netStackGo: Simple and Secure Networking

Yaohua Li and Jon A. Solworth
http://www.ethos-os.org

UIC UNIVERSITY OF ILLINOIS AT CHICAGO COLLEGE OF ENGINEERING
Department of Computer Science

Contribution: Secure Networking Service

netStackGo: security on behalf of applications

- Provides encryption, network user authentication and authorization
- Provides local user authentication and authorization
- Replacement for SSL and Kerberos
- Now available for applications written in Go

Why netStackGo has better security?

Isolation

- Application is provided transparent networking service with encryption, authentication, authorization
- Failure in application will not propagate to network service

Simplicity

- Simple and few APIs exposed to programmers
- Less chance for programmers to make mistakes

Why Use netStackGo?

- Simple Syntax: As easy as **connect** and **communicate**
- Easy Administration: Hassle-free configuration
- 0 line of application code for security
- Security through careful abstraction and design
- Security embedded into normal process of networking

Security Properties in Comparison

Property	netStackGo	Kerberos	TLS(SSL)/PKI	SSH	IPsec
Public Key	✓	✗	✓	*	✓
Authentication Server	✓	✓	✗	*	*
Protection of Private Key	✓	✓	N/A	*	✓
Timeliness	✓	✓	*	N/A	*
Availability	✓	✗	✓	✓	*
Multi-Organization	✓	✗	✓	✓	*
Low Complexity	✓	✗	✗	✗	✓
Isolation	✓	✗	✗	✗	✓
Authenticate User	✓	✓	✓	✓	*
Mutual Authentication	✓	✓	✓	✗	✓

✓– yes; ✗– no; * – partial; N/A – not applicable

Authentication with Strong Trust Model

- Based on Public Key Cryptography
- Administrator can choose which certificate issuers to trust
- Authenticate with multiple organizations

Low Complexity Networking Primitives

```
conn ← ipc(service, remoteHost)
```

- Initiate a connection to specified host and service
- Daemon retrieves host IP and public key
- *Service* is analogy to port number in TCP/UDP

```
ad ← advertise(service)
```

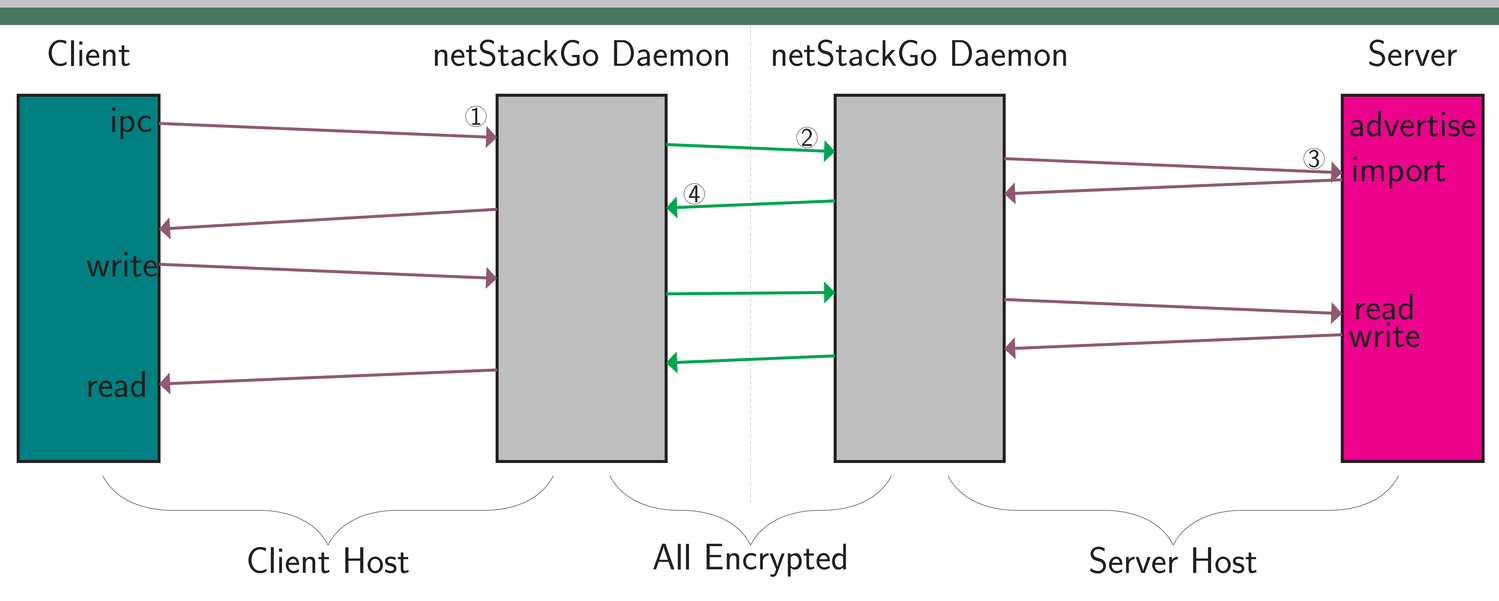
- Announce to provide a service

```
conn, user ← ad.import()
```

- Accept an incoming connection with remote user id

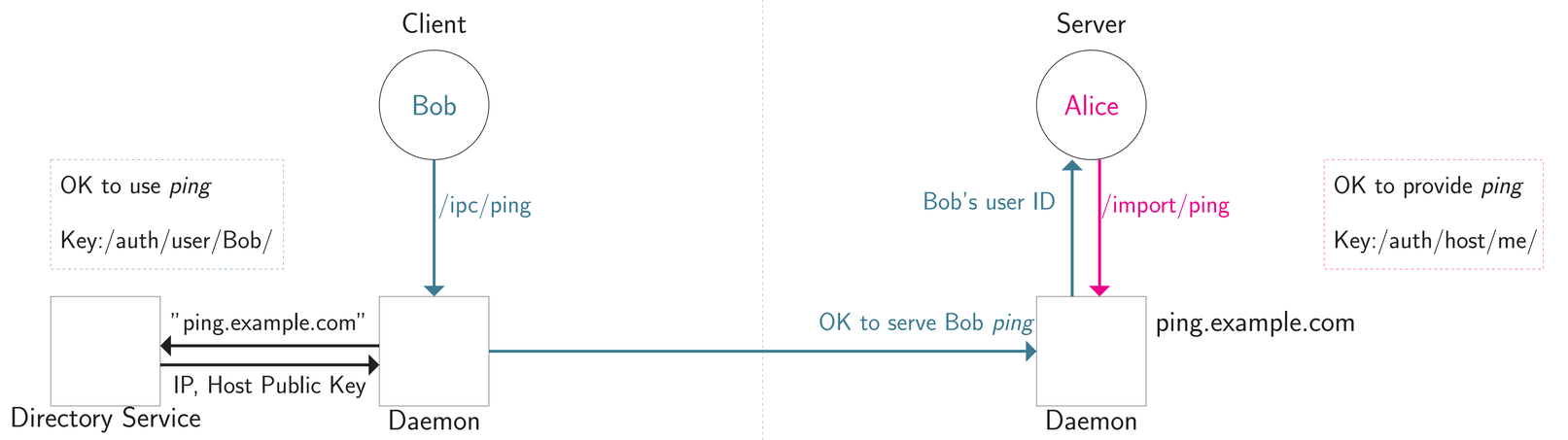
```
conn.write(data)
data ← conn.read()
```

- Write to or read from a connection



Authentication Procedure

① Only clients authorized to invoke *service* can connect to daemon. Find user's private key.
② Authenticate remote user. Authorize remote user for *service*.
③ Only *authenticated*, *authorized* connection can be *import*ed for server application
④ Mutual authentication of server's host



Ping Client

```
1 conn ← ipc("ping", "ping.example.com")
2 conn.write(data)
3 response ← conn.read()
4 conn.close()
```

Ping Server

```
1 ad ← advertise("ping")
2 while TRUE
3     conn ← ad.import()
4     data ← conn.read()
5     conn.write(data)
6     conn.close()
```

Abstraction for Security

1. Bob is authenticated and authorized to his daemon to use *ping* service
2. Bob requests to ipc to host "ping.example.com"
3. Bob's daemon looks up IP and host public key of "ping.example.com" to create an encrypted tunnel
4. Bob's daemon mutually authenticates remote host
5. Bob's daemon uses his private key for authentication
6. Alice is authenticated and authorized to her daemon to provide *ping* service
7. Alice's daemon authenticates remote user Bob and authorizes him to request *ping* service
8. Alice imports an authenticated, authorized connection, protected by encrypted tunnel
9. Bob ends up having a connection authenticated and authorized, protected by encrypted tunnel

Configuration

User Generate user public key pair
Service Create directories */ipc/ping*, */import/ping*
Authorize

- Add user to group *ipc-ping* to allow use of *ping*, for access to Unix socket */ipc/ping*
- Add user to group *import-ping* to allow to provide *ping*, for access to Unix socket */import/ping*

Enforcement of Local Security

- Use Linux identity
  - ▷ Daemon authenticates user via Unix socket
- Only daemon can read host/user private keys
  - ▷ Operations done on behalf of user
- Only specified users can provide/use a service
  - ▷ Linux user groups for advertise/ipc separate services

Authentication-related libraries

Library	Version	C LoC	# of APIs
OpenSSH	6.2p1	349,000	N/A
LibSSH2	1.4.3	296,000	166
OpenSSL	1.0.1e	274,000	227
GnuTLS	3.1.9	230,000	641
NSS	3.14.3	586,000	742
Kerberos v5	1.11.2	306,000	194
netStackGo		37,000	9

*major required libraries included in LoC

Due to high complexity, even widely used software written by experts suffer from mis-authentication.

Conclusion

netStackGo provides secure networking by lower complexity and strong trust model in authentication.