STRIDE Model

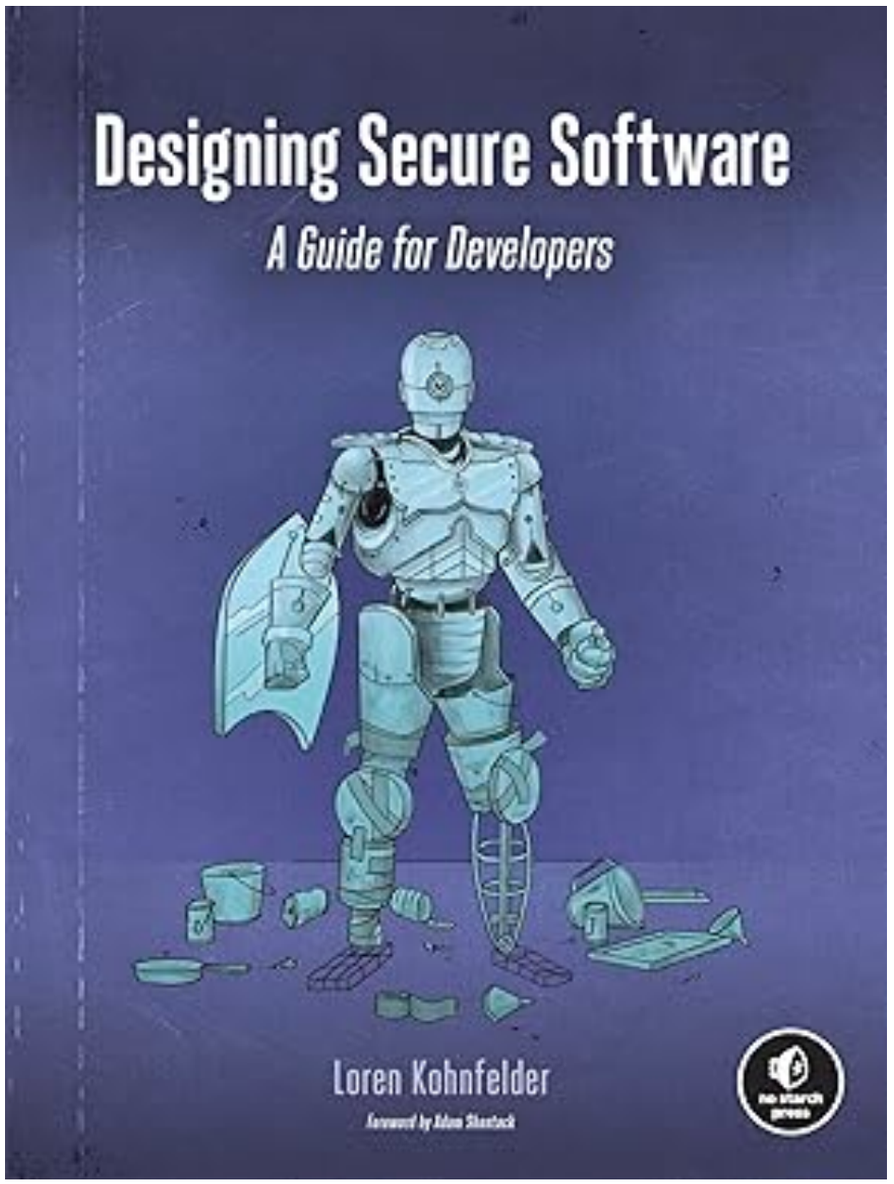
Threat	Desired property	Threat Definition
Spoofing	Authenticity	Pretending to be something or someone other than yourself
Tampering	Integrity	Modifying something on disk, network, memory, or elsewhere
Repudiation	Non-repudiability	Claiming that you didn't do something or were not responsible; can be honest or false
Information disclosure	Confidentiality	Someone obtaining information they are not authorized to access
Denial of service	Availability	Exhausting resources needed to provide service
Elevation of privilege	Authorization	Allowing someone to do something they are not authorized to do

CVSS (Common Vulnerability Scoring System)

CVSS Base Score	CVSS Severity Level
0	None
0.1 - 3.9	Low
4.0 - 6.9	Medium
7.0 - 8.9	High
9.0 - 10.0	Critical

Factors

- Attack Vector
- Attack Complexity
- Privileges Required
- User Interaction
- Scope
- Confidentiality
- Integrity
- Availability



8 Secure Programming

Topics

- The Challenge
  - Malicious Influence
  - Vulnerabilities are Bugs
  - Vulnerability Chains
  - Bugs and Entropy
- Case Study: GotoFail
  - One-Line Vulnerability
  - Beware of Footguns
  - Lessons from GotoFail

Topics

- Coding Vulnerabilities
  - Atomicity
  - Timing Attacks
  - Serialization
- The Usual Suspects

The Challenge

Security Cops

- Criticize developers in unhelpful ways
- Software is fragile and complex
- Professional developers know how to test and debug code
  - But security is another matter
    - Vulnerable code usually works (when not attacked)
- Idealized design may be secure
  - But actual implementation may introduce vulnerabilities

Malicious Influence

- Untrusted inputs may influence code
  - Directly or indirectly
- An attack string may avoid rejection and propagate deeper into the system
  - This is called ***tainting***

Vulnerabilities are Bugs

- All software has bugs
- ***Vulnerabilities*** are bugs attackers can use to cause harm
- A website layout design flaw is probably just a **bug**
- An exposed administrative interface is a ***vulnerability***

Vulnerability Chains

- Several minor bugs can combine
  - To create a serious vulnerability
- Example
  - Long ago, a developer noticed that orders without a valid warehouse ID led to automatic refunds and the order was sent to another warehouse and fulfilled
    - This bug was low-ranked because customer has no way to change the warehouse ID
- A new change put the Warehouse ID in an editable field on the order form
  - But if the customer changes it, the order should be rejected
  - So this bug is low-ranked without testing
- Now customers who change the ID get products and refunds

Bugs and Entropy

- Why do we need to reboot our phones occasionally?
- Entropy (disorder) accumulates, perturbing the system in unpredictable ways
  - Unexpected interactions between execution threads
  - Memory corruption on stack and heap

Vigilance

- Errors are common in the easy part of code
  - Where you ***aren't paying attention***
- Vigilance requires discipline at first
  - With practice it becomes second nature

Case Study: GotoFail

One-Line Vulnerability

- Apple's code to verify SSL certificate signatures
- Intention was to test for three errors
  - Return nonzero **err** if any of the three tests fail
  - Return zero **err** if all three tests pass

```
        if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
                goto fail;
        if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
                goto fail;
                goto fail;
        if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
                goto fail;
        ...

fail:
        SSLFreeBuffer(&signedHashes);
        SSLFreeBuffer(&hashCtx);
        return err;
}
```

One-Line Vulnerability

- The extra **goto fail;**
  - Causes it to skip the last test
  - And return zero if the first two tests passed
- So invalid signatures are accepted
- The outlined code is **dead** (never executed)

```
        if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
                goto fail;
        if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
                goto fail;
                goto fail;
        if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
                goto fail;
        ...

fail:
        SSLFreeBuffer(&signedHashes);
        SSLFreeBuffer(&hashCtx);
        return err;
}
```

Beware of Footguns

- The outlined line should be indented less
- This error would have been more obvious in Python
  - Which enforces correct indentation

```
        if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
                goto fail;
        if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
                goto fail;
                goto fail;
        if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
                goto fail;
        ...

fail:
        SSLFreeBuffer(&signedHashes);
        SSLFreeBuffer(&hashCtx);
        return err;
}
```

Equals Signs

- This line tests to see if **x** equals 8

```
if (x == 8)
```

- This line assigns **x** to the value 8
- And then considers the result **true**

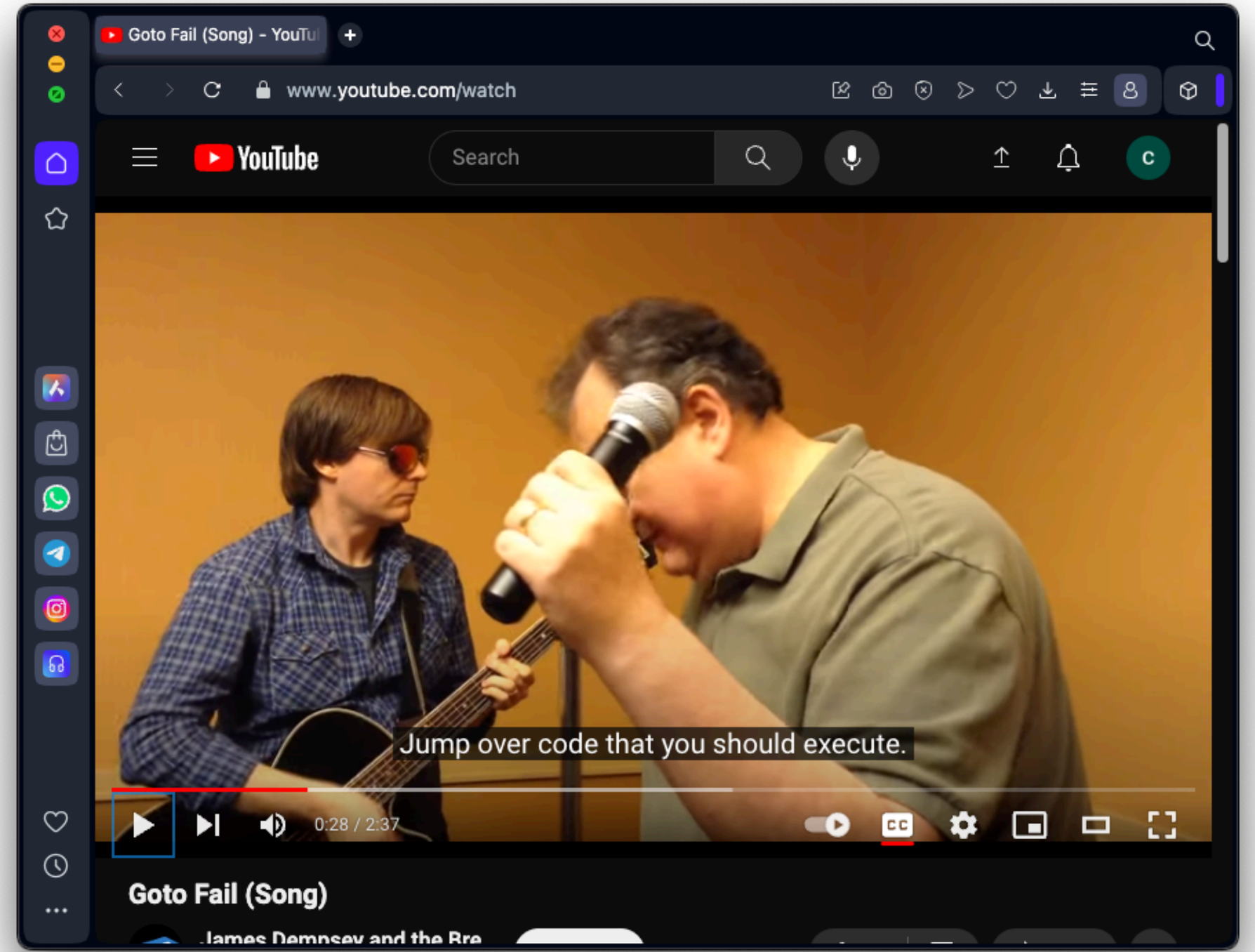
```
if (x = 8)
```

Lessons from GotoFail

- **Small slips in critical code can have a devastating impact on security.**
- **The vulnerable code still works correctly in the expected case.**
- **It’s arguably more important for security to test that code like this rejects invalid cases than that it passes the normal legit uses.**
- **Code reviews are an important check against bugs introduced by oversight. It’s hard to imagine how a careful reviewer looking at a code diff could miss this.**

Useful Countermeasures

- **Better testing**
  - Test each of those **if** statements
- Watch out for **unreachable code**
- Make code flow explicit, with parentheses and curly braces
  - Even where they could be omitted
- Source code analysis such as "linters"
- Ad hoc source code filters for recurrent errors
- Measure and require full test coverage
  - Especially for security-critical code



- https://www.youtube.com/watch?v=tQms037U72w

Coding Vulnerabilities

Atomicity

- Servers are running many threads and processes
- They may interact, creating temporary errors
  - ***Race conditions***
- ***Atomicity*** describes operations that are guaranteed to be completed as a single step
  - The whole operation either succeeds or fails

Python Example

- The **tempfile.mktemp** function returned the name of a temporary file guaranteed not to exist
- But another process might call the same function before your code writes to it, getting the same file name
  - The **tempfile.mktemp** function is now deprecated
- Use **tempfile.NamedTemporaryFile** instead
  - An atomic operation creates and opens the temporary file
  - Nothing can intervene in the process

Timing Attacks

- Measuring the time an operation takes
  - Reveals secret information
  - Such as how many 1s are in a key
- ***Meltdown*** and ***Spectre*** are timing attacks related to "speculative execution"
  - Processor races ahead, performing instructions in advance
  - If those instructions are skipped, it attempts to cancel them
- But cached results remain, changing the timing of later read operations

Mitigations

- Reduce the time differential to an imperceptible level
- Introduce an artificial delay to blur the timing signal

Serialization

- Convert data objects to a byte stream
- When received, they are ***deserialized***
  - To retrieve orignal data
- Deserializing malicious inputs may do malicious things
  - Python's **pickle** serialization can be tricked into executing arbitrary code

Mitigation

- Add a MAC or digital signature to the serialized data
  - So it cannot be altered or forged
- Or avoid serialization entirely

Java Web Token

Encoded PASTE A TOKEN HERE


```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKx
wRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```


Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

The Usual Suspects

Upcoming Topics

- Fixed-width integer vulnerabilities
- Floating-point precision vulnerabilities
- Buffer overflow and other memory management issues
- Input validation
- Character string mishandling
- Injection attacks
- Web security

Ch 9