

# Distributed Systems Security

CSE 486: Distributed Systems

Ethan Blanton

Department of Computer Science and Engineering  
University at Buffalo



# Distributed Systems Security

We have **ignored a lot of security** up to this point.

This is because:

- Systems security is **very hard**
- UB offers **entire courses on security**
- Some background in **cryptography** and **cryptographic protocols** is necessary for a thorough treatment

In this lecture, we will:

- touch on some **important security considerations**
- explore some **distributed security protocols**

# Disclaimer Redux

*This lecture is **not a replacement** for a more thorough treatment of security.*

This lecture is **greatly (over?)simplified** for time reasons.

This material is to **give you a place to get started**.

# Applying Security

Like *almost everything we talk about*, security in distributed systems depends on your application.

It is important to:

1. Define your **threat model** (remember elections?)
2. Design a **security model** to meet the threats
3. Select **mechanisms and protocols** to implement your model
4. Demonstrate (ideally *prove*) that your implementation captures the models

# Threat Model

You must **know what you are trying to secure, and what the threats against it are**, before you can secure it.

**Threat modeling** is a discipline for defining this.

You define:

- What am I trying to protect?
- How are those things vulnerable?
- *etc.*

*E.g.*, you might define:

- **authentication credentials** as an asset to protect
- the **authentication database server** as a vulnerable interface to that data

Full threat modeling is out of scope for this course!

# Cryptography

Distributed security solutions must often deal with

- Varying levels of **trust** within the system
- Multiple **authorities**
- **Untrusted infrastructure**
- *etc.*

These features make **encryption**, and in particular **public key cryptography**, an important part of distributed security.

# Threats

Threats can be divided into several categories. One division is:

- **Information Disclosure**: An attacker acquires information that was intended to remain private or controlled.
- **Unauthorized Access**: An attacker obtains access to a system or facility without appropriate permission.
- **Denial of Service**: An attacker prevents legitimate users of a service from accessing it normally.

There are **many threat taxonomies**, and this division may not cleanly capture all possible threats for all systems.

# Vulnerability

A **vulnerability** is a property of an **implemented system** which allows exploitation of a threat.

Examples:

- A **network protocol** may expose an **information disclosure** threat by failing to encrypt data or using an encryption protocol inappropriately.
- An **authentication service** may expose an **unauthorized access** threat by allowing an authentication token to be re-used.
- A web service may expose a **denial of service** threat by providing an **computation-heavy feature** to un-authenticated users.



# Threat Interplay

One vulnerability can lead to another vulnerability.

*E.g.:*

- An authentication system has an **information disclosure vulnerability**
- Leaked credentials lead to **unauthorized access**

In a **distributed system**, this can mean that a **vulnerability in a system administered somewhere else** may lead to **threat exploitation in a local system**.

There are techniques to mitigate this, such as the **principle of least authority**.

# Principle of Least Authority

The **principle of least authority** states that a user or system should have the minimum access required to accomplish their tasks.

Some common techniques to accomplish this are:

- Role-based access control
- Capability-based security
- Privilege separation

Some systems applying this principle:

- Tahoe-LAFS
- EROS: The Extremely Reliable Operating System
- Microsoft Azure

# Cryptography

Cryptography gives us tools to:

- **Encrypt data** so that it cannot be viewed by third parties
- **Agree on a secret** to be used for encryption
- **Sign data** so that its authenticity can be verified
- ...

In distributed systems, cryptography allows systems to **communicate safely and securely over untrusted networks.**

# Terminology I

Some basic terminology (**dangerously vague and sloppy!**):

**Cryptographic protocol:**

A series of steps between to be performed between two or more parties to accomplish some cryptographic goal.

**Key:**

A value used in a cryptographic function which determines its output, and is difficult or impossible to deduce from its other inputs and outputs.

# Terminology II

## Shared-key (symmetric) encryption:

Two parties **share a secret** and use that secret to encrypt.

Examples: AES, Blowfish, ChaCha20

## Public-key (asymmetric) encryption:

One party has a secret key, and shares a related value (the public key) with any other parties. Data encrypted with one key can be decrypted by the other.

Examples: RSA, ElGamal, ECC

# Terminology III

## Message digest (hash):

A data item is passed through a **one-way function**, producing a value that is dependent on the input data but from which it is difficult or impossible to predict properties of the input.

Examples: SHA-256, SHA-3 (Keccak), BLAKE2/3

## Message authentication code (MAC):

A small datum that can be used to verify the authenticity of another, possibly (much) larger, message.

An **HMAC** is a MAC constructed using a **hash function** and a **secret key**.

# Common Cryptographic Tools

**TLS**, or Transport Layer Security (formerly SSL):

A standardized suite of cryptographic protocols used to secure **streaming communication channels**, such as TCP sockets. (TLS can also be used for datagrams.)

**scrypt**:

A method of creating a **key** from a **password**. Scrypt is designed to make brute-force password retrieval from the key, or password guessing to find a key, arbitrarily difficult.

**Kerberos**:

A distributed authentication protocol using shared key cryptography.

# TLS

TLS [2] is an improvement on SSL.

- SSL was developed by Netscape in the mid 1990s.
- SSL provides:
  - authentication of the server
  - optional authentication of the client
  - protection against eavesdropping and man-in-the-middle attacks
- Early versions of SSL were rife with security flaws.
- TLS succeeded SSL, with TLS version 1.3 being current.
- TLS 1.2 is still in wide usage.



# TLS Authentication

TLS authenticates hosts via **public key encryption**.

Servers have a private key with its corresponding public key **signed by a Certificate Authority (CA)**.

Certificate authorities **sign server keys** and issue a **certificate** containing this signature.

- (We won't get into the details of signatures.)

Clients **agree to trust certain CAs** to issue certificates.

# TLS Properties

A stream protected by TLS provides:

- Proof of the server's identity, if the CA is trusted.
- Assurance that the data in the stream is un-tampered-with.
- Protection of the data in the stream from eavesdropping.

TLS supports [many ciphers, hashes, and cryptographic protocols](#).

TLS stream negotiation ([handshaking](#)) is expensive and slow. (Several RTT; TLS 1.3 improves on this.)

# Kerberos

Kerberos [1] is an **authentication** and **key distribution** protocol from MIT's **Project Athena**.

Project Athena also spawned the X Window System and is the source of the popular “MIT license” for open source software.

Kerberos handles:

- **Authenticating** a user to a server
- **Distributing encryption keys** for secure communication

Kerberos **uses only symmetric-key cryptography**.

# Kerberos Architecture

The Kerberos server is a **Key Distribution Center (KDC)**.

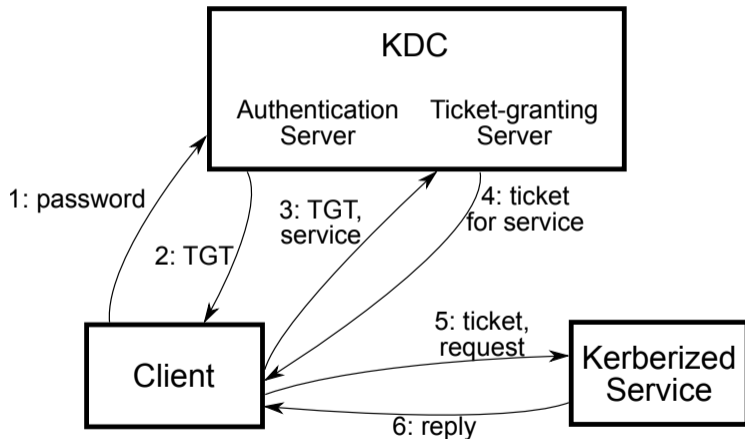
Clients authenticate with an **authentication service** on the KDC using a **password** from which an encryption key is derived.

The KDC issues a **ticket-granting ticket (TGT)** to clients upon authentication.

Services supporting Kerberos accept **tickets** from a **ticket-granting server (TGS)** on the KDC as proof of identity.

Clients use their TGT to request tickets for specific services from the TGS.

# Kerberos Architecture



# Trust and Secret Keys

The only **trusted entity** in Kerberos is the KDC.

Neither clients nor servers must be trusted.

The KDC **shares secret keys** with **every client and server**.

- Client keys are **derived from user passwords**.
- Server keys are **registered with the KDC**.

The shared keys are used only to **authenticate and exchange session keys**.

# Tickets

Kerberos is built around **tickets**, which are **encrypted with the server key for the service they grant**.

Tickets are time-limited, having a **start time** and **end time** between which they can be used.

Tickets contain:

- A client ID  $C$
- A server ID  $S$
- The begin time
- The end time
- A **shared session key** for  $C$  and  $S$

Communication between  $C$  and  $S$  is (optionally) encrypted with

# Kerberos Mechanisms: Client Authentication

To **acquire a TGT**, the user **sends a request** containing a **nonce** to the authentication server.

The authentication server replies with a **TGS session key** and the **nonce**, both **encrypted to the client key (user password)**, as well as a **TGT**.

- If **the client can decrypt the session key**, the client knows the user's password.
- If **the encrypted nonce is correct**, the client knows the server knew the user's password.
- Thus **two-way authentication is achieved**.

The session key and ticket will be used for future communication with the TGS.



# Kerberos Mechanisms: Service Requests

When the client **wants to connect to a Kerberized service**, it contacts the TGS and asks for a **service ticket** for that service.

The request is **encrypted by the session key** and **includes the TGT** and a **nonce**.

The TGS replies with a **service session key** and the nonce, both **encrypted with the TGS session key**, as well as a **service ticket**.

- Similar to the authentication server exchange, mutual decryption **proves identities**.
- The service ticket is **encrypted to the server**.

# Kerberos Mechanisms: Service Authentication

The client **presents a ticket to a server** along with a service request.

The server **decrypts the ticket**, then verifies its timestamps and the client identity.

The server's reply is **encrypted with the session key in the ticket**.

- This ensures that **only the authorized client** can decrypt it.
- A third time, **mutual decryption** proves identities.

# Summary

- Distributed security is very hard, and approaches depend on the application.
- The **principle of least authority** can be used to separate concerns and minimize collateral damage from vulnerabilities.
- Cryptography is important when **infrastructure is untrusted**.
- **TLS** is used to **protect socket communications**.
- **Kerberos** is a **distributed authentication and key exchange protocol** that requires **minimal trust** between entities.

# References I

## Optional Readings

- [1] B. Clifford Neuman and Theodore Ts'o. "Kerberos: An Authentication Service for Computer Networks". In: *IEEE Communications Magazine* 32.9 (Sept. 1994), pp. 33–38. URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.517.2080&rep=rep1&type=pdf>.
- [2] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. URL: <https://www.rfc-editor.org/rfc/rfc8446.html>.