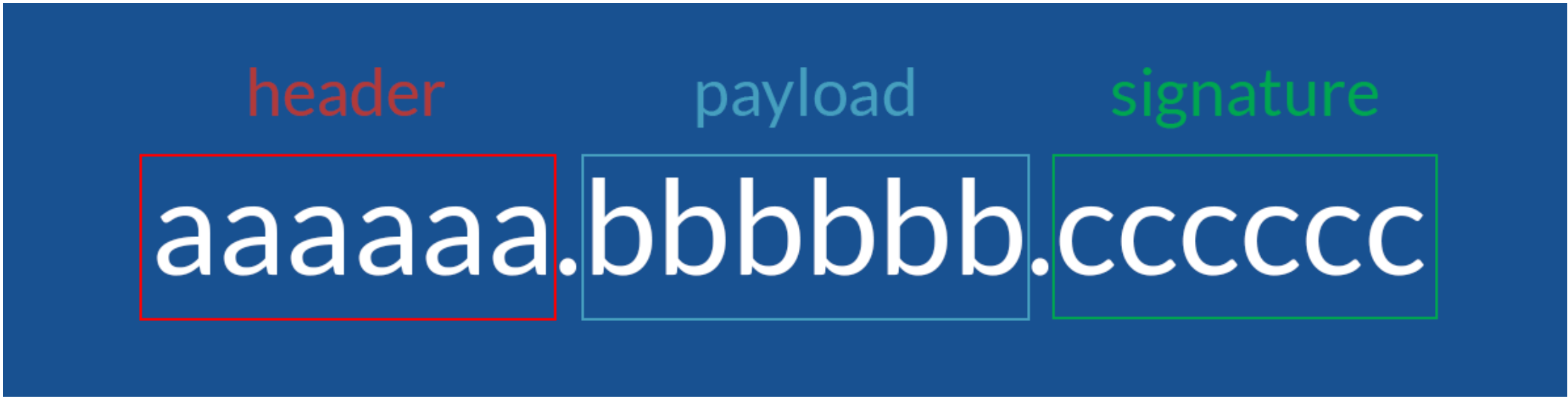
JWT Issues

A high level intro and look at the issues in JWT

JWT

- aaaaaaaaaa.bbbbbbbbbbb.ccccccccccc



Header


```
{
  "typ": "JWT",
  "alg": "HS256"
}
```

Payload

Claims that are not mandatory whose names are reserved for us. These include:

iss: The issuer of the token

sub: The subject of the token

aud: The audience of the token

exp: This will probably be the registered claim most often used. This will define the expiration in NumericDate value. The expiration MUST be after the current date/time.

nbf: Defines the time before which the JWT MUST NOT be accepted for processing

iat: The time the JWT was issued. Can be used to determine the age of the JWT

jti: Unique identifier for the JWT. Can be used to prevent the JWT from being replayed. This is helpful for a one time use token.

Signature

```
var encodedString = base64UrlEncode(header) + "." + base64UrlEncode(payload);

HMACSHA256(encodedString, 'secret');
```

Claimed Benefits

- Easier to (horizontally) scale
- Easier to use
- More flexible
- More secure
- Built-in expiration functionality
- No need to ask users for 'cookie consent'
- Prevents CSRF
- Works better on mobile
- Works for users that block cookies

Lets look Deeper

- Stateless JWT: A JWT token that contains the session data, encoded directly into the token.
- Stateful JWT: A JWT token that contains just a reference or ID for the session. The session data is stored server-side.
- Session token/cookie: A standard (optionally signed) session ID, like web frameworks have been using for a long time. The session data is stored server-side.

Misunderstanding

- “they are not sent as a cookie"
- Wrong! use TLS if you want to protect against interception
- Cookies have more security “secure” “httponly”

Protects against CSRF

- If used as a cookie, same problem as a normal cookie
- Use as local storage, brings its own set of problems
- Local storage, unlike cookies, doesn’t send the contents of your data store with every single request. The only way to retrieve data out of local storage is by using JavaScript, which means any attacker supplied JavaScript that passes the Content Security Policy can access and exfiltrate it. Not only that, but JavaScript also doesn’t care or track whether or not the data is sent over HTTPS. As far as JavaScript is concerned, it’s just data and the browser will operate on it like it would any other data

Issues

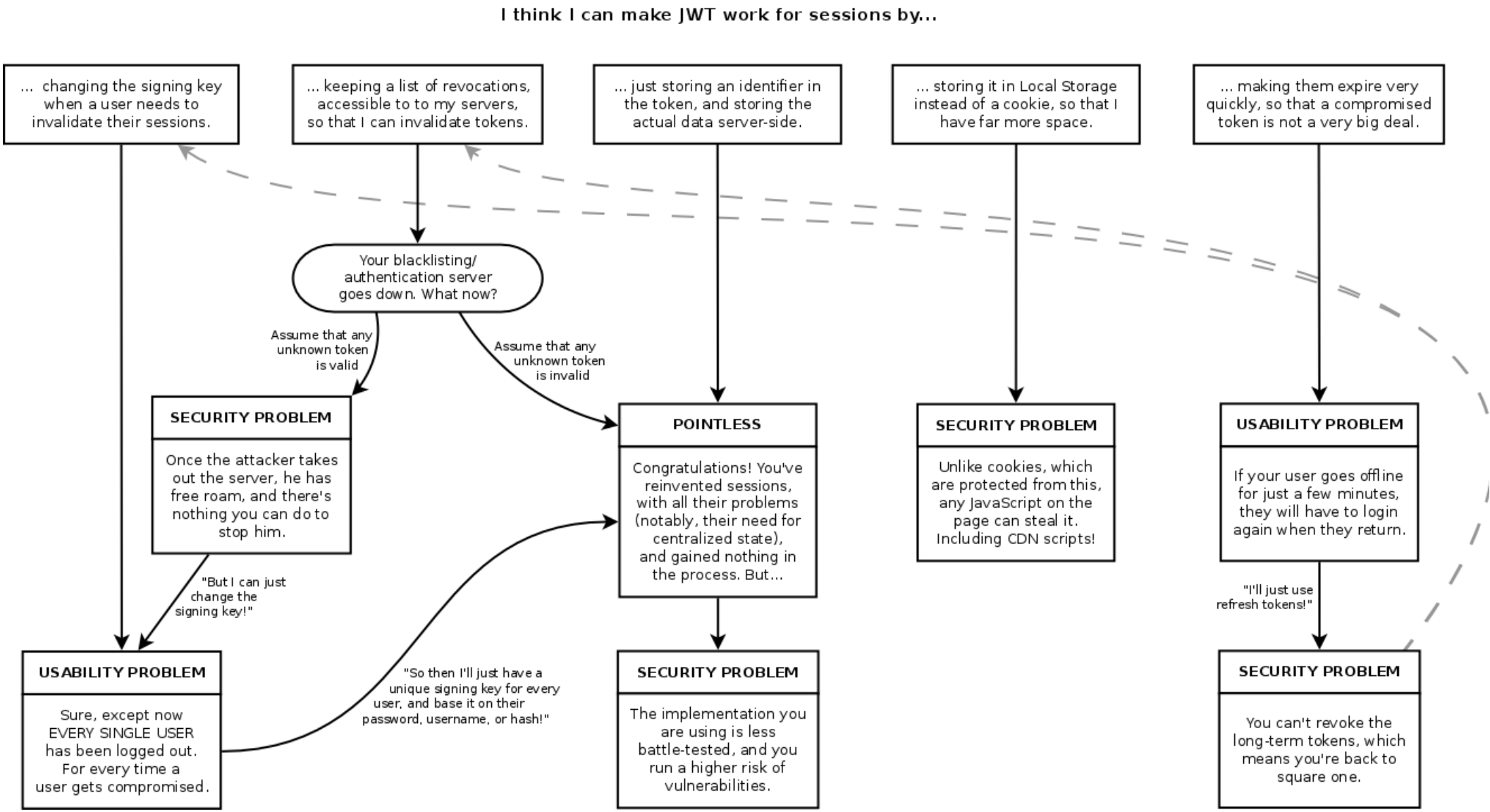
- You cannot invalidate individual JWT tokens
- It can also mean somebody has a token with a role of admin, even though you've just revoked their admin role. Because you can't invalidate tokens either, there's no way for you to remove their administrator access, short of shutting down the entire system.
-

Issues

- Implementation less battle tested
- Updating sessions cannot signout for new versions
-

Stop using JWT for sessions, part 2

A handy dandy (and slightly sarcastic) flowchart about why your "solution" doesn't work



Credits

- Most of the text + diagrams based on prior work from
- http://cryto.net/~joepie91/blog/2016/06/13/stop-using-jwt-for-sessions/
- https://scotch.io/tutorials/the-ins-and-outs-of-token-based-authentication