eBPF incrontro 3

Tesi template frontespizio

<https://github.com/micheleberetta98/>

Helpers: <https://github.com/libbpf/libbpf/blob/master/src/bpf_helper_defs.h>

Funzioni

* Delete non si può fare con array\_map, ma in hash\_map sì

<https://elixir.bootlin.com/linux/latest/source/kernel/bpf/arraymap.c#L389>

delete in array map ritorna -EINVAL (codice -22)

<https://www.man7.org/linux/man-pages/man2/bpf.2.html>

* Function che return 0 perché si poteva fare in passato

Map: oggetto non primitivo per scambiare dati tra space.

Array: dimensione vincolata -> non permette di fare la delete

* Operazione invalida : codice -22 -> EINVAL

Programmi BPF devono ritornare qualcosa oppure static

Ogni cosa dichiarata fuori da SEC è definita globale.

BPF: se non sono statiche le funzioni le mette in una sezione unica del programma.

Programma eccede la dimensione massima.

* Funzioni richiamate in .bpf.c dichiarate static (eccetto quella dentro SEC)

Ogni SEC definisce un programma in BPF.

* Funzioni definite fuori dalla SEC:
  + O ritornano qualcosa
  + O vanno dichiarate static
* Altrimenti scrivere tutto nella funzione dentro al SEC

In base al punto di attacco alla SEC sono messi a disposizione alcuni helpers o meno.

<https://github.com/libbpf/libbpf/blob/master/src/bpf_helper_defs.h>

Mappa BPF\_TASK\_STORAGE: <https://lwn.net/Articles/844391/>

Prende come chiave il puntatore a un determinato task e si può inserire un qualsiasi valore.

Allarga la struct del puntatore e aggiunge i dati in una sua sezione.

Richiamo con bpf\_get\_current\_task() -> return u64

bpf\_get\_current\_task\_btf() -> return \*task\_struct

* \*puntatore pid per ottenere il pid del processo
* Uno dei due può non esistere in Windows

Btf si basa su informazioni di debug di linux.

Per questa mappa esiste solo la GET: ultimo parametro è operazione.

bpf\_task\_storage\_get(...., BPF\_LOCAL\_STORAGE\_GET\_F\_CREATE)

Linux Security Module: libbpf ha la possibilità di far eseguire o meno un’operazione.

Tramite bpf si ha maggiore controllo dei privilegi di un processo.

Dentro SEC: coppia

* Tipologia hook:
  + kprobe/”hook”
  + lsm/”hook” (lista specifica di funzioni del kernel)
  + tracepoint/”hook” (punti particolari per essere utilizzati per vedere cosa fa il kernel in alcune fasi d’esecuzione)
  + uprobe/”hook” (documentazione arretrata, ma allega a funzioni lato user e sono usate per ottenere il simbolo della funzione)
  + varianti krpobe e uprobe: prima di entrare nell’esecuzione della funzione gli hook sono eseguiti.

Per eseguire la funzione in uscita, si usa kretprobe o uretprobe (si esegue quando si ritorna da una funzione nel kernel / user)

permettono di ottenere il valore di ritorno (kprobe e uprobe non si può fare perché si è in ingresso), ma non di accedere ai parametri.

* Nome hook stesso.

Tipo di hook: diverse cose a cui attaccarsi.

<https://elixir.bootlin.com/linux/latest/source/include/linux>

<https://elixir.bootlin.com/linux/latest/source/include/linux/lsm_hooks.h>

<https://elixir.bootlin.com/linux/latest/source/net/ipv4/tcp_ipv4.c#L200>

kprobe/funzione\_definita\_nel\_kernel

possibilità di accedere ai parametri (nome\_hook, param1, param2, …) passati della funzione per vedere le cose come le vede il kernel.

Programming for Windows

* <https://blog.subcom.tech/ebpf-programming-on-windows/?utm_medium=email&_hsmi=258521338&_hsenc=p2ANqtz-9taEQ7sh0r7EigAVYIUQAWXILfklYjy4fqJZ7urIApSoRk2T4eO39lKrq4pOSOOtZtXrM2hdkTc27ocis3sALK2Fm23Q&utm_content=258521338&utm_source=hs_email>

Sito di gente che sclera a eBPF: <https://tmpout.sh/2/4.html>

Examples BCC to convert in libbpf:

<https://github.com/iovisor/bcc>

<https://github.com/iovisor/bcc/blob/master/tools/bindsnoop.py>