1. Introduction

It has been found that it is very complex to set up VPN technologies such as OpenVPN[1] and IPsec[2]. They disconnect easily in the absence of further configuration. Ipsec and OpenVPN take a very large amount of time to negotiate reconnections, use outdated ciphers, and have massive code bases of over 400,000 and 600,000 lines of code, respectively, making debugging and auditing a very difficult task which also lead to vulnerability in security because of large attack surface. WireGuard[3]'s design aims to replace traditional VPN like IPsec and OpenVPN. It is simple, easy to implement, easy to use, requires less code, has minimal attack surface and gives high performance. It uses the most secure ciphers achieved through versioning of cryptography packages. It has minimal attack surface as it has a codebase of around 4000 lines of code which is very less as compared to OpenVPN or IPsec, making security audits easier and can be reviewed by single individuals. Wireguard is a prime candidate for kernel implementation as it is easy to audit and has small base code. It is very difficult to debug by implementing wireguard logic in kernel source code or installing it as a loadable kernel module because of complicated infrastructure and abstracted layers. Extended eBPF solves both of these problems. Extended Berkeley Packet Filter allows wireguard to run without having to change the kernel source code or adding additional modules. eBPF[4] is a lightweight, sandbox virtual machine (VM) inside the Linux kernel where BPF bytecode of Wireguard code can be run. eBPF eliminates the need to change kernel source code and streamlines the ability of WireGuard.

2. Aim

The Aim of this project is performance comparison of IPsec, OpenVPN and WireGuard and to evaluate linux kernel support for wireguard. There are 3 ways provided by linux kernel to include functionality of WireGuard in linux.

- We can Install WireGuard as a loadable kernel model(LKM).
- WireGuard is built statically into the kerlen itself.
- Sandbox WireGuard code as an eBPF program in linux kernel.

3. Objective

1. Implement IPsec as an eBPF program and sandbox in the kernel.
2. Implement Wireguard as an eBPF program and sandbox in the kernel.

3. Performance comparison of WireGuard eBPF implementation and IPsec eBPF implementation.
4. Finding best linux kernel support for WireGuard.

4. Related Work

There has already been research into the performance of VPN gateways. In 2017, Jason A. Donenfeld benchmarked wireguard with IPsec and OpenVPN. Wireguard performed better than IPsec and OpenVPN [3]. In another paper, Steven Mackay[20] WireGuard performed better than OpenVPN on multi-core machines. In 2020, Lukas Oswald[22] published a paper in which he found that IPsec with AES-based encryption performed better than WireGuard in a virtualised environment But WireGuard performed better than IPsec in non virtualised environments. Adnan Mohsin Abdulazeez[19] publish a paper in 2020 , in which he compared IPsec with WireGuard on several metrics. He found that IPSec provides good encryption, authentication but IPSec has issue with compatibility, encryption, decryption, tunneling process. WireGuard provides simple design, confidentiality and provides high speed. He also concluded that Wireguard has security hole as it stores users data and also that it is only works with UDP. In 2020, Pudelko[17] implemented WireGuard using DPDK bypassing kernel which performed better than implementation of IPSec. He found that the bottleneck for scaling is multi-core synchronisation. In 2021, Phu Ngyeun[18] evaluates SSL-VPN, IPsec, and WireGuard on the basis of performance and throughput. Wireguard performed better than other VPNs.

5. Problem Statement

There have never been evaluation of existing implementations of Linux Kernel Support of WireGuard. There have also not been implementation of WireGuard using eBPF which is published. In this project, we will be implementing WireGuard using eBPF and IPsec also using eBPF. Existing linux kernel support implementations will be compared with implementation of WireGuard with eBPF.

6. Background

WireGuard[3] implements encrypted virtual private network. It encapsulates TCP, UDP, and other IP traffic inside UDP packets with encrypted content. Traffic is passed over UDP. The cryptographic design is based on the Noise framework. WireGuard supports pre-shared symmetric key mode.. WireGuard uses UDP only and thus does not work in networks that block UDP traffic. WireGuard supports Point-to-Point,Star and Mesh topologies.We can extend Wireguard by scripts and third party programs. WireGuard has a minimal core codebase which improves its stability and security. WireGuard ensures security by limiting the options for implementing cryptographic controls, restricts the choices for key exchange processes, and maps crypto algorithms to a small subset of cryptographic primitives.

IPsec is a secure network protocol suite that authenticates and encrypts data packets to allow secure encrypted communication over an Internet Protocol network between two computers. IPsec includes protocols for establishing mutual authentication between agents at the beginning of a session and negotiating cryptographic keys to use during the connection. Secure data transfers between two hosts (host-to-host), two security gateways (network-to-network), or a security gateway and a host are all possible with IPsec (network-to-host). IPsec uses cryptographic security services to protect communications across Internet Protocol (IP) networks. Data origin authentication, data integrity, data confidentiality (encryption), and replay protection are all supported at the network level.

Extended Berkeley Packet Filter (eBPF) allows sandboxed programmes to run within an operating system kernel. It is used to enhance the capabilities of the kernel in a secure and efficient manner without the need to update kernel source code or load kernel modules. It exposes programmable hooks to the Linux kernel's network stack. We can think of it as a virtual computer that runs on top of the Linux kernel. It enables small applications to be loaded into the kernel and linked to hooks that are triggered when certain events occur. This enables the kernel's behaviour to be customised. The verifier sandboxes the application, ensuring that it can only access allowed regions of memory and that it must exit immediately. The operating system ensures the same level of safety and efficiency as if the code had been natively built using a Just-In-Time (JIT) compiler and verification engine. The combination of programmability and efficiency makes eBPF a natural fit to implement WireGuard. The programmability of eBPF enables implementing wireguard adding wireguard to kernel without ever leaving the packet processing context of the Linux kernel. The JIT compiler's efficiency allows for execution times that are comparable to natively built in-kernel programmes.

The Linux kernel supports BPF hooks in the networking stack which can run BPF programs. We can use these hooks to load BPF programs for WireGuard. Examples of such hooks are XDP(Express Data Path), Traffic Control Engress and Ingress etc. There is a compiler back end for LLVM which can compile wireguard programs written in C into BPF instructions. There is in-kernel verifier which ensures BPF wireguard program is safe to run and There is Just in Time (JIT) compiler which converts the WireGuard BPF bytecode to CPU architecture specific instructions for native execution efficiency. BPF programs can run at various hooking points in the kernel such as for incoming packets, outgoing packets, system calls, kprobes, uprobes, tracepoints, etc.

7. Methodology

7.1 First Phase of project

In the first phase of the project, Existing implementations of wireguard will be reviewed using LKM[8] and built in kernel. Cryptographic tools will be reviewed , their maturity and cryptographic strength will be analyzed, and Wireguard's features will be summarized. Several tools will be created to study different types of WireGuard implementations. Security claims will be reviewed made by WireGuard and ipsec and its suitability will be evaluated for implementing a secure tunnel. We will gather, analyse, specify, and validate the software requirements for different linux kernel support for wireguard. We will be specifying non-functional requirements such as portability, security, maintainability, reliability, scalability, and performance of wireguard’s implementations.

7.2 Second Phase of project

In the second phase of the project, after doing software requirement analysis, we will be defining the specification of different implementations of wireguard. A modular, easy-to-change design that allows fast iteration and experimentation will be implemented. After that we will define the architecture, components and interfaces. We will give architectural design and detailed design. We will be specifying system components, algorithms and the data structures. After design, we will be implementing the wireguard using eBPF. We will also install wireguard as a loadable kernel module and use a kernel build in which Wireguard is baked in.

7.2.1 Wireguard as a loadable kernel modules(LKM)

We will install wireguard using LKM[8]. WireGuard operates as either a Loadable Kernel Module (LKM) or built statically into the kernel itself. DKMS[12] (Dynamic Kernel Module Support) will build the WireGuard kernel moduleas LKM. It will be "in-tree". Wireguard

requires linux >= 3.10. We will be using kernel 5.6.13 version. We will enable CONFIG_NET for basic networking support , CONFIG_INET for basic IP support, CONFIG_NET_UDP_TUNNEL for sending and receiving UDP packets and CONFIG_CRYPTO_ALGAPI for crypto_xor. All above configuration will set with ‘y’.
Above configuration can be configured through ‘menuselect’ option.
CONFIG_WIREGUARD controls whether WireGuard is built as a module, as built-in, or not at all. CONFIG_WIREGUARD will be set with ‘m’. We will build directly from within the the kernel tree[13].

7.2.2 WireGuard statically built in kernel

We will install WireGuard statically in the kernel. CONFIG_WIREGUARD will be set with ‘y’.

7.2.3 Wireguard using eBPF

We will not create bytecode from scratch when implementing WireGuard logic using eBPF. eBPF back end for LLVM (Low-Level Virtual Machine) is already implemented which means that we will used Clang to compile C code in an eBPF object file, which will be the loaded inside the kernel for verification. We will be translating from C to eBPF, it is easier to write code in C.

To implement wireguard using eBPF, we will add a new system call bpf(). It is a multiplexer for a range of different operations. From User space , we will load an eBPF program into the kernel with a call

```
int bpf(wireguard_prog, int prog_id, enum int len).
```

Here prog_id is the number used to identify the program.

Maps will be created or deleted from user space.

```
int bpf_create_map(int map_id, int key_size, int value_size, int max_entries);
int bpf_delete_map(int map_id);
```

To store values into and retrieve values from maps, we can call following functions from user spacel:

```
    int bpf_update_elem(int map_id, void *key, void *value);
    int bpf_lookup_elem(int map_id, void *key, void *value);
    int bpf_delete_elem(int map_id, void *key);
    int bpf_get_next_key(int map_id, void *key, void *next_key);
```

We will add a new form of access to the socket filtering mechanism, allowing a program to directly attach an eBPF program to an open socket:

```
            setsockopt(sock,    SOL_SOCKET,    SO_ATTACH_FILTER_EBPF,    &prog_id,
sizeof(prog_id));
```

Here, prog_id must be the ID number of a program previously loaded into the kernel with the bpf() system call.

7.2.4 Wireguard using Cilium

We will use Cilium Platform to evaluate WireGuard implemented. Cilium (Makowski & Grosso, 2018) supports encrypting traffic between pods in the cluster using the WireGuard protocol. WireGuard allows tuning of parameters such as the key size and cipher suite, which makes it very easy to use. The encryption key pair for each node is automatically generated by Cilium and key rotation is performed transparently by the WireGuard kernel module.

At the foundation of Cilium , we have eBPF, which will insert wireguard in the Linux itself. eBPF is used to provide WireGuard functionality. Besides providing traditional network-level security, the flexibility of eBPF enables security with the context of application protocols and DNS requests/responses.

We can configure Cilium with encryption of traffic between endpoints using wireguard which is impemented using eBPF. We will Installed Cilium(v1.11.2) on kubernetes cluster. We will configure kubernetes to use Cilium as their Container Network Interface(CNI). Cilium will installed on fully functional kubernetes cluster using command

```
cilium install --encryption wireguard
```

7.2.5 Ipsec using eBPF

We will configure Cilium to use IPsec based encryption using Kubernetes secrets to distribute the IPsec keys. All traffic between Cilium-managed endpoints, as well as Cilium-managed host traffic will encrypted using IPsec. We will store Kubernetes secret. Cilium managed nodes will be using IPsec for all traffic. Packets will carry the IP Encapsulating Security Payload.

7.3 Find phase of project

In the Final and third phase of the project, We will collect evidence for different implementations by taking Wireguard measurements from a functioning program in the form of a loadable module, modified kernel build, or eBPF programas or installed on Cilum, as well as for IPsec(eBPF) installed Cilium. We will analyze the requirements, review architecture and the overall design also. We will review the Ciliumdeployment infrastructure and will implement a test strategy.

7.3.1 Comparing different VPN standard solutions

We will compare IPsec and WireGuard on following criteria:

7.3.1.1 Scaling

We will found evidence and provide to prove the hypothesis hat wireguard has better performance than IPsec and OpenVPN. We will also check that if there is bottleneck in synchronization of the security   association (SA) for IPsec. We will evaluate that whether IPsec or WireGuard are able scale to multiple cores. We will investigate the source code to check whether multiple locks to be taken for every handled packet. We will

7.3.1.2 Security

We will evaluate which VPN has smaller base code or which can be configured incorrectly, which will be easy to audit. IPsec offers many encryption options, many of which can be insecure if incorrectly configured, WireGuard limits the available choices to modern, secure encryption methods. We will also check whose code has been verified.

7.3.1.3 Ease of use

We will evaluate the specific requirements for both WireGuard and IPsec and will determine which will be more easier to configure.

7.1.3.4 Comparison of latency and throughput between IPsec and WireGuard

We will compare performance of WireGuard and IPsec on Cilum and other Linux kernel support. We will conduct in-depth performance study and will compare the throughput and latency for both VPN. We will evaluated both protocols to find which VPN can achieve very high throughput for streaming workloads which can achieve better latency, and whether encryption algorithm can be offloaded to the CPU in certain cases.

7.1.3.5 Evaluation of best Linux kernel support for Wireguard

Here we will find best kernel support for WireGuard. We will evaluate eBPF implementation of WireGuard with already established Linux Kernel Support.

For eBPF, We will write a C file which hooks a kernel function which will the then reports that back up to a python script . In LKM, we will build and maintain a kernel module that will communicate to some higher level code.

In this patch set, the CAP_SYS_ADMIN capability is required to use any of the bpf() system call functions. That restriction may limit interesting future uses of eBPF, but there are a number of potential issues (such as the single global ID namespace and resource use limits) that would have to be dealt with before that restriction could be lifted.

8. Deliverables and Evaluation

A code based solution of wireguard using eBPF will be provided which will follow specification of protocol which will be memory safe and free of side channels. We will also be providing computational analysis of the WireGuard protocol. Userspace implementation of wireguard written in C will be also provided. We will be evaluating Linux Kernel support for Wireguard. Wireguard will also be installed as a Loadable kernel module[8] . Wireguard will also be built in the kernel. We will be evaluating whether it is better to implement Wireguard using eBPF than other methods of implementation. The criteria to compare of different implementations will be code compactness, maintainability, ease of update, speed of execution, platform availability, ease of use, hacking vulnerability, and security of the protocol.

9. Technical Aspects

WireGuard through eBPF will be implemented through bpftrace and Cilium. Advantage of Cilium is that they provide abstractions so it is easy to implement the eBPF program directly. BCC will be used to compile and load the WireGuard eBPF program. Libbpf[9] will be used to build the WireGuard ebpf project. LLVM will be used to convert WireGuard eBPF programs written in c into eBPF bytecode. For userspace implementation of wireGuard. We will use wireguard-tools[10]. Wireguard-vanity-address [11] will be used to generate human memorable names to keys to identify the owner of the key. Linux kernel 5.6 will be used to build wireguard with kernel. Drago[12] will be used as configuration manager to configure WireGuard Network. Wg-install [13] will install Wireguard server and generate configuration files for clients for a secure wireguard connection. Algo VPN [14] will set up WireGuard and IPsec VPN.

10. Risk Management and Ethical Considerations

10.1 SWOT Analysis

Strengths	**Weaknesses**
● Have worked in VPN related projects earlier. ● Implemented eBPF programs and worked on Cilium and Kubernetes. ● Have setup ipsec using strongswan as well as linux XFRM modules.	● Limited tutorial for learning eBPF implementations. ● Software development can take long duration. ● Developing and debugging eBPF code is very hard.

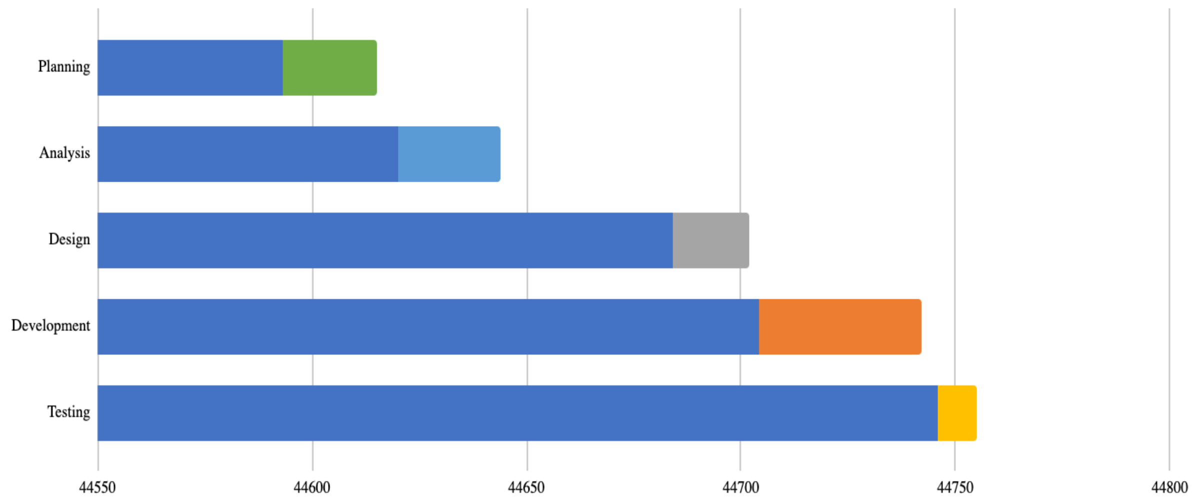
Opportunities	**Threats**
● Current lack of solutions for implementing wireguard using eBPF. ● We can publish this in a high quality research paper.	● This is a hot topic. Someone can implement and give solution already.

10.2 Ethical Considerations

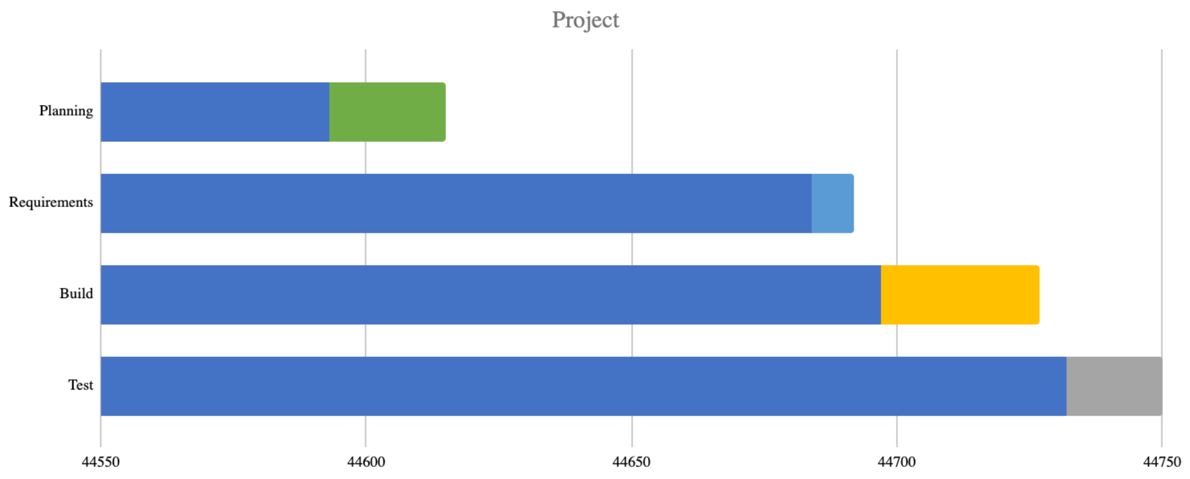
An important part of ethics in engineering is making sure that the project's goals and scope are not misrepresented. More specifically to this project, it is important in this project to make it clear that the goal is not to make a new Wireguard Algorithm or to make change in algorithm, but to implement wireguard algorithm using eBPF. In future after the prototype stage if the eBPF implementation is used by another party , it will be their responsibility to ensure their system is secure against attackers.

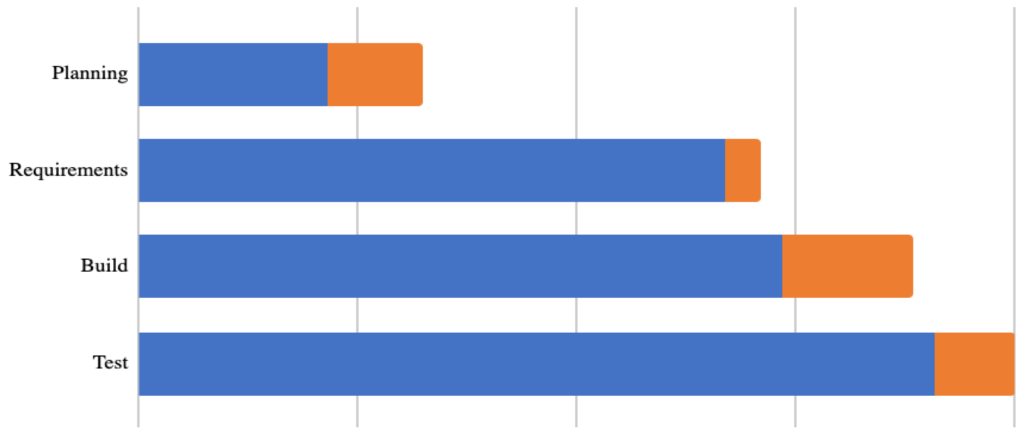
11**. Gantt Chart**

Steps	Start Date	End Date	Duration
Planning	1/2/2022	23/2/2022	22
Analysis	28/2/2022	24/3/2022	24
Design	3/5/2022	21/5/2022	18
Development	23/5/2022	30/6/2022	38
Testing	4/7/2022	13/7/2022	9



Tasks	Start Date	End Date	Duration
Planning	1/2/2022	22/2/2022	22
Requirements	3/5/2022	11/5/2022	8
Build	16/5/2022	15/6/2022	30
Test	20/6/2022	8/7/2022	18





REFERENCES

1. Feilner, M., 2006. *OpenVPN: Building and integrating virtual private networks*. Packt Publishing Ltd.
2. Doraswamy, N. and Harkins, D., 2003. *IPSec: the new security standard for the Internet, intranets, and virtual private networks*. Prentice Hall Professional.
3. Donenfeld, J.A., 2017, February. Wireguard: next generation kernel network tunnel. In *NDSS* (pp. 1-12).
4. Vieira, M.A., Castanho, M.S., Pacífico, R.D., Santos, E.R., Júnior, E.P.C. and Vieira, L.F., 2020. Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications. *ACM Computing Surveys (CSUR)*, *53*(1), pp.1-36.
5. Dowling, B. and Paterson, K.G., 2018, July. A cryptographic analysis of the WireGuard protocol. In *International Conference on Applied Cryptography and Network Security* (pp. 3-21). Springer, Cham.
6. Ferguson, N. and Schneier, B., 1999. A cryptographic evaluation of IPsec.
7. Abdulazeez, A., Salim, B., Zeebaree, D. and Doghramachi, D., 2020. Comparison of VPN Protocols at Network Layer Focusing on Wire Guard Protocol.
8. De Goyeneche, J.M. and De Sousa, E.A.F., 1999. Loadable kernel modules. *IEEE software*, *16*(1), pp.65-71.

9. Greppi, P., 2006. LIBPF: a library for process flowsheeting in C++. In *Proceeding of the International Mediterranean Modelling Multiconference* (pp. 435-440).

10. https://archlinux.org/packages/?name=wireguard-tools

11. https://aur.archlinux.org/packages/wireguard-vanity-address

12. https://github.com/seashell/drago

13. https://github.com/its0x08/wg-install

14. https://github.com/trailofbits/algo

15. Domsch, M. and Lerhaupt, G., 2004, July. Dynamic Kernel Module Support: From Theory to Practice. In *Linux Symposium* (p. 187).

16. https://www.wireguard.com/compilation/

17. Pudelko, M., Emmerich, P., Gallenmüller, S. and Carle, G., 2020, June. Performance analysis of vpn gateways. In *2020 IFIP Networking Conference (Networking)* (pp. 325-333). IEEE.

18. Hai, P.N.P., Hong, H.N., Quoc, B.B. and Hoang, T., 2021, October. A Comparative Research on VPN Technologies on Operating System for Routers. In *2021 International Conference on Advanced Technologies for Communications (ATC)* (pp. 89-93). IEEE.

19. Abdulazeez, A., Salim, B., Zeebaree, D. and Doghramachi, D., 2020. Comparison of VPN Protocols at Network Layer Focusing on Wire Guard Protocol.

22. Dekker, E. and Spaans, P., Performance comparison of VPN implementations WireGuard, strongSwan, and OpenVPN in a 1 Gbit/s environment.

Vancouver