

## DoS Defense #2

What is currently being done with respect to DDOS?

- 1 If you can distribute a service or function, do it. Examples content distribution networks (Akami) or anycast routing (7 out of 13 DNS roots). This is mostly useful for static interactions.
- 2 Over provisioning. Not useful against a high end flood. The use of this is mostly a function of your threat model.
- 3 Mitigation boxes. Examples are Arbor and Riverhead / Cisco. These systems detect floods via anomaly detection, block suspect addresses and offload TCP handshakes to hardware.
- 4 Give up and withdraw your route announcing the DDOS target. AS level granularity. Victory for the attacker.
- 5 Address Agility: change DNS mapping for systems which are targeted based only on IP address. A perfect example of this was the Code Red worm which has hard coded into it the IP address of [www.whitehouse.gov](http://www.whitehouse.gov). Clearly, attacks that anticipate this can defeat it.
- 6 Tough Luck!

### Spoofting

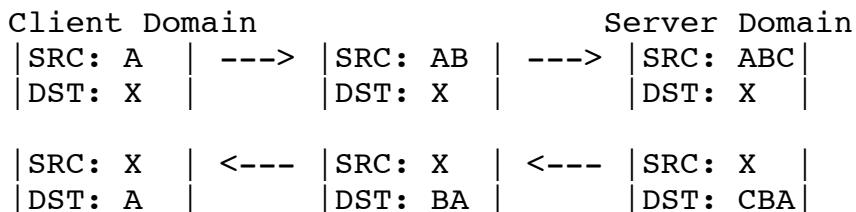
Bots do not generally spoof- why? Not clear. Two likely answers:

- No need given the number of hosts involved.
- As a response to the current set of anti-spoofing measures currently in place.

Two separate goals in preventing spoofing: (1) don't give the attacker such an easy way to evade filtering; (2) don't let them so easily induce load (by initiating zillions of requests that aren't completed).

Architectural solutions – enforce routing as a function of actual client location.

Map to domain boundaries rather than router IP's:



Hard to spoof since the path must be correct, plus spoofed packets build up additional domain elements as they travel to recipient. Spoofing is limited to within the domain.

Q: Use NAT as an example

A: Look at *Triad* as a proposal

We are looking for abstractions that have circuit-like properties to limit the ability of attackers to spoof and to influence data flow.

### SYN Cookies

In a classic 3 way handshake, all sequence numbers are a function of the local clock. This naïve implementation was replaced with random sequence number generation.

```

Client           Server
-----
  SYN SN=x  --->
    <--- SYN ACK SN=y ACK=x+1  [STATE CREATED]
  ACK=y+1  --->

```

With SYN cookies, all the state is balled up in the SYN ACK sequence number (y in the above picture). Because of this, server state is created at the end of the 3 way handshake rather than during the middle. A possible value for creating this might be:

$$y = \text{SHA-1}(\text{client\_addr}, \text{client\_port}, \text{ISN\#}, \text{server\_secret})$$

The premise here is to force a response before costly state creation. This process obviously cannot be more expensive than state creation. The current implementation needs space to encode all the state information and looks something like :

$$24 \text{ bit hash} + 5 \text{ bit timestamp} + 3 \text{ bits to remember MSS}$$

This comes at the cost of not remembering TCP options; somewhat acceptable since it only needs to be used when a server is under stress.

### Puzzles

Assume that we cannot differentiate between good and bad connections. Level the playing field by giving the client work requirements. This puzzle must be cheap to generate and validate, but expensive to solve.

Example: I give you a nonce. You take local info and MD5(nonce + info). Find and return k zero bits in the hash – requires  $2^k$  client actions.

Work factor: In SIFF there was a puzzle scheme such that at the end capability y that is l bits long. Return top l bits plus bottom l-l bits of hash of corresponding capability.

Q: what about non-solvable tests?

A: structure of test can guarantee it's solvable (e.g., the fact that MD5 covers the full range of 128-bit output values).

This is not free – e.g., disparity in CPU capability between a PDA and a high end server. Other riffs: puzzles limited by number of memory access – less disparity in memory rates.

### CAPTCHAs

Principle: set bot work factor to infinity by using a puzzle that humans can solve but computers cannot. Levels the playing field in terms of letting clients operating on behalf of individual users have a capability that bot'd hosts lack.

E.g., difficult visualization problem, such as extract from an image letters and numbers.

Problems:

- Best if cheap to generate (though if not, can sometimes pre-compute)
- “Outsourcing” attacks – botmaster relays the puzzles to humans who solve them (for example, as a means to get access to juicy web content)
- Inconvenient for users
- Arms race wrt. visualization research  
“Kitten Auth” system gets around growing ability of visualization software via soft information.

Issues:

- disabled access / assumptions about human solveability (e.g., language impediments)
- Not all robots are bad (think google)

### Pushback

Routers monitor queues. If there is sufficiently high congestion then

- 1 Construct an *aggregate description* of the load. This description could be something like “dest server and port 22” or “src net 128.32/16”.
- 2 Rate limit this aggregate.
- 3 Communicate the description *each* upstream link where it is installed.
- 4 Upstream monitors their own queue for that traffic and if it’s clogged, then recurses with possible different aggregate description.

This pushback should follow the path upstream to its source. Good traffic should be left untouched, except when it shares a path all the way back to an attack source.

This fails with diffuse attacks. There has been resistance from operational staff regarding filters crossing site boundaries; even auto-installation of filters within a single network; and skepticism about whether indeed good aggregate descriptions can be derived that have minimal collateral damage.

### Re-Feedback

Idea by Briscoe (simplified for lecture).

Key economic notion: *Externality* – cost that you impose on third parties with respect to some secondary event.

Current feelings are “Why bother? Someone else will take care of the cost.” In doing so, the public good is damaged. *Re-align* the incentive with the cost so that it is in a site’s own interest to solve problems.

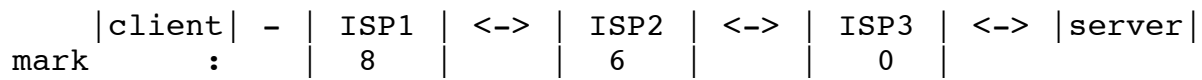
You sending a packet on a congested path imposes a cost = drops on third parties downstream.

client	-	ISP1	<->	ISP2	<->	ISP3	<->	server
congestion:		0		2		6		

packets have markings which track the level of congestion

client	-	ISP1	<->	ISP2	<->	ISP3	<->	server
mark	:	0		-2		-8		

which makes the cost = 8. *Future packets start with a cost mark = 8*



Packets that arrive at a congestion point with negative markings have a higher probability of being dropped.

At boundaries you have a pricing model – ISP1 pays ISP2 sum(markings). Also congested routers drop lowest marks first. This incents ISP1 to (1) minimize the volume of positive markings, and thus (2) police the markings set by their users. If attacker doesn't add marking to their streams, they will be discarded with high priority, alleviating the attack. If they do add markings, the ISPs will be actively rate-limiting the traffic to avoid incurring costs.

Take away: sender payment removes the externality (now senders care about congestion they impose); simple mechanism that can be negotiated between ISPs in bulk; *bi-lateral* agreements (ISPs just have to have business agreements with their peers, not with the full end-to-end path).