7-28-2021

A WireGuard Exploration

Alexander Master
Purdue University, amaster@purdue.edu

Christina Garman
Purdue University, clg@purdue.edu

A WireGuard Exploration

Alexander Master, Christina Garman
Center for Education and Research in Information Assurance and Security (CERIAS)
Purdue University – West Lafayette, USA
Email: {amaster}{clg}@purdue.edu

***Abstract* – Internet users require secure means of communication. Virtual Private Networks (VPNs) often serve this purpose, for consumers and businesses. The research aims of this paper were an analysis and implementation of the new VPN protocol WireGuard. The authors explain the cryptographic primitives used, build server and client code implementations of WireGuard peers, and present the benefits and drawbacks of this new technology. The outcome was a functional WireGuard client and server implementation, capable of tunneling all Internet traffic through a cloud-based virtual private server (VPS), with minimal manual configuration necessary from the end user. The code is publicly available.**

I. INTRODUCTION

Given the prevalence of Internet-based services and communication today, much of the effort to maintain individual autonomy of personal information focuses on Internet Protocol (IP) based applications. While many of the original protocols that became building blocks of the Internet were not designed with security or confidentiality in mind, newer software can often allow individuals to manage the amount of data being associated with them. Web proxies, VPNs, and encryption protocols can all play a part in making the Internet a more secure and private place to conduct ourselves. As such, a thorough understanding of the cryptographic primitives and their implementation in modern Internet routing will benefit the research community as well as everyday users.

For the purposes of this paper, the authors sought to perform a detailed analysis of the WireGuard VPN implementation. The research includes public key encryption, stream cipher, message-authentication code, and hashing functions as they are implemented in the publicly available versions of WireGuard as of 2021. The authors also deployed server and client instances, simulating how a commercial VPN service provider could establish communications with customers. The code for both implementations is openly available at https://github.com/smolbytes/wireguard-deployer. This study intends to illuminate this state-of-the-art protocol, as well as how it compares to current market leaders such as OpenVPN and IPSec. The associated presentation also aims to inform how these protocols can be practically applied.

II. BACKGROUND

The WireGuard project was started by Jason Donenfeld in late 2016, according to the earliest commits on https://git.zx2c4.com. Jason describes how his idea was to create a replacement for OpenVPN and IPSec in his talk given at Black Hat 2018 at Mandalay Bay Casino, Las Vegas [1]. Overall, the protocol operates at layer 3 of the Open Systems Interconnection (OSI) model, commonly referred to as the network layer. This is in contrast to IPSec, which offers layer 2 functionality. The WireGuard team began with the concept of a Linux network interface and built their protocol around that concept [2]. The protocol supports IPv4 and IPv6 traffic outside and inside the tunnel. WireGuard uses what Jason describes as "modern, conservative" cryptographic principles and primitives. He also describes the protocol as "opinionated," meaning that it provides the exact cipher suite and key exchange mechanisms to make WireGuard work. It does not allow for negotiation or administrator configuration of the underlying protocol without fundamentally redesigning it. The WireGuard team also places emphasis on the simplicity and auditability of the protocol. The intent is that a single researcher, or a small team of security professionals, can easily audit the entire code base. The Linux implementation of WireGuard has under 4,000 lines of code, significantly less than other competitors in the VPN space. Additionally, the authentication model is similar to that of Secure Shell (SSH) and its authenticated_keys; any administrator that knows how to administrate with SSH can at least fundamentally understand WireGuard's authentication. The security design principles the WireGuard team followed when designing the protocol are below [2].

A. Security Design Principles

1. **Easily auditable**. The Linux kernel implementation of WireGuard is under 4,000 lines of code, and can be reviewed by an individual person or a small team as new versions are released. In contrast, OpenVPN has over 110,000 lines of code, and IPSec (when instantiated using the XFRM and StrongSwan packages) amounts to over 400,000 lines of code. This makes performing a code review particularly daunting. The potential downside of this approach is that the protocol only implements the core functionality of allowing two peers to communicate with each other. Other desirable features common in VPN technologies, such as username and password authentication,

must be implemented on top of or around WireGuard.

2. **Simplicity of Interface**. The WireGuard team provides a few helper functions, and they are the only things needed to set up a WireGuard tunnel quickly. Configurations can be scripted and automated using these lower-level functions. Because WireGuard is a virtual network interface at its core, the interface and the traffic to and from it can be manipulated with standard operating system tools.

3. **Static Fixed Length Headers**. This feature makes WireGuard traffic easily observable, and eliminates the need for parsers because the headers of the WireGuard packets will always be formed the same way (or dropped if they are malformed). The security advantage of this principle is it eliminates an entire class of parser vulnerabilities from consideration when analyzing the protocol. The downside is the traffic is easily identifiable when traversing IP networks. If an Internet Service Provider (ISP) or nation-state wanted to restrict VPN traffic from its users, WireGuard traffic is easily identifiable by deep packet inspection and would be difficult to obfuscate.

4. **Static Allocations and Guarded State**. All state required for WireGuard to operate is allocated during configuration. Also, no memory is dynamically allocated in response to received packets – eliminating further classes of vulnerabilities associated with memory allocation.

5. **Stealth**. Interestingly, part of the inspiration for the WireGuard project came while Jason Donenfeld was working on a stealth rootkit project. He realized that many techniques required for stealth are also useful for tunnel defensive measures. For example, WireGuard is not a "chatty" protocol. If the peers have no communication to transmit, the tunnel is silent, unlike other VPN technologies that make liberal use of keep-alive messages. Additionally, non-authenticated traffic – traffic that does not match an appropriate tunnel IP or associated public key - is simply dropped and not acknowledged. Finally, being hosted on a UDP listening service makes the presence of a WireGuard peer difficult to detect with port scanning, as opposed to the ease of finding TCP-based services on the Internet.

6. **"Solid" Cryptography**. The WireGuard team focused on providing strong cryptographic key exchange, as well as strong underlying symmetric encryption for the transmission of data over the tunnel, which is explained in more detail below.

III. RELATED WORK

The authors are aware of two prominent academic studies related to WireGuard in the literature. The first is [3], which was included in conference proceedings at the International Conference on Applied Cryptography and Network Security in 2018. The study was unable to conclusively prove the security of the key exchange without making a small modification to the code and making particular simplifications in their analysis. The study received some criticism from Jason Donenfeld, who explained that the U.K. researchers were using a more traditional extended Canetti-Krawczyk (eCK) model, which relies on having a further separated key exchange. The authors express that the attacks presented require particular capability on behalf of the attacker, and regard it as a barrier to strong security proof as opposed to a practical attack.

The second analysis in the literature is [4], conducted in 2019, which analyzes the WireGuard protocol "as-is" using an authenticated and confidential channel establishment (ACCE) model [5]. The paper provides formal proofs for correctness, message secrecy, forward secrecy, mutual authentication, session uniqueness, and resistance against key compromise impersonation for WireGuard.

Regarding practical implementation, numerous resources are available on the open Internet for guides to setting up WireGuard peers in various scenarios. The WireGuard website [6] itself is an excellent resource for quick-start and installation references on various platforms. Many commercial VPN service providers have also begun to implement WireGuard into their platforms. The authors' implementation was aimed primarily as an education tool in a university setting to demonstrate the functionality of WireGuard, and to show that it can be done with reasonable ease by a user without vast programming experience.

IV. THE PROTOCOL

A. Cryptographic Primitives

As of 2021, WireGuard utilizes the following cryptographic primitives:

1. Noise Protocol Framework
2. Curve25519 within the key exchange for Elliptic-Curve Diffie-Helman (ECDH)
3. ChaCha20 for symmetric encryption session keys
4. Poly1305 for encryption authentication
5. BLAKE2 for cryptographic hashing

B. Establishing Secure Communications

The WireGuard whitepaper, published in the Distributed System Security Symposium NDSS 2017, presents the protocol and each aspect of the communication in granular detail [7]. The following is a summary for a layperson with a foundational understanding of cryptography concepts.

WireGuard begins with the assumption that two computers wish to communicate securely with one another over an untrusted (or otherwise) IP route. Public and private key pairs, utilizing Curve25519, are generated by each peer. By some other mechanism, the public keys of each peer are exchanged. Each peer then assigns an IP address, or IP address range, that their partner peer is allowed to use within a WireGuard tunnel. These IP addresses are generally RFC1918 addresses in the reserved private IP address space. They use IP addressing that

will not conflict with the logical networking of either end of the tunnel, in whatever environment a peer finds itself connected. Either peer may serve as an "initiator" or a "responder" at any time. When a peer wants to initiate communication, a "First Message" is sent from the initiator to the responder. It includes the message type, some reserved zero bytes, the sender address, an ephemeral ECDH key, the sender's static public key, a timestamp, and two MAC values (122 bytes total). Assuming the message is received, the responder will look up if the sender's public key is present. If so, it generates a "Second Message" with message type (0x2), some reserved zero bytes, sender address, responder address, an ephemeral ECDH key, and two MAC values (overall shorter than the original First Message). At this point, both peers have all of the information they need to use the Noise Protocol to generate a shared ChaCha20 session key. The NoiseIK function of Noise Protocol uses a form of "triple D.H.," in which a mixture of the static public keys and ephemeral keys generated results in a one-time-use encryption to encrypt the value of the session key *k*, and the two peers can begin communications immediately after the 1 round trip time (1-RTT) transaction takes place. This exchange enables forward secrecy, in that exposure of keys or future cracking of the encryption of the ECDH handshake only results in one session being compromised, enormously increasing cost to attackers and preventing the decryption of large sessions, even if captured and stored by an adversary. From here, WireGuard communications can begin, and each node can access resources from the other side of the tunnel as if it was connected to that broadcast network space. Plaintext messages will be encrypted by the WireGuard interface, routed out of the externally facing network interface en route to its peer, travel through the external interface of its peer, and be decrypted at the WireGuard interface on the receiving end [7].

V. IMPLEMENTATION

While WireGuard documentation prefers to refer to each node as a peer, we refer to each node as a client or server for our implementation. Our scenario aims to simulate a star network topology consisting of a central trusted server to which one or many clients may connect and funnel their IP traffic. We simulate a commercial VPN provider by enabling access to shared networking resources, allowing customers to manage the attribution of their originating traffic while using the Internet. The server is designed to be run on CentOS, while the client currently supports Ubuntu, Debian, and Fedora desktop Linux environments. Both are freely available at [8] for review.

The server script "wireguard-server-deployer.sh" performs the bulk of the work in the scenario. First, it updates system packages and ensures that necessary dependencies for the WireGuard tools are installed. The protocol is implemented in the Linux kernel in version 5.6 and above, but the helper functions from the WireGuard team are available as separate packages in various repositories. The script then generates each set of private and public key pairs, one set for the server and one for the client. These are represented as 44-character, base64 encoded strings. The script then configures the WireGuard interface on the server using the generated keys, as well as IP addresses allowed for clients and particular port values for the UDP listening service, which can be specified. Next, the script enables the WireGuard config as a service so the interface will persist through server reboots. After that, the firewall host daemon is configured to allow traffic to SSH for server administration and the UDP port for WireGuard to listen on externally. IP masquerading is also enabled in the firewall. Next, changes to the operating system to allow for IP forwarding on IPv4 and IPv6 are made since this server will act as the redirector for client traffic. Finally, the script displays the information necessary for a client to connect successfully.

Separately, a user can begin running the client script "wireguard-client-setup.sh" on the client of their choice. The script will initially request the required pieces of information: the server's WireGuard public key, the external IP address of the server, the UDP port of the service (or a default value), and the client's private key. This information is available at the bottom of the output of the server script. The client script will take the client private key a user inputs and derive its own public key from it. It will then configure a WireGuard interface, listing the server as an available peer. Next, it will bring up the WireGuard interface and send the "First Message," establishing the secure tunnel. Finally, it configures iptables rules to force all IPv4 traffic (0.0.0.0/0) through the WireGuard tunnel from a routing perspective. The client can now freely browse the Internet, scan bug bounty targets, or perform research without traffic appearing to originate from their home, public wifi hotspot, etc. - knowing the traffic is encrypted at least along the "first mile" towards the intended destination.

VI. CONCLUSION

The WireGuard protocol is an interesting new tool in the ever-increasing effort to promote cybersecurity and Internet privacy in our world today. Our increasing reliance on technology forces us to reimagine the tools we use to secure our information, and VPN technology is one such tool that has many use cases.

The WireGuard protocol has many advantages, including forward secrecy, ease of auditing and implementation, high performance, and minimal attack surface. The protocol has the potential to be a building block for many different system designs. Projects such as Tailscale demonstrate how WireGuard can be used as a building block for larger software platforms [9]. WireGuard can assist in integrating encryption mechanisms into other systems that would otherwise potentially be transmitted in the clear.

The protocol also has limitations. When faced with an ISP or a nation-state actor that wishes to block VPN-based traffic circumventing censorship, WireGuard is currently lacking. Given the nature of the WireGuard exchange format and encrypted traffic formatting, when subject to deep packet inspection (DPI), WireGuard traffic is easy to detect - and subsequently filter out. Active and passive local adversaries can easily observe the use of WireGuard. Another interesting

dilemma arises if a peer's private key were compromised (or broken in a post-quantum context). While forward secrecy persists, the identity of the peer(s) someone was communicating with will be revealed. This lack of identity-hiding may be of concern for some use cases. WireGuard also currently relies on ChaCha20Poly1305 for symmetric encryption, which has no hardware encryption instruction support as of this writing. This is a drawback for large-scale computing systems that would otherwise benefit from implementations that utilize AES-NI. On the contrary, however, low-resource computing hardware (such as Internet-of-Things devices) may benefit by using WireGuard, given its reliance on ChaCha20. Finally, the issue of "roaming mischief" is of concern. WireGuard allows peers to roam, such as leaving a network and gaining a new IP address in a mobile device scenario, and maintain the tunnel without reauthentication. This could lead to issues with malicious users replaying traffic from legitimate devices. However, the timestamp included in message initiation "resets" the relationship, and the freshest timestamp available wins in order of precedence. Along with the fact that WireGuard traffic renegotiates sessions every two minutes, these features should mitigate most adversarial issues related to roaming mischief.

For future work on the authors' implementation of WireGuard, a more robust method of key exchange would be necessary to commercialize the VPN as a service. While copying the client private key over an SSH terminal is believed to be a secure method, paying clients would not have shell access to the VPN server. Users would need another secure "out-of-band" method for receiving their private key (or requesting a new static key for a separate device). The current implementation also only routes IPv4 traffic through the tunnel. Given the inevitability of IPv6 (and the reality that both are currently interoperable on the Internet as we know it today), it may be prudent to configure tunnels to use IPv6 by default, and allow user modifications as necessary for legacy circumstances.

VII. DISCLAIMER

The views presented here are exclusively those of the authors and do not represent the views of Purdue University nor imply or constitute endorsement by the Department of Defense.

References

[1] J. Donenfeld. "WireGuard: Next Generation Secure Network Tunnel." YouTube. https://www.youtube.com/watch?v=88GyLoZbDNw (accessed April 23, 2021).

[2] J. Donenfeld. "WireGuard – Fast, Modern, Secure VPN Tunnel." BlackHat USA. https://i.blackhat.com/us-18/Wed-August-8/us-18-Donenfeld-WireGuard-Next-Generation-Secure-Network-Tunnel.pdf (accessed April 23, 2021).

[3] B. Dowling and K. Paterson. "A Cryptographic Analysis of the WireGuard Protocol", in International Conference on Applied Cryptography and Network Security 2018, p 3-21, Jun. 2018, https://www.doi.org/10.1007/978-3-319-93387-0_1.

[4] B. Lipp et al. "A Mechanised Cryptographic Proof of the WireGuard Virtual Private Network Protocol," 2019 IEEE European Symposium on Security and Privacy (EuroS&P), p. 231-246, Jun. 2019, https://doi.org/10.1109/EuroSP.2019.00026.

[5] T. Jager, F. Kohlar, S Schage, J. Schwenk. "On the Security of TLS-DHE in the Standard Model." in Advances in Cryptology - CRYPTO 2012. Lecture Notes in Computer Science, vol. 7417, Aug. 2012, https://doi.org/10.1007/978-3-642-32009-5_17.

[6] J. Donenfeld. "WireGuard – Fast, Modern, Secure VPN Tunnel." WireGuard. https://www.wireguard.com (accessed April 23, 2021).

[7] J. Donenfeld, "WireGuard: Next Generation Kernel Network Tunnel," NDSS 2017, Feb. 2017, https://doi.org/10.14722/NDSS.2017.23160.

[8] A. Master. "wireguard-deployer." GitHub. https://github.com/smolbytes/wireguard-deployer (accessed May 06, 2021).

[9] "Tailscale makes networking easy." Tailscale. https://tailscale.com (accessed April 24, 2021).