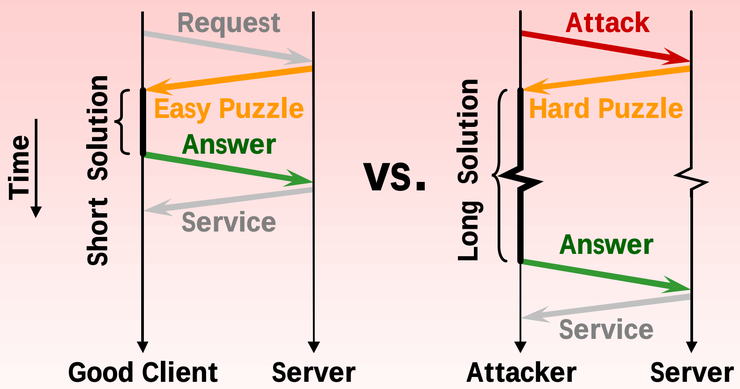
Publicly Auditable Puzzles

Ed Kaiser Advisor: Wu-chang Feng

Cryptographic Puzzles

Used to computationally throttle malicious clients.



Motivation 1: Can isolate attacker from good clients.

Motivation 2: Consume attacker resources: an attacker cannot attack multiple victims concurrently.

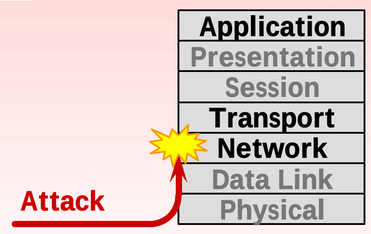
Motivation 3: Errors are more forgiving than filtering.

At Network Layer

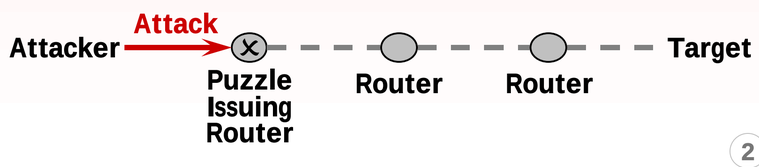
Use Network Layer cryptographic puzzles to fight:

- DoS attacks
- Worms
- Port scans
- Hacking
- Spam e-mail
- Game cheaters

Motivation 1: Upper layers at mercy of lower layer attacks.



Motivation 2: Allows attack response closer to the source.



Remaining Challenges

Limited Header Space

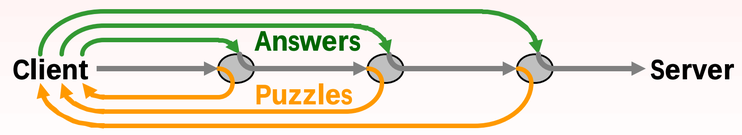
- Limited to 40 bytes in the IPv4 header
- Potentially unlimited number of issuers along path
- Thus, attaching an answer per issuer is infeasible

Parsing Complexity

- Verification must be done in the router's fast path
- Finding an answer within a list is too time-consuming

Communication Overhead

- Puzzles essentially create a three-way handshake
- Overhead grows with number of intermediate issuers



Our solution:

Publicly Auditable Puzzles

Public Auditability

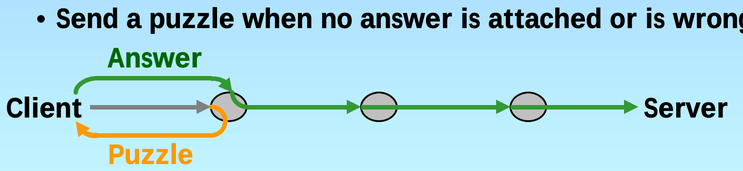
Answers that any machine can verify. This means:

Clients Attach Fewer Answers

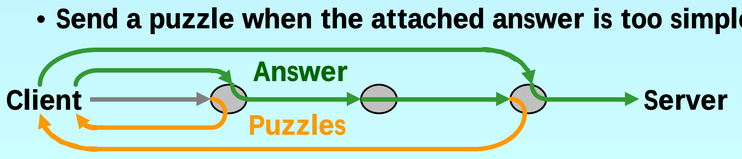
- Attach only one answer: the answer to the most difficult puzzle issued along the path to the server

Issuers Send Puzzles Less Often

- Send a puzzle when no answer is attached or is wrong



- Send a puzzle when the attached answer is too simple



Puzzle Requirements

Efficient Issuing and Verification

- Attackers must not get a new DoS opportunity
- Puzzle issuers must get computational advantage

No Solution Shortcuts

- Attackers must not be able to avoid work
- Improves control over throttling

Answers Expire

- Attackers must not be able to reuse answers
- Improves throttling responsiveness

Not Precomputable

- Attackers cannot compute answers in advance
- Avoids pre-computation attacks

Public Auditability Requirements

Public Verification

- Any machine can verify an answer
- Routers can block clients sending incorrect answers

Public Answer Expiration

- Any machine can identify expired answers
- Routers can block clients sending expired answers

Public Difficulty Estimation

- Any machine can estimate work done for an answer
- Routers can throttle traffic according to their needs: reject easy answers and issue more difficult puzzles

Definitions

Relatively One-way Function: *h*()

- Must not be reversible during the puzzle lifetime
- For example: a cryptographic hash function like SHA-1

Puzzle Difficulty: *d*

- A 32-bit integer
- 0 = no puzzle, 1 = easy puzzle, 2^{32} = hard puzzle

Flow Identifier: *f*

- Includes client port & address, server port & address, and protocol

Timestamp: *t*

- An integer (for easy timestamp comparison)

Random Nonce: *n*

String Concatenation: ||

Achieving Puzzle Requirements

Use an Efficient *h*()

- The client must find an input to *h*() that produces a specific output

Require *f* in the Input

- Full work must be done for every new connection

Require *t* in the Input

- Answers expire once *t* is older than the current time

Require *n* in the Input

- If sufficiently random this prevents pre-computation

E.g., the Hash Reversal puzzle:

$$h(f \,||\, t \,||\, n \,||\, \text{answer}) = \underbrace{00\ldots0}_{d}xx\ldots x$$

Client must find

Achieving Public Auditability

Use Only Publicly Known Inputs

- Allows any machine to compute *h*() and check validity
- *f* is publicly known since it is in the IP header
- Send *t* along with the answer
- Establish *n* publicly and securely *DIFFICULT*

Require Synchronization of *t*

- Every verifier knows when an answer expires
- A client can choose a sufficient *t* to compensate for clock jitter between itself and any issuer

Send *d* Along With the Answer

- Provides easy estimation of an answer's difficulty

Properties

Minimal State Required at Issuer

- The timestamp
- The set of recent public random nonces

Fast Puzzle Issuance

- Send the minimum acceptable *d* and the latest *n*

Fast Answer Verification

- One execution of *h*() using the client's answer
- On a 1.8 GHz Intel: 182K answers per second

Small Communication Overhead

- Only requires *t*, *d*, and the answer added to the IP header

Further Investigation

How to Efficiently Establish a Public Random Nonce?

- May be equivalent to publicly rolling an n-sided die

Future Work

Lightweight One-way Functions

- Required by hardware operating at line speed
- Investigating T-functions

Difficulty Adaptation Technique

- Response time to emerging threats
- With respect to distance from puzzle expiration
- With respect to other clients

Reputation-Based Networking

- Keep interaction history about clients
- Determine their reputability
- Use network puzzles to punish clients who are bad
- Share knowledge with other puzzle issuers

PORTLAND STATE UNIVERSITY

Project Webpage: http://syn.cs.pdx.edu/projects/puzzlenet
Funded by: intel NATIONAL SCIENCE FOUNDATION