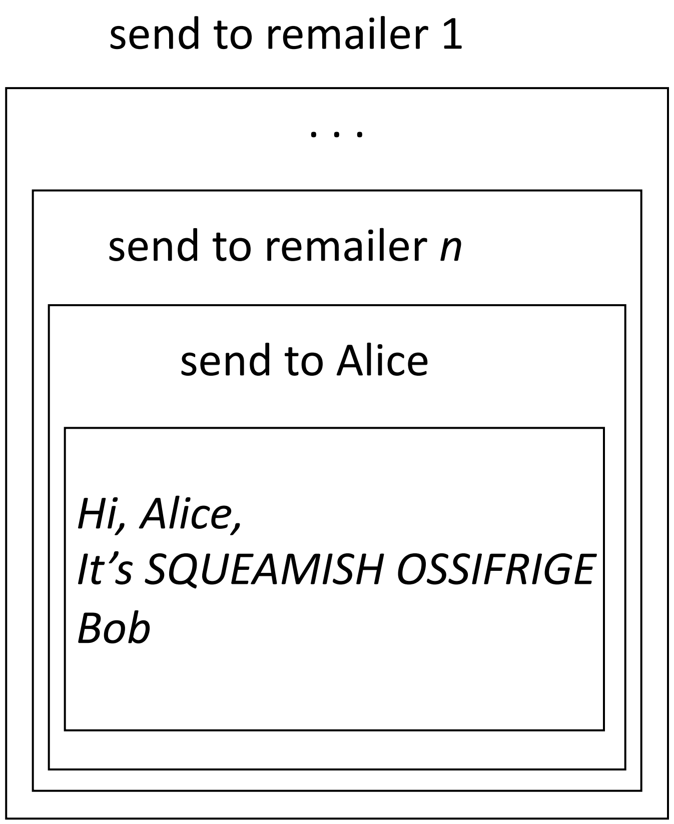
Lecture 28
November 30, 2022

Cypherpunk Remailer

- Remailer that deletes header of incoming message, forwards body to destination
- Also called *Type I Remailer*
- No record kept of association between sender address, remailer's user name
  - Prevents tracing, as happened with *anon.penet.fi*
- Usually used in a chain, to obfuscate trail
  - For privacy, body of message may be enciphered

Cypherpunk Remailer Message



- Encipher message
- Add recipient address
- Encipher and add remailer *n*'s address
- . . .
- Encipher and add remailer 1's address
- Send this to remailer 1

Weaknesses

- Attacker monitoring entire network
  - Observes in, out flows of remailers
  - Goal is to associate incoming, outgoing messages
- If messages are cleartext, trivial
  - So assume all messages enciphered
- So use traffic analysis!
  - Used to determine information based simply on movement of messages (traffic) around the network

Attacks

- If remailer forwards message before next message arrives, attacker can match them up
  - Hold messages for some period of time, greater than the message interarrival time
  - Randomize order of sending messages, waiting until at least *n* messages are ready to be forwarded
    - Note: attacker can force this by sending *n*–1 messages into queue

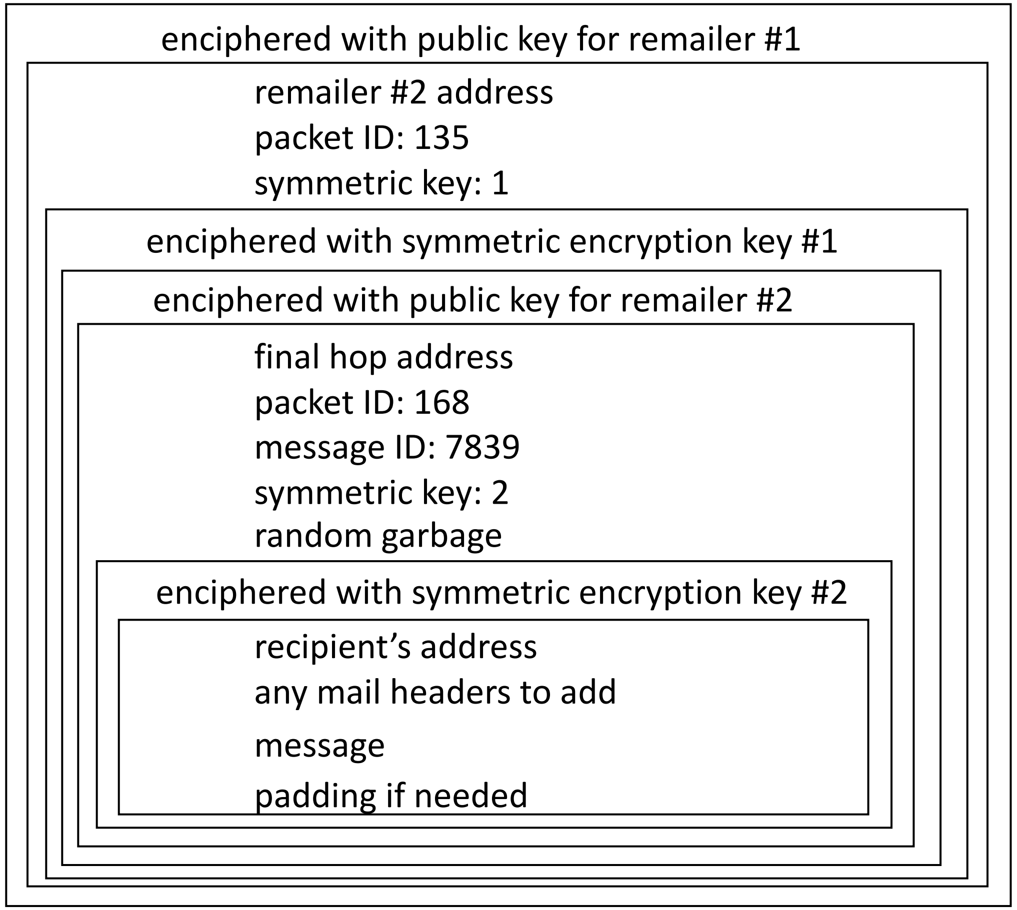
Attacks

- As messages forwarded, headers stripped so message size decreases
  - Pad message with garbage at each step, instructing next remailer to discard it
- Replay message, watch for spikes in outgoing traffic
  - Remailer can't forward same message more than once

Mixmaster (Cypherpunk Type 2) Remailer

- Cypherpunk remailer that handles only enciphered mail and pads (or fragments) messages to fixed size before sending them
  - Also called Type 2 Remailer
  - Designed to hinder attacks on Cypherpunk remailers
    - Messages uniquely numbered
    - Fragments reassembled *only* at last remailer for sending to recipient

Cypherpunk Remailer Message



Onion Routing

- Method of routing so each node in the route knows only the previous and following node
  - Typically, first node selects the route
  - Intermediate node may be able to change rest of route
- Each intermediate node has public, private key pair
  - Public key available to all nodes and any proxies
- Client, server have proxies to handle onion routing

Heart of the Onion Route

{ *expires* || *nexthop* || E_F || k_F || E_B || k_B || *payload* } pub_r

- *payload*: data associated with message
- *expires*: expiration time for which *payload* is to be saved
- *nexthop*: node to forward message to
- pub_r: public key of next hop (node)
- E_F, k_F : encryption algorithm, key to be used when sending message forward to server
- E_B, k_B : encryption algorithm, key to be used when sending message backwards to client

Notes About the Heart

- *payload* may itself be a message of this form or the data being sent
- Each router has table storing:
  - Virtual circuit number associated with a route
  - E_F, k_F, E_B, k_B for the next, previous nodes on the route
  - Next router to which messages using this route are to be forwarded
    - If last router on route, this is NULL (as is *nexthop* in the packet)

Creating a Route

- Client's proxy determines route for the message
  - Can be defined exactly, or loosely, where the intermediate routers can route messages to next hop over other routes
- Create onion encapsulating route, put it in a *create* message and add virtual circuit number
- Forward to next (second) router on path
- That router deciphers the onion using its private key ("peeling the onion")
  - Compare it to what's in table; if replay, discard

Creating a Route

- Router creates new virtual circuit number, and add to table:
  - (virtual circuit number in message, created virtual circuit number) pair
  - Keys, algorithms in onion
- Router generates new *create* message, puts assigned virtual circuit number and “peeled” onion in it
  - This is smaller than the onion received, so add padding to make it the same size
- Forward it to next hop

Sending a Message

- Sender applies decryption algorithms corresponding to each backwards encryption algorithm along the route
- Example: route begins at *W*, then through *X* and *Y* to *Z*; *W* constructs this:

$$d_X(k_X, d_Y(k_Y, d_Z(k_Z, m)))$$

- Sends this to *X*, which uses its E_B to encrypt message, getting

$$d_Y(k_Y, d_Z(k_Z, m))$$

- Forwards this to *Y*, which uses its E_B to encrypt message, getting

$$d_Z(k_Z, m)$$

- Forwards this to *Z*, which uses its E_B to encrypt message, getting *m*

Potential Attacks

- If client's proxy compromised, attacker can see all routes selected and all messages, and so may be able to deduce server
- If server's proxy compromised, attacker can see all messages but cannot deduce the routes
- If router compromised, attacker can determine only the previous, next routers in path
    - In particular, the attacker cannot read the encrypted onion
- Attacker can see all traffic on network
    - Matching client, server message sizes; that's why all messages are padded to same size
    - Observing the flow of messages; have the onion network send meaningless messages to obscure that flow

Example: Tor (The Onion Router)

- Connects clients, servers over virtual circuits set up among onioon routers (*OR*)
  - Each OR has identity key, onion key
  - Identity key signs information about router
  - Onion key used to read requests to set up circuits; changed periodically
  - All virtual circuits over TLS, and a third TLS key established for this
- Basic message unit: *cell*, always 512 bytes long
  - Control cell: header contains command directing recipient to do something
    - Create a circuit, circuit created, destroy a circuit
  - Relay cell: deals with an established circuit
    - Open stream, stream opened, extend circuit, circuit extended, close stream cleanly, close broken stream, cell contains data

Example: Tor (The Onion Router)

- Connects clients, servers over virtual circuits set up among onioon routers (*OR*)
  - Each OR has identity key, onion key
  - Identity key signs information about router
  - Onion key used to read requests to set up circuits; changed periodically
  - All virtual circuits over TLS, and a third TLS key established for this
- Basic message unit: *cell*, always 512 bytes long
  - Control cell: header contains command directing recipient to do something
    - Create a circuit, circuit created, destroy a circuit
  - Relay cell: deals with an established circuit
    - Open stream, stream opened, extend circuit, circuit extended, close stream cleanly, close broken stream, cell contains data

Setting Up Virtual Circuit

- Set up over TLS connections
  - Several circuits may use same TLS connection to reduce overhead
- Streams move data over virtual circuits
  - Several streams may be multiplexed over one circuit
- Client's onion proxy OP_c needs to know where ORs are
  - Tor uses directory services for this; group of well-known ORs track information about usable ORs, including keys, addresses
  - OPc contacts one such directory server, gets information from it, chooses path

Setting Up Virtual Circuit

- Tor uses 3 ORs (OR_1, OR_2, OR_3); client, server proxies OP_c, OP_s
- *RSA*(*x*) is enciphering of message *x* using onion key of destination OR
- *g*, *p* as in Diffie-Hellman
- $x_1, ..., x_n$ and $y_1, ... y_n$ generated randomly; $k_i = g^{x_i y_i} \bmod p$, and forward, backwards keys selected from this
- *h*(*x*) cryptographic hash of *x*
- All links are over TLS and so encrypted (TLS keys not shown on next slide)

Tor Protocol to Create Virtual Circuit

This sets up the part of the virtual circuit between OP_c and OR_1:

$$OP_c \xrightarrow{\{\, create \,||\, c_1 \,||\, RSA(g^{x_1}) \,\}} OR_1$$

$$OP_c \xleftarrow{\{\, created \,||\, c_1 \,||\, g^{y_1} \,||\, h(k_1) \,\}} OR_1$$

Tor Protocol to Create Virtual Circuit

This sets up the part of the virtual circuit between OP_c and OR_2:

$$OP_c \xrightarrow{\{\, relay \,||\, c_1 \,||\, \{\, extend \,||\, OR_2 \,||\, RSA(g^{x_2}) \,\}\,\}} OR_1$$

$$OR_1 \xrightarrow{\{\, create \,||\, c_2 \,||\, RSA(g^{x_2}) \,\}} OR_2$$

$$OR_1 \xleftarrow{\{\, created \,||\, c_2 \,||\, g^{y_2} \,||\, h(k_2) \,\}} OR_2$$

$$OP_c \xleftarrow{\{\, relay \,||\, c_1 \,||\, \{\, extended \,||\, g^{y2} \,||\, h(k_2) \,\}} OR_1$$

Tor Protocol to Create Virtual Circuit

This sets up the part of the virtual circuit between OP_c and OR_3:

$$OP_c \xrightarrow{\{\, relay \,||\, c_1 \,||\, \{\, extend \,||\, OR_3 \,||\, RSA(g^{x_3}) \,\}\,\}} OR_1$$

$$OR_1 \xrightarrow{\{\, relay \,||\, c_2 \,||\, \{\, extend \,||\, OR_3 \,||\, RSA(g^{x_3}) \,\}\,\}} OR_2$$

$$OR_2 \xrightarrow{\{\, create \,||\, c_3 \,||\, RSA(g^{x_3}) \,\}} OR_3$$

$$OR_3 \xleftarrow{\{\, created \,||\, c_3 \,||\, g^{y_3} \,||\, h(k_3) \,\}} OR_3$$

$$OR_1 \xleftarrow{\{\, relay \,||\, c_2 \,||\, \{\, extended \,||\, g^{y_3} \,||\, h(k_3) \,\}\,\}} OR_2$$

$$OR_1 \xleftarrow{\{\, relay \,||\, c_1 \,||\, \{\, extended \,||\, g^{y_2} \,||\, h(k_2) \,\}\,\}} OP_c$$

After All This . . .

- OP_c has forward keys for OR_1, OR_2, OR_3; call them f_1, f_2, f_3
  - Here, $f_i = g^{y_i} \bmod p$
- To send message m to server, client sends m to OP_c
  - OP_c enciphers it using AES-128 in counter mode, getting $\{ \{ \{ m \}f_1 \}f_2 \}f_3$
  - It puts this into a relay cell and sends it to OR_1
- OR_1 deciphers cell, determines next hop by looking up virtual circuit number in its table, puts $\{ \{ m \}f_1 \}f_2$ into another relay cell, forwards it to OR_2
- OR_2 does same, but forwards it to OR_3
- OR_3 deciphers cell, either does what m requests (eg, open TLS connection to server) or forwards payload m to server

Server Replies

- Server sends reply *r* to OR_3
- OR_3 enciphers it using its backwards key, embeds it in relay cell, forwards it to OR_2
- OR_2 uses circuit number to determine OR_1, enciphers cell using its backwards key, forwards it to OR_1
- OR_1 does same but forwards it to OP_c
- OP_c has all the forward keys, and so can decipher the message and forward it to client

Use Problems

Adversary wants to determine who is using onion routing network

- Attack: monitor the client, known entry router
  - Solution: use unlisted entry routers
  - Example: Tor uses *bridge relays* that are not listed in Tor directories; to find them, go to specific web page or email a specific set of addresses; result is a list of entry routers (bridges) that OP_c can use
- Attack: examine packets sent from a client looking for structures indicating that they are intended for onion routers
  - Solution: obfuscate packet contents; endpoint deobfuscates it
  - Example: Tor has *pluggable transports* that do this

Anonymity Itself

- Some purposes for anonymity
  - Removes personalities from debate, or with appropriate choice of pseudonym, shape course of debate by implication
  - Prevent retaliation
  - Protect privacy
- Are these benefits or drawbacks?
  - Depends on society, and who is involved

Pseudonyms

- Names of authors of documents used to imply something about the document
- Example: *U.S. Federalist Papers*
    - These argued for the states adopting the U.S. Constitution
    - Real authors were Alexander Hamilton, James Madison, John Jay, all Federalists who wanted the Constitution adopted
    - But using alias "Publius" hid their names
        - Debate could focus on content of the *Federalist Papers*, not the authors or their personalities
        - Roman Publius seen as a model governor, implying the *Papers* represented responsible political philosophy, legislation

Whistleblowers

- Criticism of powerholders often fall into disfavor; powerholders retaliate, but anonymity protects these critics
  - Example: Anonymous sources spoke to Woodward and Bernstein, during U.S. Watergate scandal in 1970s; one important source, called "Deep Throat", provided guidance that helped uncover a pattern of activity leading to impeachment articles against President Nixon and his resignation
    - "Deep Throat" later revealed as an assistant director of Federal Bureau of Investigation; had this been known, he would have been fired and might have been prosecuted
  - Example: Galileo openly held Copernican theory of the earth circling the sun; brought before the Inquisition and forced to recant

Privacy

- Anonymity protects privacy by obstructing amalgamation of individual records
- Important, because amalgamation poses 3 risks:
  - Incorrect conclusions from misinterpreted data
  - Harm from erroneous information
  - Not being let alone
- Also hinders monitoring to deter or prevent crime
- Conclusion: anonymity can be used for good or ill
  - Right to remain anonymous entails responsibility to use that right wisely

Intrusion Detection

- Detect wide variety of intrusions
  - Previously known and unknown attacks
  - Suggests need to learn/adapt to new attacks or changes in behavior
- Detect intrusions in timely fashion
  - May need to be be real-time, especially when system responds to intrusion
    - Problem: analyzing commands may impact response time of system
  - May suffice to report intrusion occurred a few minutes or hours ago

Intrusion Detection Systems

- Present analysis in simple, easy-to-understand format
  - Ideally a binary indicator
  - Usually more complex, allowing analyst to examine suspected attack
  - User interface critical, especially when monitoring many systems
- Be accurate
  - Minimize false positives, false negatives
  - Minimize time spent verifying attacks, looking for them

Principles of Intrusion Detection

- Characteristics of systems not under attack
  - User, process actions conform to statistically predictable pattern
  - User, process actions do not include sequences of actions that subvert the security policy
  - Process actions correspond to a set of specifications describing what the processes are allowed to do
- Systems under attack do not meet at least one of these

Example

- Goal: insert a back door into a system
  - Intruder will modify system configuration file or program
  - Requires privilege; attacker enters system as an unprivileged user and must acquire privilege
    - Nonprivileged user may not normally acquire privilege (violates #1)
    - Attacker may break in using sequence of commands that violate security policy (violates #2)
    - Attacker may cause program to act in ways that violate program's specification

Basic Intrusion Detection

- *Attack tool* is automated script designed to violate a security policy
- Example: *rootkit*
  - Includes password sniffer
  - Designed to hide itself using Trojaned versions of various programs (*ps*, *ls*, *find*, *netstat*, etc.)
  - Adds back doors (*login*, *telnetd*, etc.)
  - Has tools to clean up log entries (*zapper, etc.)*

Detection

- *Rootkit* configuration files cause *ls*, *du*, etc. to hide information
  - *ls* lists all files in a directory
    - Except those hidden by configuration file
  - *dirdump* (local program to list directory entries) lists them too
    - Run both and compare counts
    - If they differ, *ls* is doctored
- Other approaches possible

Key Point

- *Rootkit* does *not* alter kernel or file structures to conceal files, processes, and network connections
  - It alters the programs or system calls that *interpret* those structures
  - Find some entry point for interpretation that *rootkit* did not alter
  - The inconsistency is an anomaly (violates #1)

Denning’s Model

- Hypothesis: exploiting vulnerabilities requires abnormal use of normal commands or instructions
  - Includes deviation from usual actions
  - Includes execution of actions leading to break-ins
  - Includes actions inconsistent with specifications of privileged programs

Models of Intrusion Detection

- Anomaly detection
  - What is usual, is known
  - What is unusual, is bad
- Misuse detection
  - What is bad, is known
  - What is not bad, is good
- Specification-based detection
  - What is good, is known
  - What is not good, is bad

Anomaly Detection

- Analyzes a set of characteristics of system, and compares their values with expected values; report when computed statistics do not match expected statistics
  - Threshold metrics
  - Statistical moments
  - Markov model

Misuse Detection

- Determines whether a sequence of instructions being executed is known to violate the site security policy
  - Descriptions of known or potential exploits grouped into *rule sets*
  - IDS matches data against rule sets; on success, potential attack found
- Cannot detect attacks unknown to developers of rule sets
  - No rules to cover them

Types of Learning

- *Supervised learning methods*: begin with data that has already been classified, split it into “training data”, “test data”; use first to train classifier, second to see how good the classifier is
- *Unsupervised learning methods*: no pre-classified data, so learn by working on real data; implicit assumption that anomalous data is small part of data
- Measures used to evaluate methods based on:
  - TP: true positives (correctly identify anomalous data)
  - TN: true negatives (correctly identify non-anomalous data)
  - FP: false positives (identify non-anomalous data as anomalous)
  - FN: false negatives (identify anomalous data as non-anomalous)

Measuring Effectiveness

- *Accuracy*: percentage (or fraction) of events classified correctly
  - $((TP + TN) / (TP + TN + FP + FN)) * 100\%$
- *Detection rate*: percentage (or fraction) of reported attack events that are real attack events
  - $(TP / (TP + FN)) * 100\%$
  - Also called the *true positive rate*
- *False alarm rate*: percentage (or fraction) of non-attack events reported as attack events
  - $(FP / (FP + TN)) * 100\%$
  - Also called the *false positive rate*

Usefulness of Measurement

- Data at installation should be similar to that used to measure effectiveness
- Example: military, academic network traffic different
  - KDD-CUP-99 dataset derived from unclassified and classified network traffic on an Air Force Base
  - Network data captured at Florida Institute of Technology
- FIT data showed anomalies not in KDD-CUP-99
  - FIT data: TCP ACK field nonzero when ACK flag not set
  - KDD-CUP-99 data: HTTP requests all regular, all used GET, version 1.0; in FIT data, HTTP requests showed inconsistencies, some commands not GET, versions 1.0, 1.1
- Conclusion: using KDD-CUP-99 data would show some techniques performing better than they would on the FIT data

Specification Modeling

- Determines whether execution of sequence of instructions violates specification
- Only need to check programs that alter protection state of system
- System traces, or sequences of events t_1, … t_i, t_{i+1}, …, are basis of this
  - Event t_i occurs at time $C(t_i)$
  - Events in a system trace are totally ordered

Comparison and Contrast

- Misuse detection: if all policy rules known, easy to construct rulesets to detect violations
  - Usual case is that much of policy is unspecified, so rulesets describe attacks, and are not complete
- Anomaly detection: detects unusual events, but these are not necessarily security problems
- Specification-based vs. misuse: spec assumes if specifications followed, policy not violated; misuse assumes if policy as embodied in rulesets followed, policy not violated