Lecture 24
November 19, 2025

ECS 235A, Computer and Information Security

Administrativia

- Homework 2, extra credit 2 grades now available
  - So are the answers
  - Problem 4 (the Otway-Rees protocol) gave trouble so we'll go over it now
- No handwritten homework or project submissions will be accepted
  - We will return them ungraded
  - You can turn them in in text format (``.txt'' on many systems)
- For the completed project, please add a brief write-up of what you did in the project
  - Submit this *individually*, not as a group

Otway-Rees Protocol

Alice $\xrightarrow{n \,||\, \text{Alice} \,||\, \text{Bob} \,||\, \{ r_1 \,||\, n \,||\, \text{Alice} \,||\, \text{Bob} \} k_A}$ Bob

Cathy $\xleftarrow[\{ r_2 \,||\, n \,||\, \text{Alice} \,||\, \text{Bob} \} k_B]{n \,||\, \text{Alice} \,||\, \text{Bob} \,||\, \{ r_1 \,||\, n \,||\, \text{Alice} \,||\, \text{Bob} \} k_A \,||}$ Bob

Cathy $\xrightarrow{n \,||\, \{ r_1 \,||\, k_s \} k_A \,||\, \{ r_2 \,||\, k_s \} k_B}$ Bob

Alice $\xleftarrow{n \,||\, \{ r_1 \,||\, k_s \} k_A}$ Bob

Homework 2, Question 5a (Otway-Rees)

Consider Alice when all 4 steps of the protocol have been completed. How does Alice know that steps 2 and 3 have taken place?

What Alice sees:

$$\text{Alice} \xrightarrow{n \,||\, \text{Alice} \,||\, \text{Bob} \,||\, \{ r_1 \,||\, n \,||\, \text{Alice} \,||\, \text{Bob} \} k_A} \text{Bob}$$

$$\text{Alice} \xleftarrow{n \,||\, \{ r_1 \,||\, k_s \} k_A} \text{Bob}$$

She never sees steps 2 and 3 and assumesmthe contents of the encrypted part

Homework 2, Question 5b (Otway-Rees)

What components of the protocol does Edgar know — that is, does he know r_1, r_2, n, or $k_{session}$, or the names of "Alice" and "Bob"? How?

Here is what Edgar monitors:

Alice $\xrightarrow{n \,||\, \text{Alice} \,||\, \text{Bob} \,||\, \{ r_1 \,||\, n \,||\, \text{Alice} \,||\, \text{Bob} \} k_A}$ Bob

n, "Alice", and "Bob" are visible so Edgar can see them
r_1 is encrypted and so not visible, and Edgar does not know the key k_A
r_2 and $k_{session}$ are not there, so Edgar cannot know them

Homework 2, Question 5c (Otway-Rees)

Given this, in step 4 of the protocol, how might Edgar provide Alice with a session key that he knows?

From step 1, Edgar knows "Alice", "Bob", and n, and $\{ r_1 \mid\mid n \mid\mid \text{Alice} \mid\mid \text{Bob} \} k_A$
In step 4, the enciphered part is $\{ r_1 \mid\mid k_s \} k_A$
So in step 4, the session key k_s would be $n \mid\mid \text{Alice} \mid\mid \text{Bob}$ — which Edgar knows

Step 1 Alice $\xrightarrow{n \mid\mid \text{Alice} \mid\mid \text{Bob} \mid\mid \{ r_1 \mid\mid n \mid\mid \text{Alice} \mid\mid \text{Bob} \} k_A}$ Bob

Step 4 Alice $\xleftarrow{n \mid\mid \{ r_1 \mid\mid k_s \} k_A}$ Bob

Homework 2, Question 5d (Otway-Rees)

How might someone fix this?

Add type information indicating the encryption in step 1 is identifying information and the encryption in step 4 is a key

As Edgar cannot decipher the encrypted parts he cannot change the "id" type indicator to a "sesskey" indicator

Step 1 Alice $\xrightarrow{n \,||\, \text{Alice} \,||\, \text{Bob} \,||\, \{ \text{id} \,||\, r_1 \,||\, n \,||\, \text{Alice} \,||\, \text{Bob} \} k_A}$ Bob

Step 4 Alice $\xleftarrow{n \,||\, \{ \text{sesskey} \,||\, r_1 \,||\, k_s \} k_A}$ Bob

Lai and Gray

- Implemented modified version of Karger's scheme on UNIX system
  - Allow programs to access (read or write) files named on command line
  - Prevent access to other files
- Two types of processes
  - Trusted: no access checks or restrictions
  - Untrusted: valid access list (VAL) controls access and is initialized to command line arguments plus any temporary files that the process creates

File Access Requests

1. If file on VAL, use effective UID/GID of process to determine if access allowed
2. If access requested is read and file is world-readable, allow access
3. If process creating file, effective UID/GID controls allowing creation
   - Enter file into VAL as NNA (new non-argument); set permissions so no other process can read file
4. Ask user. If yes, effective UID/GID controls allowing access; if no, deny access

Example

- Assembler invoked from compiler
- as x.s /tmp/ctm2345
- and creates temp file /tmp/as1111
  - VAL is
  - x.s /tmp/ctm2345 /tmp/as1111
- Now Trojan horse tries to copy x.s to another file
  - On creation, file inaccessible to all except creating user so attacker cannot read it (rule 3)
  - If file created already and assembler tries to write to it, user is asked (rule 4), thereby revealing Trojan horse

Trusted Programs

- No VALs applied here
  - UNIX command interpreters: *csh*, *sh*
  - Program that spawn them: *getty*, *login*
  - Programs that access file system recursively: *ar*, *chgrp*, *chown*, *diff*, *du*, *dump*, *find*, *ls*, *restore*, *tar*
  - Programs that often access files not in argument list: *binmail*, *cpp*, *dbx*, *mail*, *make*, *script*, *vi*
  - Various network daemons: *fingerd*, *ftpd*, *sendmail*, *talkd*, *telnetd*, *tftpd*

Specifications

- Treat infection, execution phases of malware as errors
- Example
  - Break programs into sequences of non-branching instructions
  - Checksum each sequence, encrypt it, store it
  - When program is run, processor recomputes checksums, and at each branch compares with precomputed value; if they differ, an error has occurred

N-Version Programming

- Implement several different versions of algorithm
- Run them concurrently
  - Check intermediate results periodically
  - If disagreement, majority wins
- Assumptions
  - Majority of programs not infected
  - Underlying operating system secure
  - Different algorithms with enough equal intermediate results may be infeasible
    - Especially for malicious logic, where you would check file accesses

Inhibit Sharing

- Use separation implicit in integrity policies
- Example: LOCK keeps single copy of shared procedure in memory
  - Master directory associates unique owner with each procedure, and with each user a list of other users the first trusts
  - Before executing any procedure, system checks that user executing procedure trusts procedure owner

Multilevel Policies

- Put programs at the lowest security level, all subjects at higher levels
  - By *-property, nothing can write to those programs
  - By ss-property, anything can read (and execute) those programs
- Example: Trusted Solaris system
  - All executables, trusted data stored below user region, so user applications cannot alter them

Proof-Carrying Code

- Code consumer (user) specifies safety requirement
- Code producer (author) generates proof code meets this requirement
  - Proof integrated with executable code
  - Changing the code invalidates proof
- Binary (code + proof) delivered to consumer
- Consumer validates proof
- Example statistics on Berkeley Packet Filter: proofs 300–900 bytes, validated in 0.3 –1.3 ms
  - Startup cost higher, runtime cost considerably shorter

Detecting Statistical Changes

- Example: application had 3 programmers working on it, but statistical analysis shows code from a fourth person—may be from a Trojan horse or virus!
  - Or libraries …
- Other attributes: more conditionals than in original; look for identical sequences of bytes not common to any library routine; increases in file size, frequency of writing to executables, etc.
  - Denning: use intrusion detection system to detect these

Entropy for Information Flow

- Random variables
- Joint probability
- Conditional probability
- Entropy (or uncertainty in bits)
- Joint entropy
- Conditional entropy
- Applying it to secrecy of ciphers

Random Variable

- Variable that represents outcome of an event
  - X represents value from roll of a fair die; probability for rolling n: $p(X=n) = 1/6$
  - If die is loaded so 2 appears twice as often as other numbers, $p(X=2) = 2/7$ and, for $n \neq 2$, $p(X=n) = 1/7$
- Note: $p(X)$ means specific value for X doesn't matter
  - Example: all values of X are equiprobable

Joint Probability

- Joint probability of X and Y, $p(X, Y)$, is probability that X and Y simultaneously assume particular values
  - If X, Y independent, $p(X, Y) = p(X)p(Y)$
- Roll die, toss coin
  - $p(X{=}3, Y{=}\text{heads}) = p(X{=}3)p(Y{=}\text{heads}) = 1/6 \times 1/2 = 1/12$

Two Dependent Events

- X = roll of red die, Y = sum of red, blue die rolls

$p(Y=2) = 1/36$	$p(Y=3) = 2/36$	$p(Y=4) = 3/36$	$p(Y=5) = 4/36$
$p(Y=6) = 5/36$	$p(Y=7) = 6/36$	$p(Y=8) = 5/36$	$p(Y=9) = 4/36$
$p(Y=10) = 3/36$	$p(Y=11) = 2/36$	$p(Y=12) = 1/36$	

- Formula:

  $p(X=1, Y=11) = p(X=1)p(Y=11) = (1/6)(2/36) = 1/108$

- But if the red die (X) rolls 1, the most their sum (Y) can be is 7
- The problem is X and Y are dependent

Conditional Probability

- Conditional probability of X given Y, $p(X \mid Y)$, is probability that X takes on a particular value given Y has a particular value
- Continuing example …
  - $p(Y=7 \mid X=1) = 1/6$
  - $\text{p}(Y=7 \mid X=3) = 1/6$

Relationship

- $p(X, Y) = p(X \mid Y)\, p(Y) = p(X)\, p(Y \mid X)$
- Example:

  $p(X{=}3, Y{=}8) = p(X{=}3 \mid Y{=}8)\, p(Y{=}8) = (1/5)(5/36) = 1/36$
- Note: if X, Y independent:

  $p(X \mid Y) = p(X)$

Entropy

- Uncertainty of a value, as measured in bits
- Example: *X* value of fair coin toss; *X* could be heads or tails, so 1 bit of uncertainty
  - Therefore entropy of *X* is $H(X) = 1$
- Formal definition: random variable *X*, values $x_1, \ldots, x_n$; so $\sum_i p(X = x_i) = 1$; then entropy is:

$$H(X) = -\sum_i p(X=x_i) \lg p(X=x_i)$$

Heads or Tails?

- $H(X) = -\,p(X\text{=heads}) \lg p(X\text{=heads}) - \text{p}(X\text{=tails}) \lg p(X\text{=tails})$

  $= -\,(1/2) \lg (1/2) - (1/2) \lg (1/2)$

  $= -\,(1/2)\,(-1) - (1/2)\,(-1) = 1$
- Confirms previous intuitive result

n-Sided Fair Die

$H(X) = -\Sigma_i \, p(X = x_i) \lg p(X = x_i)$

As $p(X = x_i) = 1/n$, this becomes

$H(X) = -\Sigma_i \, (1/n) \lg (1/ n) = -n(1/n) (-\lg n)$

so

$H(X) = \lg n$

which is the number of bits in n, as expected

Ann, Pam, and Paul

Ann, Pam twice as likely to win as Paul

W represents the winner. What is its entropy?

- w_1 = Ann, w_2 = Pam, w_3 = Paul
- $p(W=w_1) = p(W=w_2) = 2/5$, $p(W=w_3) = 1/5$

- So $H(W) = -\Sigma_i\, p(W=w_i) \lg p(W=w_i)$

  $= -(2/5) \lg (2/5) - (2/5) \lg (2/5) - (1/5) \lg (1/5)$

  $= -(4/5) + \lg 5 \approx -1.52$
- If all equally likely to win, $H(W) = \lg 3 \approx 1.58$

Joint Entropy

- X takes values from $\{ x_1, \ldots, x_n \}$, and $\Sigma_i\, p(X=x_i) = 1$
- Y takes values from $\{ y_1, \ldots, y_m \}$, and $\Sigma_i\, p(Y=y_i) = 1$
- Joint entropy of X, Y is:

$$H(X, Y) = -\Sigma_j\, \Sigma_i\, p(X=x_i, Y=y_j) \lg p(X=x_i, Y=y_j)$$

Example

X: roll of fair die, Y: flip of coin

As X, Y are independent:

$p(X{=}1, Y{=}\text{heads}) = p(X{=}1)\ p(Y{=}\text{heads}) = 1/12$

and

$$H(X, Y) = -\Sigma_j \Sigma_i\ p(X{=}x_i, Y{=}y_j) \lg p(X{=}x_i, Y{=}y_j)$$

$$= -2\ [\ 6\ [\ (1/12) \lg (1/12)\ ]\ ] = \lg 12$$

Conditional Entropy (Equivocation)

- X takes values from $\{ x_1, \ldots, x_n \}$ and $\Sigma_i\, p(X=x_i) = 1$
- Y takes values from $\{ y_1, \ldots, y_m \}$ and $\Sigma_i\, p(Y=y_i) = 1$
- Conditional entropy of X given $Y=y_j$ is:

$$H(X \mid Y=y_j) = -\Sigma_i\, p(X=x_i \mid Y=y_j) \lg p(X=x_i \mid Y=y_j)$$

- Conditional entropy of X given Y is:

$$H(X \mid Y) = -\Sigma_j\, p(Y=y_j)\, \Sigma_i\, p(X=x_i \mid Y=y_j) \lg p(X=x_i \mid Y=y_j)$$

Example

- X roll of red die, Y sum of red, blue roll
- Note $p(X=1 | Y=2) = 1$, $p(X=i | Y=2) = 0$ for $i \neq 1$
  - If the sum of the rolls is 2, both dice were 1
- Thus

$$H(X | Y=2) = -\Sigma_i\, p(X=x_i | Y=2) \lg p(X=x_i | Y=2) = 0$$

Example (*con't*)

- Note $p(X=i, Y=7) = 1/6$
  - If the sum of the rolls is 7, the red die can be any of 1, …, 6 and the blue die must be 7–roll of red die
- $H(X|Y=7) = -\sum_i p(X=x_i|Y=7) \lg p(X=x_i|Y=7)$

$$= -6\,(1/6) \lg (1/6) = \lg 6$$

Example: Perfect Secrecy

- Cryptography: knowing the ciphertext does not decrease the uncertainty of the plaintext
- $M = \{ m_1, \ldots, m_n \}$ set of messages
- $C = \{ c_1, \ldots, c_n \}$ set of messages
- Cipher $c_i = E(m_i)$ achieves *perfect secrecy* if $H(M \mid C) = H(M)$

Basics of Information Flow

- Bell-LaPadula Model embodies information flow policy
  - Given compartments *A*, *B*, info can flow from *A* to *B* iff *B dom A*
- So does Biba Model
  - Given compartments *A*, *B*, info can flow from *A* to *B* iff *A dom B*
- Variables *x*, *y* assigned compartments $\underline{x}$, $\underline{y}$ as well as values
  - Confidentiality (Bel-LaPadula): if $\underline{x} = A$, $\underline{y} = B$, and *B dom A*, then *y* := *x* allowed but not *x* := *y*
  - Integrity (Biba): if $\underline{x} = A$, $\underline{y} = B$, and *A dom B*, then *x* := *y* allowed but not *y* := *x*
- For now, focus on confidentiality (Bell-LaPadula)
  - We'll get to integrity later

Entropy and Information Flow

- Idea: information flows from x to y as a result of a sequence of commands c if you can deduce information about x before c from the value in y after c
- Formally:
  - s time before execution of c, t time after
  - $H(x_s \mid y_t) < H(x_s \mid y_s)$
  - If no y at time s, then $H(x_s \mid y_t) < H(x_s)$

Example 1

- Command is $x := y + z$; where:
  - x does not exist initially (that is, has no value)
  - $0 \le y \le 7$, equal probability
  - $z = 1$ with probability 1/2, $z = 2$ or 3 with probability 1/4 each
- s state before command executed; t, after; so
  - $\mathrm{H}(y_s) = \mathrm{H}(y_t) = -8(1/8) \lg (1/8) = 3$
- You can show that $H(y_s \mid x_t) = (3/32) \lg 3 + 9/8 \approx 1.274 < 3 = H(y_s)$
  - Thus, information flows from y to x

Example 2

- Command is

  if $x = 1$ **then** $y := 0$ **else** $y := 1$;

  where x, y equally likely to be either 0 or 1
- $H(x_s) = 1$ as x can be either 0 or 1 with equal probability
- $H(x_s \mid y_t) = 0$ as if $y_t = 1$ then $x_s = 0$ and vice versa
  - Thus, $H(x_s \mid y_t) = 0 < 1 = H(x_s)$
- So information flowed from x to y

Implicit Flow of Information

- Information flows from x to y without an *explicit* assignment of the form $y := f(x)$
  - $f(x)$ an arithmetic expression with variable x
- Example from previous slide:

if $x = 1$ **then** $y := 0$ **else** $y := 1$;

- So must look for implicit flows of information to analyze program

Notation

- $\underline{x}$ means class of *x*
  - In Bell-LaPadula based system, same as “label of security compartment to which *x* belongs”
- $\underline{x} \leq \underline{y}$ means “information can flow from an element in class of *x* to an element in class of *y*
  - Or, “information with a label placing it in class $\underline{x}$ can flow into class $\underline{y}$”

Compiler-Based Mechanisms

- Detect unauthorized information flows in a program during compilation
- Analysis not precise, but secure
  - If a flow *could* violate policy (but may not), it is unauthorized
  - No unauthorized path along which information could flow remains undetected
- Set of statements *certified* with respect to information flow policy if flows in set of statements do not violate that policy

Example

if *x* = 1 **then** *y* := *a*;

else *y* := *b*;

- Information flows from *x* and *a* to *y*, or from *x* and *b* to *y*
- Certified only if $\underline{x} \leq \underline{y}$ and $\underline{a} \leq \underline{y}$ and $\underline{b} \leq \underline{y}$
  - Note flows for *both* branches must be true unless compiler can determine that one branch will *never* be taken