P2P
CS 475, Spring 2018
Concurrent & Distributed Systems
Passwords

• How we authenticate *users* is going to vary based on our environment
• Authenticating you when you log in to your local computer is going to be different than in a distributed system, right?
• Plus: what can we use besides passwords?
  • Biometrics?
  • Tokens?
Authentication Examples

- Parties: Prover (P), Verifier (V), Issuer (I)
- Issuer supplies credentials; Prover tries to log in to Verifier
- How many verifiers?
- How many different provers?
- What sort of networking is available?
- What sort of computer is P using?
- What is the relationship of P, V, and I?
- What are the adversary’s powers?
Passwords: Consumer Website

- Low-value logins
- Can’t afford customer care
- Use email addresses as login names; email new password on request (but why not send out old password?)
- Don’t worry much about compromise
Passwords: Financial Services Site

- High-value login
- Protecting authentication data is crucial
- Customer care is moderately expensive; user convenience is important, for competitive reasons
  - Perhaps use tokens such as SecurID, but some customers don’t like them
  - Today, perhaps use smart phones as second factor
- Do not let customer care see any passwords
- Require strong authentication for password changes; perhaps use physical mail for communication
- Guard against compromised end-systems
Authentication - High level

• The many different forms of authentication have a great deal in common:
  • Secondary authentication
  • Dealing with server compromise
  • Credential loss
  • Susceptibility to guessing attacks
  • Administrative infrastructure
• These pieces interact
• No perfect solution… best seems to be still… passwords
Distributed Denial of Service Attacks (DDoS)

- Model: Attacker has (hundreds of?) thousands of machines at disposal to attack
- Most common form of DoS today
- Exhausts network bandwidth
- Typically rooted in a botnet - some command and control infrastructure setup by an attacker, who then controls all of these machines
Heuristic Defenses

- Overprovision
- Black-hole routing
- Filter anomalies
- Replication
Billion lolz

```xml
<xml version="1.0"?>
<!DOCTYPE lolz [
  <!ENTITY lol "lol">  
  <!ELEMENT lolz (#PCDATA)> 
  <!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">  
  <!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">  
  <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">  
  <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">  
  <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">  
  <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">  
  <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">  
  <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">  
  <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">  
]> 
<lolz>&lol9;</lolz>
```

After parsing: this document contains “lol” repeated literally a billion times... ~3GB of RAM
Announcements

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

• Today - P2P:
  • Masterless systems
  • Discussion of course structure
  • Course evaluation
Why P2P?

• Spreads network/cache costs across users instead of provider
• No server might mean:
  • Easier to deploy
  • Less chance of overload
  • Single failure won’t take down the system
  • Harder to attack
Why not P2P?

- Hard to find data items over millions of users
- Computers might not be as reliable as a managed server
- Less secure (?)
P2P

• Goal: IF there must be a master, all that it knows is the address of a few clients using the system
• Otherwise, everyone talks to each other, figures it out
• Replicate files, store them on clients, let clients find files from each other
• Challenges:
  • Where to find data?
  • What to do when clients come and go?
P2P

• Break it down into four operations:
  • **Join** the network and begin participating
  • **Publish** a file to the network, letting others know you have it
  • **Search** for a file that you want
  • **Fetch** a file once it is found
Napster

- Single master (centralized DB) stores metadata and client status
  - **Join**: Client contacts master
  - **Publish**: Client reports list of files to master
  - **Search**: Query the server, find who has the file you want
  - **Fetch**: Get directly from that peer client
Napster

Hi, I signed on, I have files f1, f2, f3

Napster Master

Who has f1?

client 1 does

Can I have f1?

Here is f1

client 2

client 1

Doesn’t everything just look like GFS, even things that predated it? :)

J. Bell

GMU CS 475 Spring 2018
Napster

• The good:
  • Simple
  • Finding a file is really fast, regardless of how many clients there are - master has it all

• The bad:
  • Server becomes a single point of failure
  • Server does a lot of processing
  • Server having all of metadata implies significant legal liabilities
Gnutella 1.0

- **Join**: Client contacts a few other clients to find “neighbors”
- Requires some initial mechanism to bootstrap
- **Publish**: N/A
- **Search**: Client asks neighbors for file, who ask their neighbors for file, who asks their neighbors out to some depth
- **Fetch**: Clients directly communicate with each other
Can I have f1?

where is f1?

client 1

where is f1?

c2 has f1

c2

client 2

client 3

client 4

client 5

Gnutella 1.0
Gnutella

• This is called "flooding"

• Cool:
  • Fully decentralized
  • Cost of search is distributed - no single node has to search through all of the data

• Bad:
  • Search requires contacting many nodes!
  • Who can know when your search is done?
  • What if nodes leave while you are searching?
BitTorrent

- "Swarming"
- **Join**: Contact master "tracker," get list of peers
- **Publish**: Run a tracker server
- **Search**: Out-of-band (e.g. google)
- **Fetch**: Download chunks of files from peers
BitTorrent vs Napster

- Focus on **less** files, each of which is **larger**
- Files are broken into chunks -> can get different pieces of a file from different clients
- Anti-freeloading mechanisms - if you don't share, you don't get to play!
  - Since a big file is many chunks, once you get a chunk you can immediately share it with others
- Trackers are still single-points of failure, but assumption is 1 tracker per file
BitTorrent
BitTorrent

- "Tit-for-tat" sharing strategy
- A is getting data from B, C, D
  - A will let the fastest of those get data from A
  - A will be optimistic though, and let nodes who haven't shared anything yet have some data so that they can have a chance to share
DHT (Distributed Hash Table)

• Goal:
  • Guarantee that a file is always found within some bounded and reasonable number of steps

• Abstraction:
  • Create a lookup table, mapping from file to node that has that file (much like Napster)
  • BUT distribute this lookup table amongst the nodes participating (no single master)
DHT

- **Join**: Contact some other node to bootstrap: integrate yourself into the DHT, get a node ID and list of participating nodes
- **Publish**: Tell "mostly the correct" node that you have a file
- **Search**: Query for a file, asking first a "mostly correct" node
- **Fetch**: Contact node that has it directly
- How do we know where to route? Consistent hashing!
Reminder: Consistent Hashing

Example: hash key size is 16
Each $\odot$ is a value of hash $\% 16$
Each $\bullet$ is a bucket
Example: bucket with key 9?
DHT

• Pros:
  • Guarantees that if the data is in the network, you'll find it in $\log(n)$ time (compare to Gnutella - pseudo-random search)
  • Good for caching, infrequently written data

• Cons:
  • Can really only match on exact keys
  • The node join/leave story is really bad - if we are distributed across the internet, a node leaving/joining might involve moving hundreds of GBs around
DHT Applications

• Use a DHT instead of a tracker for BitTorrent!
• Bootstrap: find a DHT peer
• Application: As you acquire files or look for files, add those facts into the DHT
Discussion of course structure

• More/less concurrency discussion at start?
• More/less programming?
• More/less detail on some topics?