

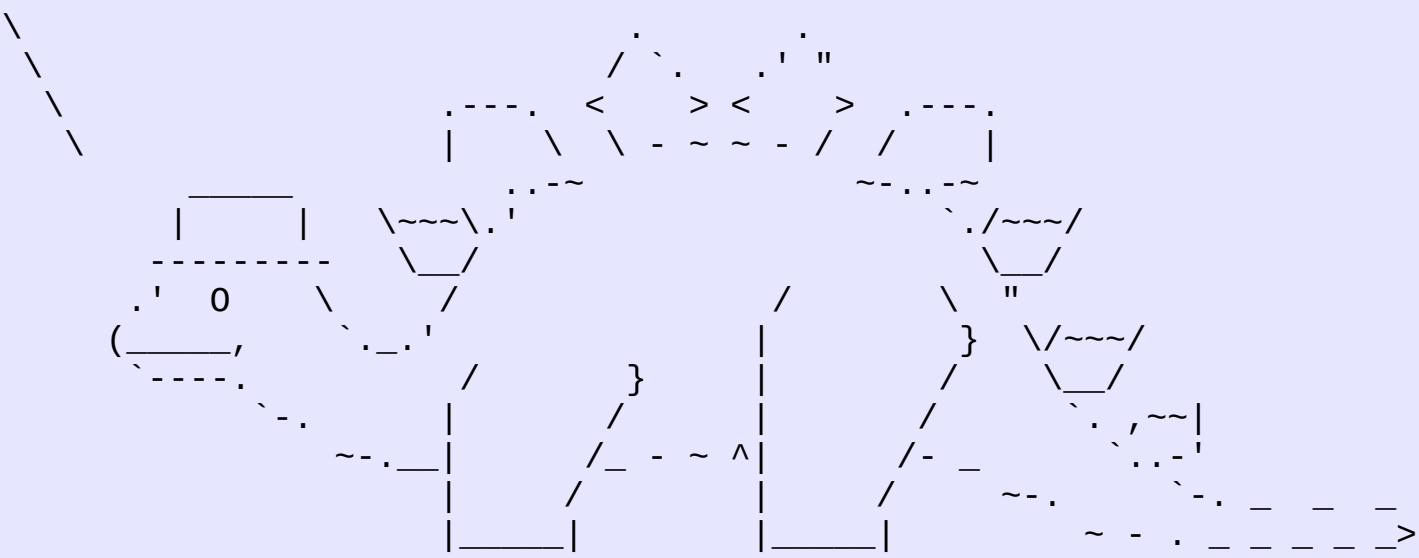
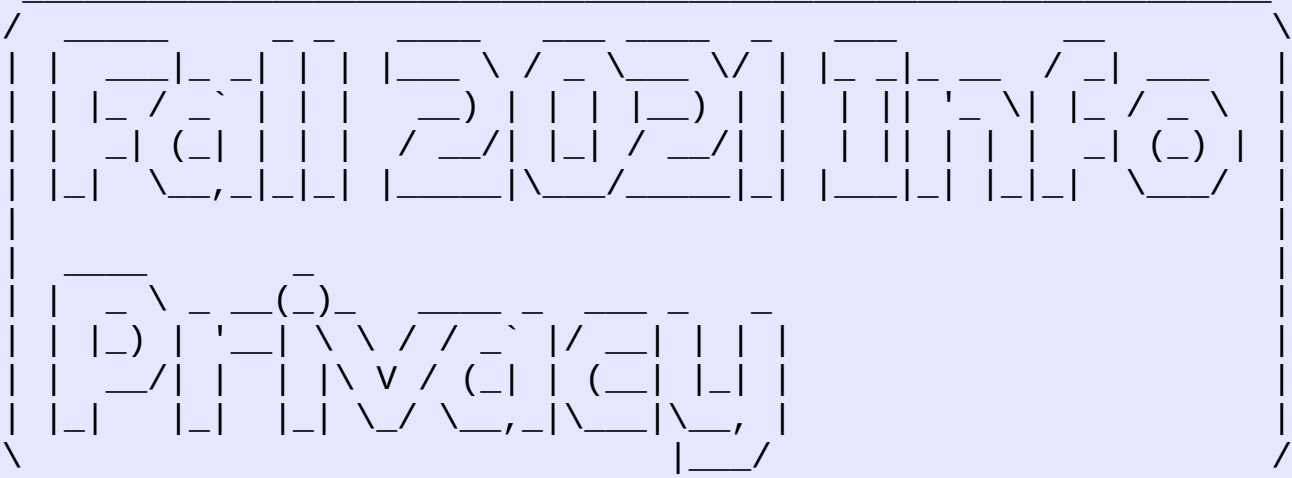


Group Privacy Technology III

Information Privacy with Applications
David Sidi (dsidi@email.arizona.edu)

Administration

- Thoughts on the integrated session?
- Internet is a bad neighborhood: reviewing our logs
- Server Assignment II: Cracking



WELCOME TO THE FALL 2021 SANDBOX! ENJOY YOURSELF...

Last login: Tue Aug 24 16:59:00 2021

fa21-course-1vcpu-1gb-sfo1-01:dsidi \$

Today

- Finishing trusting trust
- Privacy as confidentiality: threat modeling

Philip Zimmerman on trust

- “When examining a cryptographic software package, the question always remains, why should you trust this product? Even if you examined the source code yourself, not everyone has the cryptographic experience to judge the security. Even if you are an experienced cryptographer, subtle weaknesses in the algorithms could still elude you.”
- What is the *further* problem that Ken Thompson pointed out with source verification?

The “trusting trust” attack



The actual bug I planted in the compiler would match code in the UNIX "login" command. The replacement code would miscompile the login command so that it would accept either the intended encrypted password or a particular known password. Thus if this code were installed in binary and the binary were used to compile the login command, I could log into that system as any user. ...

Two ingredients for “trusting trust”

- Self-reproducing programs (*quines*), and compiling other compilers

Compiling compilers and hidden inheritance

```
...  
c = next( );  
if(c != '\\')  
    return(c);  
c = next( );  
if(c == '\\')  
    return('\\');  
if(c == 'n')  
    return('\n');  
if(c == 'v')  
    return('\v');  
...
```

FIGURE 2.1.

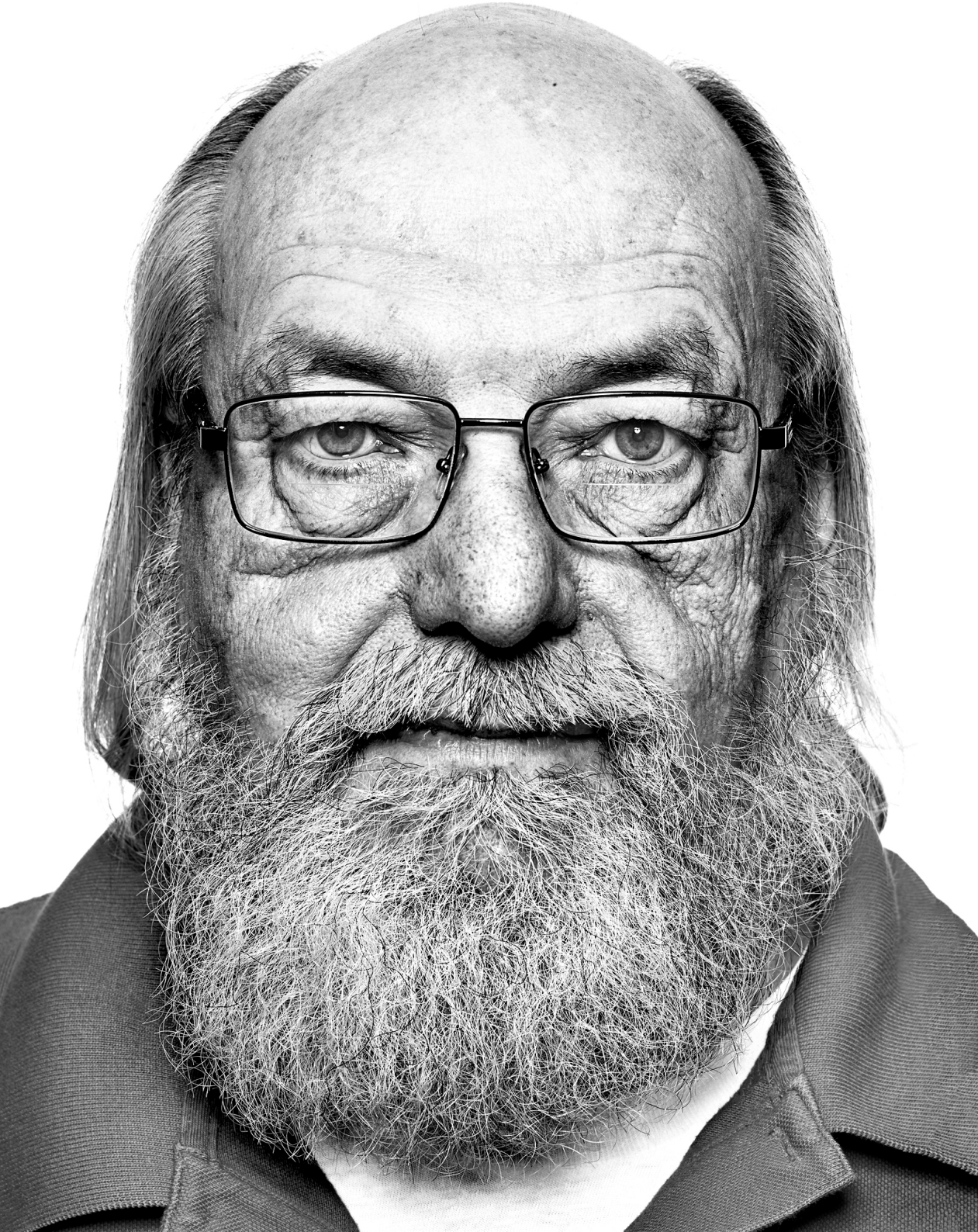
```
...  
c = next( );  
if(c != '\\')  
    return(c);  
c = next( );  
if(c == '\\')  
    return('\\');  
if(c == 'n')  
    return('\ n');  
if(c == 'v')  
    return(11);  
...
```

FIGURE 2.3.

“that’s worrying... let’s audit the source code for both the login command and for the compiler we use, and recompile everything cleanly”

“that’s worrying... let’s audit the source code for both the login command and for the compiler we use, and recompile everything cleanly”

nope



The moral is obvious. You can't trust code that you did not totally create yourself. (Especially code from companies that employ people like me). No amount of source-level verification or scrutiny will protect you from using untrusted code.

Ken Thompson, ACM Turing Award Speech, "Reflections on Trusting Trust"

Q: What is 'trust' here?

Can it be helped?

Don't think---Look!

- A binary seed: bootstrap binaries to compile the first compiler
- the ultimate seed: an easily-inspectable binary, readable as source
- Is this folly? Nope:
 - <http://www.joyofsource.com/guix-reduces-bootstrap-seed-by-50.html>
 - **<http://www.joyofsource.com/we-did-it.html>**

Threat modeling

Goals of threat modeling

- To build a description of threats using an organized process
- The process should help to prevent mistakes and oversights

Goals of threat modeling

- To build a description of threats using an organized process
- The process should help to prevent mistakes and oversights
- Broadly
 - What are you building?
 - What can go wrong?
 - What should you do when things go wrong?
 - Did you do a decent job in your analysis?

Strategies for threat modeling

- Assets
- Attackers
- Software



- experts
- less technical input to your project
- prioritization

Strategies for threat modeling

- **Assets**
- **Attackers**
- **Software**
- usually: what attackers want, that you want to protect
- Stepping stones really requires understanding attackers and software



Figure 2-2: The overlapping definitions of assets

Strategies for threat modeling

- **Assets**
- **Attackers**
- **Software**
- **what kinds of attackers do we face?**

- Competitor
- Data miner
- Radical activist
- Cyber vandal
- Sensationalist
- Civil activist
- Terrorist
- Anarchist
- Irrational individual
- Government cyber warrior
- Organized criminal
- Corrupt government official
- Legal adversary
- Internal spy
- Government spy
- Thief
- Vendor
- Reckless employee
- Untrained employee
- Information partner
- Disgruntled employee

Strategies for threat modeling

- **Assets**
- **Attackers**
- **Software**
- **what kinds of attackers do we face?**
- **A space of attacker features: personas**
 1. Identify behavioral variables.
 2. Map interview subjects to behavioral variables.
 3. Identify significant behavior patterns.
 4. Synthesize characteristics and relevant goals.
 5. Check for completeness and redundancy.
 6. Expand descriptions of attributes and behaviors.
 7. Designate persona types.

Strategies for threat modeling

- **Assets**
- **Attackers**
- **Software**
- what kinds of attackers do we face?
- A space of attacker features: personas
 - motivation and skill
- what will the attacker do?

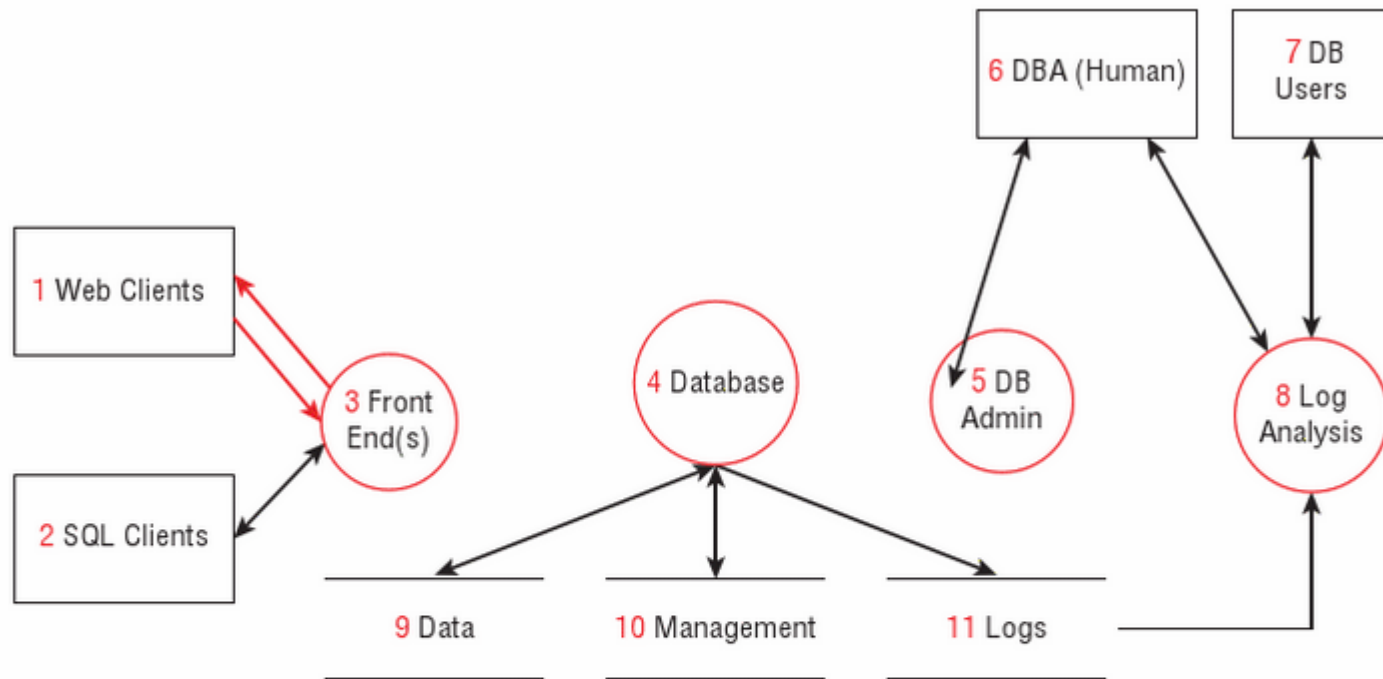
Strategies for threat modeling

- **Assets**
- **Attackers**
- **Software**
- modeling software in a way that helps to unearth threats
- security retrofitted at the level of software design (broadly)

Focusing on Software

- Point is to diagram how the system works
- Done in a group with a wide variety of people (you want them to turn out to be across trust boundaries): if there is disagreement, this usually means a security problem is nearby

Data flow diagrams



Key:

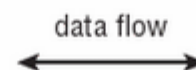
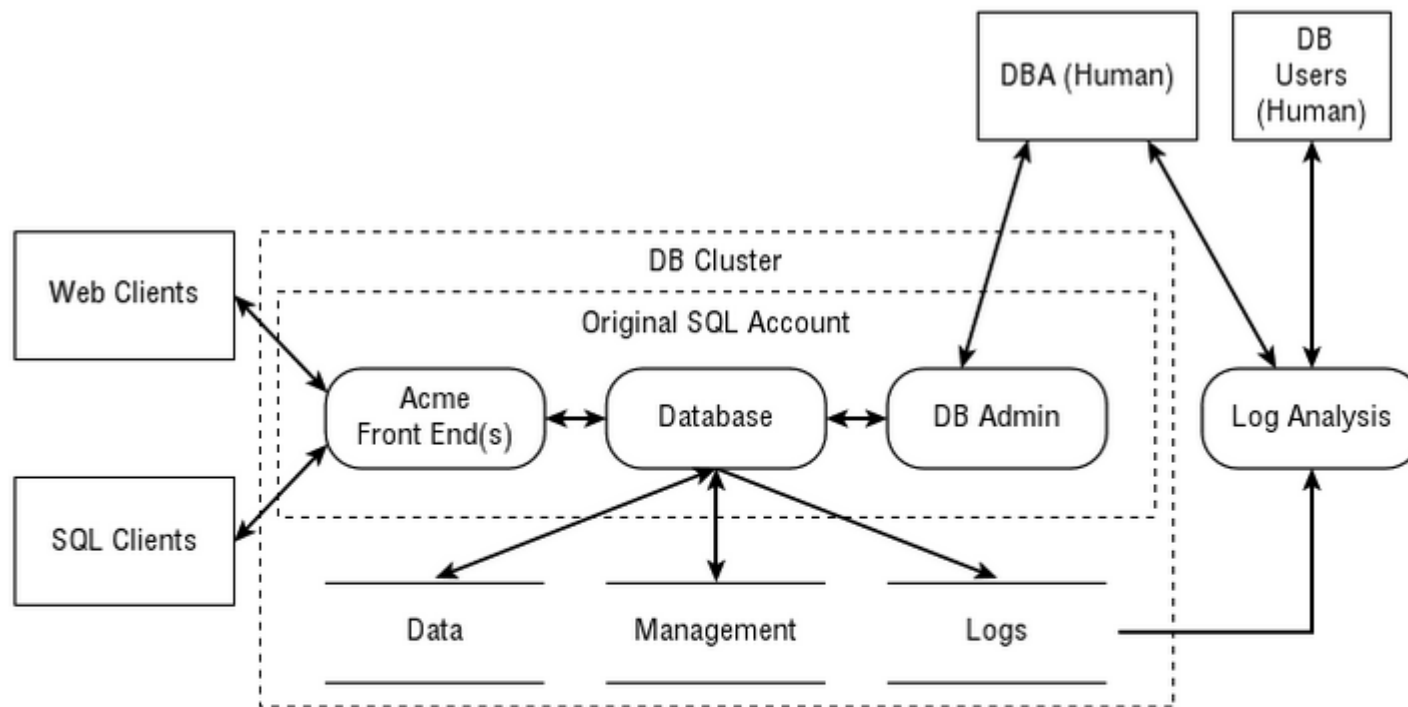


Figure 2-3: A classic DFD model

Data flow diagrams



Key:

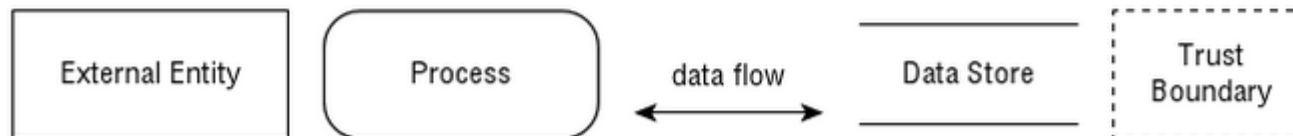


Figure 2-4: A modern DFD model (previously shown as Figure 2-1)

Swim lane diagrams

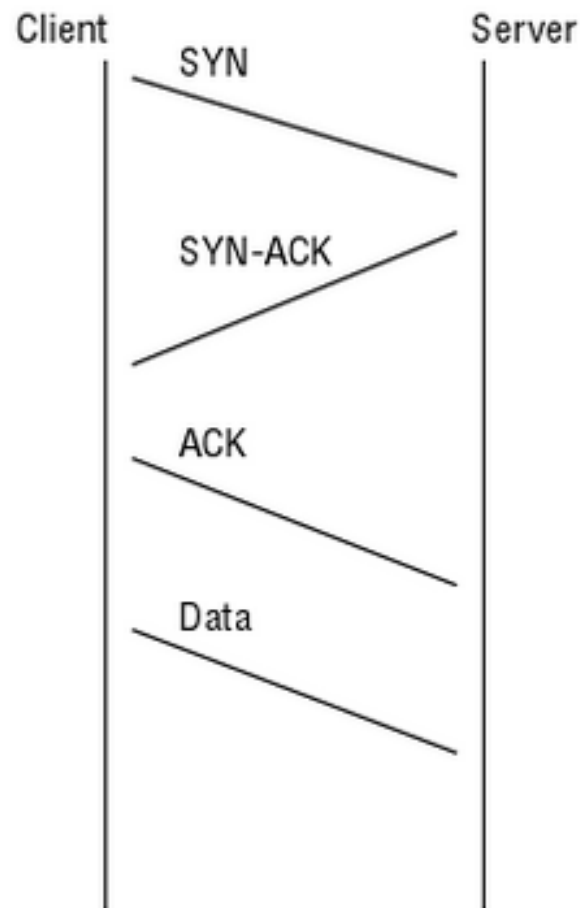


Figure 2-6: Swim lane diagram (showing the start of a TCP connection)

Processing and Managing Threats

