Running system tests with active authn/z

The story starts with /.well-known/security.txt

Objective

Encourage teams to divert more effort on to system testing with authn/z (and other security features) activated.

Active = Components running in "production mode", security features like authn/z not mocked or disabled

Why do we pursue this objective?

It will have an impact on mitigating well known security risks!

OWASP Top 10:2021 risks

- A01-Broken Access Control
- A05-Security Misconfiguration
- A07-Identification and Authentication Failures
- A09-Security Logging and Monitoring Failures

OWASP API Top 10:2019 risks

- API1 Broken Object Level Authorization
- API2 Broken User Authentication
- API5 Broken Function Level Authorization
- API7 Security Misconfiguration
- API10 Insufficient Logging & Monitoring

Disclaimer

The more I learn about the topic at hand, the more I realize how complex it is. I am not an expert. Life is not simple. All solutions have trade-off's. There are always more than one solution. Context matters - a lot!

Code examples is not production quality!!
Some patterns that are discussed are quite risky - be cautious
Doing a threat model exercise of your test system would be the smart thing to do!

Whoami

- Lars Kåre Skjørestad - **Lars** will do just fine 🙂
- Application Security Advocate @Equinor (www.equinor.com)
- Curious since late 60's, abusing computers since mid 80's
- Living at the west coast of Norway
- 1 wife, 3 grown-up kids
- Hobbies with too little time to pursue (besides coding), cycling, fly-fishing, hunting, hiking
- larskaare @ linkedIn, twitter and github

Outline

- Observations
- Presenting our demo system
- Test principles - Technical Challenges
- Exploring one potential solution
- Wrapping up
- Q&A

Slides / code will be made available.

"There is no end state for application security, we just learn and improve"

Observations

Teams reports that

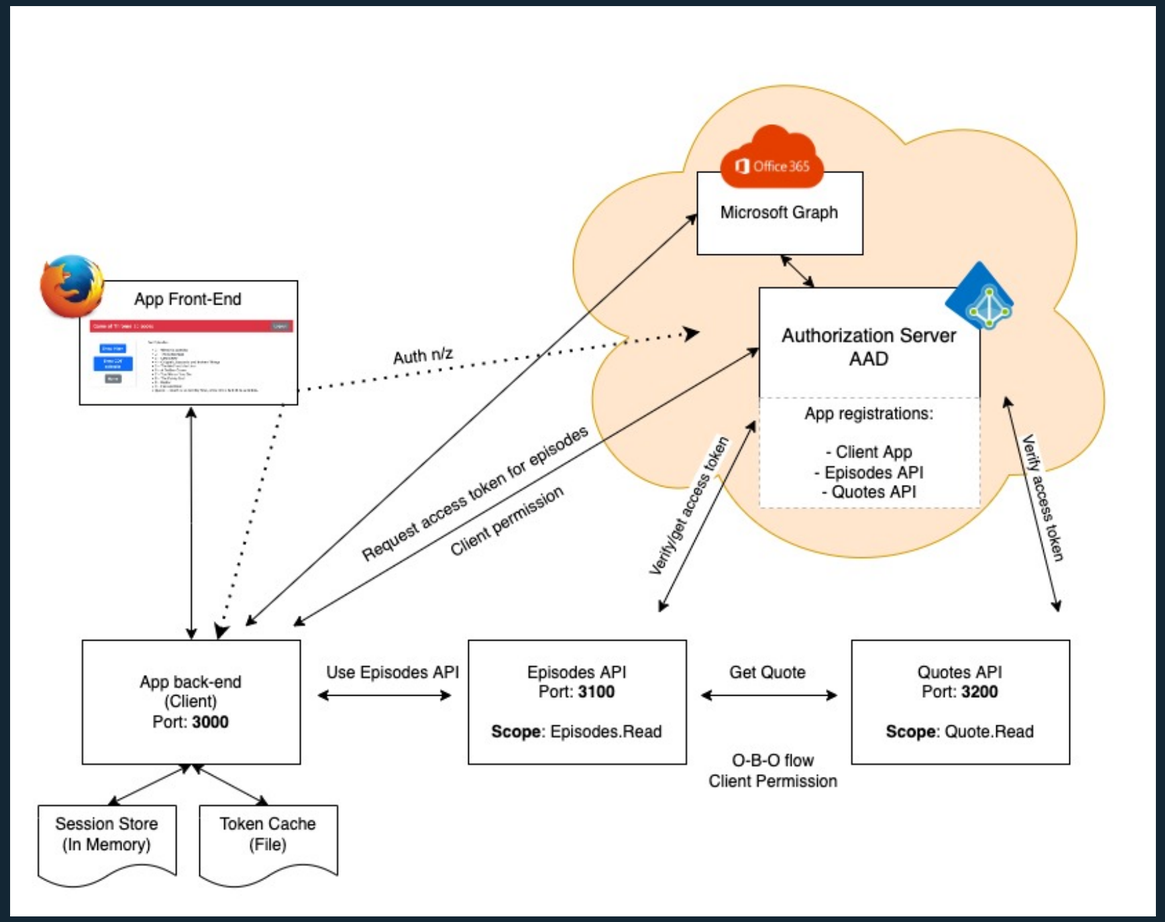
- 80%+ of test effort is spent on unit testing, very little on system testing.
- Authn/z are either mocked/stubbed or disabled when testing
- In testing, security features are often disabled/mocked
- We don't write security related tests
- In "modern cloud native systems", especially when we are hurrying into the cloud and shifting security left - we really should do more system testing
- System testing is hard !/?
- The challenge have multiple perspectives
  - technical - personal skills/proficiency - team compositions - team capacity
  - and not the least - team/company culture!

Presenting our demo system

... context matters ...

The Application

System Components



We are using OICD and oAuth2 for authn/z
We are using the Microsoft Authentication Library (MSAL) in the client back-end
We are using a "back-end for front-end" pattern for the client component
We will be using oAuth2 ROPC rather than Code Grant for the client login in test mode

The Feature we test

For our testing we are focusing on one specific feature;
using the browser to retrieve a list of Game of Thrones episodes for a logged in user.

Further details are documented in our scenario documentation.

Test Principles - Technical Challenges

Test Principles

We could benefit from having a few principles to guide our testing.

A few selected could be:

1. We automate more or less all of our testing
2. Non-functional tests are as important as the functional ones (performance, usability, reliability, security ....)
3. Don't test in production
4. Use synthetic test data
5. Running code should not know that it's under test
6. Testing should not involve components that we don't control/own

The list is not exhaustive. Teams should have their own list!

Technical challenges

- We must do a login through the browser (to create a valid session)
  - This comes with many issues on it's own.
  - Handing over authentication to a different site will typically break many web drivers.
- There are no tokens in the browser, we must utilize the backend.
- Could we acquire tokens in the tests and manipulate the token cache?
- Could we mess around with the session objects in the back-end?
- Corporate environments/users have MFA enabled.
  - Makes login automation more or less impossible...
  - And we should not put test users in our prod system anyway?
- Which IAM solution to use (Corporate Prod, Development)?
  - The same vendor we use in prod, Azure AD (AAD) for us.
- Could we mock the IAM with an "oAuth2 Mock Server" component?

The choices we make should, as far as possible, not disable the security features of our system!
We would like to develop as little as possible of "test supporting code"

A more elaborate discussion is available.

Time to make some decisions

... all solutions/choices are trade-off's trying to balance risk/impact? ...

- We use Azure AD, using the MS O365 developer program to acquire separate Azure test environment
- We create synthetic test users with out MFA to support automated tests
- Our Client component will "know" that's its under test
  - The "test footprint" should be as little as possible
  - If in test, we use a different flow (ROPC) to login and get the initial tokens
- We use Docker and docker-compose to build and serve the system infrastructure
- We drive the test from the browser and from the perspective of the end-user.
  - For this we use Cypress

The system tests must be fully automated and be able to execute in a pipeline.

Exploring one potential solution

Running the application

- We are putting the system components into containers
- We have created Azure AD objects for the components
- We user Docker-compose to run the application

Demo of running application.

Testing in "private" mode, discussing some of the testing challenges

Putting the system in test mode

- We use the NODE_ENV environment variable
  to tell the system it is under test
- When testing, we switch from the oAuth2 **Auth Code** flow to **ROPC**
- We provide user name/pwd from the back-end as config params.

Demo of running application in test mode

Exploring the test mode code and the login

Running the test

- We use Cypress to drive the tests

Demo of Cypress running the test

Bringing it all together

- We use docker-compose to
  - Start the system components
  - Put the system under test
  - Run headless tests in Cypress
  - Use exit-code from Cypress for further actions

```
./source ~/path-to-my-env-file/pawa-compose.env
export NODE_ENV=test
docker-compose up --abort-on-container-exit --exit-code-from cypress
```

Wrapping up

Current status

For our "complex and modern system" we now have:

- A fully automation system test scenario
- A fully synthetic test environment System tests with active Authn/z
- System tests with active security controls
- We have very little test support code

We have broken a few principles to reach this state -
the trade-offs could be acceptable?

Again, a threat modeling exercise on the test system would be smart.
What could possibly go wrong 🙂

Our Objective

Encourage teams to divert more effort on to system testing with authn/z (and other security features) activated.

It is hard, not only a technical challenge
- but it is doable - and necessary ...

Q&A

Thank You

I am larskaare @ linkedIn, twitter and github

Visit loop.equinor.com to learn more about the software developer community in Equinor.

Slides and code are available at https://github.com/larskaare/pawa-system-testing