WireGuard: Trusting Network Layer Authentication in a New Protocol

Wesley Jones
Iowa State University
CPRE Network Security, 530
Ames, US
wesleyj@iastate.edu

***Abstract*—WireGuard: a software layer virtual private network you can use in the modern era. WireGuard is built into the Linux kernel and supports necessary extensible options with features like pre-shared keys, a configurable access control layer, and flexibility of functioning on top of virtual interfaces. Can we trust the network layer authentication (Layer 2) provided by the routing and handshake in WireGuard?**

***Index Terms*—WireGuard, VPN, cryptography, network authentication**

I. Introduction

WireGuard is a virtual private network (VPN) protocol and subsequent software utility that deviates from some of the expectations that standard technologies in this space have followed since their inception. The protocol acts as a method of communication to first negotiate a trusted connection between two or more endpoints. In order to instantiate this trusted connection, WireGuard relies on an existing software layer. It was introduced in 2016 to a small group on a popular Linux news site and mailing list. As the original software was defined by its creator it features encryption and authentication. These were tunneled using a virtual network interface established within the Linux kernel [1]. WireGuard should not be confused with a traditional VPN as it does not rely on the same process of transmitting and receiving packets that a traditional VPN does. Instead of waiting for the kernel to speak to the application layer which then presents the payload to the userspace, WireGuard implements an interface directly into the Linux kernel. Using a software layer to manage trust on top of this interface has little effect on the speed gains – transmitted and received packets are able to be presented at near real-time wire speeds [2].

The creator of WireGuard, Jason Donenfeld, focuses on the dysfunctional state of Internet Protocol Security when discussing motivations for the software and protocol. Donenfeld suggests that while the principles of the Internet Protocol Security protocol are sound, the implementation is challenging and leads to unsafe systems because of the inherent hurdles in the technology [3]. The inclusion of WireGuard into the Linux kernel has been swift. Donenfeld provided a short explanation of the software to a popular Linux mailing list in the summer of 2016 and by March of 2020 the software had been merged into the Linux 5.6 kernel [1], [4].

WireGuard is a serious tool, but it is moving fast. The philosophy seems sound, but without enough "real-world" implementation can we trust *philosophy* alone?

Motivation

I have watched WireGuard's progression as it moved from a suggested kernel component to a commonly available component of networking systems. After reviewing some of the current writings on the topic I am left with questions that can be summarized into one: can we trust the network layer authentication (Layer 2) provided by the routing and handshake in WireGuard? I intend to provide a in depth background on the functionality of WireGuard since it is not considered traditional networking yet and I will also provide some analysis on common discussions surrounding authentication security in WireGuard.

Terms

- **Initiator**: The WireGuard participant who begins the connection. This role is flexible and can change when a connection is updated [2].
- **Responder**: The WireGuard participant who accepts the connection. This role is flexible and can change when a connection is updated [2].
- **Handshake**: The process WireGuard uses to create a trusted connection. It involves the initiator sending the responder enough information to complete a Diffie-Hellman key exchange process [2].
- **Cryptokey Routing**: A WireGuard specific method of network layer trusted routing by relying on the previously acquired handshake key to build routing connections [2].
- **Roaming Mischief**: A name given to a suggested vulnerability in the WireGuard protocol. This vulnerability deals with changing the flow of traffic during WireGuard connection process.

II. Methodology

In my reading, I sought to answer a few questions. While much of the answers are subjective – the resulting analysis is put forth to be objective in presentation with some subjective discussion at the end.

1) Have the provided referenced materials fully explored WireGuard in a way that provides enough background on its network layer trust?

2) Does roaming mischief pose a threat? Is there potential issues with trusting the cryptokey routing process if the network authentication layer can be tampered with?
3) Should post-quantum threats be considered when assessing WireGuard's security offerings?
4) Is WireGuard a *wise* tool? Does it account for long-term implementation needs?

III. Literature Review

WireGuard is still young. There are a few notable contributors to the study of this new protocol and its implications in the network security space. This analysis combines many previously uncombined resources. There are a handful of technical discussions sourced directly from Donenfeld's online presence. These primary sources can fit in well against the additional analytical articles. These articles typically rely on the WireGuard white paper with technical background regarding information security concepts from their relevant authorities. The most common focus is on cryptography.

A. Works by Jason Donenfeld

Donenfeld is the creator of WireGuard. His white paper is referred here many times – a trait shared among several of the other sources [2]. The presentation and justification of the technology is sensible. Donenfeld supplies two other contributions to this paper from his mailing list, [4] and [5]. Donenfeld provides more background through his posting on LWN.net, a popular Linux forum [1]. Donenfeld is a prolific developer and has contributed to cryptography and security systems as an independent software developer and researcher [6]. Donenfeld focuses on justifying the security of WireGuard.

B. Contributions involving Peter Wu

References [7] and [8], involve Peter Wu, a research student who has provided a thorough analysis of WireGuard. Wu provides a broad overview of the WireGuard protocol, dissecting the protocol, software, weaknesses, and countermeasures, as well as some discussion about the post-quantum security improvements [7]. Wu teams up with a group of researchers to propose a tweak to WireGuard in "Tiny WireGuard Tweak". This paper focuses on the state of risks quantum system pose to encryption. By capturing data now and storing it until quantum decryption becomes available, it puts current invulnerable systems at risk. It might be tempting to dismiss this proposal, but Wu and colleagues argue that the risks are closer than comfortable. The tiny tweak in question: add a requirement for pre-shared keys [8]. Donenfeld acknowledges this from the inception of WireGuard – implementers of the protocol should take note [2].

C. Dowling and Paterson

Continuing with the cryptography analysis Dowling and Paterson explore the viability of the cryptography system imposed in WireGuard. Their review of the protocol supplies authority to the previous topics offered by Wu and others.

Fig. 1. Example of commands to set up a virtual Linux network interface.

```
$ ip link add dev wg0 type wireguard
$ ip address add dev wg0 10.19.12.3/24
$ ip route add 10.0.0.0/8 dev wg0
```

They assert this review to be the first for WireGuard, a high-level security review. A great deal is condensed into a succinct discussion showing that the duo's assessment of the protocol security; ultimately urging the practice of expanding the key exchange packets to outside the same scope as the session keys exchange. Their premise for this assertion relies on a modified process of testing WireGuard since they acknowledge the implementation design does not allow them to separate authentication (session) keys from handshake keys. This aspect of trust changing from handshake to authentication phase was of particular interest in writing this paper [9].

D. Venter and Eloff

Venter and Eloff approached information theory taxonomy methodically many years before the creation of WireGuard. They cataloged a selection of works at the time and itemized which topics were covered in the conversation surrounding security technology. This approach has provided a future-proof work that offers context to the approach of deciphering which attack surface to expect in differing applications. Their care to separate *proactive* approaches from *reactive* helps to define the current state of security considerations within networking security (nearly two decades later!). Every good assessment of an application starts with a firm grasp of the taxonomy – something this reading does well. I was especially drawn to their categorization of VPNs as a tangent from cryptography in general, something WireGuard emphasizes. [10].

IV. How it Works

Overview

Review these brief steps. Further information follows that will detail the components and ultimate expound on the steps to provide situation context.

1) Create an interface and add routable networks to it. It can be done using operating system kernel APIs or in the newer (5.6) Linux kernels with the actual `wg` interface option added [2], [4]. This implementation step deals with bringing the protocol into the networking stack of your device. You can see the examples of these commands on modern Linux distributions in Figure 1.
2) Create client connection. This involves instantiating the trust between clients. The WireGuard software layer works to extend a set of *internal* routing methods (called "Cryptokey Routing") based on this shared trust [2]. This implementation step is the authentication phase of WireGuard.

Fig. 2. A simple diagram of the 1.5 RTT during the WireGuard handshake process [7].

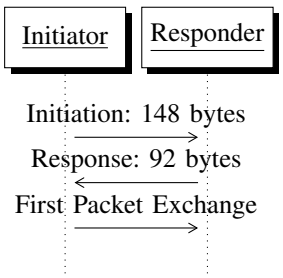
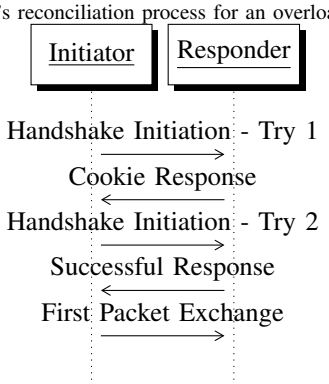


Fig. 3. WireGuard's reconciliation process for an overloaded responder [2].



Virtual Network Interface

WireGuard's integration with the kernel is purely a speed gain. The route-based VPN functionality could exist in userspace (as is common in other commercial and freeware VPN solutions) [1]. This kernel integration is trivial to the functionality of the *networking* aspects of WireGuard, so we will set it aside.

Further research on exploiting kernel features to horizontally attack WireGuard might be worthwhile. WireGuard in essence is a virtual interface in the kernel. The process to create a socket to receive and transmit from involves utilizing builtin tools from the Linux kernel. Software exists that extends WireGuard into BSD, MacOS, and Windows. In the sense of functionality these all work the same. Treat the functional aspects of "Linux virtual interface" to be all-inclusive to the further development of WireGuard interactions. Security implications per-OS is worth another paper.

Communication

WireGuard optimizes for efficiency. Messages are encapsulated in a Universal Datagram Packet (UDP) and submitted to an exposed port, **initiator** to **responder**. The handshake to negotiate this connection is efficient (1.5 RTT, similar to Transmission Layer Security 1.3) [7]. This can be seen during the negotiation phase of the protocol between initiator and responder. The creator chose to offload the burden of establishing trust during the connection phase by requiring the key exchange to occur out of band. This exchange requires knowledge of the the connection point (network address or domain name) and a static public key that will remain unchanged, ideally generated and shared over a trusted network [8]. The key is generated by instantiating the WireGuard software layer on a client (remember WireGuard is flexible between clients and servers – the *client* referred to here could be a server.) See Figure 2 for a visual of the packet exchange during a normal initiator and responder communication process. This handshake exchange works by the initiator sending just enough information to create a Diffie Hellman key exchange. The responder completes the request. The *first packet exchange* provides a Transmission Control Protocol-like acknowledgment (WireGuard is only UDP, do not misread this comparison) to complete the trust between the pair. This last packet exchange is debatable, since completing the trust exchange in such a fashion implies that each party already trusted each other [9].

In situations where the connection is unreliable the parties remain silent until a successful request and response exchange is completed. Unless the responder is under load there will not be any effort to track or ban connection attempts. A special process exists for situations where the responder is unable to respond. The circumstances could occur as an attempt at denial of service (simply with empty one-half RTT communications from an initiator) or simply natural load experienced by the responder [2]. Wu notes that constraints within the Linux implementation along with more constraints in the Go language software layer can result in a situation where high traffic environments will rely on a retry response method [7]. WireGuard protocol contains a process as follows [2]:

- Initiator sends first packet for handshake negotiation.
- Responder is overloaded. Responder generates a cookie with a time to live of 120 milliseconds. It provides the initiator with this.
- Based on the standard settings for the WireGuard the initiator will retry a connection to the specified responder, providing the cookie response data.

You can see this process detailed in Figure 3.

Assuming an environment has this configuration implemented correctly, two WireGuard clients will be able to communicate across the Internet relying only on an exposed UDP port. To handle reliable routing WireGuard introduces cryptokey routing. Donenfeld describes the `wg` interface as an interface that WireGuard performs cryptography over – he is highlighting the seamless integration of cryptography into routing and thus authentication portion of networking [1].

Recall that the out of band initialization is required to use WireGuard. This process establishes a public key listing on each member of a WireGuard connection. The public key is added a routing table that can be utilized for access control or any other expected network routing needs [2]. Since the key is required to decrypt the incoming message from the initiator, network layer authentication is established while access control mechanisms operate on the appropriate routing. This does not serve to add *more* authentication, but,

interestingly, contributes to the general trust in the routing since the routing table entry can contain correlated information that has previously been verified.

Donenfeld provides a simple step-by-step process of this send/receive chain. There are several points at which data is dropped should it deviate from to standards. That includes things like non-internet protocol packets within the received data, packets destined to an unknown endpoint, and of course access control restrictions on packet sources and destinations based on the offered cryptokey [2].

It is tempting to keep digging into WireGuard's routing mechanics, but despite the protocol's typical simplicity there is a level of safeguards and intricacies here that warrant better discussion outside of this paper.

V. Threat Surfaces

Security in the Protocol

Peter Wu discusses this UDP relationship between initiator and responder – continuing to list the security considerations that come into play with the protocol running within UDP. UDP might be thought of as a bygone protocol, but it is uniquely qualified to handle WireGuard traffic. Technical considerations come into play since the robustness offered by the now more common, Transmission Control Packets in favor of UDP, actually pose a threat to VPN and WireGuard traffic. WireGuard *must* handle common UDP simplicity, such as packet ordering and limited packet trust [7]. The secure encapsulation process helps to ensure a network level security, but WireGuard smartly pairs this protocol with an application layer to extend network authentication up the stack. Using application layer and network layer security in tandem ensures a robust proactive approach to network security [10].

Those familiar with managing network security on the user connection side might know a trick to ensure a safe environment: block everything. Users who *cannot* access bad decisions (fraudulent or insecure sites) *will not* access bad decisions. Stepping back from this diminutive approach to security for humans, protocols can be secured using a similar cut-throat methodology.

- A unauthenticated request to the open UDP port will not result in *any* response [2].
- A packet field that is either incorrectly filled (including overfilled) or not filled is discarded [7].
- Special care was taken for timeouts and overload scenarios. WireGuard implements a cookie-based approach to these scenarios: only approving registrations for a short duration provided the party has the correct token (cookie) [2].
- A limited set of cryptography options. Instead of a standard plug-and-play approach to cryptography offerings, WireGuard insists on a small set (ChaCha20Poly1305) [2].

Quantum Analysis

A common thread is to assume a state (or any sufficient large and capable) actor would collect and store captured data for later quantum decryption. This assumes the standard negotiation encryption methods of ChaCha20Poly1305 is able to be broken within the time frame of quantum cracking existing. Setting aside the complications of predicting the future, our security first decision policies would tell us to *assume* we will enter a post-quantum era where all previous encryption methods are crack-able. Appelbaum, Martindale, and Wu discuss this very scenario in their suggestion for WireGuard: "Tiny WireGuard Tweak". They explain that historic captures, even when currently unbreakable can become a risk when quantum computing is available to literature hashes, keys, and signatures [8]. The conclusion to their analysis: *use pre-shared keys (PSK)*. This tweak is subtle to implement and since the age of internet has adjusted to widely distributed Transport Layer Security certificates and keys, it is reasonable to expect our trusted protocol layers would be shored up with similar treatment. WireGuard comes with support for PSKs – something Donenfeld highlights in his initial discussion on the protocol [2]. The proactive methodology of both PSKs and a very decisive cryptography standard can help to ensure WireGuard's longevity [10]. It might seem that the quantum era poses a looming threat to non-PSK, but by embedding the technology into the kernel and tightly controlling the software packages the protocol can be changed via normal operating system updates. Clever packaging systems would allow for gradual adjustments to this security baseline; something currently done in systems like SSH by deprecating cipher suites, except by explicit request from the client or peer [11].

Authentication

Donenfeld has been open about a potential corruption of the WireGuard protocol when discussing the roaming component of WireGuard. In a note to the WireGuard parent organization mailing list Donenfeld argues the exploitation of roaming functionality is *at most* trivial. He presents the process to exploit the implementation, but argues the real-world functionality does not gain the exploiter anything [5]. Dowling and Paterson do not explicitly disagree with this assessment, but from their analysis the entanglement of the handshake and authentication/session keys is an issue. They focus on the the reliance of the security on a first message sent by the initiator. This process can jeopardize the entire handshake [9]. Their concern could be founded.

The likelihood of this authentication being compromised is low, but the cascading risk of a granted authentication is real. The packet header contains *verification* data, but it does not contain a second factor of authentication (remember the efficiency of a 1.5-RTT handshake!) [2]. The reliance on public key cryptography routing means that a packet that is trusted is immediately routed. This process is expected on network routing equipment, but it does depart from traditional virtual interfaces, since these typically will rely on either a full-fledged software VPN or the local firewall and route settings for access control limitations [7] and [10].

A final contention, well publicized, deals with roaming mischief. Donenfeld addresses this concept head-on in a mailing

list post [5]. The assumed process is this:

1) Alice initiates a connection to Bob. Eve intercepts this (an active man in the middle attack). Eve then forwards the message to Bob with her IP as the source of traffic.
2) Bob responds, again this is captured by Eve who forwards the message back to Alice, though again, changing out Bob's address for her own. A successful handshake will be complete since Alice and Bob *believe* they are communicating directly with each other. Eve is able to snoop on the traffic however – acting as a silent proxy.
3) Alice might roam off the original network and again reconnect later still relying on a forward with Eve to get to Bob. At some point, a session timeout will remove Eve from the negotiation process.

In that scenario, Alice and Bob would not necessarily know they are being snooped on. Eve might feel clever with the traffic captured, but since this is encrypted using a Diffie Hellman exchange there is not any immediately tangible value in it. Donenfeld is quick to point out that intercepting traffic is not a unique trait on networks – calling into question the viability of the attack as a whole [5]. In this process, the suggestions by Appelbaum, Martindale, and Wu serve to further guard traffic against post-quantum analysis [8]. There is no reason to expect networks to be entirely free of traffic monitoring either. In an idealistic sense roaming mischief would be resolved, but as such the standard of un-monitored traffic seems unlikely.

VI. Discussion

On available literature content

I set out with a concern that this protocol being new enough might not offer enough insight to potential threats. I was concerned that the coverage of it would not hold up to scrutiny. I believe it does hold up. I found several of the cryptography sources very in depth for their field of interest, but a correlation with this discussion is reliable when applying trust to the network layer trust.

I suspect that there are many more areas to cover regarding WireGuard. I think the literature should focus on WireGuard's implementation *within* the Linux kernel as well as WireGuard's implementation within varieties of networking situations. These questions for the future might be:

1) Does WireGuard scale well within crowded networks?
2) Analysis of the administrative benefits of WireGuard versus existing common similar protocols and software.
3) Is the kernel implementation posing long-term risks for maintaining security by extending the network interfaces with another software layer package?

On authentication security and cryptokey routing

I am inclined again to agree with Donenfeld's assessment of the risks posed by a threat misusing the authentication steps of the protocol as being lackluster [5]. The threat is contrived at best. If there was a verifiable path to exploitation (*even* of compromising components of the network security) I would consider the roaming mischief proposal a potential issue, but after reviewing the process I fail to see validity. On paper the process to compromise the trust layer looks weak and in practice it is virtually inaccessible. I have laid out an attack such as this above. Stepping through what data is exposed results in a meager return. The same Diffie Hellman exchange we rely on for Transport Layer Security and Secure Shell Protocol is more prone to expose data in a WireGuard communication.

I included cryptokey routing within the discussion surrounding network authentication. The reliance of out of band enrollment to help build a table of routable destinations and allowed sources is clever in my opinion. By populating the route table with cryptokeys associated with IPs/ranges of IPs, a layer of abstraction is achieved. This layer serving as *both* the unique identifier in the table lookup and for access control is a valuable level of efficiency. Administrators should be able to assess WireGuard's functionality easily – thus improving the overall security of the protocol.

I recommend a more in depth discussion regarding potential ramifications in imposing a software layer route on top of the standard route tools built into the network.

On quantum analysis

I do not see anything remarkable in WireGuard's coverage of a post-quantum threats. The interest in this field is a "hot topic" as the horizon almost certainly contains quantum computing, but for a specific protocol that encapsulates into existing architecture, the threat is unremarkable. WireGuard implements a pre-shared key optional system — a methodology widely available throughout networked systems today. As Donenfeld highlights, this is the endgame for today's mitigation against a potential future threat. [2]

In context of WireGuard's inherent security benefits (or lack thereof), I do not place quantum threats in the category of viable. I do not seek to oppose the analysis and suggestions provided by Appelbaum, Martindale, and Wu when offering the tweaks to the protocol; they are right to suggest a requirement of the PSK integration [8]. Assessing the network security dilemma of the post-quantum world is a massive undertaking and almost certainly if done today, would be premature. This is well out of scope of this paper.

On implementation

Global networking has been around long enough that expectations for new entrants into the system networking stack are expected to meet a litany of baseline standards. WireGuard accounts for so many things since its inception is an honest iteration of long-term existing models. WireGuard does not seek to reinvent *networking* – and by extension networking authentication – instead it continues to develop the marketplace of tools at the administrator's disposal. This is a wise position.

VII. Conclusion

Recent world events have changed the perspective of the remote network access needs of computer users everywhere.

No network should be expected to be so isolated that all authentication relies on physical presence. Those wishing to remotely access networks will turn to VPNs and WireGuard.

I have confidence that a network administrator will encounter WireGuard over the next ten years. Short of sounding like a market forecaster, I would propose that in an environment where computing is centralized to large global data centers organizations will rely on point-to-point communication that has typically been owned by traditional VPNs. The efficiency and ease at which WireGuard can be implemented and maintained could prove to be a major success on distributed infrastructure networks.

WireGuard is built into the Linux kernel and supports necessary extensible options with features like pre-shared keys, a configurable access control layer, and flexibility of functioning on top of virtual interfaces. These features should ensure the protocol will remain for the long haul.

REFERENCES

[1] J. A. Donenfeld. (2016, June) Wireguard: a new vpn tunnel. [Online]. Available: https://lwn.net/Articles/693015/

[2] ——. (2020, June) Wireguard: Next generation kernel network tunnel. [Online]. Available: https://wireguard.com/papers/wireguard.pdf

[3] ——, "Wireguard: Next generation kernal network tunnel," in *Proceedings 2017 Network and Distributed System Security Symposium*. Reston, VA: Internet Society, 2017.

[4] ——. (2020, March) [announce] wireguard 1.0.0 for linux 5.6 released. [Online]. Available: https://lists.zx2c4.com/pipermail/wireguard/2020-March/005206.html

[5] ——. (2017, November) Roaming mischief. [Online]. Available: https://lists.zx2c4.com/pipermail/wireguard/2017-November/001957.html

[6] Jason donenfeld. [Online]. Available: https://www.blackhat.com/us-18/speakers/Jason-Donenfeld.html

[7] P. Wu, "Analysis of the wireguard protocol," Master's thesis, Eindhoven University of Technology, Eindhoven, Netherlands, June 2020.

[8] J. Appelbaum, C. Martindale, and P. Wu, "Tiny wireguard tweak," in *Progress in Cryptology – AFRICACRYPT 2019*, J. Buchmann, A. Nitaj, and T. Rachidi, Eds. Cham: Springer International Publishing, 2019, pp. 3–20.

[9] B. Dowling and K. G. Paterson, "A cryptographic analysis of the wireguard protocol," in *Applied Cryptography and Network Security*, ser. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 3–21.

[10] H. Venter and J. Eloff, "A taxonomy for information security technologies," *Computers & Security*, vol. 22, no. 4, pp. 299–307, 2003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404803004061

[11] Openssh legacy options. [Online]. Available: https://www.openssh.com/legacy.html