Security: Authentication

CS 475, Spring 2018
Concurrent & Distributed Systems
Byzantine Faults

Set A=5

"OK"!

Read A

Set A=5

"OK!"

"6"!
Oral BFT Example ($n=4$, $m=1$)

Commander

Lieutenant 1

Lieutenant 2

Lieutenant 3
Oral BFT

- At best, can tolerate $m$ failures from $3m+1$ participants
  - Ensures you always have a majority of valid participants
- If the loyal lieutenants decide the general is a traitor, they need to have some predefined behavior
- This is really expensive (communication)
  - To tolerate $m$ traitors among $n$ participants, or OM($m$), each of $n-1$ participants will invoke this OM($m-1$) times
  - OM($m-1$) will cause $n-2$ participants to call OM($m-2$)
  - Overall number of messages: $O(n^m)$
  - Example: tolerate 3 failures from 10 participants: 1,000 messages
Blockchains

- Solution: make it hard for participants to take over the network; provide rewards for participants so they will still participate
- Each participant stores the entire record of transactions as blocks
- Each block contains some number of transactions and the hash of the previous block
- All participants follow a set of rules to determine if a new block is valid
Announcements

• Form a team and get started on the project!
  • http://jonbell.net/gmu-cs-475-spring-2018/final-project/
  • AutoLab available

• Today:
  • More security
Threat Models

• What is being defended?
  • What resources are important to defend?
  • What malicious actors exist and what attacks might they employ?
• Who do we trust?
  • What entities or parts of system can be considered secure and trusted
  • Have to trust something!
Blockchain & Trust

- Miners don't trust people submitting transactions
  - If you accept an invalid transaction then try to include it in your block, block is rejected
- Miners don't trust each other
  - If you include invalid transactions: rejected
- Nobody trusts miners
  - Requires expending effort to get a new block in
What does it mean for a distributed system to be secure?

• Maintain a secure channel between nodes:
  • Authenticity (Who am I talking to?)
  • Confidentiality (Is my data hidden?)
  • Integrity (Has my data been modified?)
  • Availability (Can I reach the destination?)

• Maintain some security about who participates in the system?

• What cryptographic tools are available to us?
Security isn't (always) free

• You just moved to a new house, someone just moved out of it. What do you do to protect your belongings/property?
• Do you change the locks?
• Do you buy security cameras?
• Do you hire a security guard?
• Do you even bother locking the door?
Security: Managing Risk

• Security architecture is a set of mechanisms and policies that we build into our system to mitigate risks from threats

• Threat: potential event that could compromise a security requirement

• Attack: realization of a threat

• Vulnerability: a characteristic or flaw in system design or implementation, or in the security procedures, that, if exploited, could result in a security compromise
Costs & Benefits

• Increasing security might:
  • Increase development & maintenance cost
  • Increase infrastructure requirements
  • Degrade performance
• But, if we are attacked, increasing security might also:
  • Decrease financial and intangible losses
• So: How likely do we think we are to be attacked in way $X$?
Example Threat: Web Server

HTTP Request

HTTP Response

client page
(the “user”)

server
Example Threat: Web Server

client page (the “user”) → HTTP Request → HTTP Response ↔ server

Do I trust that this request *really* came from the user?
Example Threat: Web Server

Do I trust that this request *really* came from the user?

Do I trust that this response *really* came from the server?

client page (the "user")

server
Example Threat: Web Server

client page (the “user”)

malicious actor “black hat”

server

Do I trust that this response really came from the server?

Do I trust that this request really came from the user?
Example Threat: Web Server

Might be “man in the middle” that intercepts requests and impersonates user or server.

client page (the “user”)

malicious actor “black hat”

server

HTTP Request

HTTP Response

HTTP Request

HTTP Response

Do I trust that this response really came from the server?

Do I trust that this request really came from the user?
Other Risks

- Is our network well behaved?
- Is our network malicious?
- Who can access our system?
- Are our users well-behaved?
- Are our users malicious?
- Is our system well behaved?
Protection Concerns

• Secure channels of communication
  • Authentication: is everyone who they say they are?
  • Confidentiality & integrity: is a third party interfering in our communication?

• Access Controls
  • Authorization: Who has access to an operation/resource?
  • Accountability: Maintaining an audit trail
  • Non-repudiation: A participant can not deny some action that they took with the system
Symmetric encryption, aka shared secret key

- \( M = D_K (E_K (M)) \)
- \( M \) is the data, \( D \) is decrypt, \( E \) is encrypt, and \( k \) is the key
- Computationally efficient (relatively)
- Can have hardware support too (e.g. iPhone)
Asymmetric encryption, aka public key/private key

- $M = D_{K_{priv}}(E_{K_{pub}}(M)) = D_{K_{pub}}(E_{K_{priv}}(M))$
- When a node B wants to send a message to node A, it obtains A’s public key and uses $K_{A_{Public}}$ to encrypt the message.
- Only A can decrypt the message using its private key.
- Computationally expensive.
Public/Private Key Encryption

- Encrypt with public key: only private key holder can decrypt
Public/Private Key Encryption

- Encrypt with private key: anyone with public key can decrypt

<table>
<thead>
<tr>
<th>Private Key</th>
<th>Public Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain text Message</td>
<td>Signed Message</td>
</tr>
</tbody>
</table>
Hashing

- $S = H(M)$
- $S$, aka digest, is a unique representation of data such that an accidental or intentional change to the data will change the representation
- Fixed size and independent of size of $M$
- Computationally efficient

![Diagram of hash function with inputs and corresponding hash values]

Inputs
- Leo McGarry
- Josh Lyman
- Sam Seaborn
- Toby Ziegler

Hash
- 0
- 1
- 2
- 3
- 4
- 5
- 6
- 7
Hashing to verify messages

• Just sending a hash of the message isn't enough!
• How do we know that a third party didn't tamper with the message and the hash?
• Solution: encrypt the hash using your private key. Anyone can verify that the hash was "signed" by you
## Symmetric vs Asymmetric Crypto

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Symmetric Crypto</th>
<th>Asymmetric Crypto</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requires a pre-shared secret</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Relative speed</td>
<td>Very fast</td>
<td>Very slow</td>
</tr>
</tbody>
</table>
Asymmetric Cryptography

- So, great: no need to pre-share anything, right!
- Widely used for instance... HTTPS! SSL!
- But: there's a bootstrapping problem
- When you visit amazon.com, the site will sign its content using its private key
- You can use amazon.com's public key to verify it's really from amazon.com
- How do you know what amazon.com's public key is though?
- “PKI” - Public Key Infrastructure
Certificate Authorities

• A certificate authority (or CA) binds some public key to a real-world entity that we might be familiar with

• The CA is the clearinghouse that verifies that amazon.com is truly amazon.com

• CA creates a certificate that binds amazon.com's public key to the CA’s public key (signing it using the CA’s private key)
Certificate Authorities

Amazon

amazon.com private key

amazon.com public key

Some real-world proof that we are real

amazon.com certificate
(AZ’s public key + CA’s sig)

Certificate Authority

CA private key

CA public key

My Laptop

CA public key
Certificate Authorities

• Note: We had to already know the CA's public key
• There are a small set of “root” CA’s (think: root DNS servers)
• Every computer/browser is shipped with these root CA public keys
Certificate Authorities

- What happens if a CA is compromised, and issues invalid certificates?
- Not good times.

Security

Fuming Google tears Symantec a new one over rogue SSL certs

We've got just the thing for you, Symantec ...

By Iain Thomson in San Francisco 29 Oct 2015 at 21:32

Security

Comodo-gate hacker brags about forged certificate exploit

Tiger-blooded Persian cracker boasts of mighty exploits
Authentication Protocols

- So, we can build SSL based on this public key infrastructure
- How do we support user authentication?
Authentication with multiple service providers

- What happens if we want to build some big distributed system that a user logs into once?

Prof Bell

Todos

REST service

Database

Google Calendar API

Logs into, posts new todo

Connects as user, creates new event

How does Todos tell Google that it’s posting something for Prof Bell? Should Prof Bell tell the Todos app his Google password?
We’ve got something for that…

- Example App would like to:

  - View your basic profile info
  - View your email address

By clicking Allow, you allow this app and Google to use your information in accordance with their respective terms of service and privacy policies. You can change this and other Account permissions at any time.
OAuth

- OAuth is a standard protocol for sharing information about users from a “service provider” to a “consumer app” without them disclosing their password to the consumer app
- 3 key actors:
  - User, consumer app, service provider app
  - E.x. “Prof Bell,” “Todos App,” “Google Calendar”
- Service provider issues a token on the user’s behalf that the consumer can use
- Consumer holds onto this token on behalf of the user
- Protocol could be considered a conversation…
An OAuth Conversation

Goal: **TodosApp** can post events to **User’s** calendar. **TodosApp** never finds out **User’s** email or password.

1: intent
2: permission (to ask)
3: redirect to provider
4: permission to share
5: token created
6: Access resource

**User**

**TodosApp**

**Google Calendar**
A token is a **secret value**. Holding it gives us access to some privileged data. The token identifies our users and app.

Decoded:

```json
"iss": "https://securetoken.google.com/authdemo-72a42",
"name": "Jonathan Bell",
"picture": "https://lh5.googleusercontent.com/-m-OocFU5GLw/AAAAAAAAAAI/AAAAAAAAAH0/BUWkN6DmMRk/photo.jpg",
"aud": "authdemo-72a42",
"auth_time": 1477529371,
"user_id": "JMQrQiu9SRTdx64bTygA3xxDcuH2",
"sub": "JMQrQiu9SRTdx64bTygA3xxDcuH2",
"iat": 1477530885,
"exp": 1477534485,
"email": "jonbellwithnoh@gmail.com",
"email_verified": true,
"firebase": {
  "identities": {
    "google.com": ["109040352574312154216"],
    "email": ["jonbellwithnoh@gmail.com"]
  },
  "sign_in_provider": "google.com"
},
"uid": "JMQrQiu9SRTdx64bTygA3xxDcuH2"
```
Trust in OAuth

- How does the Service provider (Google calendar) know what the TodosApp is?
- Solution: When you set up OAuth for the first time, you must register your consumer app with the service provider
- Let the user decide
  - … they were the one who clicked the link after all
Authentication as a Service

- Whether we are building “microservices” or not, might make sense to farm out our authentication (user registration/logins) to another service
- Why?
  - Security
  - Reliability
  - Convenience
- We can use OAuth for this!
Using an Authentication Service

1: intent
2: permission (to ask)
3: redirect to provider
4: permission to share
5: token created
6: Access resource

User

Firebase