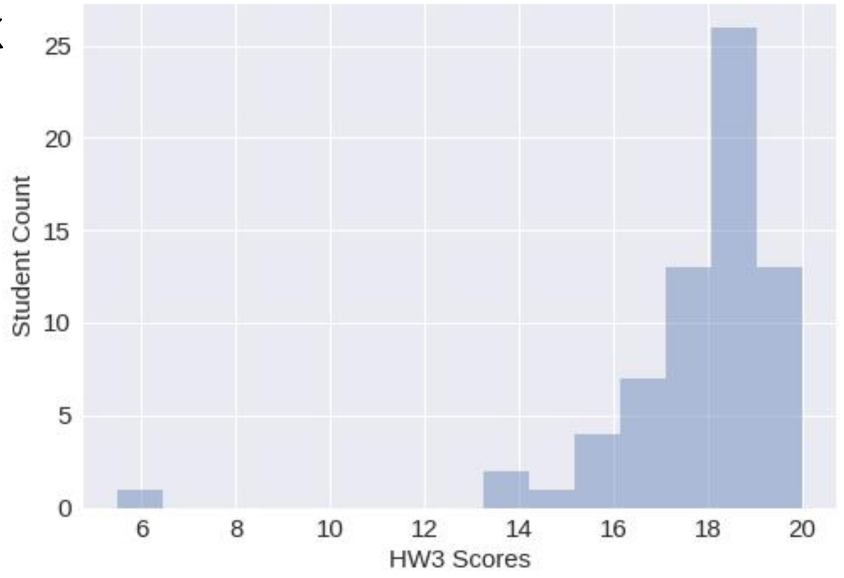
Applications of Factor Graphs in Security Modeling
Mini Project 3 – In Class Lab

Ravi K. Iyer,
Yuming Wu with Phuong Cao

Announcements

- HW3 grades have been released
  - Regrade requests to be submitted within 1 week
- MP3 released
  - Checkpoint 1 due on April 10



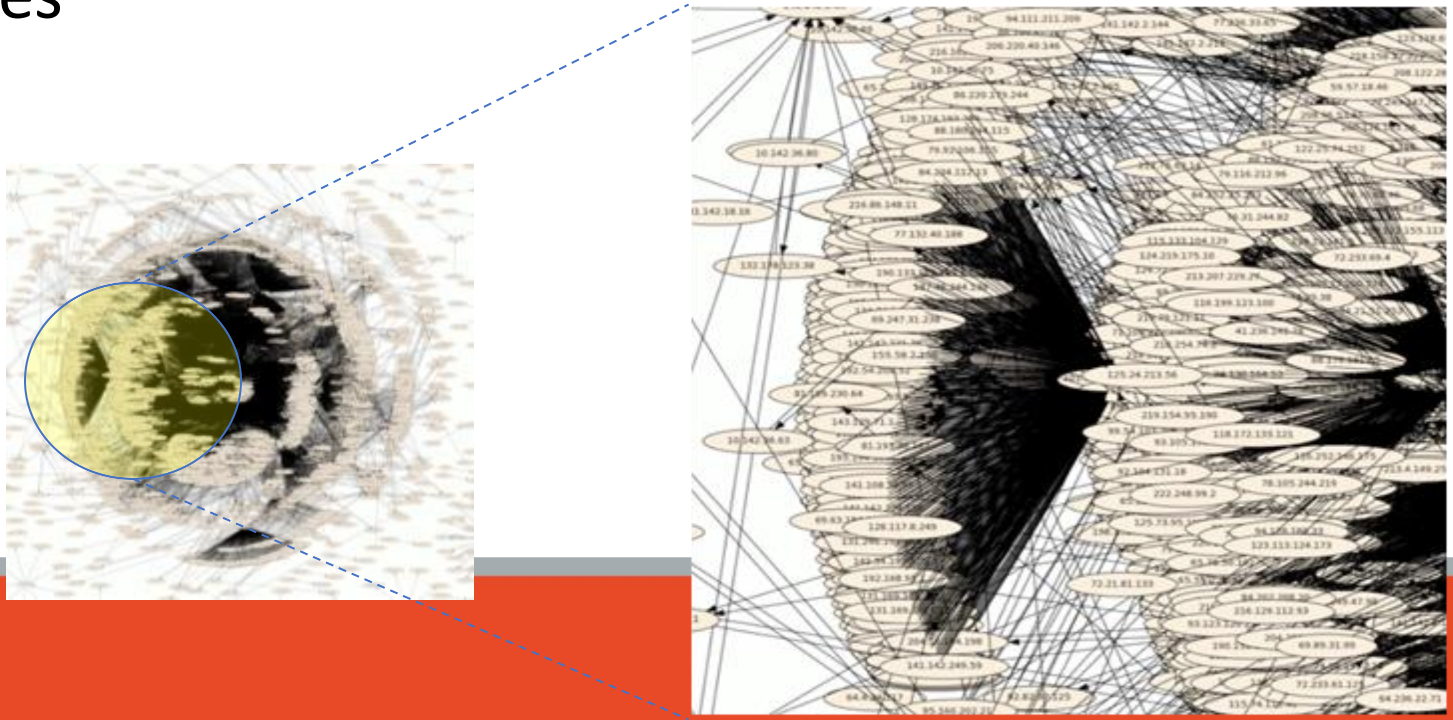
Mean	Std	Median	Max	Min
17.95	2.11	18.50	20.00	5.50

Outline

- Overview of security incidents at NCSA
- Case studies
  - A credential-stealing attack at NCSA
  - An emulation of a real-world data breach at Equifax
- Characterization of attacks
- Probabilistic modeling of attacks using Naïve Bayes, Bayesian Networks, and Factor Graphs

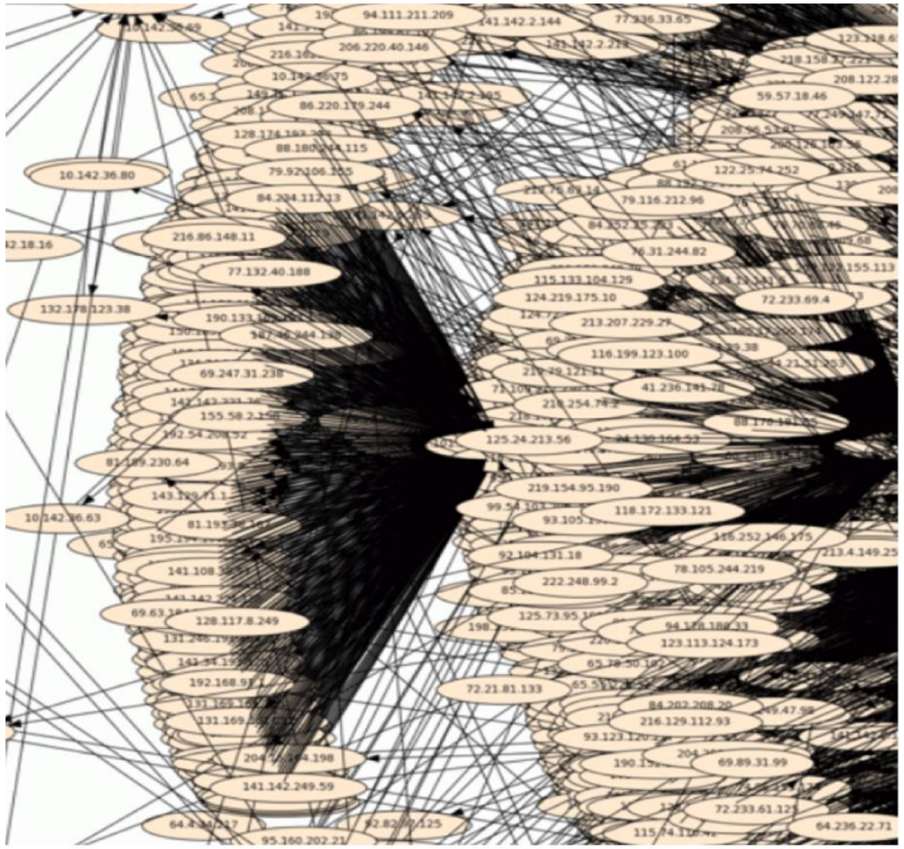
Measurements from NCSA@Illinois: Five minute Snap Shot

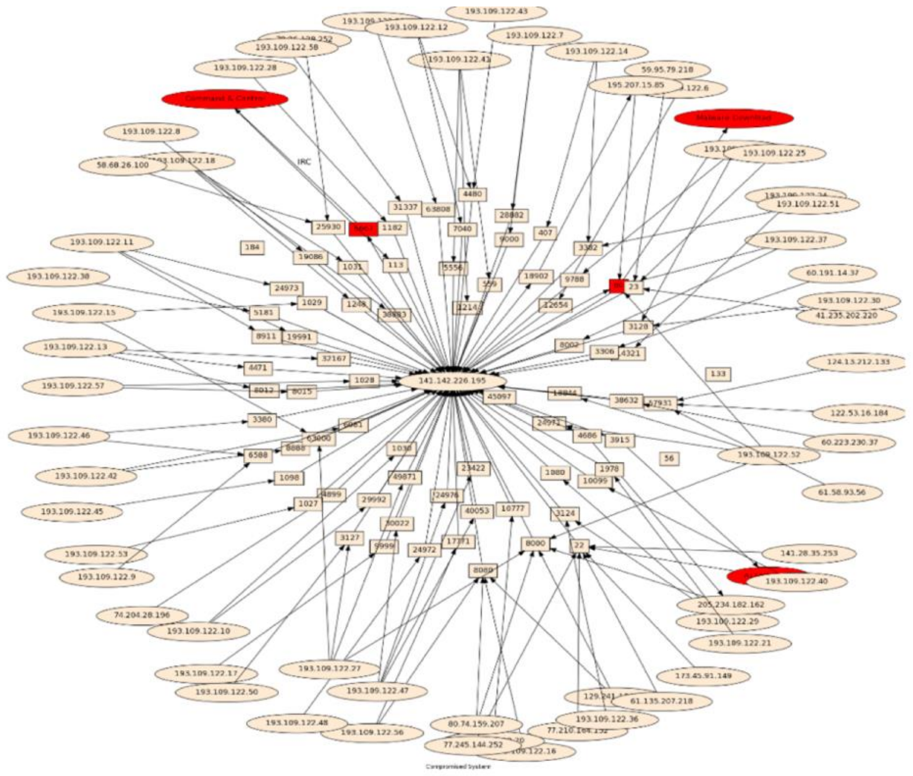
- Goals:
  - Provide a system-level characterization of incidents and evaluate the intricacies of real-time diagnosis
  - Design protection strategies to reduce missed incidents and false positives
  - Experimentally Demonstrate new techniques in a sandbox
- Challenges



Five-Minute Snapshot of In-and-Out Traffic at NCSA

Five-Minute Snapshot of In-and-Out Traffic within NCSA@Illinois


(a)



(b)

An attack in the wild: the Equifax data breach (2017)

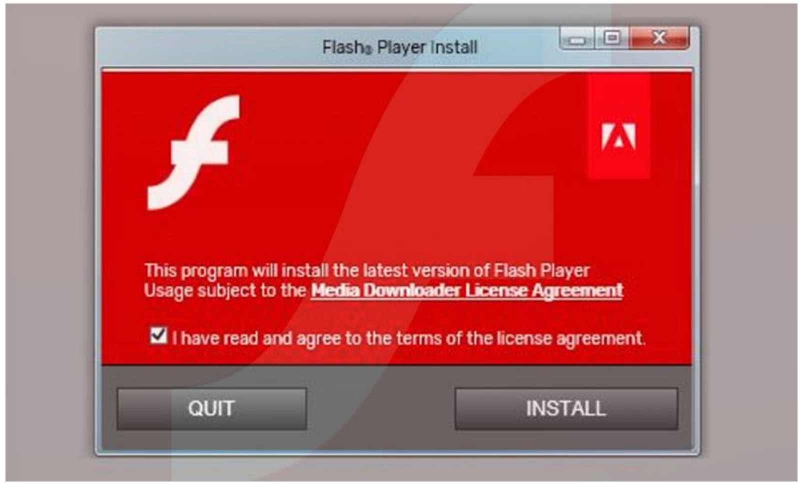
Background: Equifax portal has a remote code execution vulnerability (CVE-2017-5638) in its Apache Struts server since March 2017, but the vulnerability has only been patched in July 2017

What happened?

- Attackers executed malicious commands to gain an initial access to the vulnerable server
- Attackers accessed database and extract 143M sensitive records
- Attackers injected malware on Equifax website after the breach

Very few details of the attacks are released to the public

([watch Equifax CEO describes the attack for the Senate Banking Committee](#))



This program will install the latest version of Flash Player
Usage subject to the Media Downloader License Agreement
☑ I have read and agree to the terms of the license agreement.

A fake Flashplayer that links to a malware displayed on Equifax website after the breach

CVE-2017-5638

- The Jakarta Multipart parser in Apache Struts 2 2.3.x before 2.3.32 and 2.5.x before 2.5.10.1 has incorrect exception handling and error-message generation during file-upload attempts, which allows remote attackers to execute arbitrary commands via a crafted Content-Type, Content-Disposition, or Content-Length HTTP header, as exploited in the wild in March 2017 with a Content-Type header containing a #cmd= string.
- *Ref: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5638*

Reproducing the Equifax Attack Scenario in our testbed

1. Exploit CVE-2017-5638 (container-based environment for MP3 you only need to analyze the traces)
   - Attach an arbitrary length command to the specifically crafted HTTP request.
   - Send the malformed HTTP request to the vulnerable Apache Struts server.
   - Example: if the command is 'whoami', and the server is vulnerable, the server responds with the output of the command, e.g., 'root'. That means the attacker has the superuser (root) privilege.
   - **Attacker has the root privilege**
2. HTTP sensitive request
   - Attacker downloads a rootkit (backdoor) via HTTP sensitive request
3. Insert a new system service
   - Attacker compiles and install the rootkit as a system service (i.e., a kernel module)
4. Read the secret key
   - Attacker opens the private RSA key in the ~/.ssh/ folder
5. Extract the secret key via DNS tunnel (because the backdoor only receives basic commands)
   - Attacker encodes the secret key as legitimate DNS requests to extract a significant amount of data
   - (uses the secret key to move laterally in the system)

Attacker can use the secret key to move laterally to other servers in the internal network.

Reproducing the Equifax attack

Goal. To understand how such attack will progress when legitimate user activities are underway.

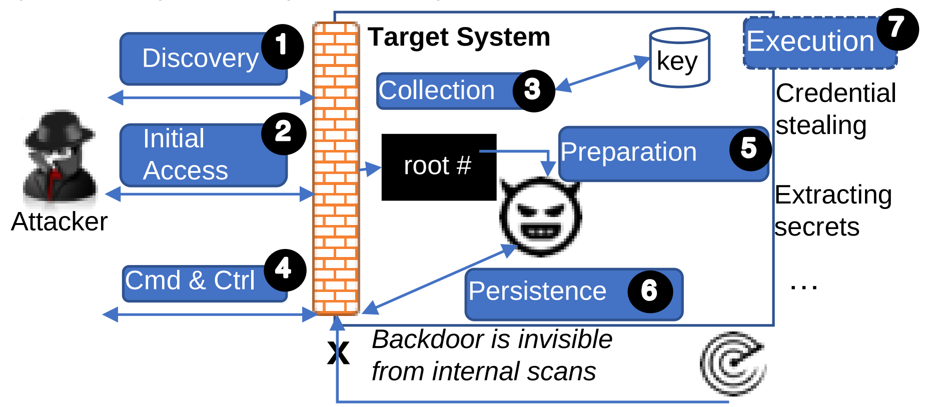
Attack traces were collected from security monitors and fed into an intrusion detection system (a machine learning model) to detect the attack.

At the end of the project, students would have built a factor graph-based detection system to detect each attack stages in the provided traces.

How we obtained the attack traces

- Model 7 attack stages, e.g., discovery, access, privilege escalation, …, execution, using the MITRE ATTCK framework.
- Define attacker action at each stage using publicly available exploit
  - Scan for vulnerability, login, get privilege escalation code, collect ssh keys, build a DNS tunnel, and extract logs.
- Execute the exploits targeting vulnerable servers
- Collect traces of the exploits, e.g., network flows or system logs

a) Attack stages of the "port-knocking" attack



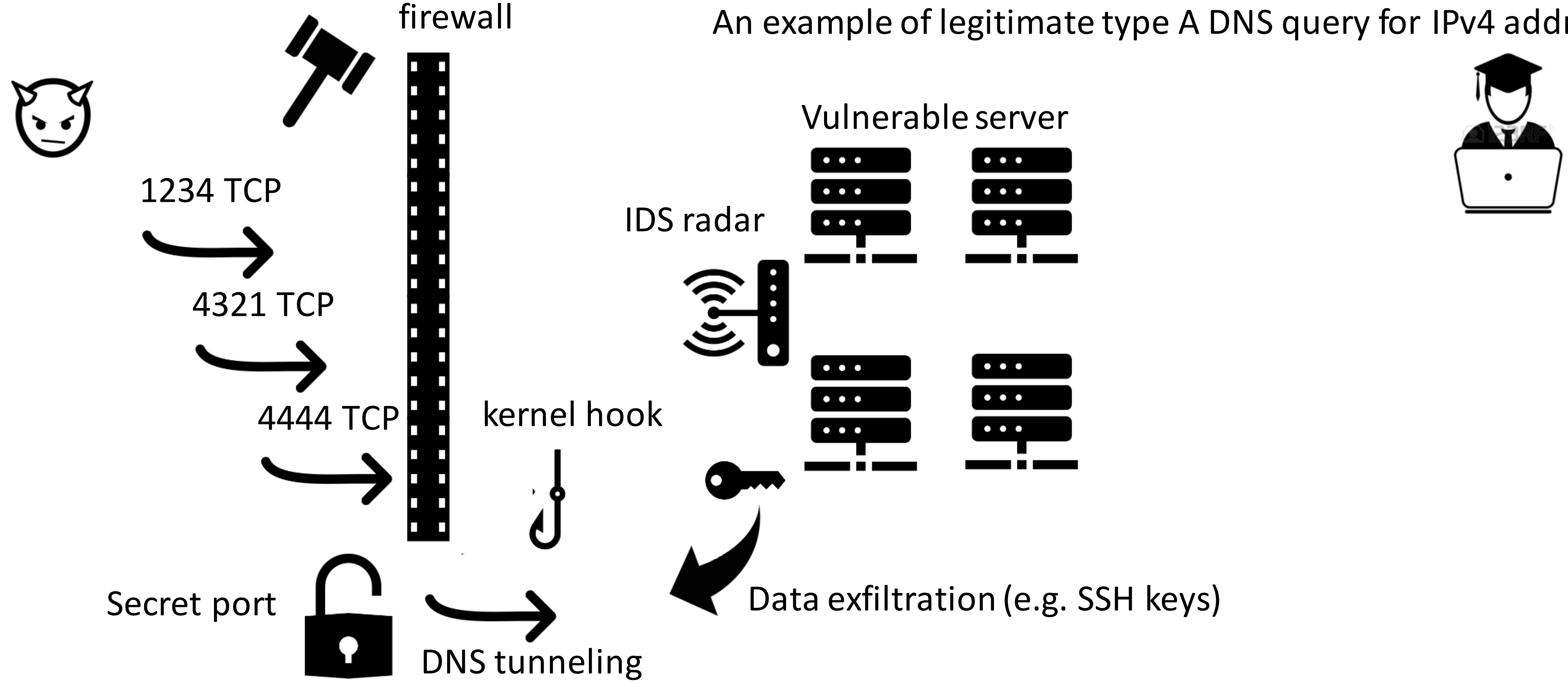
b) Security events and raw logs corresponding to the attack stages

ID	Name	Raw logs from host and network security monitors
1	VULN_SCAN	`11:57:49 hotcluster bro::scan::struts (#container=#context[opensymp`
2	LOGIN	`11:57:49 hotcluster bro::http::struts (#cmd =/bin/bash 200 OK`
3	SENSITIVE_URI	`11:59:02 hotcluster bro::http::get http://srvx[redacted]/knockd.c 200 OK`
4	COMPILE	`00:01:05 osquery::compile in volatile directory /dev/shm`
5	NEW_KMOD	`00:02:22 hotcluster osquery::new kernel module (knockd.ko) loaded size 16384`
6	SSH_PRIV_KEY_ACCESS	`00:03:14 hotcluster osquery::inotify fopen $HOME/.ssh/id_rsa`
7	DNS_TUNNEL	`00:03:59 hotcluster bro::dns::oversized answer TXT record length: 1054 bytes`
8	CAPTURE_LOSS	`00:06:22 hotcluster bro::capture loss script detected a loss rate above 86.92%`

Port Knocking to control rtkt

```
DNS      74  Standard query 0xe465 A www.google.com
```

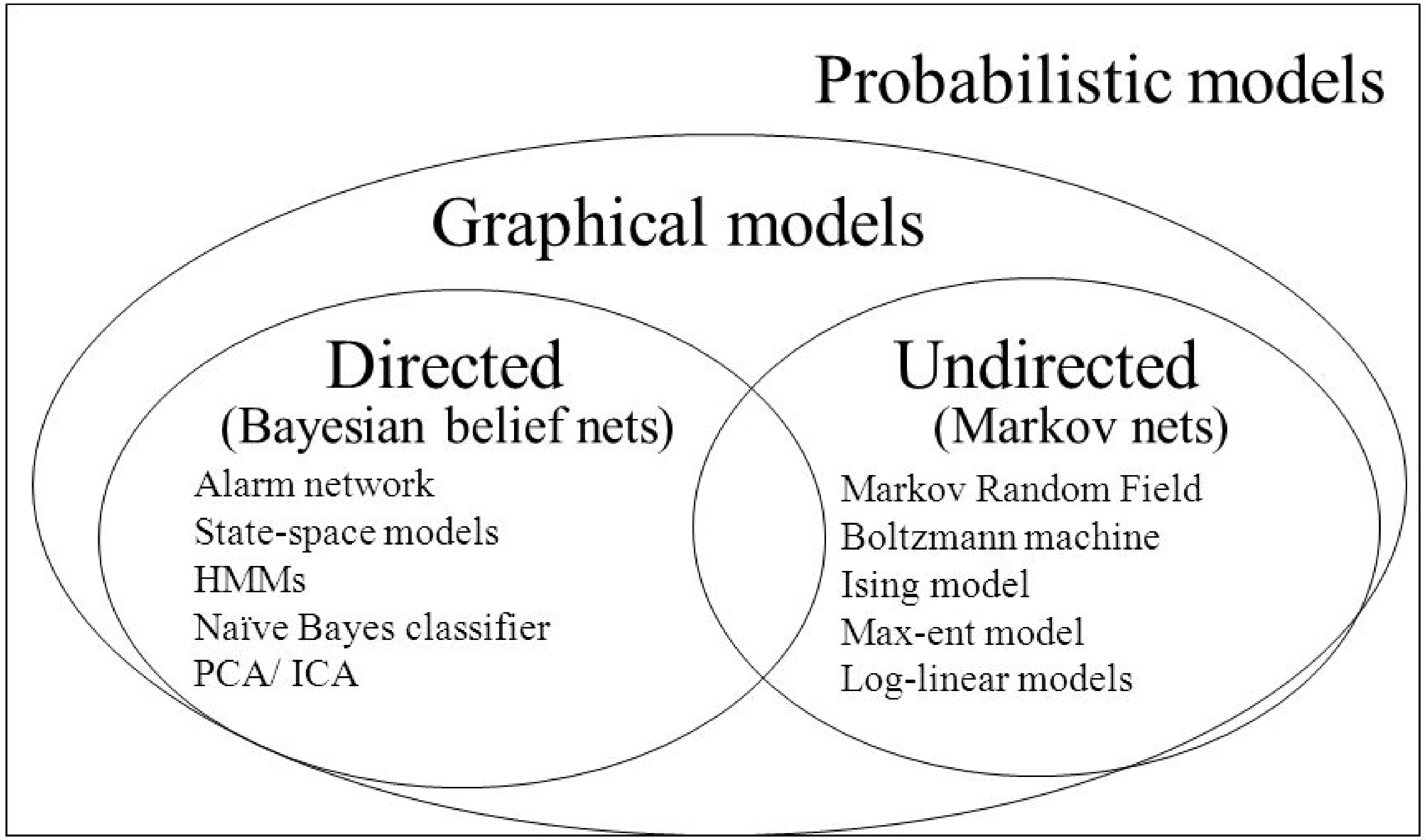
An example of legitimate type A DNS query for IPv4 address



```
DNS      158  Standard query 0x1973 TXT 02bbs82\3122hb\276\356\360\326gh\313\311Fb\336\3365\335j\332Udb\302\340dF4\357\3
```

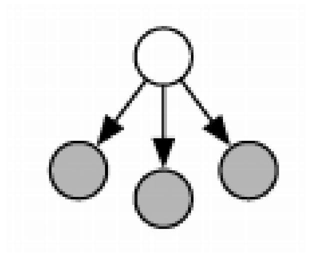
An example of using TXT records for DNS tunneling

Taxonomy of graphical models (cont.)

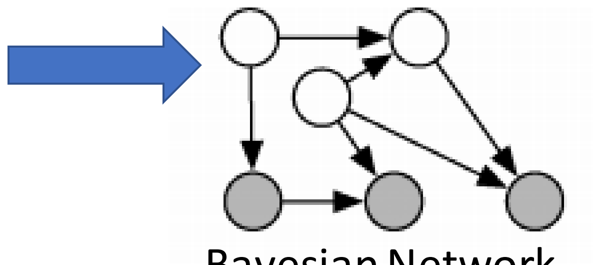


Machine Learning, A Probabilistic Perspective, Kevin Murphy, MIT Press

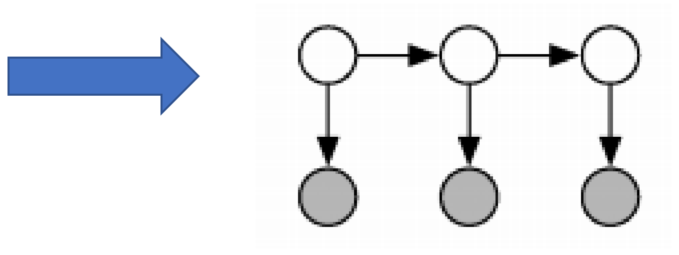
Taxonomy of graphical models



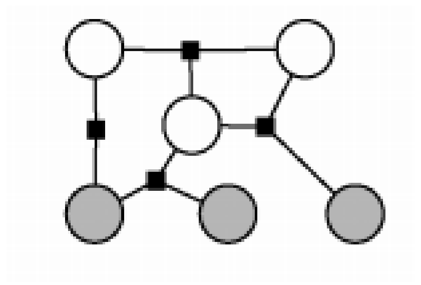
Naïve Bayes



Bayesian Network



Hidden Markov Model



Factor Graph

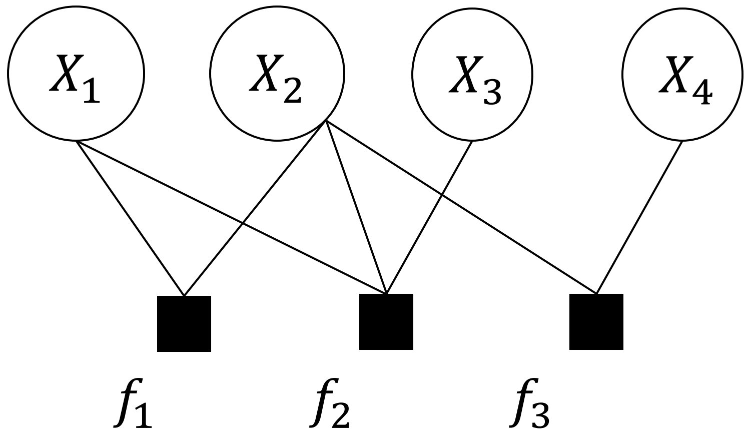
Conditional probabilities and statistical dependencies can be represented by a general type of graph: Factor Graph

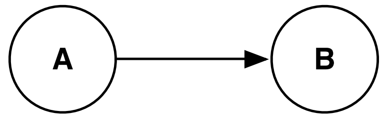
Definition of a Factor Graph

A factor graph is a **bipartite, undirected graph** of **random variables and factor functions. [Frey et. al. 01].**

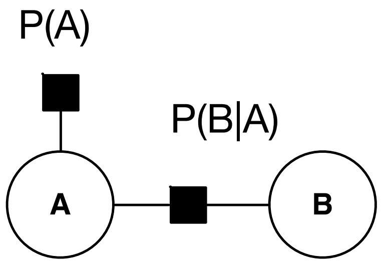
G (graph) = (X,f,E); E denotes the edges

FG can represent both ***causal and non-causal relations****.*

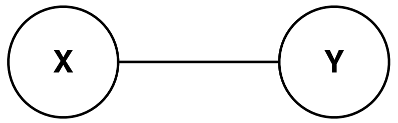




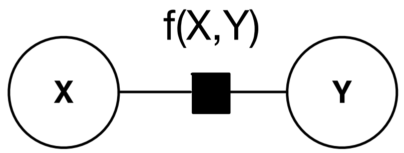
Bayesian Network (BN)



Factor Graph equivalent of BN



Undirected Graph



Factor Graph equivalent of UG

Applications of Factor Graphs in Security Domain

Problem statement. Given a set of security events, infer whether an attack is in progress?

Modeling Approach.

Each security event is a known variable **e**, each takes value from a discrete set of events **E**.

An attack happens in a chain of exploits, thus we have a sequence of events in time dimension.

Each event is associated with a corresponding attack state **s,** which is unknown. The simplest approach is to classify **s** as a binary {0,1}. However, when we can infer **s** it is often too late (the attacker is already in the system)

Thus, we want to discretize **s** to smaller attack stages and provide update on such stages as soon as an event is observed.

Applications of Factor Graphs in Security Domain (cont.)

Problem statement. Given a set of security events, infer whether an attack is in progress?

Formally, the problem becomes

1. Define a joint probability distribution function (joint pdf)

$P(e_1,e_2,..,e_n,s_1,s_2,\dots,s_n)$

2. Derive a conditional probability

$P(e_1,e_2,..,e_n|s_1,s_2,\dots,s_n)$

However, the search space is exponentially large (by the order of the number of observed stages and events) and the joint pdf is sophisticated.

We want to break the joint pdf into smaller components that are easier to compute, i.e., factorize the joint pdf.

Modeling the credential stealing attack using Factor Graphs - Data

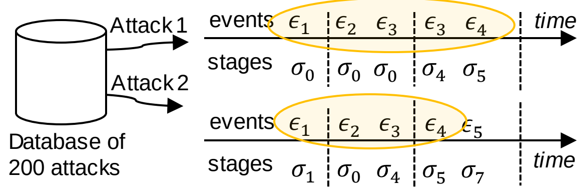
State space of variables

Attack stage: $X = \{\sigma_0, \sigma_1, \ldots, \sigma_7\}$

(Observed) Events: $E = \{\epsilon_1, \ldots, \epsilon_5\}$

OFFLINE ANNOTATION ON PAST ATTACKS

a) Annotated events and attack stages in a pair of attacks



Attack 1
Attack 2
Database of 200 attacks
events ϵ_1 ϵ_2 ϵ_3 ϵ_3 ϵ_4
stages σ_0 σ_0 σ_0 σ_4 σ_5
time
events ϵ_1 ϵ_2 ϵ_3 ϵ_4 ϵ_5
stages σ_1 σ_0 σ_4 σ_5 σ_7
time

b) Event-stage annotation table for the attack pair (Attack 1 and Attack 2)

Event	Attack stage
$\{\epsilon_1\}$	$\{\sigma_0 \vert \sigma_1\}$
$\{\epsilon_2\}$	$\{\sigma_0\}$
$\{\epsilon_3\}$	$\{\sigma_4\}$
$\{\epsilon_4\}$	$\{\sigma_5\}$
$\{\epsilon_5\}$	$\{\sigma_7\}$

- Multi-stage credential stealing attack where the attack stage is not observed; however events which are related to the attack stage are observed
- Goal is to detect and pre-empt the attack
- **Model assumptions**
  - There are multivariate relationships among the events
  - There is no restriction on order of the relationships (can be non-causal or correlation based)

- Markov Model and Bayesian Networks cannot be used in this scenarios

- Factor graphs can be used for modeling highly complex attacks, where the causal relations among the events are not immediately clear.

ϵ_1	vulnerability scan	σ_0	benign
ϵ_2	login	σ_1	discovery
ϵ_3	sensitive_uri	σ_4	privilege escalation
ϵ_4	new_library	σ_5	persistence

Modeling the credential stealing attack using Factor Graphs

OFFLINE LEARNING OF FACTOR FUNCTIONS

Example patterns, stages, probabilities, and significance learned from the attack pair

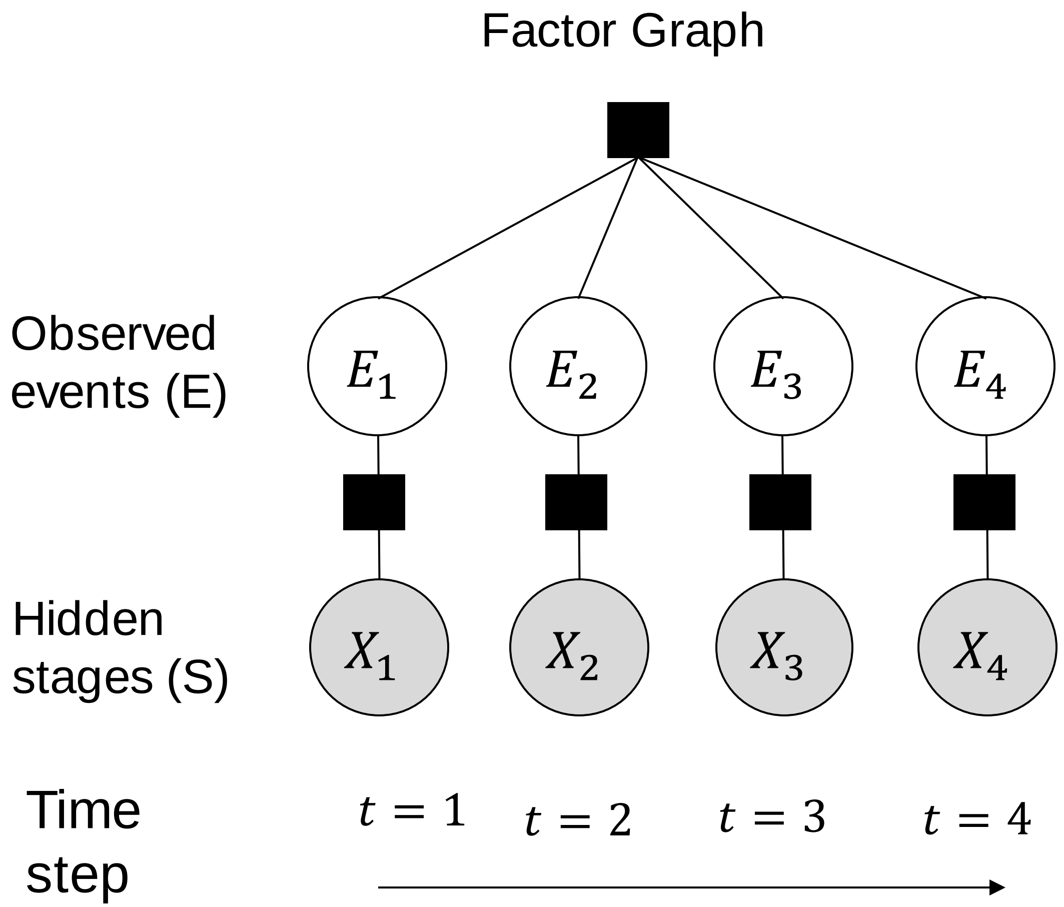
Pattern	Attack stages	Probability in past attacks	Significance (p-value)
$[\epsilon_1, \epsilon_3, \epsilon_4]$	$[\sigma_1, \sigma_4, \sigma_5]$	q_a	p_a
$[\epsilon_1]$	$[\sigma_0 \vert \sigma_1]$	q_b	p_b

…

$$f(E) = \exp\{q_E(1 - p_E)\}$$

A factor function defined on the learned pattern, stages, and its significance

DETECTION OF UNSEEN ATTACKS



Problem statement of MP3

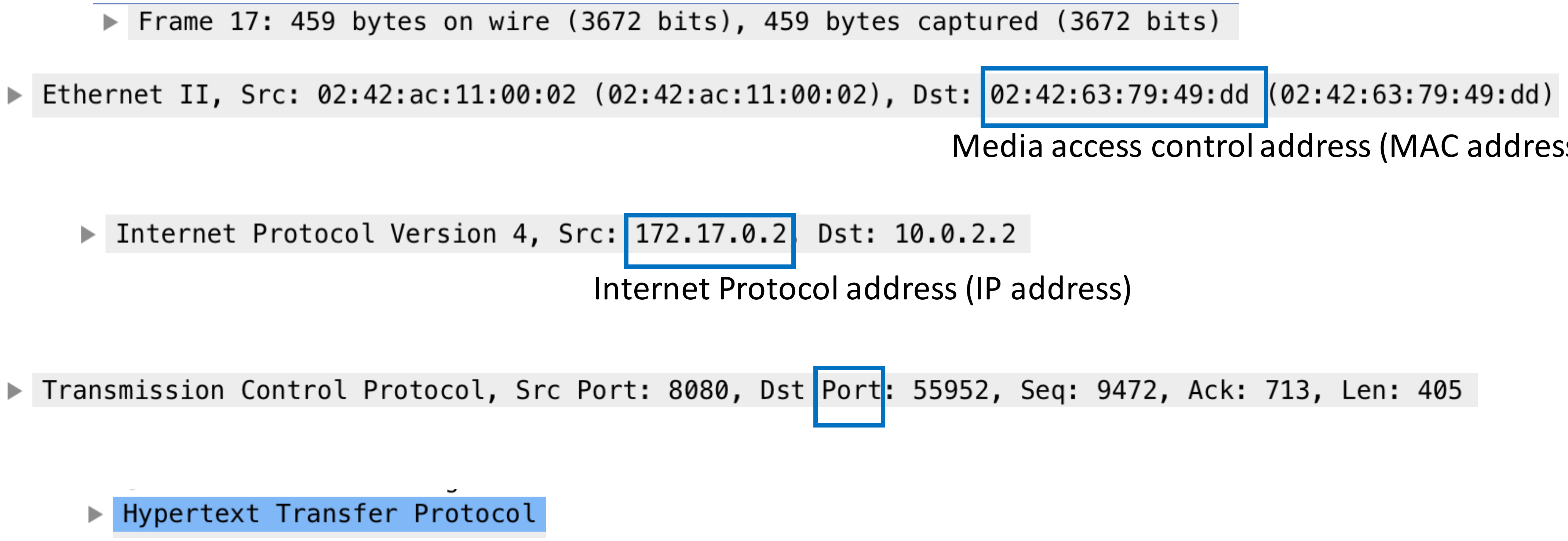
The goal of this project is to understand the progression of a multi-stage attack that aims to leak secret keys from target system. In this project, you will perform analysis of a multi-stage attack recreated from publicly available information on the Equifax breach. You will explore the use of signature-based, anomaly-based, and factor graph-based techniques.

Task description

- Task 0 (Wireshark, Pyshark, Pandas, etc.)
  - In this task, you will familiarize yourself with networking and operating system (OS) concepts
  - Data pre-processing of network traffic and OS event logs (Pyshark, pandas)
  - Differentiate between malicious and benign network activity given the CVE (vulnerability) in HTTP and HTTPS traffic (Wireshark)
- Task 1
  - HTTP, TCP, IP, and DNS traffic analysis (Pyshark, Wireshark)
  - Host level OS activity
- Mainly use Pyshark and pandas for project task deliverables in a Jupyter Notebook;
- Only use Wireshark as a supplementary tool for intuitive visual inspection
  - Provide screenshot if the evidence to task solutions is from Wireshark

Wireshark

- A network packet analyzer
- Packet is layered (MAC address, IP address, port numbers, etc.)



```
▶ Frame 17: 459 bytes on wire (3672 bits), 459 bytes captured (3672 bits)
```

Physical layer

```
▶ Ethernet II, Src: 02:42:ac:11:00:02 (02:42:ac:11:00:02), Dst: 02:42:63:79:49:dd (02:42:63:79:49:dd)
```

Data Link layer

Media access control address (MAC address)

```
▶ Internet Protocol Version 4, Src: 172.17.0.2, Dst: 10.0.2.2
```

Network layer

Internet Protocol address (IP address)

```
▶ Transmission Control Protocol, Src Port: 8080, Dst Port: 55952, Seq: 9472, Ack: 713, Len: 405
```

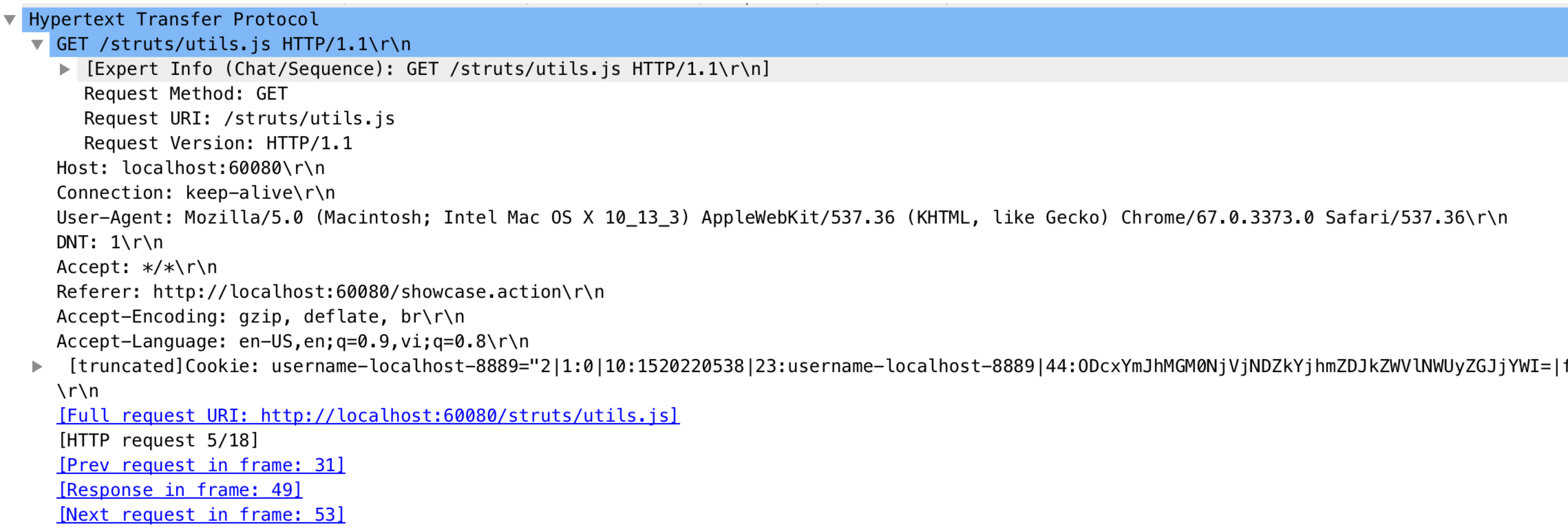
Transport layer

```
▶ Hypertext Transfer Protocol
```

Application layer

Wireshark

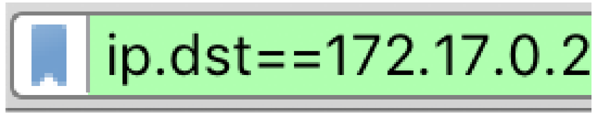
- HTTP traffic: GET request, hostname, data, etc.
  - GET request: retrieval of information from the server



```
▼ Hypertext Transfer Protocol
  ▼ GET /struts/utils.js HTTP/1.1\r\n
    ▶ [Expert Info (Chat/Sequence): GET /struts/utils.js HTTP/1.1\r\n]
      Request Method: GET
      Request URI: /struts/utils.js
      Request Version: HTTP/1.1
    Host: localhost:60080\r\n
    Connection: keep-alive\r\n
    User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/67.0.3373.0 Safari/537.36\r\n
    DNT: 1\r\n
    Accept: */*\r\n
    Referer: http://localhost:60080/showcase.action\r\n
    Accept-Encoding: gzip, deflate, br\r\n
    Accept-Language: en-US,en;q=0.9,vi;q=0.8\r\n
  ▶  [truncated]Cookie: username-localhost-8889="2|1:0|10:1520220538|23:username-localhost-8889|44:ODcxYmJhMGM0NjVjNDZkYjhmZDJkZWVlNWUyZGJjYWI=|
    \r\n
    [Full request URI: http://localhost:60080/struts/utils.js]
    [HTTP request 5/18]
    [Prev request in frame: 31]
    [Response in frame: 49]
    [Next request in frame: 53]
```

Wireshark

- Display filters



```
ip.dst==172.17.0.2
```

- Filter expressions: https://www.wireshark.org/docs/wsug_html_chunked/ChWorkBuildDisplayFilterSection.html
- more display filter examples: https://wiki.wireshark.org/DisplayFilters)

Pyshark

- Python wrapper for tshark (Wireshark command-line utility)
- Allow for parsing from a capture file or live capture (pcap files in mp3)
- Capture files can be used
  - as an iterator to get their packets
  - to receive various filters to save some of the incoming packets

- Tutorial available on course website: https://courses.engr.illinois.edu/ece498dsu/sp2019/secure/MP/MP3/pyshark_tutorial.zip