WireGuard

Florian Preinstorfer

florian@nblock.org
https://nblock.org

Linuxwochen Linz
23.06.2018

VPN?

- Virtual Private Network
- Ein privates (meist verschlüsseltes) Netzwerk
- Typische Anwendungsfälle: Remote Access, Site-to-Site

Ein gelöstes Problem

- IPsec
- OpenVPN
- Tinc
- Kommerzielle Lösungen
- WireGuard

Meta

- Jason A. Donenfeld
- Komponenten: Kernel Modul, Userspace Tools, Implementierungen für verschiedene Plattformen
- Lizenz: GPLv2
- Wird aktiv entwickelt

Historie

. . . zum WireGuard Kernel Modul[1]

- Seit 2016
- 932 Commits in Summe (96% von Donenfeld)
- Lines of code: ∼4000

[1] https://git.zx2c4.com/WireGuard

Cryptokey Routing

- Basis von WireGuard
- Fixe Beziehung zwischen Public Key und Source IP-Adressen
- Jeder Host benötigt:
  - Private Key
  - Public Key
  - Erlaubte Peers (Public Key + Source IP-Adressen)

Cryptokey Routing: Beispiel 1

Host Public Key	**Host Private Key**
aaaa	AAAA
Peer Public Key	**Source IP-Adressen**
bbbb	10.90.90.2/32, 192.168.40.0/24
cccc	10.90.90.3/32

Cryptokey Routing: Beispiel 2

Host Public Key	**Host Private Key**
cccc	CCCC
Peer Public Key	**Source IP-Adressen**
aaaa	0.0.0.0/0

Konfiguration

- Konfiguration via WireGuard Interface
- „reguläres“ Netzwerk Interface (`eth0`, `wlan0`, ...)
- Gängige Tools nutzbar: `ip`, `ipconfig`, `iptables`, ...
- stateless

Konfiguration

- Jedes WireGuard Interface benötigt:
  - Private Key
  - Erlaubte Peers
  - UDP Port für eingehende Verbindungen
- Peer:
  - Public Key
  - Erlaubte Source IP-Adressen
  - Optional: Internet Endpunkt + Port

Konfiguration

Host Public Key	**Host Private Key**	**UDP (listen)**
cccc	CCCC	12345
Peer Public Key	**Source IP-Adressen**	**Internet Endpunkt**
aaaa	0.0.0.0/0	192.95.5.64:21841

Ablauf: Host cccc sendet Daten an Peer aaaa

Sender:

- Userspace: Paket erstellen
- Linux: Paket zu WireGuard
- WireGuard: Welcher Peer?
- WireGuard: Verschlüsseln
- WireGuard: Zum Internet Endpunkt von aaaa senden

Empfänger:

- WireGuard: Paket empfangen
- WireGuard: Paket entschlüsseln?
- WireGuard: Welcher Peer?
- WireGuard: Peer erlaubt?
- Linux: Paket Zustellen

Demo

Screencast von `https://www.wireguard.com/quickstart`

stateless?

Nein,

- Kein Perfect Forward Secrecy
- DoS durch das Versenden von Paketen
- Amplification Attacks?

Key Exchange

- Handshake vor dem 1. Paket nötig
- Aushandeln von Session Keys
- 1-RTT

1-RTT

Quelle: Jason A. Donenfeld, WireGuard, 15. Juni 2018

Auswirkungen des 1-RTT Designs

- Perfect Forward Secrecy möglich
- Keine Antworten bei Problemen (stealth)
- Keine Amplification Attacks
- DoS-Schutz durch Cookie
- Keine Allokation von Speicher bei ungültigen Paketen
- Keine Cipher- oder Protokoll Agilität

Timer

- State wird intern mittels Timer State Machine verwaltet
- Rekey nach 120 Sekunden oder $2^{64} - 2^{16} - 1$ Nachrichten
- Abweisen 180 Sekunden oder $2^{64} - 2^{4} - 1$ Nachrichten

Was noch?

- Symmetrischer Schlüssel für den Handshake
- Periodische Keepalives (bei Bedarf)

WireGuard ist . . .

- sicherer Layer 3 Netzwerk Tunnel für IPv4/IPv6
- UDP-basiert
- ausgelegt für den Linux Kernel
- einfach zu nutzen

WireGuard ist . . .

- nutzt Keys für die Identität der Peers
- stateless
- konservativ
- (möglicher Ersatz) für OpenVPN, IPsec . . .

Implementierungen

- Als externes Modul für den Linux Kernel
- Alternative Implementierungen in: Go, Rust, Haskell, ...

Verfügbarkeit

- Linux (Arch, Debian, Ubuntu, . . .): Ja, externes Kernel Modul
- OpenWRT/LEDE: Ja
- Android: Ja, Custom ROM oder Go
- MacOS: Ja, via homebrew
- Windows: Nein

Fazit

- Neue VPN Lösung für Linux
- Einfach und sicher zu nutzen
- Nur wenige Integrationen
- Schnell
- Nicht im Mainline Kernel
- Kaum konfigurierbar

WireGuard

Florian Preinstorfer

florian@nblock.org
https://nblock.org

Linuxwochen Linz
23.06.2018

Quellen

- Webseite: `https://www.wireguard.com`
- Logo: `https://www.wireguard.com/img/wireguard.svg`
- Protokoll: `https://www.wireguard.com/protocol/`
- Key Exchange: `https://www.wireguard.com/talks/sstic2018-slides.pdf`
- Whitepaper: `https://www.wireguard.com/papers/wireguard.pdf`
- Screencast (Demo): `https://www.wireguard.com/talks/talk-demo-screencast.mp4`

Algorithmen

- ChaCha20: Symmetrische Verschlüsselung, authentifiziert via Poly1305 (RFC7539's AEAD)
- Curve25519: ECDH (RFC7748)
- BLAKE2: Hashing und keyed hashing (RFC7693)
- SipHash24: Hashtable keys
- HKDF key derivation (RFC5869)