Foundations of Network and Computer Security

John Black

Lecture #17
Oct 27th 2005

CSCI 6268/TLEN 5831, Fall 2005

Backscatter Technique

- CAIDA (San Diego) owns large block of IP address space
  - They have a lightly-used Class A network
  - They assumed
    - All source addresses uniformly chosen
      - Misses reflection attacks
    - All attack packets reliably reach victim
    - All replies reliably leave victim
    - Any unsolicited replies seen by CAIDA were backscatter

Approach

- Backscatter packets revealed
  - Type of attack
    - SYN/ACK means SYN flood
    - ICMP messages from routers indicated other types of attacks like UDP floods
  - IP of victim
    - Source address of backscatter
  - Intensity of attack
  - Duration of attack

Results

- 12,805 distinct attacks against over 5,000 hosts in 2,000 organizations in three weeks
- About 6000 packets per sec on average

DDoS: Preventative Measures

- Tracing and filtering
  - If source addresses could not be forged, filtering would be a reasonable solution
- Ingress filtering
  - Idea: if you are an ISP, don't let packets leave your IP address space if they have source addresses out side your address space
  - Old idea
  - Simple
  - Still a lot of ISPs don't do this
  - Even with ingress filtering, attackers can jump around within a range of IP addresses
  - Note that this limitation meant some backscatter numbers were probably a bit off

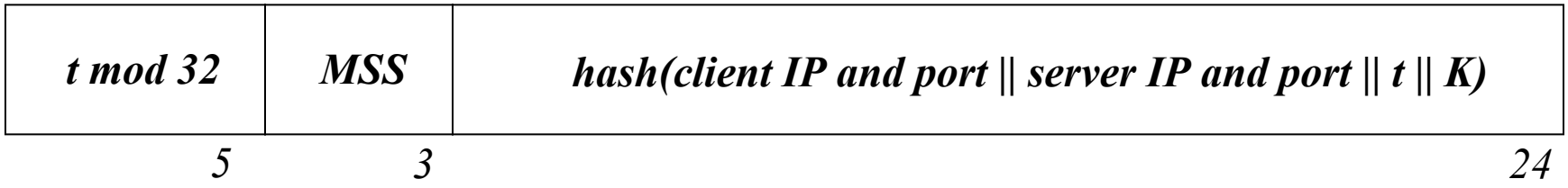
SYN Cookies

- A SYN flood leaves half-open connections
  - The “SYN queue” is a data structure which keeps track of these half-open connections
  - We track the source IP and port of client, server IP and port, seq# of client, seq# of server
  - Idea: we don’t really need to keep all of this
    - We just need enough to recognize the ACK of the client
    - Can we get away without storing *anything* locally?

SYN Cookies: The Idea

- Store nothing locally
  - ISN: Initial sequence number
  - Encode all we need to remember in the ISN we send back to the client
  - t: a 32-bit counter which increments every 64 seconds
  - K: a secret key selected by server for uptime of server
  - Limitations: MSS limited to 8 values

Server ISN



SYN Cookies: Details

- MSS: Maximum Segment Size
  - Suggested by client, server then computes best value
    - Details depend on whether they are on the same network, MTU on network, etc
    - Server can have only 8 values to encode here
- What happens when client replies with ACK?
  - Client will reply with ISN+1 of server in the ACK
  - Server then subtracts 1 and checks against hash of client IP and port, server IP and port, t which matches in the lowest 5 bits, and K
    - If match, put in SYN queue
    - If not, ignore

SYN Cookies: Limitations

- Note that this will NOT prevent bandwidth-saturation attacks
  - This technique seeks only to prevent SYN queue overflows
  - If attacker can saturate bandwidth, this doesn't help
    - But note that bandwidth saturation is not going to be TCP-based
    - UDP and ICMP can be used for bandwidth saturation, but we are often less dependent on these protocols

SYN Cookies: Implementation

- Standard in Linux and FreeBSD

echo 1 > proc/sys/net/ipv4/tcp_syncookies

- As far as I know, Windows does not implement them yet

Tracebacks Methods

- One basic problem with fighting DDoS is that we cannot find the *source IP* of the attacker
  - Finding the attacker would allow us to shut down the attack at the source
    - This assumes ISPs will cooperate and that there is a mechanism in place for reporting the source
    - Both of these assumptions are questionable as we saw in the Gibson story
  - The Internet Protocol (IP) makes it hard to find out where things are coming from
    - Easy to forge source IPs
    - No tracing mechanism available
      - This is on purpose

Adding Traceback

- Perhaps we could add a mechanism to IP to implement traceback
  - Still doesn't stop reflectors
  - Needs to be backward-compatible with current routing protocols
    - If not, too expensive and no one will do it
  - There have been several suggestions
    - Probabilistic traceback
    - Algebraic traceback
    - Others
  - We'll look just at probabilistic traceback

Probabilistic Traceback

- Original idea due to Savage, Wetherall, Karlin, and Anderson
  - "Practical Network Support for IP Traceback"
- Improved scheme due to Song and Perrig
  - "Advanced and Authenticated Marking Schemes for IP Traceback"
- We'll focus on the first paper, even though it is still far from a complete solution

First Try: Link Testing

- Idea: Manually trace source of traffic
  - Too labor intensive
  - Some tools developed, but requires a lot of cooperation between ISPs and backbone companies
    - Not much economic incentive to cooperate
  - Could use “controlled flooding”
    - Induce traffic from upstream routers and see which traffic is dropped
    - But this is a DoS attack itself… ethical?
    - Relies on being able to generate traffic
    - Requires good map of the Internet… hard to get
  - Both are useful only *during* an attack

How about Logging?

- Idea: select routers log all packets as they pass through
  - Then what?
  - Data mining techniques to try and figure out which packets were part of an attack
  - Then trace back upstream
  - Huge resource requirements!
  - Large-scale inter-provider database integration problems

Packet Marking

- Idea: mark packets as they pass through routers
  - The mark should give information as to what route the packet took
  - One idea is to mark every packet that traverses a given router
    - Just append their IP address to a list in the IP header
    - Drawback is that this is a HUGE burden to put on routers
      - They would have to mark EVERY packet
      - Packets would get enormous if they travel a long route
      - Packets might be caused to fragment

Probabilistic Packet Marking

- First, some assumptions:
  - Attackers can generate any packet
  - Attackers can conspire
  - Packets can be lost or reordered
  - Route from attacker to victim is mostly stable
  - Routers are not widely compromised

PPM: Continued

- Each router writes its address in a 32-bit field only with probability p
  - Routers don't care if they are overwriting another router's address
  - Probability of seeing the mark of a router d hops away is $p(1-p)^{d-1}$
  - This is monotonic so victim can sort by number of packets received and get the path
    - Smallest number is received by furthest router, etc

PPM: Difficulties

- We have to change the IP header any time a router marks a packet
  - This means storing the mark (has drawbacks)
  - Updating the header checksum
    - But this is already done for TTL decrements
- But we may need a LOT of packets to reconstruct a path
  - Suppose p=0.51 and d=15, then we need more than 42,000 to get a single sample from the furthest router
  - To get the order right with 95% probability requires around 300,000 packets
- Multiple attackers complicates matters
  - With multiple attackers at the same distance, this all breaks down

Next Try: Edge Sampling

- Reserve two address-sized fields in the IP header: “start” and “end”
- Reserve a small “distance” field as well
- When a router decides to mark a packet, it writes its address in the “start” field and zeroes the distance field
- When a router sees a zero in the distance field, it writes its address in the “end” field
- If a router decides not to mark a packet, it increments the distance field only
  - Must use saturating addition
  - This is critical to minimize spoofing by the attacker; without it, attackers could inject routers close to the victim
  - Now attacker can only spoof marks with distance counts equal or greater than its distance from the victim
- Note that we can now use any probability p we like
  - We’re not sorting based on packet counts any longer

Edge Sampling (Cont)

- The expected number of packets needed for the victim to reconstruct the entire path is at most $\ln(d)/p(1-p)^{d-1}$
  - Example: p=0.1, d=10, reconstruction requires about 75 packets
  - This is related to the coupon-collection problem
- Edge sampling allows reconstruction of the whole attack tree
- Encoding start, end, and distance is a problem
  - Not backward compatible if we change the IP header!
  - There are ways around this

Digression: Coupon Collection

- Suppose you have t types of coupons, C_1, C_2, … , C_t
  - Each time you open baseball cards, you get a coupon of type i with probability 1/t
  - How many coupons do you need before you have a complete set?
    - Note that in real competitions, all types are not 1/t
    - Call total number you need N (a random variable)
  - Define a random variable N_i indicating the number of draws you need to use when you hold i-1 coupon types and you want a new type
    - Then $N = N_1 + N_2 + \ldots + N_t$

Coupon Collection (cont)

- Then $E(N) = E(N_1) + \ldots E(N_t)$
  - Linearity of expectation
- What is $E(N_i)$?
  - Probability of getting a new type if you have i-1 types is (N-i+1)/N, so expectation is N/(N-i+1)
    - For geometric random variables, expectation is the inverse of the parameter
    - If you have a fair die, it takes an expected 6 rolls to get a 4 (for example)
  - So $E(N) = 1 + N/(N-2) + N/(N-3) + \ldots + N$ or $N H_N$
    - Here H_N is the Nth harmonic number
    - This is approximately N ln N.

Back to Reality

- No one does this
  - Yet?!
- DDoS attacks are still a huge problem and are still quite common
- But fortunately there is even more to worry about

TCP Session Hijacking

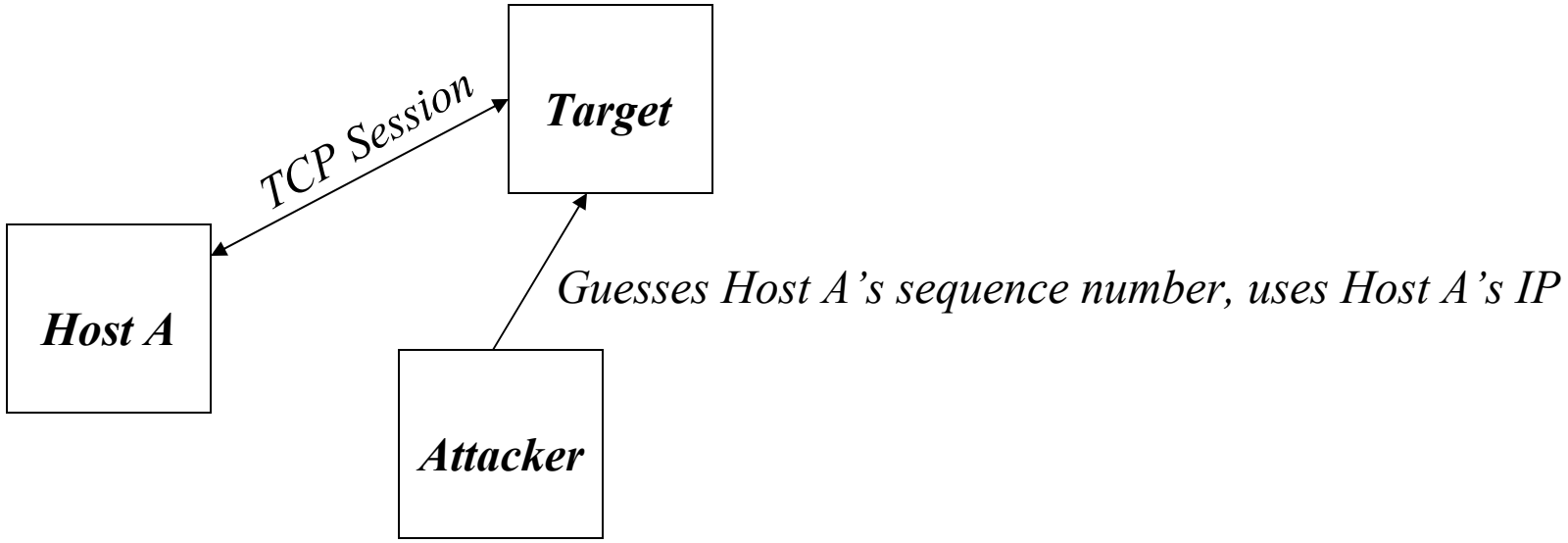
- This is the last topic on network-based attacks for a while
- We'll do vulnerabilities next
- We'll come back to some network protocols and some more crypto later in the course

Session Hijacking

- How might we jump in on an established TCP session?
  - If we could sniff the connections and inject traffic, we could do this with no problem
  - If we can only inject traffic (by sending unsolicited TCP segments to the victim) it's harder
    - Must guess the proper sequence number
    - Successfully used by Mitnick

Hijacking

- If attacker uses sequence number outside the window of Target, Target will drop traffic
- If attacker uses sequence number within window, Target accepts as from Host A
  - Result is a one-sided connection
  - Can be used to crash Target, confuse, reset connection, etc



Preventing Hijacking

- Make sequence number hard-to-guess
  - Use random ISNs
    - Note that SYN cookies in effect do this by using a hash of some stuff which includes a counter and a secret key
- There are many other kinds of hijacking techniques
  - We'll later look at ARP cache poisoning

Project #2: Secure Email System

Our goal is to provide a secure email system to each member of the class (including your grader).

We are going to use both symmetric-key and public-key techniques in this project, thus tying together several of the concepts discussed in lecture. As usual, we'll use OpenSSL as our toolkit, either via the command-line interface (easiest) or via system calls (you'll need the OpenSSL book for this!)

The program you write will have three main functions:

1. A mini-database utility to keep track of certs you have acquired from Martin's web site
2. A method to send encrypted and signed email
3. A method to verify and decrypt received email

Format of the Message

- We'll start by describing what a message will look like. Then we'll back-fill the details about how to generate and digest messages in this format. Messages will look like this:

```
-----BEGIN CSCI 6268 MESSAGE-----
<session pwd encrypted under target's public key>
<blank line>
<message encrypted under session pwd above>
<blank line>
<signature of above content>
-----END CSCI 6268 MESSAGE-----
```

Message Format

- First `-----BEGIN CSCI 6268 MESSAGE-----` must appear exactly as shown; this is the indicator that the message begins immediately after this line. (This allows the message to be embedded in a bunch of other text without confusing the recipient's parser.)
- The next line is the session password encrypted under the target's public key. This password is a random string of 32 characters using A-Z, a-z, and 0-9 generated by the sender; the sender then encrypts his message with AES in CBC mode using this password.
- There is a blank line, followed by the AES-CBC encrypted message in base64 format. This is followed by another blank line.
- Next comes the signature of the sender which is generated using the sender's private key. This signature will be the RSA sig of the SHA-1 hash of every line above from the first line after the BEGIN marker to the line just before the blank line ending the message. Exclude newlines (since they are different between Unix and DOS apps).
- Finally, `-----END CSCI 6268 MESSAGE-----` concludes the encrypted message.

The Cert Database

Your program should maintain a simple catalog of certs which you have collected from the web site. You may store them in whatever format you prefer (a flat file is the simplest, but if you prefer to use MySQL or something fancier, be my guest).

A cert should always be verified using the CA's public key before being inserted into the database.

A cert should always be verified using the CA's public key after being extracted from the database (to ensure someone hasn't tampered with it while you weren't watching).

You need not store the person's email address in your database since this is embedded in the cert, but it might be easier to go ahead and store the email addresses as an index field in the file. Of course, you must not rely on these index names as the validated email addresses; always make sure the email in the cert matches!

Sending Secure Mail

Your program should accept a plain-text message along with a destination email address and output an encrypted and signed message as we described a moment ago. Here is the algorithm:

1. Get the cert of the target from the database, using the email address as the index; if the email is not there, you must extract it from the web page.
2. Verify the signature on this cert for your email target.
3. Generate a 32-character passphrase. Normally we would use a very strong random-number generator for this, but feel free to use random() or the rand function of OpenSSL if you like.
4. Encrypt the message with AES in CBC mode with the session key and a random IV (OpenSSL does this for you). Use base64 encoding, and save the output.
5. Encrypt the session password with the target's public key.
6. Sign the stuff generated so far as described previously, using SHA-1 and your private key (you will need to type in your passphrase to do this).
7. Format and send.

Receiving Secure Mail

This is how you will process incoming secure email:

1. Obtain sender's email address from mail header
2. Find sender's cert in your database, or obtain from the class website. Verify sender's cert is signed by CA; output sender name from the cert (not from the email header!)
3. Verify signature on received message using SHA-1 and public key of sender. If invalid, reject the message. Else, continue.
4. Decrypt session key with your private key (you will need to type in your passphrase for this).
5. Use session key to decrypt message; print out resulting message.

Hints for Success

- You already know many of the OpenSSL commands you will need for this project; using the command-line interface is probably the easiest way to get this task done.
- You can call the command-line interface from C or C++, or you can write your whole system in Perl, Python, or sh.
- A text-based menu system is fine, but if you want to build a GUI, feel free. As long as Martin can get it to run! ☺
- You can test your program by sending messages to yourself. Additionally, Martin will provide a test message to each of you that you can use for testing.
- The most useful advice I can give is this: don't wait until the last minute to start this project! It's more work than you think, and we have other things yet to come in the class.

Important Information

- Due Date: Tues, 11/29 in class
- What to hand in:
  - Complete source for your program in *printed* form (not on a disk or CD)
  - An example run of each of the main functions (list database, send msg, receive msg)
  - Runs on the test messages Martin sends to each of you, showing the outputs