Samba, Quo Vadis?

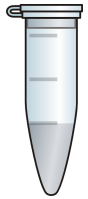
Experimenting with languages the Cool Kids™ use

Kai Blin
Samba Team

SambaXP 2017
2017-05-03

Intro

- M.Sc. in Computational Biology
- Ph.D. in Microbiology
- Samba Team member



Overview

- Programming languages

Overview

- Programming languages
- Hello, world!

Overview

- Programming languages
- Hello, ~~world~~ Kerberos!

Overview

- Programming languages
- Hello, ~~world~~ Kerberos!
- Implementing it

Overview

- Programming languages
- Hello, ~~world~~ Kerberos!
- Implementing it
- Conclusions

If someone claims to have the perfect programming language, he is either a fool or a salesman or both.
– Bjarne Stroustrup

Programming Languages

Why?

"The [Samba] project does need to consider the use of other, safer languages."
– Jeremy Allison, last year

Why?

No, honestly, why?

- Avoid whole classes of bugs
- New languages, new features
- It's getting harder to find C programmers

Currently

- 80% C, 7% Python (2)
- ~ 2 million lines of code
- ~ 15-20 regular contributors

Languages

Cool Kids™ approved

- Go
- Rust
- Python 3

Go

- Announced 2009
- C-like, compiled language
- Concurrency core part of the language
- Package management with `go get <package>`
- Programmers call themselves "Gophers"

Go

Hello, World!

```
package main

import "fmt"

func main() {
    fmt.Println("Hello, World")
}
```

Rust

- Announced 2010
- C-like, compiled language
- Focus on memory safety
- Package management with `cargo`
- Still a bit of a moving target
- Programmers call themselves "Rustacians"

Rust

Hello, World!

```
fn main() {
    println!("Hello, world!");
}
```

Python 3

- Announced in 2008
- dynamically-typed, interpreted language
- Focus on simplicity and readability
- Package management with `pip install <package>`
- Mainly for comparison
- Programmers call themselves "Pythonistas"

Python 3

Hello, World!

```
print("Hello, world!")
```

Or maybe

```
def main():
    print("Hello, world!")

if __name__ == "__main__":
    main()
```

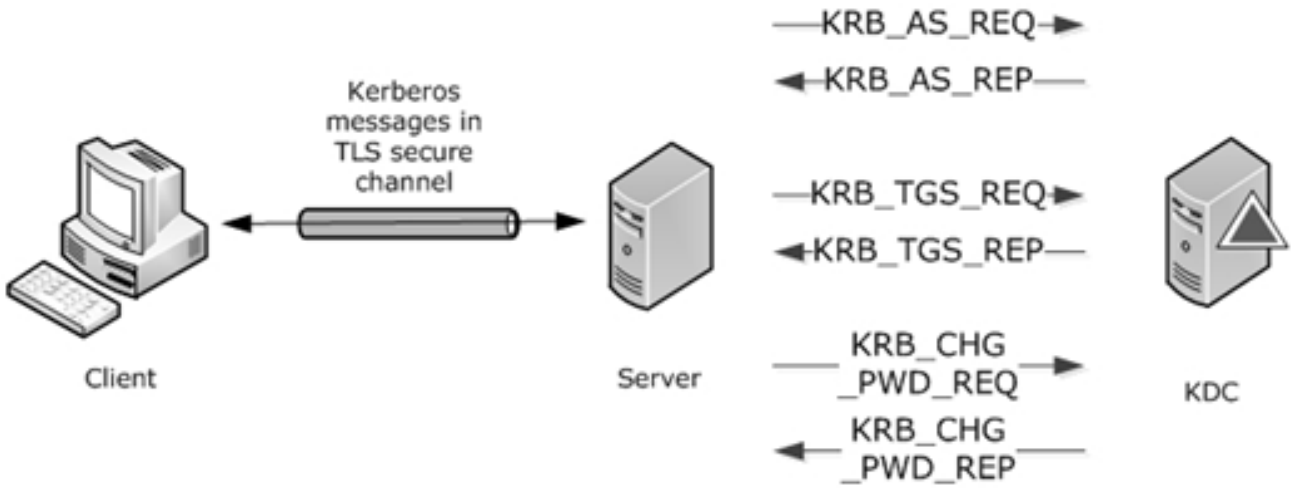
Hello, Kerberos!

Introducing the example project.

KKDCP

Kerberos **K**ey **D**istribution **C**enter **P**roxy

Proxy Kerberos TGT and service ticket requests via HTTPS



KKDCP

- Client sends ASN.1-encoded HTTPS POST request to proxy
- Proxy decodes
- extracts Kerberos ASN.1 request
- Proxy sends krb request to KDC
- KDC responds
- Proxy encodes response
- Proxy replies to client

Existing implementations

- MS-KKDCP server
- kdcproxy

Plan

- Write an implementation in all three languages
- Validate against reference
- Compare on both implementation ease and performance

In theory, there is no difference between theory and practice. But, in practice, there is.
– Jan L. A. van de Snepscheut

Implementing it

Languages

- Go: 1.8
- Rust: 1.16.0
- Python 3.5.2

Environment

- Server: Active Directory KDC (Samba or MS)
- Client: MS Windows
- Client: MIT Kerberos (kinit)

Challenges

- How to set up KKDCP with Windows
- Use kdcproxy as reference
- Only use MIT Kerberos kinit

Challenges

Part II

- Ubuntu ships MIT krb5 1.13.2
- KKDCP seems to fail on 1.13.2, reason not obvious from logs
- Use 1.16 pre-release instead
- Doesn't like HTTP, use HTTPS

New Reference: kdcproxy

- Created by RedHat (https://github.com/latchset/kdcproxy)
- Written in Python (v2 compatible)
- WSGI app

Set-Up

Server

- WSGI app in gunicorn
- Use Nginx for HTTPS

Set-Up

Server

```
server {
    listen 443;
    include ssl_params;

    location /RhKdcProxy {
        proxy_pass http://127.0.0.1:8123/;
        include proxy_params;
    }
}
```

```
gunicorn -w4 kdcproxy:application -b 127.0.0.1:8123
```

Set-Up

Client

```
[realms]
        DEMO.KBLIN.ORG = {
                http_anchors = FILE:/etc/ssl/certs/kdcproxy.pem
                kdc = https://kdcproxy.demo.kblin.org/RhKdcProxy
                admin_server = kdc.demo.kblin.org
        }
```

Demo Time!

```
# as root
./switch.sh rh

# as user
./run_demo.sh
```

Python 3

- Uses asyncio/aiohttp
- Uses new async/await keywords
- Uses type hints
- Not compatible with Python 2.x
- pyasn1 for ASN.1 handling
- See full code at https://github.com/kblin/py3-kkdcp

Python 3

```
async def handle_kkdcp(request):
    data = await request.read()
    proxy_request = codec.decode(data)
    loop = asyncio.get_event_loop()

    krb5_response = await asyncio.wait_for(
        forward_kerberos(proxy_request.message, loop=loop),
        timeout=15, loop=loop)
    return web.Response(body=codec.encode(krb5_response),
        content_type="application/kerberos")

app = web.Application()
app.router.add_post("/", handle_kkdcp)
```

Python 3

```
async def forward_kerberos(data: bytes, loop=None) -> bytes:
    reader, writer = await asyncio.open_connection(
        'kdc.demo.kblin.org', 88, loop=loop)
    writer.write(data)
    resp = await reader.read()
    return resp
```

Demo Time!

```
# as root
./switch.sh py3

# as user
./run_demo.sh
```

Go

- Apart from the HTTP handler dispatch, no attempt at async code
- Uses stdlib ASN.1 encoder
- No error handling in code samples
- See full code at https://github.com/kblin/go-kkdcp

Go

```
func HandleKkdcp(w http.ResponseWriter, req *http.Request) {
    w.Header().Set("Content-Type", "application/kerberos")
    defer req.Body.Close()

    data, _ := ioutil.ReadAll(req.Body)
    proxy_request, _ := codec.Decode(data)

    krb5_response, _ := ForwardKerberos(proxy_request.Message)
    reply, _ := codec.Encode(krb5_response)

    w.WriteHeader(http.StatusOK)
    w.Write(reply)
}

func main() {
    router := mux.NewRouter()
    router.HandlerFunc("/", HandleKkdcp).Methods("POST")
    log.Fatal(http.ListenAndServe(":8124", router))
}
```

Go

```
func ForwardKerberos(data []byte) (resp []byte, err error) {
    conn, _ := net.Dial("tcp", "kdc.demo.kblin.org:88")
    defer conn.Close()

    conn.Write(data)

    resp, _ = ioutil.ReadAll(conn)

    return resp, nil
}
```

Demo Time!

```
# as root
./switch.sh go

# as user
./run_demo.sh
```

Rust

- Documentation missing
- "Just use the nightly version of the language"
- Could not find working ASN.1 library.
- So, no demo ☠

Truth is subjectivity.
– Søren Kierkegaard

Conclusions

Conclusions

- Python 3 and Go both worth investigating further
- Rust not quite ready for prime time yet
- Would probably pick Go myself

Thank you