EECS 498/598: Encrypted Systems
Winter 2022
Lecture 3: Security Preliminaries

Paul Grubbs

paulgrub@umich.edu

Beyster 4709

Agenda for this lecture

- Announcements
- Security fundamentals, abstractly
- Case study: messaging app
- Discussion

Agenda for this lecture

- Announcements
- Security fundamentals, abstractly
- Case study: messaging app
- Discussion

Announcements

- List your preferences for presentations in the Google doc
- My PCR test was negative – see you all in person on Tuesday!

Agenda for this lecture

- Announcements
- Security fundamentals, abstractly
- Case study: messaging app
- Discussion

Lecture Goals

- Review basics of security “thinking”
  - Learn to critically read the “threat model” section of papers
- Practice skills with a case study

Security Fundamentals

1. Understand system architecture
2. Enumerate security goals + non-goals
3. Identify potential attacks
4. Propose countermeasure
5. Analyze countermeasure, argue it prevents attack

Security Fundamentals

1. Understand system architecture
2. Enumerate security goals + non-goals
3. Identify potential attacks
4. Propose countermeasure
5. Analyze countermeasure, argue it prevents attack

System architecture

- What computers are there?
  - Software/hardware
  - How do they communicate?
    What do they communicate?

- Identify actors (human)
  - Users
  - Administration
  - Develops?

- Correctness goals / functionality
  - Purpose of system
  - How are inputs processed?

Security goals and non-goals

- what security properties might be desired
  - Data privacy, authenticity
  - No denial of service
- Non-goals: what is impossible to provide?

Identify attacks

- Who could the attackers be?
  - → Actors could be malicious
- What are the capabilities of attackers? What can they try to do?
- Articulate security goal that is violated

Propose countermeasure

- what changes can be made?
- what techniques can be used
  - cryptographic vs not
- Impact on efficiency/correctness?
- Write it out in detail!

↑ prevents mistakes

Analyze countermeasure

- Formally / informally
  - → verification, model checking
    Hard (but... getting easier)
  - → make precise model, prove countermeasure } common in our papers
  - → Thinking hard, making informal argument

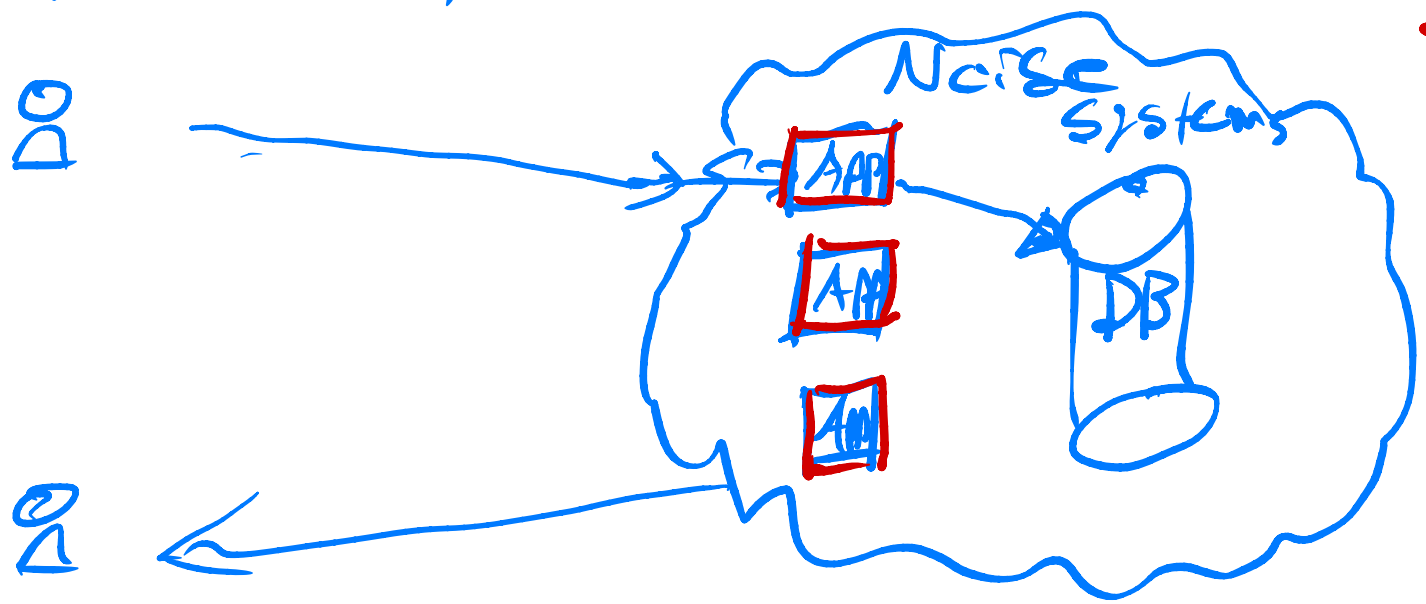
Agenda for this lecture

- Announcements
- Security fundamentals, abstractly
- Case study: messaging app
- Discussion

Case study: messaging apps

Noise Systems has a messaging app

- Users download to phones
- Communicate with other user



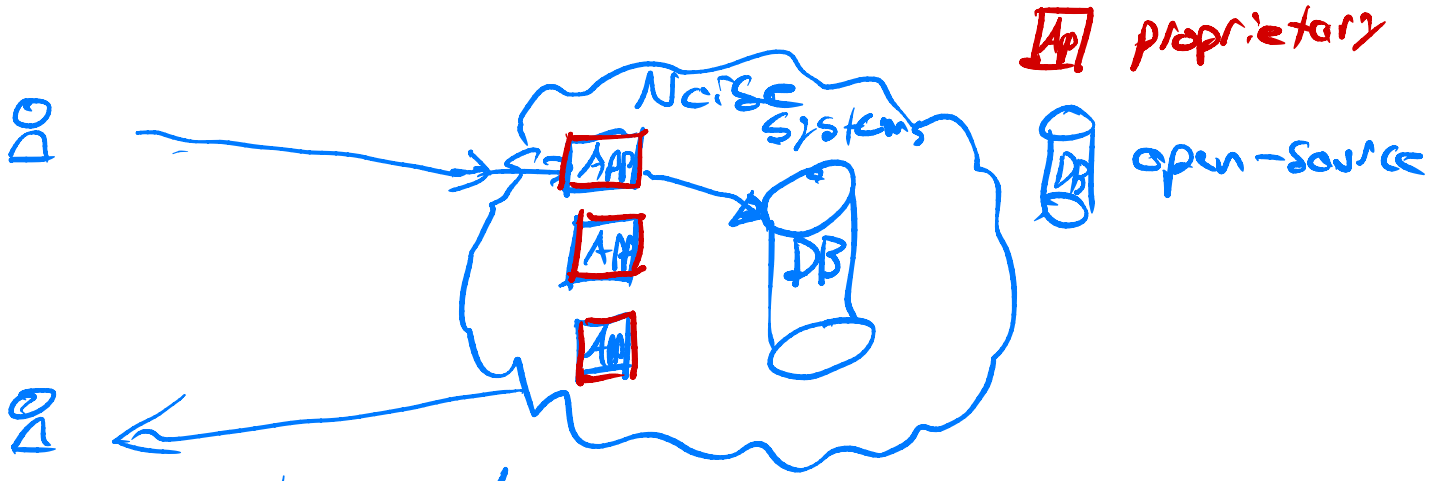
App proprietary

DB open-source

Case study: messaging apps

System architecture

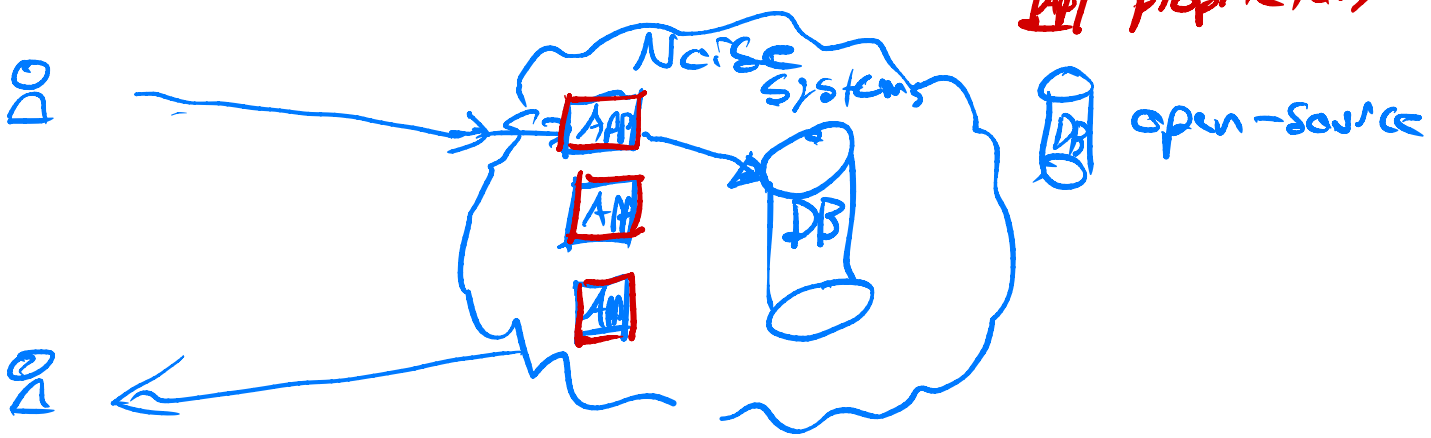
Digital:
- user apps/phones
- servers (App)
- DB servers
- Network
  - external to Noise
  - internal Noise



Human:
- Noise users
- Noise admins
- Network admins
- Noise developers
- DB developers

Correctness/functionality:
- Deliver messages
  - B gets a message from A iff
    - A sent that msg to B
    - messages delivered as sent
      - No changes
      - message ordering is correct
    - Hard!!

Case study: messaging apps



Security goals:

- No user can see other user's messages
- No access to DB servers except through app servers
- No message modifications (integrity)
- No impersonation (authenticity)
- No unauthorized leaking of messages by receivers
- Third-party apps have access control for sensitive APIs
- Availability in the face of DoS attacks
- Messages hidden from (external) network
- messages hidden from noise systems
- metadata hiding

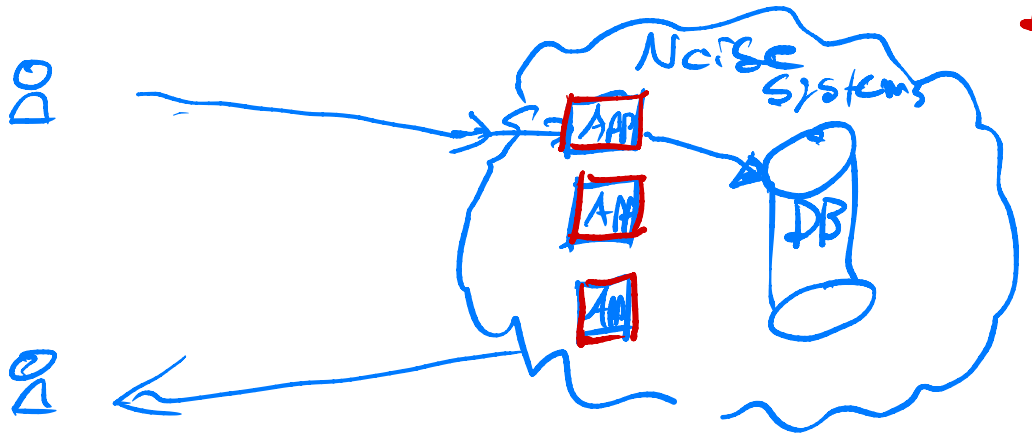
Security non-goals:

- content moderation (stopping certain kinds of content)
- Ability to delete your own content
  - Deniability?
- Admins can't take system down
- Denying use of app (phone vs. network)

Case study: messaging apps

Attacks:

- impersonate Noise systems server, read messages
- use botnet to DDoS servers
- SQL injection on DBs
- eavesdrop on user → Noise traffic
- stealing physical DB hard disks
- Login to user accts, send messages + read messages
- Developers backdoor Noise server app
- Deleting messages from DB
- DoS by crashing servers from within Noise
- Clone messages from within Noise app, re-route messages
- Noise devs distribute fake or corrupted mobile apps
- Trick user into installing malware, read Noise messages
- user → user malware



App proprietary

DB open-source

- social engineering through Noise to steal credentials / PII
- user impersonates other user (logs in as other user)
- DB developer exfiltrates data using covert channel

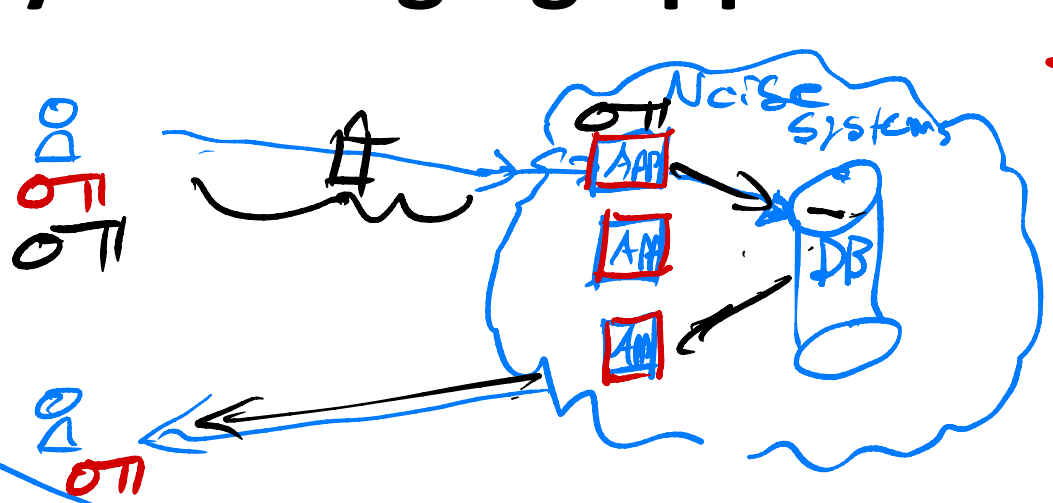
Case study: messaging apps

Attacks:

- impersonate Noise systems server, read messages
- use botnet to DDoS servers
- SQL injection on DBs
- eavesdrop on user → noise traffic
- stealing physical DB hard disks
- Login to user accts, send messages + read messages
- Developers backdoor Noise server app
- Deleting messages from DB
- DoS by crashing servers from within Noise
- Clone messages from within Noise app, re-route messages
- Noise devs distribute fake or corrupted mobile app
- Trick user into installing malware, read Noise messages
  - user → user malware

Counter measure?

? Mafia



proprietary

open-source

Use prepared statements to prevent SQL injection

Bluc checks!

Encrypt data in transit

exchange info out of band! Security codes?

each pair has new key. Authentication? PGP keys? Two-factor auth?

2FA

- social engineering through Noise to steal credentials / PII
- User impersonates other user (logs in as other user)
- DB developer exfiltrates data using covert channel

Prevent data from being sent externally to Noise app. Limit third party permissions

verify app hash to prevent malicious app. stops external but not internal attacks. Threshold signing for binaries?

Agenda for this lecture

- Announcements
- Security fundamentals, abstractly
- Case study: messaging app
- Discussion

Discussion

Resources:

- *Computer Security: Art and Science,* Bishop
- *Security Engineering,* Anderson
- *Computer Security and the Internet,* van Oorschot