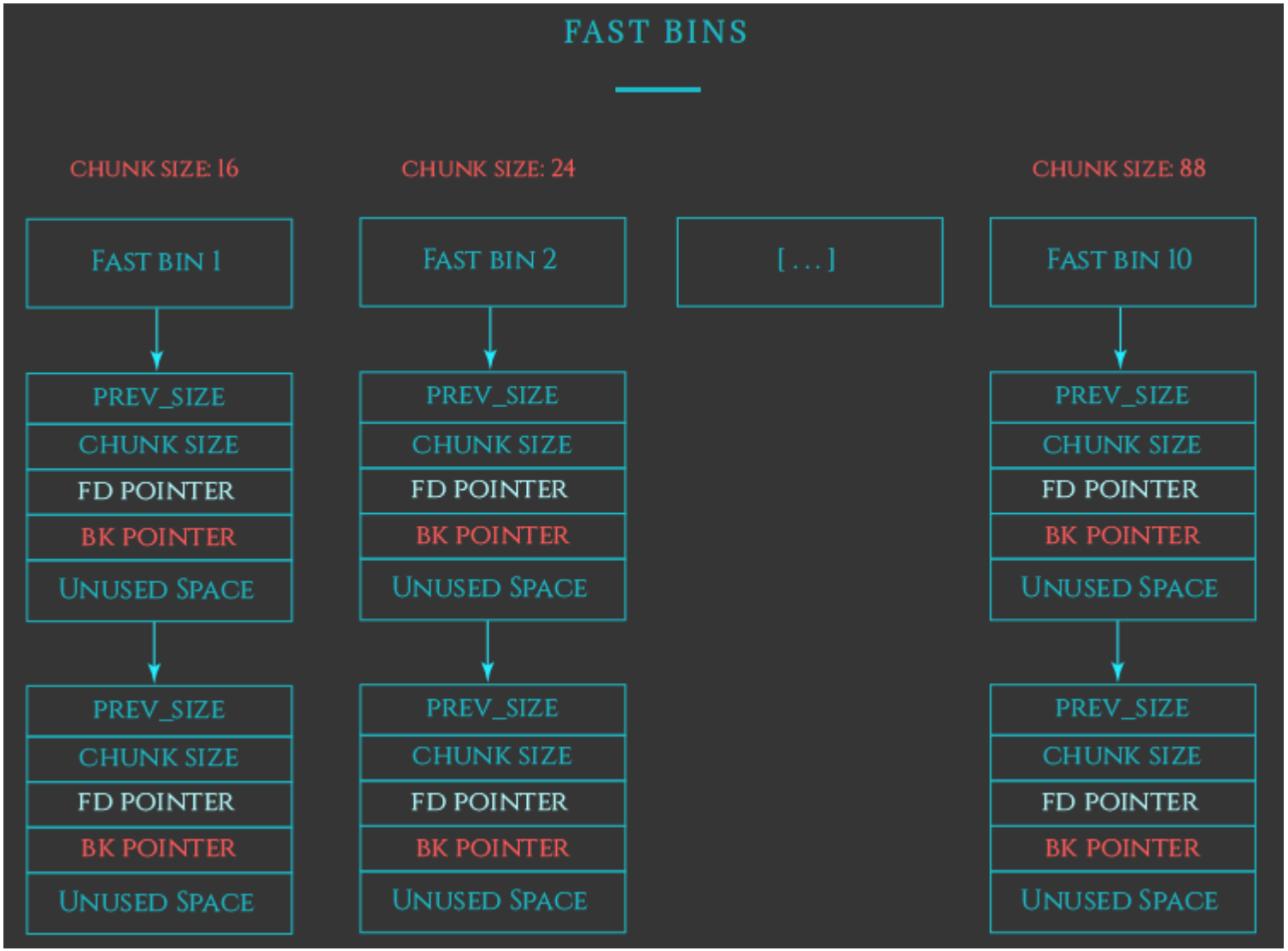
- Ideja je da kada korisnik oslobodi chunk, on bude stavljen unutar liste
  - Jer se nalazi u listi lakše je ponovno pronaći chunk takve veličine
    (recikliranje -> ušteda memorije) te olakšava konsolidiranje s ostalim chunkovima (manja segmentacija)

Razlike u binovima

- Svaki bin sadrži raspon veličina chunkova koje može sadržavati
- Zapravo postoji dosta preklapanja (tcache, fastbin i small bin mogu imati iste veličine, a unsorted prima sve veličine tako da se preklapa sa svima)
  - Kasnije o samim funkcionalnostima i interakcijama binova u sveukupnoj slici

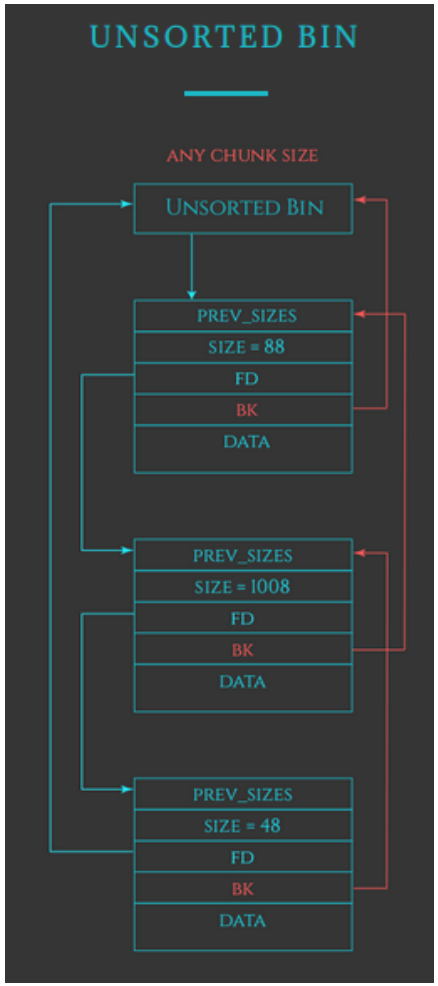
Fastbin

- Dio arene
- 10 listi s rasponom veličina (s headerom) 32-128 bajta
- Svaka lista može imati beskonačno chunkova
- Jednostruko povezane liste i LIFO način rada


FAST BINS
CHUNK SIZE: 16
CHUNK SIZE: 24
CHUNK SIZE: 88
FAST BIN 1
FAST BIN 2
[. . .]
FAST BIN 10
PREV_SIZE
CHUNK SIZE
FD POINTER
BK POINTER
UNUSED SPACE
PREV_SIZE
CHUNK SIZE
FD POINTER
BK POINTER
UNUSED SPACE
PREV_SIZE
CHUNK SIZE
FD POINTER
BK POINTER
UNUSED SPACE
PREV_SIZE
CHUNK SIZE
FD POINTER
BK POINTER
UNUSED SPACE
PREV_SIZE
CHUNK SIZE
FD POINTER
BK POINTER
UNUSED SPACE

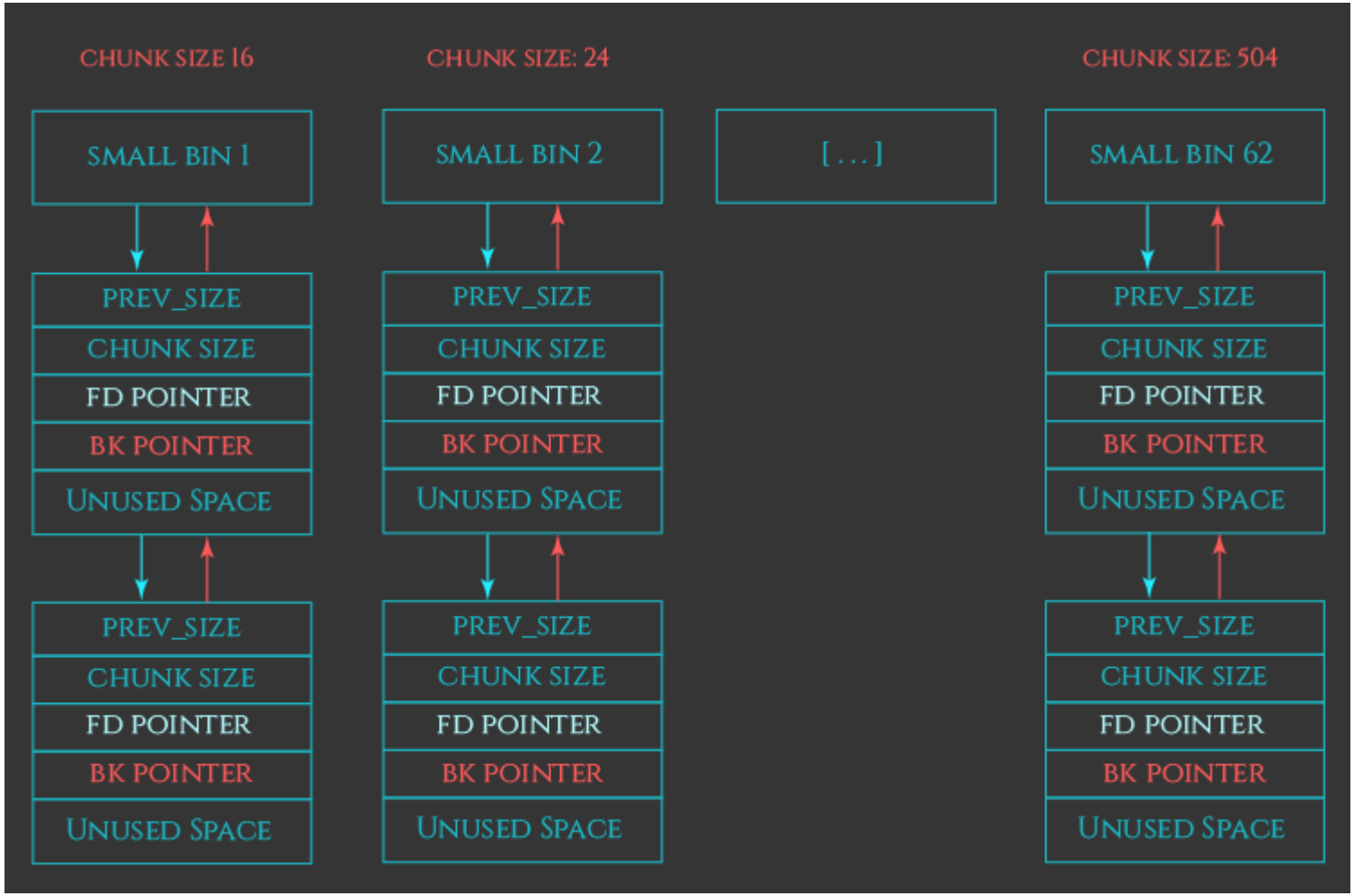
Unsorted bin

- Jedna lista koja sadrži chunkove svih veličina
- FIFO
- Dvostruko povezana lista (koristi se i bk pointer)
  - fd pointer arene pokazuje na prvi chunk
  - bk pointer arene pokazuje na zadnji chunk
  - fd pointer zadnjeg chunka pokazuje na adresu arene
    - A main arena je u libc-u, dakle ako uspijemo to pročitati imamo libc leak
  - bk pointer prvog chunka pokazuje na adresu arene


UNSORTED BIN
ANY CHUNK SIZE
UNSORTED BIN
PREV_SIZES
SIZE = 88
FD
BK
DATA
PREV_SIZES
SIZE = 1008
FD
BK
DATA
PREV_SIZES
SIZE = 48
FD
BK
DATA

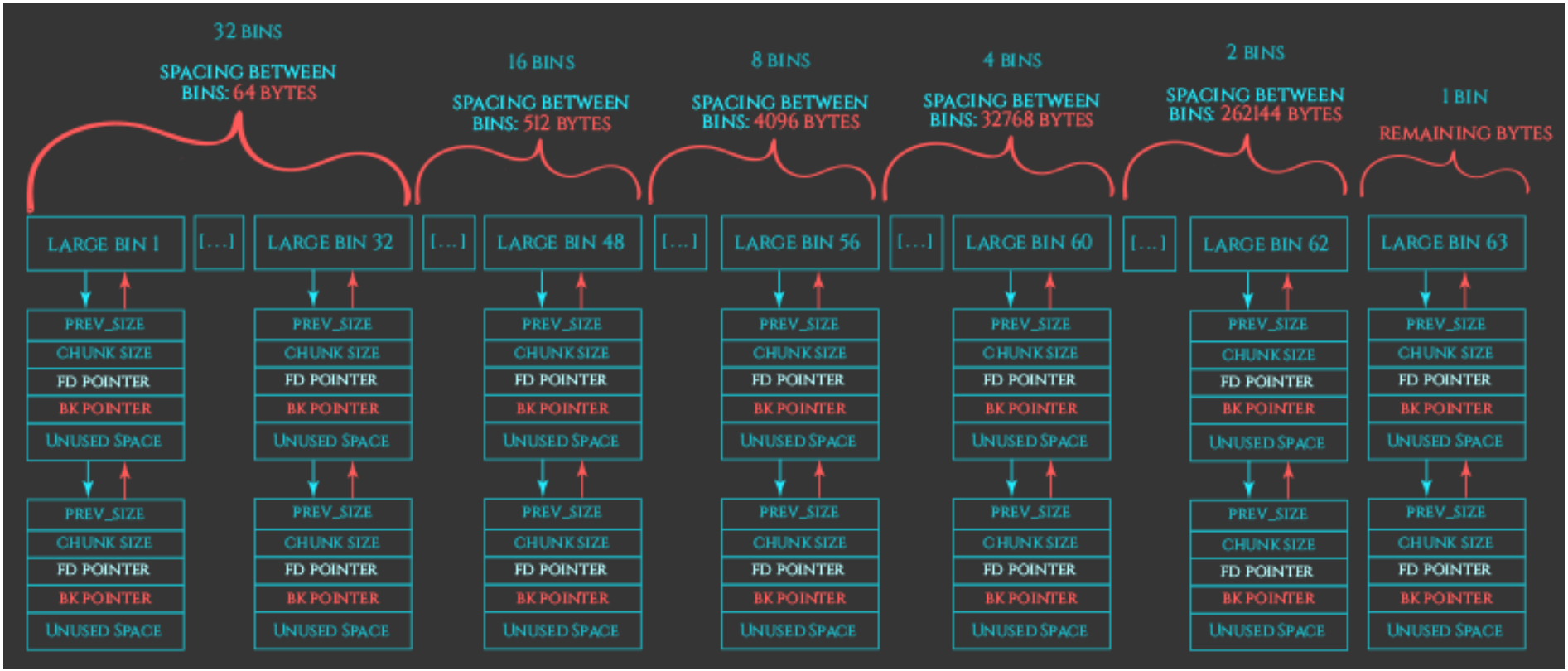
Small bin

- 62 liste
- Raspon veličina (s headerom) 32-1040 bajta
- Dvostruko povezane liste
- FIFO način rada


CHUNK SIZE 16
CHUNK SIZE: 24
CHUNK SIZE: 504
SMALL BIN 1
SMALL BIN 2
[. . .]
SMALL BIN 62
PREV_SIZE
CHUNK SIZE
FD POINTER
BK POINTER
UNUSED SPACE
PREV_SIZE
CHUNK SIZE
FD POINTER
BK POINTER
UNUSED SPACE
PREV_SIZE
CHUNK SIZE
FD POINTER
BK POINTER
UNUSED SPACE
PREV_SIZE
CHUNK SIZE
FD POINTER
BK POINTER
UNUSED SPACE
PREV_SIZE
CHUNK SIZE
FD POINTER
BK POINTER
UNUSED SPACE

Large bin

- 63 liste - raspon veličina (s headerom) 1056 - nadalje
- Koristi uz fd i bk, fd_nextsize i bk_nextsize
- FIFO


32 BINS
SPACING BETWEEN BINS: 64 BYTES
16 BINS
SPACING BETWEEN BINS: 512 BYTES
8 BINS
SPACING BETWEEN BINS: 4096 BYTES
4 BINS
SPACING BETWEEN BINS: 32768 BYTES
2 BINS
SPACING BETWEEN BINS: 262144 BYTES
1 BIN
REMAINING BYTES
LARGE BIN 1
[...]
LARGE BIN 32
[...]
LARGE BIN 48
[...]
LARGE BIN 56
[...]
LARGE BIN 60
[...]
LARGE BIN 62
LARGE BIN 63
PREV_SIZE
CHUNK SIZE
FD POINTER
BK POINTER
UNUSED SPACE

Tcache

- Pointeri na tcache liste se nalaze unutar TLS-a (ne u Areni)
- Per-thread struktura kako threadovi nebi morali koristiti mutexe
- Pokriva veličine (s headerom) 32 - 1040 bajta
- Tcache (kao i ostali binovi) jest zapravo naziv za grupu listi
  - Konkretno, tcache sadrži 64 liste
  - Svaka tcache lista može sadržavati najviše 7 chunkova
  - Prva lista sadrži chunkove veličine 32
  - Svaka sljedeća lista sadrži za 16 bajta veće chunkove

Tcache

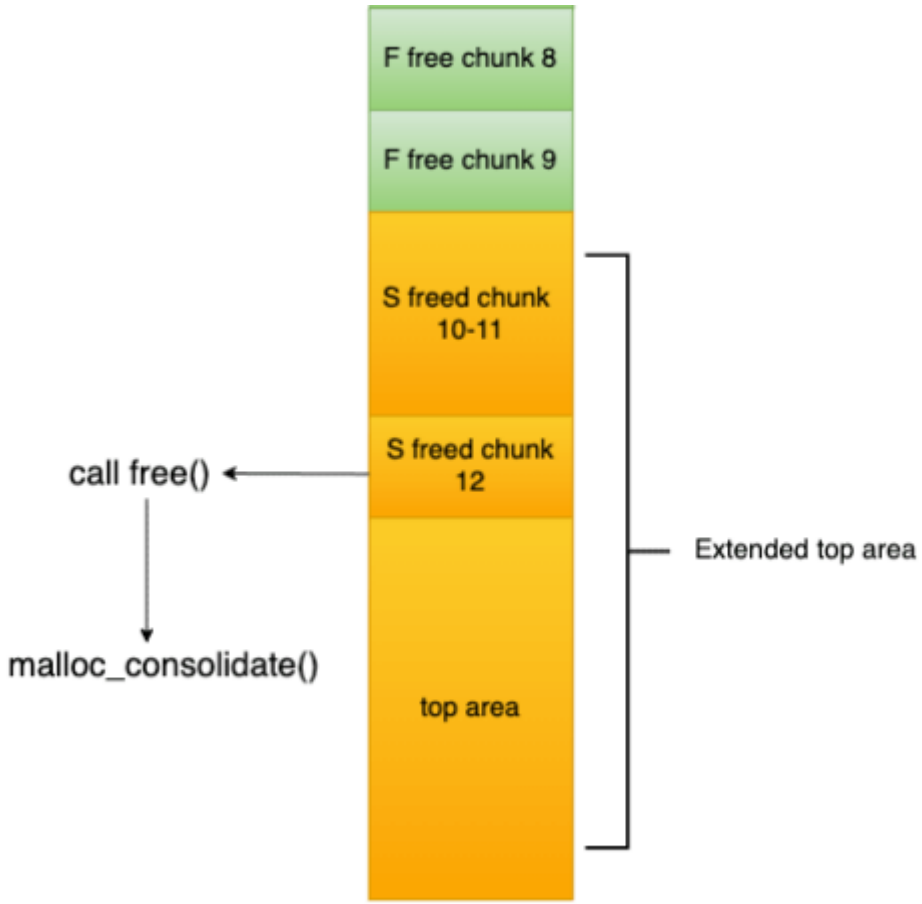
- Svaka lista jest jednostruko povezana
  - Svakom chunku, kada je dodan u listu, se unutar fd polja zapiše adresa prvog chunka unutar liste, a unutar polja za tcache pointer se izmjeni da pokazuje na adresu novog chunka
  - Kada se chunk vadi (unlinka) van liste, uzima se onaj s vrha
    - Pointer unutar polja tcachea poprima vrijednost fd koju izvađeni chunk ima (fd chunka se ne briše -> potencijalni heap leak)
  - Zbog toga tcache radi kao LIFO
  - Tcache jest dodan nakon libc 2.26 s ciljem da ubrza rad threadova jer svaki thread ima svoj tcache bin (jer se pointer nalazi unutar TLS, a struktura na heapu) -> time ne mora koristiti mutexe i blokirati rad ostalih threadova

Top chunk

- Zove se još i “wilderness”
- Nije bin
- Main arena ima pointer na njega
- Pod nekim okolnostima, ostali chunkovi se konsolidiraju s njim (više o tome uskoro)

Napomena

- Konsolidacija (consolidate) jest postupak spajanja 2 chunka na način da se chunku koji je na nižoj adresi (prethodi drugom chunku) size zbroji s veličinom sljedećeg chunka
  - Taj proces uključuje i unlinkanje (vađanje iz liste), više o tome kasnije...


F free chunk 8
F free chunk 9
S freed chunk 10-11
S freed chunk 12
top area
call free()
malloc_consolidate()
Extended top area

Funkcionalnost

- U pravilu su najbitnije 2 funkcije:
  - Malloc()
  - Free()
- Postoje i druge (realloc(), calloc(), itd...) koje imaju dodatne funkcionalnosti
- Najprije ćemo opisati funkcionalnost free(), a zatim malloc()

Free()

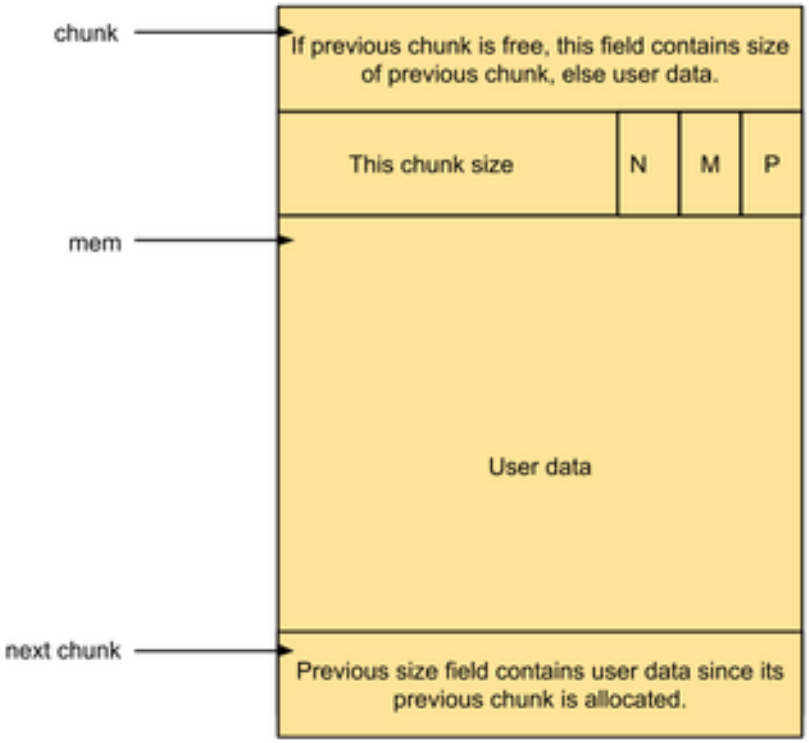
- Free je funkcija koja prima pointer na user data dio chunka kako bi “oslobodila” chunk, tj. omogućila ponovno alociranje tog prostora
- Jednom kada se funkcija pozove najprije se gleda pripada li veličina u raspon tcache bina te ima li prostora u listi za tu veličinu (max 7)
  - Ako pripada i postoji mjesto, chunk je oslobođen tako da je umetnut na prvo mjesto u listi za tu veličinu (in use bit sljedećeg chunka se ne mijenja tj. i dalje je postavljen na 1)
  - Ostavljanjem in use bita izbjegava se konsolidiranje radi bržeg rada
  - Ako uvjeti nisu zadovoljeni (da pripada u raspon te da ima slobodnog mjesta prelazi se u sljedeći korak)

Free()

- Ako chunk nije umetnut unutar tcache-a postoje dvije opcije:
  - Veličina chunka pripada u fastbin range
  - Veličina chunka ne pripada u fastbin range
- Ako pripada, pronađe se lista koja zadovoljava njegovu veličinu, umetne na prvo mjesto, a in use bit sljedećeg chunka se ne promijeni (isto kao tcache)

Free()

- Ako veličina ne pripada u raspon fastbina, najprije se dohvaćaju veličine prev_size i size (oslobođenog chunka) zbog okolnih chunkova
- Ako prethodni chunk nije korišten (zna se preko in use bita trenutnog chunka) trenutni i prethodni chunk se konsolidiraju
- Zatim se pregledava sljedeći chunk. Ako je sljedeći chunk top chunk ili slobodan ponovno se konsolidiraju.
  - Ukoliko je chunk konsolidiran s top chunkom funkcija je završena (top chunk nikada ne ulazi u bin)
  - Ako nije konsolidiran s top chunkom, novonastali chunk odlazi u unsorted bin



Allocated Chunk

Malloc()

- Malloc jest funkcija kojom se alocira prostor u heapu (te inicijalizira heap ukoliko je prvi poziv)
- Prima veličinu koja se interno postavi na aligment od 16 (43 -> 48 itd...)
- Vraća pointer na user data dio chunka koji je navedene veličine + 16 (headeri prev_size i size)

Malloc()

- Kada se funkcija pozove, naprije se pogleda veličina argumenta
- Ovisno o veličini pregledava se bin koji je zadužen za taj raspon
- Pripada li veličina u raspon tcache-a, pregleda se lista za tu veličinu
  - Ako chunk postoji unutar liste, unlinka se i dodijeli korisniku
  - Ako ne postoji nastavlja funkcija se nastavlja
- Razlikovat ćemo fastbin chunk, small chunk i large chunk ovisno o njihovim rasponima veličina (zapamtiti da se fastbin i small chunk preklapaju, stoga fastbin uvijek ima prednost nad small chunk u tom rasponu)

Malloc()

- Ako je fastbin chunk:
  - Dohvati se lista fastbina za tu veličinu i pregleda sadrži li chunkove
  - Ukoliko postoji, chunk se unlinka te proslijedi korisniku

Malloc()

- Ako je small chunk:
  - Pregleda se lista za navedenu veličinu
    - Ako chunk postoji unlinka se i dodijeli korisniku
    - Ako ne postoji pokreće se tvz. Large loop
- Ako je large chunk:
  - Odmah se pokreće se Large loop

Large loop

- Large loop je ciklus koji se pokrene kako bi se smanjila segmentacija na optimalan način
- Najprije se konsolidiraju svi chunkovi unutar fastbina s ostalim oslobođenim chunkovima
  - Svi chunkovi unutar fastbina (neovisno o listi) koji se mogu međusobno konsolidirati se konsolidiraju
  - Od novonastalih chunkova oni koji se mogu konsolidirati s top chunkom ili s ostalim chunkovima u drugim binovima se konsolidiraju

Large loop

- Zatim se iterira po unsorted binu (FIFO):
  - Ako chunk zadovoljava traženu veličinu na način da je veličina ista, iteracija prestaje i chunk se vraća korisniku
  - U suprotnome chunk se raspoređuje u bin za njegovu veličinu (ili small bin ili large bin)

Large loop

- Jednom kada se svi chunkovi rasporede i unsorted bin je prazan odabere se prva neprazna lista (small bina ili large bina) čiji raspon zadovoljava veličinu traženog chunka (FIFO)

Large loop

- Odabrani chunk zadovoljava veličinu tako da je isti ili veći
  - Ako je isti unlinka se i vraća korisniku
  - Ako je veći dijeli se na dva dijela gdje je prvi tražene veličine, a drugi ostatak (osim ako ostatak nije manji od 32 bajta, u tom slučaju korisniku je vraćen cijeli chunk)
  - Prvi chunk se vraća korisniku, a drugi je vraćen u unsorted bin (pamćen kao last remainder_chunk unutar arene)

Malloc()

- Ako nije pronađen chunk koji zadovoljava veličinu, pregledava se top chunk
- Ako je size top chunka veći od tražene veličine onda se dijeli na dva dijeli, prvi dio (tražene veličine) se vraća korisniku, a drugi dio je postavljen kao novi top chunk

Malloc()

- Ako veličina top chunka ne zadovoljava traženu veličinu, najprije se proba alocirati chunk pomoću mmap() (zahtjev za memorijom kernelu)
- Ako mmap() ne uspije pokuša se stvoriti novi heap
- Ako stvaranje novog heapa ne uspije pokuša se proširiti trenutni (sbrk()/mmap())
- Ako proširivanje ne uspije pokuša se stvoriti nova arena
- Ako ni to ne uspije vraća se NULL

Razmisliti

- Prvi malloc koji se poziva u programu – odkuda će uzeti chunk
- Alocirali smo 5 chunkova od 32 bajta, zatim oslobodili 2, zatim tražimo opet 1 od 32 bajta – od kuda će uzeti
- Alocirali smo 10 chunkova od 32 bajta, oslobodili 1 , a onda zatražili 1 – od kuda će uzeti

Zadatak “Karte”

- Prođimo zajedno kroz zadatak i pokušajmo razumijeti što se događa

Ranjivosti

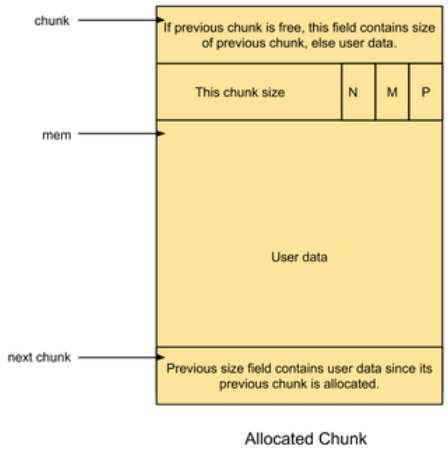
- Iskorištavanje heapa događa se kroz tzv. primitive
- To su općenite ranjivosti (overflow, UAF, double free itd...) koji se zatim koriste kako bi se heap doveo u posebno stanje (heap grooming) i time izveo napad
- U pravilu cilj napada jest dobiti chunk kao korisnik na zanimljivome području kako bi se taj dio memorije čitao/prepisao (arbitrary write/arbitrary read)
  - Mijenjane nekih bitnih struktura programa, mijenjanje return adrese, izmjenjivanje funkcije kroz got, malloc_hook...

Overflow

- Pisanje po memoriji više nego li je predviđeno
- Overflow na heapu se iskorištava na različite načine ovisno o stanju heap-a
- Moguće prepisati – neku varijablu specifičnu za taj program (npr. int is_admin), ili chunk metapodatke

Use after free (UAF)

- Pointer na chunk se koristi u programu iako je oslobođen
- Ako se čita iz njega, npr. provjerava neka vrijednost u njemu u if-statementu → ako s malloc-om dobijem kontrolu nad tim chunkom (možemo pisati u njega) možemo manipulirati tom vrijednošću (primjer1-uaf.c)
- Ako možemo pisati u njega – možemo manipulirati fd i bk pointerima → možemo kontrolirati gdje će neki sljedeći chunk biti alociran → arbitrary write/read
- Osim toga I potencijalni libc leak (npr. ako je chunk nakon free-a završio u unsorted binu)



Allocated Chunk

Use-after-free primjeri

- 3 primjera
  - Read, write, leak

Double free

- Oslobađanje istog chunka 2 puta
- “Dvaput” uđe u bin, tj. stvori se cikličnost u povezanoj listi
- To znači da i sljedeći pozivi malloca mogu vratiti isti chunk dva puta → 2 pointera na istu adresu – izmjena vrijednosti jednog će izmijeniti vrijednost drugog
- Zloupotreba – pozove free nad jednim od tih pointera, s drugim overwriteamo chunk metapodatke (npr. fd pointer) i tako utječemo na to gdje će neki sljedeći chunk biti alociran (arbitrary write ili read)

Double free provjere

- free(a); free(a) u fastbinu će crashati program
- free(a); free(b); free(a) neće
- Tcache na glibc > 2.28 ima double free zaštitu (mora se imati mogućnosti manipulirati i “bk” pointerom (kojeg zapravo tcache ne koristi kao bk pointer))
- Tcache <=2.28 još slabija zaštita od fastbina , free(a); free(a); - neće crashati program

Malloc hook

- Malloc hook – globalna varijabla (u libc-u) koja sadržava pointer na funkciju koja će se izvršiti onda kad se malloc pozove
- Ako imamo libc leak (npr. preko unsorted bina) i arbitrary-write , možemo overwriteati _malloc_hook
- One_gadget - https://github.com/david942j/one_gadget
  - Instant RCE, skokom na samo jednu adresu – nije garantiran u svakoj verziji glibca – zapišemo one-gadget na malloc_hook i dobijemo rce
  - Slično radi i _free_hook
- UKLONJENI u glibc 2.34 !!!

Iskorištavanje konsolidacije

- Overflowamo prev_size chunka i P bit stavimo na 0
- Zatim oslobodimo chunk
- Oslobađanjem chunka dogoditi će se konsolidacija unatrag s početnom granicom određenom s prepisanim prev_size poljem
- Možda smo tako stavili nešto što je zapravo alocirano u bin

Konsolidacija -“Nightmare”

- primjer

Mitigacije

- Novije verzije glibc-a rade razne provjere kako bi detektirale memory corruption na heapu (npr. provjeravaju koliko smisla imaju metapodaci chunka)
- Razne tehnike kako napraviti “heap grooming” da postignemo arbitrary read/write
- https://github.com/shellphish/how2heap - popis što radi od koje do koje verzije glibc-a
- U CTF zadacima se često koriste starije verzije glibca

Prava zaštita

- Postavi pointer na NULL prije free() - najbitnije
- Provjeri je li pointer već NULL prije free()
- Provjeri je li NULL prije korištenja

Infobip CTF 2022 - lolpass

- Fastbin Dup napad
- Libc leak jednostavan – kako do RCE-a?
- Koristi se libc 2.31 – malloc_hook i dalje postoji
- Program je ranjiv na double free
- 1) saznaj adresu malloc_hooka
- 2) Napuni tcache (ne želimo double free preko tcachea jer je glibc >= 2.29 , pa bi trebali manipulirati bk pointerima, inače bi to moglo biti ok)
- 3) Double-freem alociraj fastbin chunk blizu malloc_hooka – preduvjet, alignment user-data mora biti na 16, a size polje chunka mora zaista biti onoliko koliko smo alocirali (drugi preduvjet nije prisutan kod tcachea za < glibc 2.29)
- 4) Pronađi “one_gadget” i zapiši ga u malloc_hook
- 5) triggeraj malloc_hook

Ostalo

- Na temelju “nightmare” i “how2heap” proći što više demonstracija (dokle imamo vremena)
  - Unlink
  - House of Spirit
  - Poison nullbyte
  - House of Lore
  - House of Einherjar
  - House of Botcake

    ...