Web Authentication With Golang Google's Go Language

Web Authentication, Encryption, JWT, HMAC, & OAuth

Todd McLeod & Daniel Hoffmann

Table of Contents

Getting started

Welcome

Welcome to the course. You are taking a great step by enrolling in this course. Better skills create a better life. ***You are on your way to a better life.*** As you learn new skills, you are building a better life. ***I commend you for your efforts to improve your life.*** As you improve your life, you are improving the world --- one person at a time. You are making the world better, and you are making your life better. This is a win-win for everybody. Great work! Also, ***this is***

your course. Use it in the way which is best for you. If you want to skip ahead, skip ahead. As your teacher, my job is to help you succeed. The content here is designed to help you succeed both with visual studio code, and also as a student and in life.
Video file: 01 Welcome

Credentials

I am ***a tenured professor in California.*** I have taught at both the college and university level. I have also taught in multiple disciplines including business, information systems, computer science, and online education. In 1997, I was one of the first professors at the university to teach online. I did post-graduate work in online education at UC San Diego. I co-founded and taught in the "Online Teacher's Training Program" which trained professors how to teach online. In 2008, I was selected as one of the best instructors in the entire California Community College system. Currently, when measured by the number of students served, I am the world's leading trainer in Google's new programming language which is one of the fastest growing, highest paying programming languages in America. My background in business, information systems, computer science, and online education has prepared me to teach this course!
Video file: 02 Cred

Course resources

You can find everything I use in the course and all of the courses resources here on Udemy, including the COURSE OUTLINE which you can download from this video.
Video file: 03 Course Resources

How to be successful

Understanding what has made others successful can help you become successful. These are principles which have helped me become successful. I learned these principles from others and from my own experience. I share these **principles to help you succeed in this course and in life:**

- **GRIT**
    - Grit - Angela Duckworth
- **DRIP LEARNING**
    - Time on task
    - Small frequent engagements
    - Multiple perspectives, multiple engagements
    - my teachers
        - drop by drop, the bucket gets filled
        - persistently, patiently, you are bound to succeed
- **FOCUS**
    - Bill Gates & Warren Buffett

- **PLANNING**
    - Bill Gates, “If you want to be successful, get in front of what’s coming and let it hit you.”
- **RESOURCES**
    - The 7 Habits of Highly Effective People
        - https://drive.google.com/drive/folders/0B22KXlqHz6ZNYm4wZWxwLTB5RVE?resourcekey=0-tXNS0UyMIbh1mrqZQrzXfA&usp=sharing

Video file: 05 Success

JSON encoding

Setup

Setting up resources for our course:

- github
- ssh
- adding a collaborator
- adding workspaces to gitignore
- adding tags for versioning

tag v0.0.0
video 12 Setup

Installing Go & Go modules

video: 13 Installing Go & Go Modules

Marshal

Housekeeping

- add version to previous video

tag v0.2.0
video: 14 Marshal

Unmarshal

tag v0.3.0
video: 15 Unmarshal

Launching a service
tag v0.4.0
video: 16 Launching A Service

Encode
server sends to client
tag v0.5.0
video: 17 Encode

Decode
client sends to server and server decodes
https://curlbuilder.com/
tag v0.6.0
video: 18 Decode

Hands-on exercises - ninja level 1

Hands-on exercise #1
Create a server. Use Encode to send a slice of values of type person to the client as JSON. Use CURL to make a request to the server. https://curlbuilder.com/
tag v0.7.0
video: 19 Hands On 1

Hands-on exercise #2
On the server, decode this JSON into the appropriate data structure:

- [{"First":"Jenny"},{"First":"James"}]
- https://curlbuilder.com/

tag v0.8.0
video: 20 Hands On 2

Authentication basics

Difference between authentication and authorization

- Authentication
    - Determines who you are
    - Verifies that no-one is impersonating you

        - three ways to authenticate
            - who you are (biometrics)
            - what you have (eg, atm card; key; phone)
            - what you know (username; password, ….)
        - two-factor authentication
    - Authorization
        - What permissions you have on a system
        - Says what you are allowed to do
        - The name of the http header used for authentication

https://www.cyberciti.biz/faq/authentication-vs-authorization/
video: 21 Diff Between Authentication & Authorization

Http Basic Authentication

- Basic authentication part of the specification of http
    - send username / password with every request
    - uses authorization header & keyword "basic"
        - put "**username:password**" together
        - converts them to **base64**
            - puts generic binary data into printable form
            - **base64 is reversible**
                - never use with http; **only https**
        - use basic authentication to login

tag v0.9.0
video: 22 HTTP Basic Authentication

Exploring authentication and state

- every request?
- is this secure?

video: 23 Exploring Authentication And State

Password storage

Storing passwords

- never store passwords
- instead, store one-way encryption "hash" values of the password
- for added security
    - hash on the client
    - hash THAT again on the server
- hashing algorithms
    - bcrypt - current choice

        - https://godoc.org/golang.org/x/crypto/bcrypt
    - scrypt - new choice

video: 24 Storing Passwords

Hashing passwords

- bcrypt

tag: v0.10.0
video: 25 Hashing Passwords

Exploring HMAC

Bearer Tokens & Hmac

- bearer tokens
    - added to http spec with OAUTH2
    - uses authorization header & keyword “bearer”
- to prevent faked bearer tokens, use cryptographic “signing”
    - cryptographic signing is a way to prove that the value was created/validated by a certain person
    - HMAC
- https://godoc.org/crypto/hmac

video: 26 Bearer Tokens & Hmac

HMAC in action

- HMAC is a cryptographic signing function

tag v0.11.0
video: 27 HMAC in Action

Discussing HMAC

video: 28 Discussing HMAC

Discussing HMAC II

- https://play.golang.org/p/HarO409e8tu
- https://play.golang.org/p/G1Z7UelAuA6
- https://play.golang.org/p/m76hqC55G-q
- https://play.golang.org/p/sPZLQQC6YCF
- other
- https://play.golang.org/p/G1Z7UelAuA6
- https://play.golang.org/p/lImDdhk0Xax

- https://play.golang.org/p/m76hqC55G-q
- https://github.com/GoesToEleven/SummerBootCamp/blob/a40ab4ac3f7a497d49e73f336ccae6dc29107a5b/05_golang/02/03/11_sessions/11_03_caleb_sessions_HMAC/main.go

video: 29 Discussing HMAC II

Exploring JWT

JWT explained

JSON Web Token
{JWT standard fields}.{Your fields}.Signature
video: 30 JWT Explained

JWT documentation & claims

Analyzing a package to see if it's go module compatible

- go list -m -version github.com/dgrijalva/jwt-go

https://github.com/dgrijalva/jwt-go
tag v0.12.0
video: 31 JWT Docs & Claims

Creating JWT

tag v0.13.0
video: 32 Creating JWT

Validating JWT

tag v0.14.0
video: 33 Validating JWT

Overview of concepts

video: 34 Overview Of Concepts

Rotating Keys

tag v0.15.0
video: 35 Rotating Keys

Discussion

- cryptography

    - large field
    - You don't need to understand it fully to use it
- Hashing
    - MD5 - **don't use**
    - SHA
    - Bcrypt
    - Scrypt
- Signing
    - Symmetric Key
        - HMAC
        - same key to sign (encrypt) / verify (decrypt)
    - Asymmetric Key
        - RSA
        - ECDSA - better than RSA; faster; smaller keys
        - private key to sign (encrypt) / public key to verify (decrypt)
    - JWT

NOT DISCUSSED IN VIDEO

- Encryption
    - Symmetric key
        - AES
    - Asymmetric Key
        - RSA

video: 36 Discussion

Exploring encryption

Base64 - not encryption

tag: v0.17.0
video 37 Base64 - Not Encryption

Encrypting an email

- symmetric key - encrypt the message
    - AES
- compare sent / received
    - SHA

tags: **v0.16.0** & v0.18.0
video: 38 Encrypting An Email

Reviewing encrypted email

tag: v0.19.0
video: 39 Reviewing Encrypted Email

Review and overview

video: 40 Review And Overview

SHA256 a file

tag v0.20.0
video 41 SHA256 A File

Authentication with HMAC

HMAC values in a cookie intro

https://play.golang.org/p/U1IMaqURYe_7
Examine example code
Video: 42 HMAC Values Intro

HMAC values in a cookie - 1

setting up a server and the html
tag: v0.21.0
video: 43 HMAC Values In A Cookie - 1

VS code liveshare

Extension: **Live Share** by Microsoft
video 44 VS Code Liveshare

HMAC values in a cookie - 2

Creating and viewing the hmac cookie
tag: v0.22.0
video: 45 HMAC Values In A Cookie - 2

HMAC values in a cookie - 3

using HMAC to ensure a cookie has not been tampered with
tag: v0.23.0
video: 46 HMAC Values In A Cookie - 3

Authentication with JWT

Security high-level

https://research.swtch.com/deps
video: 47 Security High-Level

JWT overview of docs

Range: Why Generalists Triumph in a Specialized World Hardcover – May 28, 2019 by David Epstein
video: 48 JWT Overview

JWT in a cookie

tag: v0.24.0
video: 49 JWT In A Cookie

JWT parsing Part 1

tag: v0.25.0
video: 50-a JWT Parsing

JWT parsing Part 2

tag: v0.25.0
video: 50-b JWT Parsing

Running

tag: v0.25.1
video: 51 Running

Hands-on exercises - ninja level 2

Hands-on exercise #1

For this hands-on exercise:

- Create a server with two endpoints
    - Endpoint **/** should display a webpage with a register form on it
        - This form should take in at least a username and password
        - It should post to **/register**
    - Endpoint **/register** should save the username and password in a map
        - The password should be securely stored using bcrypt
        - Redirect the user back to endpoint **/** afterwards

tag: v0.26.0
video: 52a Hands On 1 - Part 1
video: 52b Hands On 1 - Part 2

Hands-on exercise #2

For this hands-on exercise:

- Modify the server from the previous exercise
- Add a login form to the webpage
    - The form should take in a username and password
    - The form should post to a **/login** endpoint
- Add a new endpoint **/login**
    - The endpoint should compare the given credentials with stored credentials in the user map
        - Make sure to use the bcrypt function to compare the password
    - If the credentials match, display a webpage saying login successful
    - If the credentials do not match, display a webpage saying login failed

tag: v0.27.0
video: 53 Hands On 2

Hands-on exercise #3

For this hands-on exercise:

- Modify the server from the previous exercise
- Create two functions
    - **createToken**
        - Should take in a session id
        - Should use **HMAC** to create a signature of the session id
        - Should combine the signature with the session id and return the signed string

        - Note, you will need to convert the hmac to a printable form
            - Hex
            - Base64
    - **parseToken**
        - Should take in a signed string from createToken
        - Should separate the signature from the session id
        - Should verify the signature matches the session id
        - Should return the session id

HINT:

- cryptography
    - large field
    - You don't need to understand it fully to use it
- Hashing
    - MD5 - **don't use**
    - **SHA**
    - Bcrypt
    - Scrypt
- Signing
    - Symmetric Key
        - **HMAC**
        - same key to sign (encrypt) / verify (decrypt)
    - Asymmetric Key
        - RSA
        - ECDSA - better than RSA; faster; smaller keys
        - private key to sign (encrypt) / public key to verify (decrypt)
    - **JWT**

NOT DISCUSSED IN VIDEO

- Encryption
    - Symmetric key
        - AES
    - Asymmetric Key
        - RSA

tag: v0.28.0
video: 54a Hands On 3 - Part 1
video: 54b Hands On 3 - Part 2

Hands-on exercise #4

For this hands-on exercise:

- Modify the server from the previous exercise
- On **/login**
    - Generate a session id
        - Use a map **sessions** between session id and username

        - Use **createToken** with the generated session id
        - Set the token as the value of a session cookie
- Change login endpoint to redirect the user back to **/**
- On **/**
    - Use **parseToken** to get the session id
    - Use the sessions map to get the username
    - Display the username in the page if one has been found
        - No username will be displayed if
            - No session cookie set
            - Session cookie validation failed
            - No session in the sessions map to a username

tag: v0.29.0
video: 55 Hands On 4

Hands-on exercise #5

For this hands-on exercise:

- Modify the server from the previous exercise
- have the db map store a struct of user fields, including bcrypted password
- display a user field (not bcrypted password) when someone’s session is active

tag: v0.30.0
video: 56 Hands On 5

Hands-on exercise #6

For this hands-on exercise:

- Modify the server from the previous exercise
- Modify **createToken** and **parseToken** to use JWT
    - Create a custom claims type that embeds **jwt.StandardClaims**
    - The custom claims type should have the **session id** in it
    - Make sure to set the **ExpiresAt** field with a time limit
        - time.Now().......
    - Use an HMAC signing method
    - Make sure to check if the token is valid in the **parseToken** endpoint
- Question
    - will we still need our sessions table / database?
    - YES!

tag: v0.31.0
video: 57a Hands On 6 - Part 1
video: 57b Hands On 6 - Part 2
video: 57c Hands On 6 - Part 3

Hands-on exercise #7

For this hands-on exercise:

- Modify the server from the previous exercise
- Add a new form to **/**
  - It should just have a submit button labeled logout
  - It should post to **/logout**
- Add a new endpoint **/logout**
  - Delete the session of the current user from the **sessions** map
  - Set the cookie of the user to be deleted

tag: v0.32.0
video: 58a Hands On 7 - Part 1
video: 58b Hands On 7 - Part 2

OAuth2 overview

What is OAuth2

Oauth2 allows

- we also use Oauth to login
  - example: login with facebook
- a user authorizes one website to do something at another website
  - example: give one website access to your dropbox account to store files there, or retrieve files from there

Two most common ways to do Oauth

- **authorization code**
  - more secure, but requires a server
  - “three legged flow”
- **implicit**
  - less secure

Here are the four flows, FYI

- **Authorization Code**
  - for apps running on a web server, browser-based and mobile apps
- **Password**
  - for logging in with a username and password (only for first-party apps)
- **Client credentials**
  - for application access without a user present
- **Implicit**
  - was previously recommended for clients without a secret, but has been superseded by using the Authorization Code grant with PKCE.
- https://aaronparecki.com/oauth-2-simplified/#authorization

video: 59 What Is OAuth2

Overview of the OAuth2 process

- user is as spacex.com (for example)
    - logs in with Oauth2 using google Oauth2
    - Redirects user to Google Oauth login page
        - user is asked to grant permissions
        - what to share from google account
    - google Redirects back to spacex.com with a code
    - Spacex.com exchanges code and secret for access token to google
    - Spacex.com uses token to get who the user is on google, including user id on google

video: 60 Overview Of The OAuth2 Process

To err is human

"It is not the critic who counts; not the man who points out how the strong man stumbles, or where the doer of deeds could have done them better. The credit belongs to the man who is actually in the arena, whose face is marred by dust and sweat and blood; who strives valiantly; who errs, who comes short again and again, because there is no effort without error and shortcoming; but who does actually strive to do the deeds; who knows great enthusiasms, the great devotions; who spends himself in a worthy cause; who at the best knows in the end the triumph of high achievement, and who at the worst, if he fails, at least fails while daring greatly, so that his place shall never be with those cold and timid souls who neither know victory nor defeat."
~ Theodore Roosevelt

video 61 <- no vid

Examining the OAuth2 package

- create a config struct
- authcodeURL
    - state is some string, anything, some unique ID for this login attempt
    - returns string that you want to redirect your users to

- Exchange
  - converts a code into a token
- TokenSource
  - gives us a token source
- NewClient
  - gives us an http client
  - this client is authenticated with the oauth resource provider

https://godoc.org/golang.org/x/oauth2
video 62 Examining The OAuth2 Package

Finding documentation for an OAuth2 provider
video: 63 Finding Docs For An OAuth2 Provider

Oauth2 in action

Setting up a server
- create a server
- have a page for github oauth login
  - a form with a button
  - action = /oauth/github
- have a route to handle
  - /oauth/github

tag v0.33.0
video: 64 Setting Up A Server

Starting the oauth process
- create a config struct
- authCodeURL

tag v0.34.0
video: 65 Starting The OAuth Process

Getting the http client
- Exchange
  - converts a code into a token
- TokenSource
  - gives us a token source
- NewClient
  - gives us an authenticated http client
  - this client is authenticated with the oauth resource provider

tag v0.35.0
video: 66 Getting The HTTP Client

Using the http client

Github is using graphql now so here is how to do it with that
tag v0.36.0
video: 67 Using The HTTP Client

JSON decoding user id

tag v0.37.0
video: 68 JSON Decoding User ID

Hands-on exercises - ninja level 3

Hands-on exercise #1

For this hands-on exercise:

- Choose an OAuth2 provider
- Find the documentation for the authentication process
    - Find the **AuthURL** and **TokenURL** in the documentation
        - find these in the documentation from the oauth2 provider
        - hints
            - tokenURL is often a POST
            - AuthURL is often (maybe always) a GET
    - If there is a sub-package in the oauth2 package ...
        - (https://godoc.org/golang.org/x/oauth2)
        - … make sure the endpoints match what is in the documentation
- Find how to get a user ID from the OAuth2 provider
    - Make sure you know how to make the call yourself
    - See if you need any special scope permissions to get the information
- Sign up to use the OAuth2 provider
    - If your provider wants a redirect url, set it to **http://localhost:8080/oauth/<your provider>/receive**
    - Make sure you have your ClientID and ClientSecret

tag: v0.38.0
video: 69 Hands On 1

Hands-on exercise #2

For this hands-on exercise:

- Modify the server from Ninja Level 2

- Create an oauth2.Config for your provider
    - Fill it in with information from hand-on-exercise 1
        - ClientID
        - ClientSecret
        - Endpoints
            - AuthURL
            - TokenURL
        - RedirectURL
            - **http://localhost:8080/oauth/<your provider>/receive**
        - Scopes
            - IF NEEDED
- Create an endpoint **/oauth/<your provider>/login**
    - This should be a POST endpoint
    - This should generate a uuid to use as the state value
    - A new map should be used to save the state and the expiration time for this login attempt
        - Key is a **string** state
        - Value is a **time.Time** expiration time
            - One hour in the future is a reasonable time
    - Redirect the user to the oauth2 **AuthCodeURL** value
- Modify your **/** index page to include an oauth login form
    - The form should post to **/oauth/<your provider>/login**
    - The login form should only have a submit button
- LET'S TEST OUR CODE!
    - Verify that attempting to login sends you to the oauth provider's login page and approving sends you back to **/oauth/<your provider>/receive**
        - You do not have this endpoint yet, if you are using the http.ServeMux, your index page will serve this endpoint

tag: v0.39.0
video: 70 Hands On 2 - Part 1
video: 70 Hands On 2 - Part 2
video: 70 Hands On 2 - Part 3
video: 70 Hands On 2 - Part 4
video: 70 Hands On 2 - Part 5

Hands-on exercise #3

For this hands-on exercise:

- Modify the server from the previous exercise
- Create an endpoint **/oauth/<your provider>/receive**
    - Should get the query parameter values **state** and **code**
    - State should be used to see if this state has expired

        - **time.Now().After()**
    - Code should be sent through the **config.Exchange** function to get a token
    - Create a **TokenSource** from the token
    - Create an **http.Client** with the **TokenSource**
    - Make your call you found in exercise 01 to get a user ID
    - Print out the result from your call

tag: v0.40.0
video: 71 Hands On 3 - Part 1
video: 71 Hands On 3 - Part 2

Hands-on exercise #4

For this hands-on exercise:

- Modify the server from the previous exercise
- Create a new map **oauthConnections**
    - Key should be user IDs from the oauth provider
    - Value should be user IDs in your own system
- In endpoint **/oauth/<your provider>/receive**
    - Extract just the user ID from the result of your call in the previous exercise
        - This will usually require creating a struct and unmarshalling/decoding json data
    - Get the local user ID from the **oauthConnections** map
    - If there was no value in the map, set the user ID to a random user ID from your users map
    - Create a session for your user just like how **/login** does it
        - Good candidate for pulling out into a separate function
    - Redirect the user back to **/**
- **Good luck!**
- **NOTES:**
    - to establish oauth2
        - interact with oauth2 api at some site offering oauth2 connections
        - sign up
        - get **clientsecret** and **clientid** from them
        - provide them with a **REDIRECTURL**
    - mysuperawesomesite.com
        - user can choose to oauth2 with form button
    - user gets sent to oauth2 website
        - our code calls
            - **AUTHCODEURL** which sends over
                - state
                - scope
                - clientid

        - user is asked to do this at oauth2 website provider
            - sign in
            - grant permissions
    - oauth2 website sends back to my **REDIRECTURL** (in config) and adds query parameters to the URL
        - sends back
            - state
            - code
    - we now call **EXCHANGE**
        - takes in
            - **context**
            - **code**
            - **client secret**
        - makes a call to oauth2 site sending this stuff over
            - calls to **TOKENURL**
        - we get back a token
    - we don't know if we have an **accesstoken** that has expired or a **refreshtoken** so we call **TOKENSOURCE** which is a method that will give us back a value of type **TOKENSOURCE** which has a method called **TOKEN()** that can be called to return a valid **TOKEN**
        - calls to **TOKENURL**
    - call **NEWCLIENT** to get a client
    - now, using the client, make a get request to oauth2 provider
        - this will be some endpoint from oauth2 provider not already established in code
    - now we have the USERID of the user on the oauth2 provider's site

```
{"email":"tuddleymc@gmail.com","name":"Todd
McLeod","user_id":"amzn1.account.AFKG2XU7BJYC27TS7BBNSVXYISUQ"}
```

video: 72 Hands On 4 - Part 1
video: 72 Hands On 4 - Part 2

Hands-on exercise #4 - continued

- Modify the server from the previous exercise
- Create a new map **oauthConnections**
    - Key should be user IDs from the oauth provider
    - Value should be user IDs in your own system
- In endpoint **/oauth/<your provider>/receive**
    - Extract just the user ID from the result of your call in the previous exercise

                - This will usually require creating a struct and unmarshalling/decoding json data
        - maybe somebody has previously registered with this oauth2 provider with our site
            - so get the local user ID from the **oauthConnections** map
                - If there was no value in the map, set the user ID to a random user ID from your users map
        - Create a session for your user just like how **/login** does it
            - Good candidate for pulling out into a separate function
        - Redirect the user back to **/**

tag v0.41.0
video: 73 Hands On 4 - Continued - part 1
video: 73 Hands On 4 - Continued - part 2
video: 73 Hands On 4 - Continued - part 3

Hands-on exercise #5

For this hands-on exercise:

- Modify the server from the previous exercise
- In endpoint **/oauth/<your provider>/receive**
    - When there is no value in the **oauthConnections** map
        - Sign the oauth provider's user ID
            - Your **createToken** function for creating a JWT token should work
        - Redirect the user to **/partial-register**
            - Include the signed user ID in a query parameter
            - Also include any extra information you may get from the oauth provider (name, email, etc.)
                - Make sure to query escape the values
- Create endpoint **/partial-register**
    - Send users an html page
    - Page should have a form
        - Form should let the users fill in any information needed for their account
            - Pre-fill the values of any fields with the values from the query parameters, so the user may edit them if they wish
            - Example extra information:
                - Name
                - Age
                - Agree to the Terms of Service
                - Email
            - Use an input type **hidden** to include the signed user ID
        - Form should post to **/oauth/<your provider>/register**

tag: v0.42.0
video: 74 Hands On 5

Hands-on exercise #6

For this hands-on exercise:

- Modify the server from the previous exercise
- Create endpoint **/oauth/<your provider>/register**
    - Extract the oauth provider's user ID from its token
        - Your **parseToken** function should work
        - Send a user back to **/** if there is a problem
    - Create an entry in your user map
        - Fill in all information using the submitted form
        - Leave the bcrypted password field blank
    - Create an entry in your **oauthConnections** map
        - Key will be your provider's user ID
        - Value will be the new user's ID
    - Create a session for your user just like how **/login** and **/oauth/<your provider>/receive** does it
    - Redirect the user back to **/**
- Make sure your **/login** endpoint will not log in anyone if they have no password

tag: v0.43.0
video: 75 Hands On 6

Reflections

Review

Look at all we've learned!
video: 76 Review - Part 1
video: 76 Review - Part 2

Farewell

Congratulations!

- Great job completing the course. I am so proud of you.

Video file: 78 Congrats

Bonus lecture