6.033: Security - Tor
Lecture 25
Katrina LaCurts, lacurts@mit.edu

```
**********************************************************************
* Disclaimer: This is part of the security section in 6.033. Only    *
* use the information you learn in this portion of the class to      *
* secure your own systems, not to attack others.                     *
**********************************************************************
```

0. Introduction
   - We've covered how to provide confidentiality, integrity, and authenticity
   - Today we're talking about anonymity, and about what secure channels don't provide.

1. Crypto facts
   - Two ways to encrypt data
     - Symmetric-key cryptography
       - Alice and Bob share a key k, use it to encrypt and decrypt. k is secret, known only to Alice and Bob
       - Key-exchange is an issue, we typically use Diffie-Hellman key exchange (L22)
       - Generally very fast
     - Public-key cryptography
       - Alice and Bob each have their own key pair: (secret key, public key)
       - Alice's secret key is known ONLY to her; Bob's secret key is known ONLY to him. Public keys are known to everyone
       - To encrypt a message to Alice, Bob uses her public key. She decrypts it with her secret key
       - Aside: you saw public/secret keys used for signatures, where signing was done with the *secret* key and verification with the public one
         - Mathematically, signature keys have to be constructed different than encryption keys, but that's out of scope
       - Everyone can do an action using the public key, but only the owner of the corresponding secret key can do the reverse action
     - In practice: use public-key cryptography to exchange an initial secret, which is used to generate a symmetric key, which is used to encrypt the rest of the conversation
       - Happens in TLS

2. Tor
   - Goal: hide some information from a network adversary
   - Secure channel model: encrypt data
   - Adversaries still know that Alice and Bob are communicating, even in this data (bc we can't encrypt packet headers). Concerning if, e.g., Alice is communicating with a sensitive website

```
  - Tor will provide anonymity for Alice: Only she will know that
    she's communicating with a particular server. The server won't
    even know that Alice is talking to it
  - Starting idea: proxy server
    - Alice sends data to proxy server.  Header shows
      "To:Proxy|From:Alice"
    - Proxy receives packet, rewrites header, sends packet to server
      - Header: "To:Server|From:Proxy"
      - Traffic back from server goes to proxy, who sends it back to
        Alice (proxy keeps some state to do this)
    - Adversary between Alice and proxy only knows that Alice
      communicated with proxy; Adversary on network between proxy and
      server only knows that proxy communicated with server
    - Problem: Proxy knows that Alice is communicating with server
  - Better idea: A network of N proxies
    - Alice chooses three (or more) proxies.  Say P1, P2, P3
    - Traffic to server, S, goes

        A --> P1 --> P2 --> P3 --> S

    - Nodes on this path -- "circuit", in Tor parlance -- set up the
      following state.  Here, the "circuit ID" is 5.

       A -- P1 --- P2 ---- P3 --- S
       5:P1 5:A,P2 5:P1,P3 5:P2,S

    - State at each node only gives previous and next hop.  Allows
      nodes to send traffic in forward and reverse directions
    - Each node in circuit makes changes to packet header. In
reality,
      circuit ID would also be encrypted; we'll come to that
        A --- [from:A|to:P1|cir:5|XXX] --->
        P1 -- [from:P1|to:P2|cir:5|XXX] -->
        P2 -- [from:P2|to:P3|cir:5|XXX] -->
        P3 -- [from:P3|to:S|XXX] ---------> S
    - Problem: Adversary that can observe network between A and P1
      and between P3 and S will see the same packet data (even if
      it's encrypted, it didn't change), and know that A is talking
      to S.
  - Tor: Network of proxies + encryption
    - Each proxy gets its own keypair (which is why we can encrypt
the
      circuit ID too)
    - Alice encrypts here data with all three keypairs

      PK_A(circuit:K|PK_B(circuit:K|PK_C(circuit:K|XXX)))

    - Each proxy strips off a layer of encryption

      A -- [to:P1|from:A|PK_P1(circuit:K|PK_P2(circuit:K|
PK_P3(circuit:K|XXX)))] -->
       P1 -- [to:P2|from:P1|PK_P2(circuit:K|PK_P3(circuit:K|XXX))] -->
       P2 -- [to:P3|from:P2|PK_P3(circuit:K|XXX)] ------------------->
       P3 -- [to:S|from:P3|XXX] ------------------------------------->
S
```

- Layers are stripped off like onions. Tor = The Onion Router
  - Tor's encryption method is a bit different, but this is the basic idea

3. Attacks on Tor
   - Most popular attack is a traffic-correlation attack
     - If the adversary can observe traffic into the entry node (P1) and out of the exit node (P3), they will see different data in the packets, but some things will remain preserved: packet sizes (roughly), timing (roughly)
     - Can use that info to correlate traffic, infer that A is communicating with S
     - Tor does not defend against this, but does have users use only a few entry nodes, in the hopes that they are trusted
       - Their argument is that having your traffic identified some of the time is as bad as having it identified all of the time. This approach means there is a nonzero chance that it will *never* be identified, unlike a set-up where users choose a new random entry node each time.
   - A few other attacks exist, mostly due to details of Tor that we didn't cover in 6.033
   - The Tor developers are very up front about what it protects against: https://www.torproject.org/docs/faq.html.en

4. Performance
   - Tor can be slow at times, and latency is high
     - Partly inevitable: traffic bounces all around the globe, plus involves decryption at every step
     - Partly due to some implementation details that are (possibly) fixable https://svn.torproject.org/svn/projects/roadmaps/2009-03-11-performance.pdf