**Tutorial: Create Your First EBPF Program to Monitor Your System Using Bumblebee - Adam Sayah**

[Immagine che contiene testo, screenshot, bigliettodavisita, clipart

Descrizione generata automaticamente](https://www.youtube.com/watch?v=os2f0yfcgIU&list=RDCMUCfX55Sx5hEFjoC3cNs6mCUQ&start_radio=1&rv=os2f0yfcgIU)

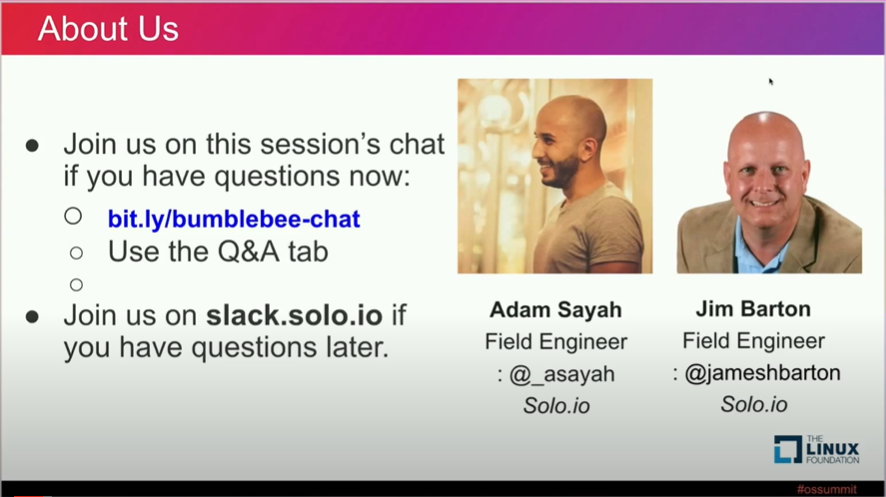
<https://www.youtube.com/watch?v=os2f0yfcgIU>

Immagine che contiene testo

Descrizione generata automaticamente

thank you thanks for for joining this session today we're going to be talking about ebpf

programs and ways to simplify (to monitor) your system and we're going to be talking about an open source project we have at solar.io, called bee, a way to simplify your your ebpf.



so just uh a quick intro to the speaker today my name is adam.

i'm a field engineer at solo.io basically i've been dealing with uh

api gatways and service mesh for quite a while now

but i'm you know we started looking into kind of a deeper layer now uh even not

solo.io and we're operating at a way deeper layer so instead of being at l7

where it's mainly application now we're looking to the l3 l4 dealing with

you know for example now we're trying to deal with ebpf uh

seeing how we can leverage it to enhance monitoring security and so on so it's pretty new to me uh

i'm still learning it and it's really interesting because that falls in exactly where

where bumblebee is to simplify the way to provide the ebpf application to

someone that is just getting up to speed in that technology and with me jim barton i

think he just went to get his uh his laptop he's a film engineer with me in the team um you know repair work together all right so again thanks thanks for joining

Immagine che contiene testo

Descrizione generata automaticamente

okay so quick intro to ebpf what it is

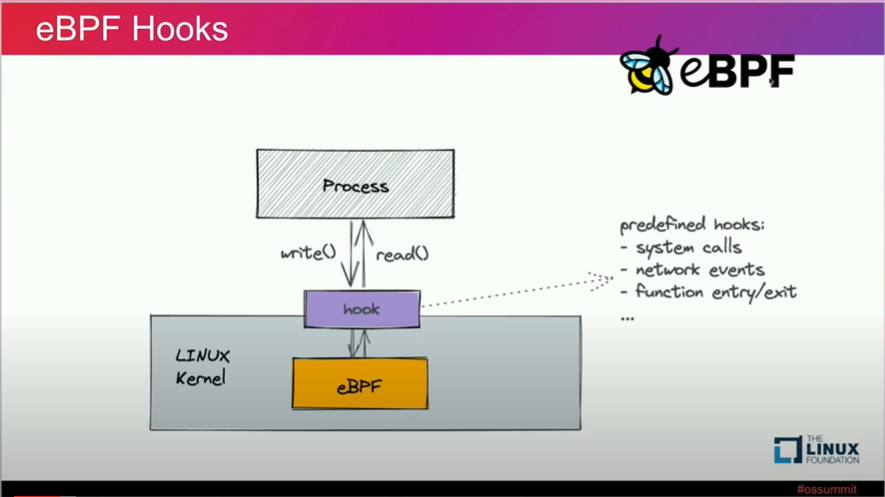
bpf stands for extended beckery packet filter i think everyone using tcp dump for

example have been using the ppf technology in the past uh it's even based basically as a

way to create some code that's runs in the kernel in kind of a sandboxed way so

secure manner and we can interact with that code right we can direct either to get data

or to provide features like security and so on okay



so the way we see it again so it's uh you're gonna create a code that will run in the kernel

that code will will be tied to a hook right so an event and then we're gonna

have a process that will interact with that code to read data and process it okay and

we're going to see that more into details later in the workshop all right

Immagine che contiene testo

Descrizione generata automaticamente

so why ebpf why would you care about this um i was talking a little bit earlier what we do at solo.io mentioned that

you know been historically dealing with l7 uh type uh you know systems where you

have the gatway gas we can produce some monitoring stack like metrics and so on

or security and so on but that's a way to high layer so if you want a security system you want to secure it in

depth what i mean by that is what matters first in terms of security obviously if you want to secure a

network to not talk to another i think the first one is basically having these two networks not being

connected to each other okay so that's physical that's the most secure uh from

there i think the layer after that and in term of security let's say they're connected now is to deal with you know

lower levels in in the model for example l3 l4 when you do tcp and network you're

saying hey okay so now we are segregating these two networks or these two services they can't talk to each other we block in the communication

there because when it hits the l7 layer so it hits like the application layer it's uh sometimes pretty late

right so uh and can be you know it's more it's more challenging to

enforce a strong security when it deals only with the top level uh l7

policies which is for example http okay so yeah why ebpf it operates super

low in in the model you can create code that goes directly into kernel so you have access data fast

operate fast enhance security i think one of the main advantages of ebpf is observability so

you have a program running in your kernel listening to events for example network

connections or file opens or things that are really low level and acts on that

either operate there to enforce a policy or capture that metric and transform it

send it to a send it to a program that can read it and for example display metrics and so

on okay so observable i think that's one of the key features of ebpf and

what we can do with it and

Immagine che contiene testo

Descrizione generata automaticamente

i talked a bit about the other features mainly

networking allowing you to kind of uh you know use this technology to

to plug multiple system or re-route or deal anything when it comes to traffic itself right and when i talk about that

also can i meant i can mention security as part of same bundle security securing

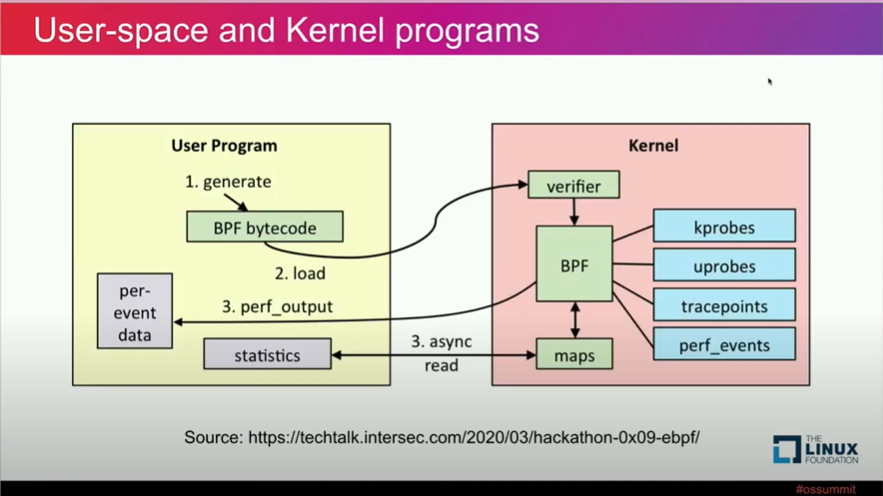
services to service communication and so on at a way lower level for example i'm

not allowing this this group of ips to talk to this group of ips right so it touches into

networking plus security and i think i don't know if you guys are familiar with psyllium for example right

psyllium use ebpf for networking but that's a good example

of you know use cases or we can use ebpf



all right so the way how it works um as i mentioned earlier

there's a program that we call like a kernel program that runs within the kernel it's it's co it's a code that

would be listening on certain events and reacting producing you know producing data when

this data is produced is it's put into a map

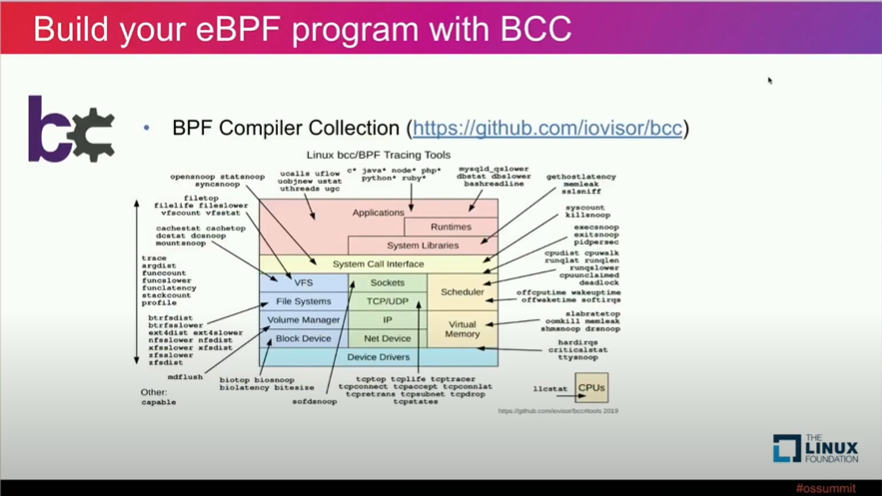
into a map and then you're going to have what we call a user space program that this one will

look into that map read data from it and do something like either i don't know print it format it into metrics do

anything you want but basically it's two-part system you have one running in the kernel producing the that's the most

important one producing what you want and there's another one which is the user space program that's the one that

actually listen to the data and do something with it



all right so now if i'm gonna write an ebpf program

what can i do today there's multiple ways i think historically the first one is

bcc right so bpf compile collection this one will allow you to create some code

i think it's easier to see this way

Immagine che contiene testo

Descrizione generata automaticamente

it allows you to create some code in in c for

what a called kernel program so the code that runs in the kernel is written in c and

then you're going to execute that code you're going to embed it into you know

a binder a binding technology for example you can use python where python you're going to use your user program uh to execute the user part of it so

in in the same in the same python script you see the example here you're gonna

have the user program so the one that actually the user space program the one that's going to read data and you

can also have c like a kernel like a kernel code

c embedded into your python right and this

Immagine che contiene testo, screenshot, monitor, schermo

Descrizione generata automaticamente

this is fine right it allows you to write your ebpf programs

though it has some drawbacks and i think the main one is that

it is actually compiled on the go so every time you run that specific

program it's going to compile everything so take some time there to run the system to run the binary right so time

plus it is uh really dependent on the platform so

let's say you have um you know you run it on certain system you can't guarantee that this binary or this code will work somewhere else it's gonna really tie to your environment

so yeah not that great okay so let's think about what's a better solution there

Immagine che contiene testo

Descrizione generata automaticamente

a better solution would be to run what you call it bpf plus uh ppf core

okay core stands for compile ones well once run everywhere okay so

it's more um like if you're looking into like just a bad comparison here but you look at the

go code for example you're probably compiled once right you're going to create your program compiled once in a

specific environment you're going to be you know should be able to copy this somewhere else and it's going to still

work okay so it is it is a a e in better way it is better than the previous way like

the with the vcc lib vpf is you know allow us to create the code

**only once it's compiled once so you don't have to combine it every time and you can transport it from an environment**

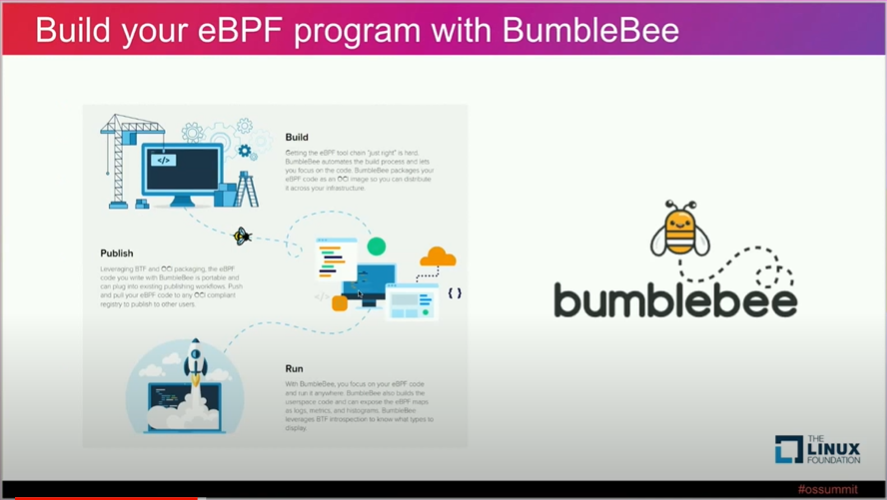
to another though what you have to do here you still have to create your

user space program plus the kernel one so both of them right you still need to to create both of them

and that will work with any uh you know ebpf we always need to create both but this here

is better than bcc and we're going to see an example of all this and we're going to create multiple programs using

the platform later on in the workshop we're going to compare different technologies and see how it works



all right um now why bumblebee so we talked about bcc we

said it's all right now i talked about lbbbpf we said hey it's better now let's make it great okay and to make it great

like what if i have a library or a program or like you know

like something that helped me to create an ebpf program i would care only

about the kernel code so i'm not going to care about the user space code i'm just going to

you know just write what matters and what matters is basically the code running in the kernel

and what if uh i want a tool that packages all that so build it and

package it and put it somewhere where i can be able to reuse you know have a good user experience

where i can be able to reuse my binaries okay and that's actually what bumblebee is

trying to solve here okay it's a tool that allows us to care only about the kernel code so i'm going to create only

the kernel code plus i'm going to be having a good user experience into

building and compiling building and just distributing my uh my binary and if you

guys use docker i'm pretty sure everyone used docker here uh it's kind of the same similar experience right so you're gonna

do you know create your code then you're gonna build it and then you're gonna package it into an oci image then you're going to be able to push it somewhere to registry and then you're going to be able to pull it when when you need it on a specific environment okay so we're going to do we're going to do this later

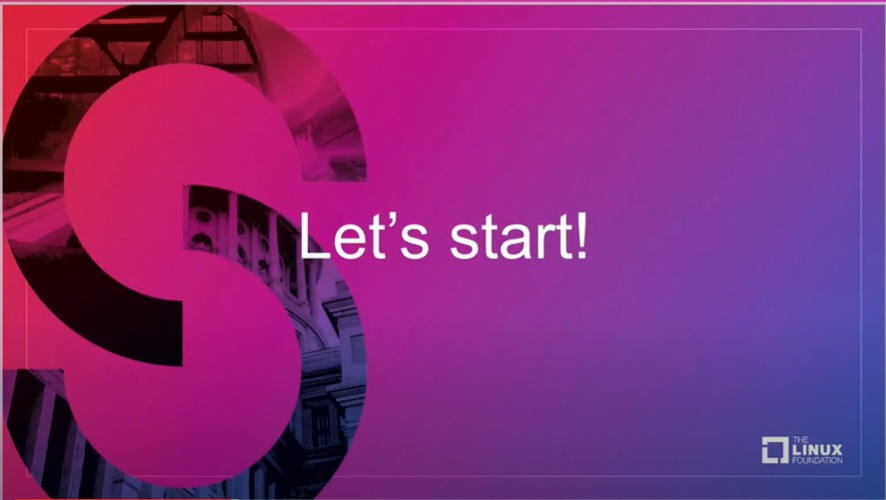
Immagine che contiene testo

Descrizione generata automaticamente

so again um that just what i said right now focusing on

kernel code first uh not dealing much with with the user space one and also

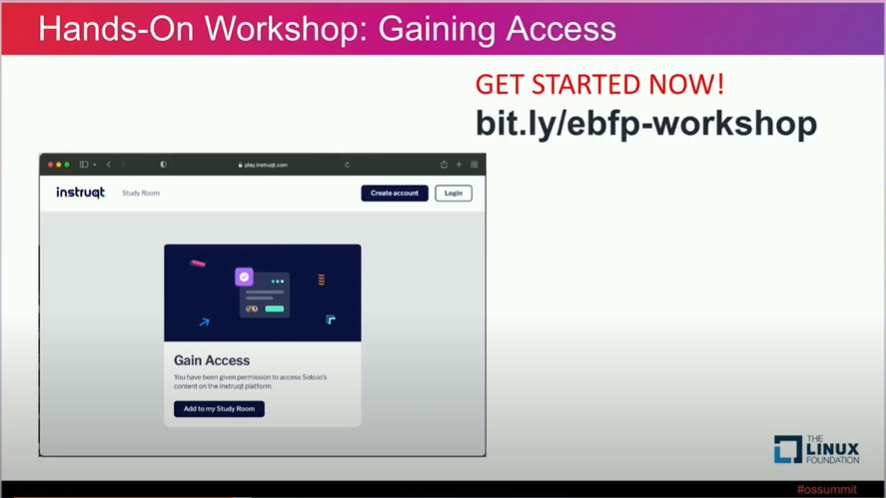
all uh the the user experience around it



so okay let's

gets hands on here let's let's start on this um i don't know if you guys were in the workshop yesterday kind of same thing

we will just



everything you guys need today going to be on a browser so you need

nothing else you just need a terminal a browser you're going to have to go on the specific link here and then

gain access to your environment and then we're going to do the same thing i'm going to do the exact same instructions

with you guys okay i'm going to do it here on if you don't have any laptops you can follow with me

here on on the on the screen if you want to do it just you know use your your laptop right

now i'm going to do some do it in the same time i just want to mention that uh at the end of this uh

workshop we have uh we're offering a certification okay so if there is a quiz at the end and if you

pass 80 uh of the questions you receive a badge that can be you know you can link it to

linkedin or you know it's basically a certification based on the fundamentals to ebpf since

we're gonna you know be creating uh some ebpf programs uh today

uh and i'm gonna share the link to the to the quiz at the end of the session okay

so again please join this link here

i'm going to click on it to show you guys how it's going to look like

and i'm pretty sure i'm going to do it into let's say i'm gonna do it in like a

incognito one just to make it easier

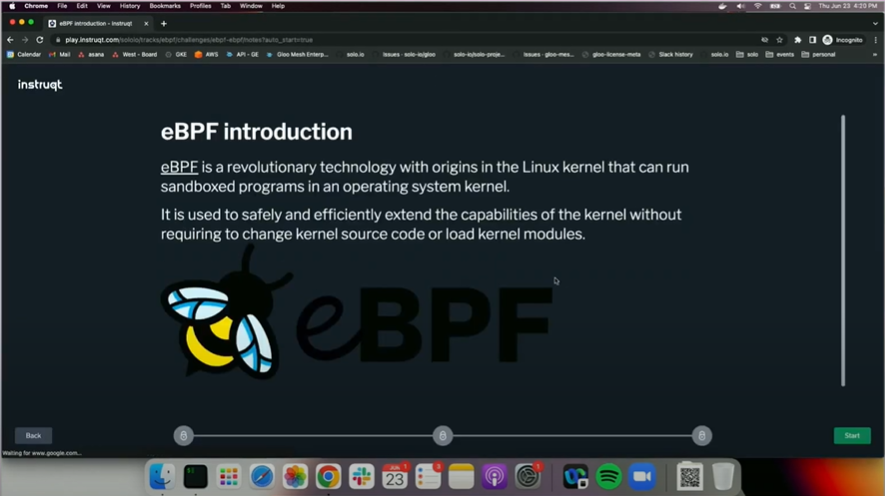
all right um there you go so

what you guys gonna see is uh you're gonna have to gain access to your

your class and you're gonna see here ebpf workshop

load it into my exclusives right and just click on the track

and then um click on start track okay so three steps



there first click on the link and then i'm going to add it and then click on start track once you click here

it should well because it's incognito it's going to recapture me here so i'm doing the

exercise very good and we see that now the environment is creating right it's going

to take a couple seconds to get um probably a minute or two to get ready and after that i'll um you know

i will do i'll do the examples in front of you so going back to my slides just

to go back and show you basically the certification so yeah first thing click on the link you're

going to have access to your to the workshop just add it and then

click on start track then you're going to be ready to follow follow with us

and then once ready you're going to see like a green button in a corner saying that your environment is ready to start

and then we're going to click on it and we're going gonna you know start our workshop uh



so again the the badge uh gonna be a

fundamentals for ebpf uh all the thing needed is to pass with eighty percent

uh if you you can always retake the the test if you missed the first time the only thing

you have to create it will use a different email uh to to to really retry the quiz

and uh the badge will be issued in a couple weeks okay so we're gonna send that bash the specific email in in a

couple couple weeks all righty so

there you go i think we start here we good we go to start

introduction to ebpf so in this workshop we're going to do three things

we're going to start by uh creating a a ppf program using

bcc right and then we're going to create uh a same program using libf and at the

end we're going to use the same thing for creating the same program but using bumblebee okay

BCC

so first step let's see how to create a bpf program using vcc

so to do this we're going to go straight to

the code and as i mentioned earlier when you create a vcc program program

you're going to you know create your kernel code plus

you're going to create a user space your user space program

if you guys have yeah are you all good i'm sorry

but it's okay oh okay sorry [Laughter]

oh and i didn't notice that sorry about that all good but uh let's go back to it maybe there's

someone on the chat that uh missed it yeah

so where where was the oh man

i needed a coffee when i created this link [Laughter]

all right sorry about that uh it's ebpf but yeah it's it just click on that link

i guess here with typo um yeah sorry about that

that's what we do when we multitask right um okay going back

so yeah can can you check if in the chat i don't know if you have access

yeah just check if if anyone needs help there all right uh well

you guys good now awesome awesome all right so again let's try to create uh

an ebpf program using vcc this time so as i mentioned to create one using bcc

you're gonna create everything in pi like you're going to write in python you're going to have

the c plus plus the c the c code for the kernel sorry i just run it directly but

you're going to yeah you're going to write the the c the c code in python you're going to have also the

user program written in python right so we're going to show you guys this in a second so let's see

start by that so let's start here

and take a look at an ebpf program in uh

using bcc so what we do here is gonna see that we have some c code which is starting

from here all the way i don't know if it's too small yeah but starting from around here

all the way down to i don't know what's going on my ui now

i'm gonna just refresh

yeah so let's go back you're gonna see that you have

some c plus plus code sorry some c code all the way down here that's basically

for our kernel code that's the one the program we're writing right now just gonna grab any ip communication

and print it okay so it's going to be attached to um basically a tcp a tcp for

connect right and gonna it's gonna print anything like if if there is any

ip interaction let's say you call out somewhere or there's a service calling something we're gonna print it okay

that's the program you're writing right now so to do this we're gonna have to do two things mainly the first part would

be creating what we call the kernel code the current program and that in c and

that is tied to the the the tcp connection event and

then it's going to put that address i'm going to put that ip into

a map as i mentioned okay

i was like maybe is a question here and then once so once the the kernel

code will grab that ip when detect there is a connection there you're gonna put it in a map and then you're gonna have

uh the user program that's that's one is in pi and python gonna read that map and

print it and that's basically what's happening here on the second part of the the python code here we'll see that we

are just looping on all everything in that map and then we're going to print it okay so again

kernel code will get the data is listening to the events grabbing the data any connection

grabbing the ip putting it in the map the user program is connecting that specific map

and fetching data there and just printing it okay that's basically how ebpf programs are

now let's try to do the same thing so okay again let's go back here and

let's just run it so again if you if you run this python 3 data

if you if you run the same you know the same python code we're just looking at here we see that we are capturing all

the ip comm communication here okay so source address to destination address

what's happening in your in your system right now okay and you see this is really powerful so

now we can definitely monitor what's going on all the interaction ip interaction they have within your same system

LIBBPF

now let's try to do the same thing but this time we're going to try to do it using

lib bpf all right to do this

let's go to a specific folder and then we're just going to copy all the code that we need

again as i mentioned earlier the main difference between

bcc and libpf in the bpc in in a vcc what we did we had python that you know

taking care of our user program code and we just embedded the c code for the

kernel within the same python script in lib bpf everything is in c both a

user program and your kernel code okay so we're going to see how this one

works now we copied everything our example here let's just do a make

and test it out first so we're going to make our binary i think also something i mentioned with

dbpf the binary you're going to create this one can be reused on other platform

okay it's not only tied to a specific environment compared to the first code

we did now that we have our binary let's just run it

okay and we see basically it's kind of the same thing we are capturing the traffic we're kept in the source

destination and uh you know the the target destination and printing both ips here

okay it's kind of the exact same functionalities if we take a look at how this one is implemented

you're just going to see that let's look at the source code

i think we should focus on mainly we should focus on mainly the

what we need here is defining sorry just messed up here

let's talk about this here right so in your code

as i mentioned earlier you always use a map to communicate between your kernel program and your user space

program and through that you're going to use a map

and there's basically two main ways of doing it either you can

use a hash map which is basically you know a key value store or you can

use a ring buffer right so basically it's just like a queue so you're putting

data in a certain queue and then do you are you know you are we are

fetching it on the other side that that basically like what we're using right now right

uh no so using sorry we're using a hashmap here but basically you define you're

gonna define the structure of the data you're gonna you know you're gonna send to your user program

all right so once this one is defined then we're going to look into what are

we doing and the other part important here is that we are tying our

kernel code to a k-prop tcp v4 connect so again as i

mentioned earlier when you create your kernel code in um in um

in the eppf in the kernel you tie to a specific event right so

this code is watching for something and react reacting to it in our use case

here we are listening to tcpv for connect okay so we are

checking any communication going on and then processing this data and putting it in the map kind of exact same situation

we have previously only difference here that both codes are in

c okay so now you know we saw

the the kernel code at this stage we saw how we defined the structure of the data we need like we see here that we need

you know a map uh and and the data actually that didn't

mention that but the data you see that we are sending here we're looking for we're looking for we know we're

capturing this this two the two field here basically that's what we are sending through our map we are sending

the address the what the source address and the destination address kind of you know like if you're looking at like if

you're writing any code you know like you're familiar with like different definition of structures you're going to define like what kind of object you want

to transport from like you know through a map or through a like a you know channel if you're you're

doing go or something so basically here we're defining that we are you know we need we need to capture or populate the

source address and the destination address we're going to you know define all this dimension and use it to put it

through the through the through the map all right so that was the kernel code

now let's take a look at the user space

application right so as i mentioned earlier once the data is processed on the kernel

now we need to connect to it and read it and do something with it okay and in this case

it's basically just going through the list right

it's just going through the list and printing whatever it gets right um

we are going to connect to that map capture the data and just print it

there is nothing there is nothing magic here now

uh the thing is so the thing is that that you need to to

to be aware of here um there is no mainly difference like in it

like at the end the two the bcc program and the lbpf program are behaving exactly the same though this time we

create something more efficient so we just you know we we built it once

**okay we had to create our user we had to create our user space program but basically we just built it once and now**

**we can reuse the same binary on a different environment right you can even like copy it to somewhere else and run**

**the same thing it's going to always work i mean as long as you're running on kind of a new a new kernel**

**all right so i hope you guys see at least the difference between bcc and libby pf**

BUMBLEBEE

now let's simplify this even a bit further so let's let's try to

make a simplified application

to do this we're going to use bumblebee okay

first uh just a question here are you guys on the same um on the same lap are you guys right here you're good

all right awesome okay okay so we we we saw how to do

an ebpf program so at this stage you guys can create ebpf programs right

right all right proud of you guys uh

yeah so let's say now i'm kind of you know that's that's what

i was like mentioning when i would start a talk i think the the

the beauty of bumblebee and you're going to see it now it's still i mean still like it's going to still be c c code and all that stuff

it's kind of simplifying the way we create abpf programs i think that's the main

goal of it we're trying to simplify things uh through technology if you if we have to still deal with

like low-level libraries it's going to be complicated though here we still have to deal partially with some low-level

libraries but there's a lot of other tools that can simplify simplify our life and that's the goal of bumblebee we see the complexity let's

try to reduce it and i'm going to solve it completely but let's try to make it better okay so

let's install bumblebee here

okay now we have bumpy bumble bee installed so you should if you run just b sorry

too many e's you should see it installed so let's go

and do be init so being this will just create kind of a

placeholder type project within in your system and allow you to create your first ebpf

program so you do be in it and then okay so which

kind of language we want to we want to use today uh today we're just you know

uh we can only use c but in the future we're looking into using rust too so okay i wanna see

uh based eppf program now let's take a look on what we to do are

we looking into doing network stuff are we looking into like uh capturing

data in from the network or we are looking more into kind of file system things right are we looking to

capturing who is opening certain files or so on okay at this time in this example let's do a network point

now in term of what we want to do in you know i mentioned the map

right i mentioned the map that contains the data sent from the kernel code to

the user space program what kind of type what type of map you

want at this stage let's say i want a hash map okay so i don't want just a

queue i want a key value store and now the last one is basically

how so the data i'm going to get from my kind of um the data i'm going to get from my kernel

program what i want to do with it am i going to want it as a metric like a counter or

one as a judge i want it to just maybe to print it and this what we're going to do right now we're just going to print

the data coming from the kernel [Music] kernel program okay

and then let's say we just want this into a file called probe

dot c all right now we're good

all righty so let's take a look at probe dead c

probe dot c

and we can see that now at this stage it's kind of just a placeholder of like what kind of data

you're looking for okay in this case uh i mean you saw earlier

we define our structure of the data we we're looking for and here we didn't define anything right dimension is is

empty so we didn't find any data we want to look we know we know process but we're going to do it later

and then we're going to define basically the configuration of the map that that we need for transporting data from from

from the kernel all the way to the user program and in this stage we are actually connecting basically

just a kind of a haloware example but in this case we are connecting to tcp v4

connect okay so because we said it's a network abpf program we're creating right now we

are you know watching the event for any tcp v4 connects right

DIFFERENCE LIBBPF AND BUMBLEBEE

um now let's let's do something let's do a difference between a diff between the

you know the previous like we do it let's go do a diff between

how we do create a user kernel program in just lbpf

comparing that with how we do it when in bumblebee if we run it

here we're gonna see that the only difference so basically bumblebee behind the scene

to use to to create like uh counter programs it's still using lib

bpf okay so the only difference here is we added like a way to say how we want

to process this specific data okay and we see here like

like in in the in the all basically on the libby pf1 we did previously we just used a map

this time we're saying okay we want a map but basically we want also a counter based on that okay and we'll see how

useful is this in in the future

BUMBLEBEE CODE

all right so now let's just

get the code of the libyapf we did previously we're going to grab the same one this time

just the the kernel program one okay we're not gonna get the the user program one we're

just gonna do the counter program copy it and just modify the specific line

that calls only the put data in map we're gonna put maps dot uh counter all right so

let's copy this now we have it all right

so okay so at this stage we have a lib bpf

type program so have we have a bumblebee program only for the kernel

uh program right we didn't write the user the user space one that actually reads

the data and the only thing when modified here is that on the map we did map dot counter

so we are going to count the data now we describe that as metric you know a

counter metric now um why bumblebee is important here how it's

gonna make our life easier at this stage we'll see no difference right we just saw we still

wrote you know we still have have to write a libby lib bpf uh program on the kernel

well the good news is you don't need to create um the user space program you don't have to

write that anymore so this is going to be taken care of by bumblebee

right so if you go back to the previous i don't know if you can go back to previous example but if you guys remember the second example

we had to create two c files and this time we have to create only one

which is just the kernel one only the one that matters okay

all right so let's do uh and now let's show you also

how we can simplify all the process using b when i say b is like

bumble bee actually right so let's do a b build here

and you guys see that i'm compiling my bpf program

so it's going to compile the uh the example i have here for tcp connect

taking some time and then

all right now it's compiled okay now i have my bpf program ready i didn't

have to create a user space program because that's a win right

now okay so i have it compiled i have it in oci image i mentioned

earlier it's kind of same docker kind of a docker experience now that i have my program i can push it

somewhere right i can push in any oci compliant registry anything that can store for example a

docker a docker image so here locally you can start like a

docker images three in your environment and then we're gonna be able to push

that program okay so now think about it you have

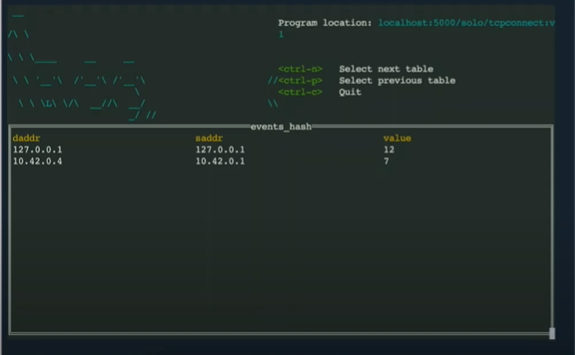
your bpf program created and you have it now in a registry that you can reuse

okay all right so now we have the program there

so let's uh let's now just run it

okay we're just going to do b run and point to the specific

you know build and look at that okay



basically we didn't have to do much we had to create obviously the kernel code one

but you see that we have a counter automatically included and we see the values here how

many communication we have between that specific address to that specific address and we can if you even if you go

on the second terminal you can try that go on like terminal 2 and do like let's

say we do curl sorry girl google.com

i run that go back to your terminal you see have new addresses okay these ones are coming

from any traffic operating on your system okay so think about the power here what

you can do now we create a ppf program that we push to a registry

that we can deploy anywhere we want is actually printing really valuable data

okay all right everybody uh so let's take a look at what we can

do with uh what's other data we can use

METRICS

since

b is running right now like bumblebee is writing it's actually also providing metrics

okay we said that this is one of the most important things providing metrics easily so if we do a curl

slash metrics pointing like on the port 1991 since our bumblebee service is running let's go to another terminal

just run this command here you guys going to be able to see that we

are getting a lot of metrics that we can be able to use we're going to see that later but you see the first one here on

top right i don't know if you guys see it or i can just probably zoom in but

basically we got the data we are getting the data from source on that destination like the data that we really need uh as

as a prometheus metric right and this this kind of component we

already wrote we can reuse and i mean in within the same uh you know enterprise or

company people can reuse like abpf programs without having to to write them they be able to run it somewhere that's

going to collect data from a specific con like let let's take an example here let's say

part of a company and i'm maybe an sre or maybe just a

security guy but the thing is they asking me to provide a way to monitor every single cluster communities

cluster to get all the data from the traffic going from a service to another okay so what i'm gonna do

let's say have like 50 clusters the only thing again like this it's

pretty straightforward which i can do here i can build my bpf program compile it build it push it into a registry and

then just deploy it on every single cluster i can just have like a for example argo cd or something

like that that deploys this bpf program everywhere it's going to collect data from everywhere and then

i'm going to be able to see a full graph or communication between all my services okay

so let's put um actually we've been seeing a lot of code now

EXAMPLE

let's see a use case let's see how we can use ebpf

in one cool example

all right so in the next lab we're going to put all this together

and try to see the power of actually an ebpa program with bumblebee

are you guys excited are you guys ready

all right so are you guys done at least are you guys

at this stage here or am i going too fast good good awesome awesome yeah cool

all right so too much code let's see the fun stuff

all right we already wrote our bpf program already pushed it to a

registry is already ready you don't have to do anything you're done with that part let's put it

in a real use case here so let's say i have uh a demo

application right let's say in this case we're going to use the book info demo application let's say i want to use that

one i'm going to deploy it on my current test cluster

there you go this will create my

demo application with a couple services if you do quick still get bod

you're gonna see some containers getting created okay so

the booking for application is just a it contains like four services it has new ui

and you can see like you know you can have i think it has like a a product page and

reviews and details ratings does anyone here use this too

no okay uh

uh it's basically the dim the demo example that is you use always okay

all right so i have a demo application running think about that as your application

right like whatever you're using internally it's on communities today now

let's now let's uh let's actually deploy

prometheus to you know collect all this data and put it put it in in prometheus

oh sorry i need to click on the right things

all right let's create namespace for prometheus and then let's just install prometheus

do you guys use primitives here all right go

all right so the goal here is to collect metrics and put them

like all this data all the the connectivity data we we saw you like

in the previous lab when i put that into primitives and then leverage this data

all right now parenthesis is installed that's cool

now let's install actually deploy bumblebee so again previously we mentioned that

we can package the avpf programs into an oci image that can be in the registry

so now what we're going to do is that we deploy this ebpf program on our cluster

and to do that we're going to use a daemon set okay and we're going to make sure that

we have at least one pod running on every single node in our kubernetes cluster to

collect this specific data so just a demon set here nothing crazy and you can see here the only difference

here so we have bumblebee running the you know the the just

the bumblebee docker image we have a package here but the thing here what matters is that we are pulling

data from this uh oci image that we pushed in the previous

lab you guys did b push to a registry right so

that's the only thing we are doing so we are running a bumblebee container but we are running actually this

the the bpf program within the same with the same the same pod and then we're exposing everything

through a port 1991 all right now once i deploy this

this is done so this will create a pod on every single

node and that pod will have our program that we created previously

running okay and the metrics we saw gonna be exposed on a specific port

now the only thing we have to do is to configure um

you know prometheus uh pod monitor

in this case to look into the this part and you know collect data

from it collect all the metrics to and send them to primitius and we see that here okay

all right good uh all right so at this okay so this stage here

we have our demo application installed we have our abpf program running on every single

node and we have prometheus configured to listen to metrics on that specific bots

now the only thing we're gonna do is to run some traffic

all right so here just generating traffic imagine that's a customer playing with your environment or

anything you want so we are running traffic through multiple microservices right so going

from a pod to another and we just you know just need some data here now

so at this stage we have all our data in prematures so only thing we're going

to do right now actually one of my colleagues created this small program called

key bpf key abpf it's just going to look into primitives like for the data and

then going to format it in like a you know in a graph so that's only thing happening right now we're looking into

prometheus and then we're going to collect all this data show that as a graph all right

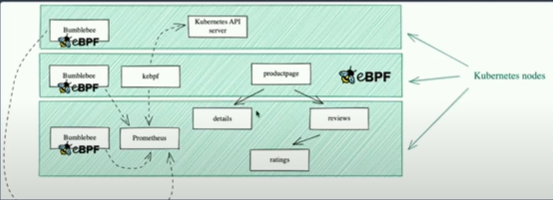
there you go we're going to just create this system and here we are just

deploying this spot key epf

and it's looking into prometheus right we're looking we're pointing to itself directly and that's all we need

to do this is running this is cool so let's

see what's happened here i don't know if you guys can see but this is what we just created



okay so we have we have a cumulative cluster of multiple

nodes we have bumblebee installed on every single node that's the one collecting

the data and then we have our product page and details and reviews

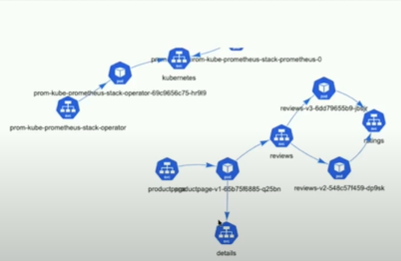
and ratings installed on our cluster i don't know which nodes basically it's just installed there and then we're

running some traffic here to see what's going on okay and all this data from bumblebee is pushed into prometheus and

then we have this k ebpf is just a server to process

promote this data and show them in a certain way so if you go to kbpf there you go

look at this magic really cool all right so



you guys saw what happened here so like that that the program we created earlier in

uh in the previous lab it's producing data here it's producing really valuable data we can see what's

going on we can see which service is calling which service you see here we can we can know that the traffic is

originating from the product page going to the reviews going to details all this data got captured by the bpf

program we just created okay we deployed in our clusters and then how we collect

them and how we process them to show some valuable data and if you guys uh

going to be using with like for example celium psyllium do something kind of similar with a project called humble

that will collect data and produce some sort of graph or some sort of charts okay so this is just a quick example of

what you can do with epf it's really powerful grab data directly from your from your nodes you can process them you

can display them this way you can create policies to you know uh secure the

traffic and so on so i hope you guys uh saw the power here that's uh

basically the end of this uh of this workshop i hope you guys saw saw that what we can

do with eppf it's a really powerful tool um i invite you guys to to test

bumblebee locally see maybe that's maybe you guys are going to find a use case for it

END

we're going to go back to the

i'm going to go back to this first thing i want to mention basically what you know if you guys interested in all these

technologies right like appf uh psyllium interested into istio

uh envoy we are hiring we're growing the company fast you take a look there

always you know we're always looking for talented people second thing we are we have a booth here

uh s25 you know you guys have any questions right now or after this talk if you want

to just continue the conversation please visit us there it's going to be always good to see you there and

lastly i want to just to mention our badge for today

so if you if you go to oh this time area i spell it right i guess

yeah so if you go to ebpf exam or just scan the qr code pretty small

but i don't know if you can do it uh that will take you to a quiz and it's prob like i think there's like a 15

questions on and if you answer them all correctly or more than 80 percent we

gonna provide you a fundamental for a bpf badge if you just wanna redo that

specific exam just use a different email and that totally be fine all right and

about that well thank you guys for listening to me that was hopefully that was uh some

valuable information here

and the best thing we did in only in one hour at five pm in the afternoon i guess you

guys were done if you guys didn't have any questions i'll be here around

COMMENTS

hey

yeah so let's go back to that i'll show you here

let's go back to it so if you go back here

all right so remember like bumblebee was providing metrics directly on the server once runs

it provides metrics on the port i remember it was 1991 when we came for units right

primitives have a way to act to collect data and that through what they call

a pod monitor i think where is it did you look for it

yeah there you go so basically this cr here right will tell

will tell prometheus to go and look for data from

slash metrics on a specific port that we call like for example this time we we're putting the port http monitoring okay

this cr here will configure primitius to aggregate data from from any

from any any service any pod basically labeled bumblebee

and that has a slash metrics on a port http monitoring we don't have to put

specifically 1991 just we need to label the port itself and that's going to take care of the

data collection

PORTABILITY

all right any other questions we have some from online online okay

yeah um don't mind me sorry if i mispronounce anything you're good

so adam you mentioned that the ebpf program you prepared would work on any

other machine is it do so would work so yeah is it due to btf

what code changes if any need to be made for an e b p f program to be

portable oh okay good um all right so

mainly the portability part of it is based on two things um like if we look

just between uh bcc right and libby leave vpf which let's

take the comparison here because that's the main one that is allowing us to do portability

uh if if we go back to not sure i have any program here running

well let's see if i can do that just give me a second i'm going to go back to the example

ebpf introduction [Music] i'm going to show you something here so

once once you if you create any bcc program you're

dealing directly with with binding methods that talk directly to the kernel okay

where if you do leave bpf you have to use specific libraries right

that kind of creating an abstraction there to allow the portability okay i'm gonna

just try to find you an example here it's gonna be easier to understand uh well this is this is still the first

example of bpc but yeah let's let's first start with bcc here um

when you yeah when you do like when you deal with like vcc code you're gonna write

directly you know you're gonna trigger bcc binding function directly though

in libf and let's be finding this right now

okay let me just find you this

if well maybe i can just do it here okay um

okay maybe not just

oh yeah maybe here all right there you go

one wrong spot oh maybe other rocks but

okay let's do this now

okay so every time you're dealing with uh every time you're dealing with like

you're trying to do portability the way you do it in libya pf is to call this binding command for example this bpf

core read into this one will guarantee the portability across

multiple kernels because it has actually a binding on top it is the one dealing with this uh you

know building a binary for multiple multiple kernels it has this awareness

because we're using an abstraction as a framework on top of just direct code

like in in bcc okay so that's the first part of the portable t the second part of the

portability is once you create that like once you create a lib bpf program it is

running on docker image right it's running on a docker image and so the porsche plus is going

to be there because even if you have another environment it's still a docker image running okay

so that's that's how possibilities is a is assured hope i answered the question

ERROR

why why do i get assertion error expected

zero to be at least one okay

you got the same thing

oh ah okay it yeah you just do uh i think it's uh

when you miss to do a b push there's like this command bumblebee

push basically if you missed that specific command i think it had the same previously it's going to say you know

you have to be at least

okay awesome

well thank you guys uh really really uh

thank you for for joining this session and uh hope to see you soon hope to see you on that we offer a lot of you know

this kind of same uh some same um workshops we we provide them online so

if you join solitaire you're gonna see uh we do that for istio we do that for like eco we have

three we have one which is fundamentals one is like more advanced and one is expert right we do that for envoy we do

that for um like now for abpf so there's multiple certification you can have so

yeah invite to see you to see you there all right thank you guys

you