CSCI-GA.2620: Networks and Mobile Systems **Spring 2018**

Lecture 9: 04/05/2018 - Network Security

Lecturer: Anirudh Sivaraman *Scribes: Harish Karthikeyan*

1 Background Material

- Paper was drafted and designed as fast as possible.
  - Competing Research Groups
  - Bug expected to be fixed quick. Might not be a phenomenon to investigate one year after it happened.
- Cloudflare Competition: Cloudflare felt that the bug might not be a big deal. However they were unsure and decided to host a competition.
  - Hosted a server with heartbleed vulnerability.
  - Asked hackers to retrieve the private key.
  - Someone hacked it within a couple of days.
  - Showed the seriousness of vulnerability, even though you can just read 64KB of random memory.

2 Introduction - Network Security Basics

- SSL/TLS: What is it?
  - SSL is deprecated.
  - Provides abstraction. Suppose there exists a sender $\mathcal{A}$ and receiver $\mathcal{B}$ who want to transfer data.
  - Open TCP connection. TCP only guarantees in-order delivery of data. A malicious party $\mathcal{C}$ snooping on the connection can read the bits. TCP only guarantees reliability.
  - TLS - Transport Layer Security: Takes the data and "garbles" it. It provides:
    - **Confidentiality**: No one but the parties involved can read the data being sent.
    - **Authenticity:** The sender is assured that they are talking to a particular receiver with a particular identity and vice-versa.

    - **Integrity:** An adversary cannot tamper the byte stream without the receiver realizing that it has been transfered.
    - DOES NOT PROVIDE **PRIVACY**
  - TCP is the underlying transport protocol. TLS runs on top of TCP, i.e TLS header and payload becomes the TCP payload. HTTPS runs inside TLS meaning HTTPS header and payload becomes the TLS payload.
- TLS Crypto Functions:
  - PubEncrypt(publickey, cleartext) → ciphertext
  - PubDecrypt(privatekey, ciphertext) → cleartext
    - This Public Process is too slow for data transfer. Usage of two keys makes it difficult. Instead go for Symmetric Encryption where same keys are used.
  - SymEncrypt(key, cleartext) → ciphertext
  - SymDecrypt(key, ciphertext) → cleartext
    - Need to exchange these shared keys.
    - In Connection Setup, use Public Key Encryption-Decryption to set up the key (TLS Handshake)
    - Each party generates a random number which is encrypted using Public Key Cryptography.
  - Sign(privatekey, message) → signature
  - Verify(publickey, signature) → message (Can use the same key for sign and verify. In this case, it's called a message authentication code)
  - How they come together:
    - Take Cleartext. Encrypt it to get Ciphertext.
    - Sign the Ciphertext to get Signature (Called Tag)
    - Send Ciphertext + Tag
    - Receiver verifies if the Tag is for the ciphertext received.
    - It then decrypts to get Cleartext.
  - How do you establish that it is indeed the server (before Key Agreement happens)?
    - Certificate: Entity name, public key, signature by an endorser, endorser name
    - The server, through the certificate, announces this is my name, this is my public key. The certificate is signed by this particular Certifying

Authority and this is the signature.
    * Using the public key of the CA, one can verify the validity of the certificate by running Verify on the signature and the certificate.
- Putting it All Together:
  - Server sends the certificate. Client verifies if the certificate is indeed authentic.
  - Note that certificate is public. Anyone can take this certificate and claim it to be theirs.
  - In the next step, Client sends a message encrypted with the public key mentioned on the certificate.
  - If the certificate indeed belongs to that server, they are in a position to decrypt the message with the private key known to them. And then the procedure for shared key establishment continues with the server sending its random number, encrypted with the public key of the client.
  - If the certificate was being falsely used, the impersonating server would decrypt a garbage random number, different from the one generated by the client. The key establishment can proceed but at the end of the process, the client and the server have completely different keys which makes symmetric encryption impossible.

3 Heartbleed

- OpenSSL: The main implementation of TLS. Powers about two-thirds of sites.
- The bug was in extension of TLS - DatagramTLS, which runs on top of UDP.
- Very few people use TLS for UDP. Hence, the code was rarely used and maintained.
- The bug had been present for nearly 2.5 years.
- The Bug Itself:
  - Say, the server sends a message saying "HELLO" with length 5 bytes. The receiver responds with "HELLO".
  - If the length says 50 bytes, it copies the 50 bytes of message starting at the location where "HELLO" was copied to the buffer, thereby getting 45 bytes of buffered data from a prior process.
  - It's called Buffer Over-read.
  - The Fix: If the message was smaller than the length mentioned, ignore the request.

- Affected Applications: Tor clients and relays, Bitcoin clients, chat servers, email servers, Android devices

4 The paper's scanning methodology

- Started scanning 48h after the disclosure. Used tool called ZMap which is used to scan open hosts.
- Sent heartbeat requests, using ZMap, with payloads of size 0.
- Why size 0? Ethical reasons. If size>0, are they hacking themselves?
- Sets length to be 0 and waits for response. Response is normally just padding.
- In theory, it is possible that the response is actually because of another bug, unrelated to heartbleed. However, they found that once the bug was patched, the responses did not come through. Hence, it is very likely the case that the response was symptomatic of the Heartbleed bug.
- False Negatives: Vulnerable Servers which are declared immune by their scan.
  - Caused by Bug in Tool. Servers where timeouts occurred were marked immune (Default setting was False, rather than unknown).
  - Could not merely fix and rerun the experiment - Measuring a phenomenon which was time-varying. Phenomenon is highly susceptible to have been patched or fixed.
  - Use redundant data, after the fact, to estimate the rate of false negatives.

5 Impact

- Alexa Top 100
  - They were patched even before they began scanning.
  - Al-Bassam conducted a scan 22 hours after disclosure.
  - 22 hours after disclosure his scan found Yahoo, Stack Overflow, Flickr, OkCupid, and a few more vulnerable.
  - They estimate 44 of the Alexa Top 100 were vulnerable before Heartbleed was discovered.
- Alexa Top 1M
  - They used their own scan.
  - Upper bound: All Apache/Nginx servers out of the HTTPS servers they

found (91% of 60% = 55%). Estimating how many could be vulnerable.
  - Lower bound: All vulnerable servers with TLS 1.1 and 1.2 (Came After Heartbleed) prior to Heartbleed disclosure (72.7% of 32.6% = 23.7%)
  - Summary: 24 to 55% of the top 1M was vulnerable before disclosure.
- Hosting Companies, devices such as VoIP systems and Printers, Firewalls

6 Aftermath

- Top websites did really well. Only 5% of Top 100 were vulnerable in Al-Bassam Scan.
- Among the unpatched websites during their scan, their top ranked website was at 689.
- Sharp drop when one AS patched many computers in a short period of time. This happened in internet-wide scan
- But patching plateaued after two weeks: 3% of the Alexa Top 1 Million were vulnerable 2 months after disclosure.
- Basically the more popular the web site the more security conscious it seems to be.

7 Certificate Revocation

- Need for revocation of certificates: If hacked and private key compromised, reusing certificates (which are public) would mean that any malicious adversary could masquerade as the compromised server and establish symmetric keys.
- Revoking Certificates is messy. It costs network bandwidth.
- Only 10.1% replaced certificates even after patching.
- Some replaced certificates, but did not revoke the old ones.
- Some replaced certificates, but never changed private keys.