Anonymous communications: Crowds and Tor

Basic concepts

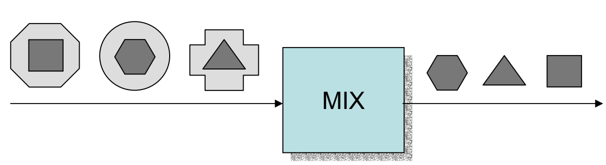
- What do we want to hide?
  - sender anonymity
    - attacker cannot determine who the sender of a particular message is
  - receiver anonymity
    - attacker cannot determine who the intended receiver of a particular message is
  - unlinkability
    - attacker may determine senders and receivers but not the associations between them (attacker doesn't know who communicates with whom)
- From whom do we want to hide this?
  - external attackers
    - local eavesdropper (sniffing on a particular link (e.g., LAN))
    - global eavesdropper (observing traffic in the whole network)
  - internal attackers
    - (colluding) compromised system elements (e.g., routers)
  - communication partner

Anonymizing proxy

- application level proxy that relays messages back and forth between a user and a service provider
- properties:
  - ensures only sender anonymity with respect to the communicating partner (service provider does not know who the real user is)
  - a local eavesdropper near the proxy and a global eavesdropper can see both the sender and the receiver information
  - proxy needs to be trusted for not leaking information (it may be coerced by law enforcement agencies!)
  - even if the communication between the user and the proxy, as well as between the proxy and the server is encrypted, a naïve implementation would have the same properties (weaknesses)

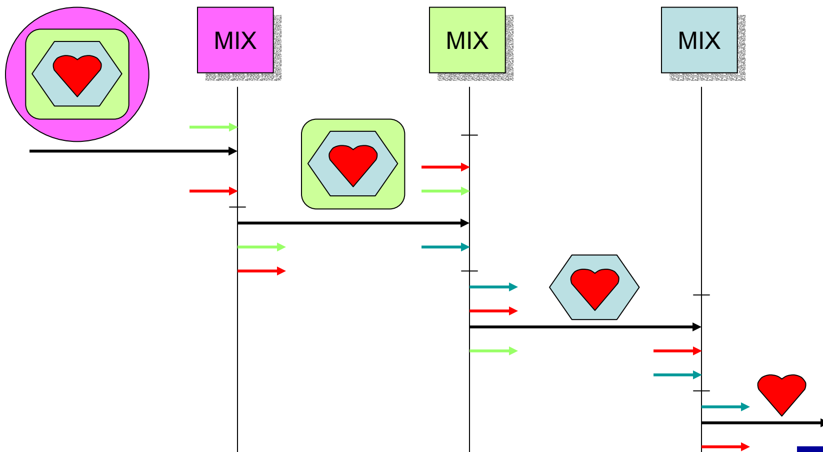
A better idea: MIX (by D. Chaum)

- a MIX is a proxy that relays messages between communicating partners such that it
  - changes encoding of messages
    - $\{ r, m \}_{K_{MIX}} \rightarrow MIX \rightarrow m$
      where m is the message, r is a random number, and K_{MIX} is the MIX's public key
  - batches incoming messages before outputting them
  - changes order of messages when outputting them
  - (may output dummy messages)
- properties:
  - sender anonymity w.r.t. communication partner
  - unlinkability w.r.t. global (and hence local) eavesdroppers
  - the MIX still needs to be trusted
  - how about reply messages ???



MIX cascade

- defense against colluding compromised MIXes
  - if a single MIX behaves correctly, unlinkability is still achieved



Return addresses

- a return address is an iteratively encrypted message, where layer i is encrypted with the public key of the i-th MIX on the return path and contains
  - the identifier of the next MIX on the return path
  - a secret key to be used for encrypting the content of the reply
  - layer i-1 of the return address
- the user pre-determines the return path and pre-computes the return address, which is sent to the receiver in the body of the (forward) message
- the return address is attached to the reply message
- each MIX on the return path decodes the next layer of the return address, encrypts the reply with the secret key found, and forwards the reply to the next MIX on the return path
- the user decrypts the reply with the secret keys iteratively
- example:
  - return address attached to the reply M:
    MIX3, $\{MIX2, K3, \{MIX1, K2, \{SRC, K1, -\}_{Kmix1}\}_{Kmix2}\}_{Kmix3}$
  - MIX3 does the following:
    - decodes the return address and sees that the next MIX is MIX2
    - encrypts M with K3 (result is M')
    - sends M' with MIX2, $\{MIX1, K2, \{SRC, K1, -\}_{Kmix1}\}_{Kmix2}$, PADDING attached

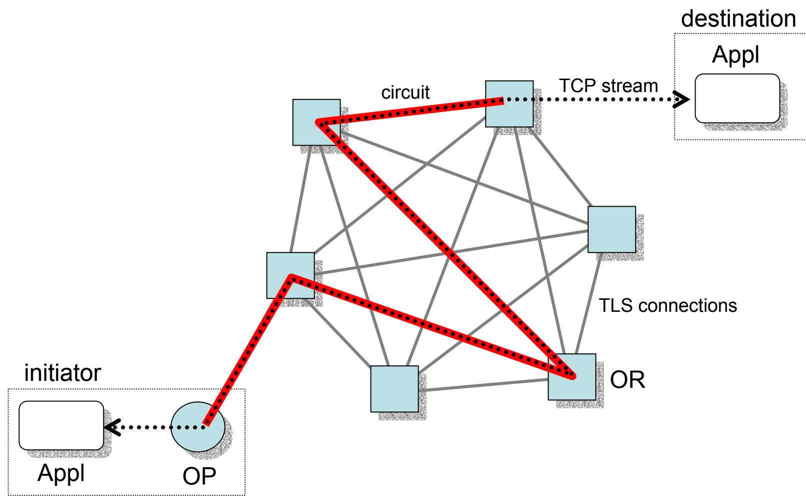
Tor

- low-latency (real-time) mix-based anonymous communication service
- tries to provide unlinkability of senders and receivers against an adversary who can
  - observe some fraction of the network traffic
  - generate, modify, delete, or delay traffic
  - compromise some fraction of the participating routers
- does not try to provide unlinkability with respect to a global observer
  - end-to-end traffic confirmation attacks are possible

The Tor network

- the Tor network is an overlay network consisting of onion routers (OR)
- ORs are user-level processes without special privileges operated by volunteers in the Internet
- each OR maintains a TLS connection to all other ORs
  - a few special directory servers keep track of the ORs in the network
  - each OR has a descriptor (keys, address, bandwidth, exit policy, etc.)
- each user runs an onion proxy (OP) locally
- OPs establish virtual circuits across the Tor network, and they multiplex TCP streams coming from applications over those virtual circuits
- the last OR in a circuit connects to the requested destination and behaves as if it was the originator of the stream

The Tor network illustrated



Cells

- data within the Tor network are carried in fixed sized cells (512 bytes)
- cell types
  - control cells
    - used to manage (set up and destroy) circuits

2	1	509
CircID	Cmd	DATA

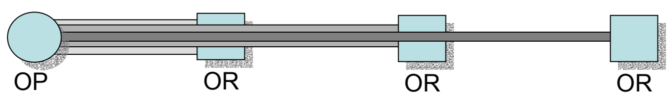
  - relay cells
    - used to manage (extend and truncate) circuits, to manage (open and close) streams, and to carry end-to-end stream data

2	1	2	6	2	1	498
CircID	Rly	StreamID	Digest	Length	Cmd	DATA

Setting up a circuit

- circuits are shared by multiple TCP streams
- they are established in the background
  - OPs can recover from failed circuit creation attempts without harming user experience
- OPs rotate to a new circuit once a minute

- a circuit is established incrementally, in a “telescoping” manner
  - a circuit is established to the first OR on the selected path by setting up a shared key between the OP and that OR
  - this circuit is extended to the next OR by setting up a shared key with that OR; this already uses the circuit established in the previous step
  - and so on…



Establishment of shared keys

- Diffie-Hellman based protocol:

  OP → OR: $E_{PK\_OR}(g^x)$
  OR → OP: g^y | H(K | “handshake”)

  where K is the established key g^{xy}

- properties:
  - unilateral entity authentication (OP knows that it is talking to OR, but not vice versa)
  - unilateral key authentication (OP knows that only OR knows the key)
  - key freshness (due to the fresh DH contributions of the parties)
  - perfect forward secrecy
    - (assuming that OR deletes the shared key K when it is no longer used)
    - if OR is later compromised, it cannot be used to decrypt old (recorded) traffic

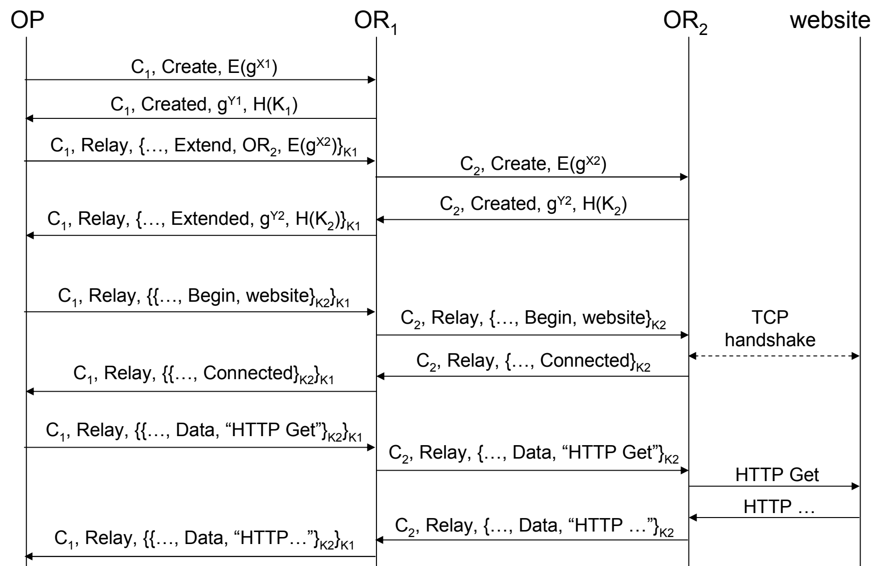
Relaying cells on circuits

- application data is sent in relay cells
- OP encrypts the cell iteratively with all the keys that it shares with the ORs on the path (onion-like layered encryption)
- each OR peals off one layer of encryption
- last OR sends cleartext data to the destination
- on the way back, each OR encrypts the cell (adds one layer), and the OP removes all encryptions
- AES is used in CTR mode (stream cipher) → encryption does not change the length

Opening and closing streams

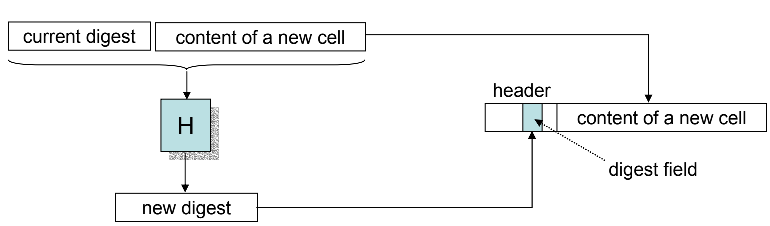
- opening:
  - the TCP connection request from the application is re-directed to the local OP (via SOCKS)
  - OP chooses an open circuit (the newest one), and an appropriate OR to be the exit node (usually the last OR, but maybe another due to exit policy conflicts)
  - OP opens the stream by sending a "relay begin" cell to the exit OR
  - the exit OR connects to the given destination host, and responds with a "relay connected" cell
  - the OP informs the application (via SOCKS) that it is now ready to accept the TCP stream
  - OP receives the TCP stream, packages it into "relay data" cells, and sends those cells through the circuit
- closing:
  - OP or exit OR sends a "relay end" cell to the other party, which responds with its own "relay end" cell

Operation illustrated



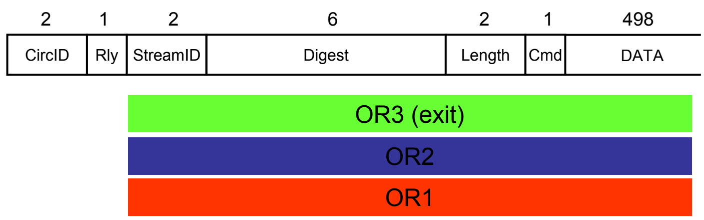
Integrity checking

- ORs are connected through TLS connections → external adversaries cannot modify or forge cells
- attacks from internal adversaries (compromised ORs) are detected by checking the digest field in the cells
- digest is verified by the exit OR
  - in fact, correct digest determines who is the exit OR (leaky-pipe circuits)
- when OP establishes a shared key with an OR in a circuit, they both initialize a SHA-1 digest with a key derived from the shared key
- each time one party creates a relay cell (intended to the other party), it adds the content of the new cell to the digest, and puts the first few bytes of the resulting digest value into the digest field of the cell



Leaky-pipe mechanism

- any OR in the circuit can be chosen as the exit point of a stream
- digest field is computed with the key shared with the chosen exit OR
- layered encryption scrambles the digest field (too)
- when the cell arrives to the chosen exit OR, all layers of encryption are pealed off, and the digest verifies correctly
- this signals to the OR that it is the exit point



Exit policies

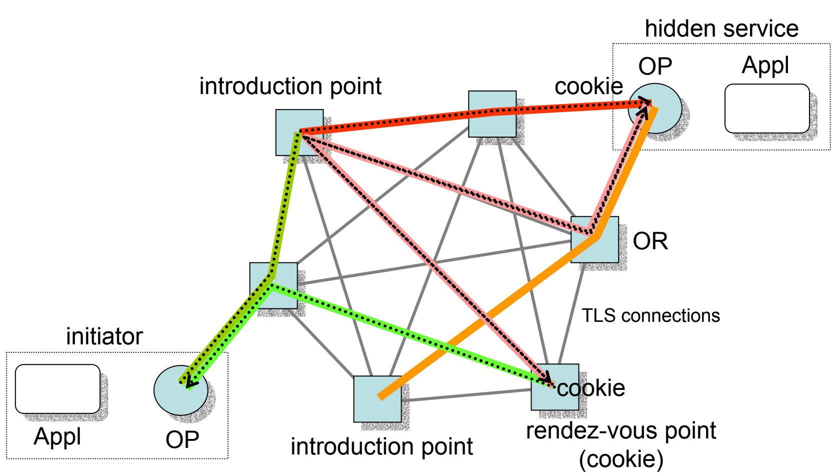
- hackers can launch their attacks via the Tor network
  - no easy way to identify the real origin of the attacks
  - exit nodes can be accused
  - this can discourage volunteers to participate in the Tor network
  - fewer ORs means lower level of anonymity

- each OR has an exit policy
  - specifies to which external addresses and ports the node will connect
  - examples:
    - open exit – such nodes will connect anywhere
    - middleman – such nodes only relay traffic to other Tor nodes
    - private exit – only connect to the local host or network
    - restricted exit – prevent access to certain abuse-prone addresses and services (e.g., SMTP)

Rendez-vous points and hidden services

- renedez-vous point enable responder anonymity (one can offer a TCP-based service without revealing his IP address to the world)
- the server's OP chooses some ORs as introduction points and advertises them on an anonymous lookup service
- the OP builds a circuit to each of these introduction points
- the client learns about the service out-of-band
- the client's OP chooses an OR as the rendez-vous point
- the OP builds a circuit to the rendez-vous point, and gives it a random rendez-vous cookie
- the client OP builds a circuit to one of the introduction points, opens an anonymous stream to the server, and sends the cookie
- the server OP builds a circuit to the given rendez-vous point and sends the cookie
- the rendez-vous point verifies the cookie and connects the client circuit to the server circuit
- the client establishes an anonymous stream through the circuit and uses the anonymous service

Rendez-vous point illustrated



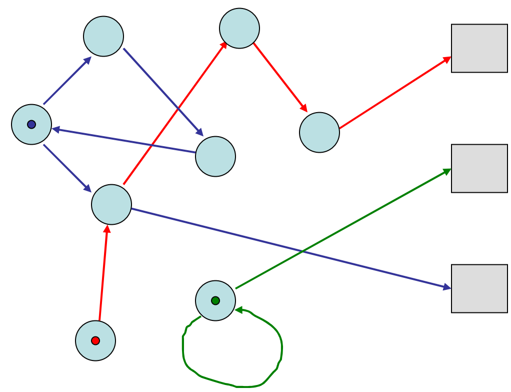
Some attacks

- end-to-end timing (or size) correlation
  - an attacker watching traffic patterns at the initiator and the responder will be able to confirm the correspondence with high probability
  - it was not the goal of Tor to prevent this
- website fingerprinting
  - an attacker can build up a database containing files sizes and access patterns for targeted websites
  - he can later confirm a user's connection to a given website by observing the traffic at the user's side and consulting the database
  - in case of Tor, granularity of fingerprinting is limited by the cell size
- tagging attacks
  - an attacker can "tag" a cell by altering it, and observing where the garbled content comes out of the network
  - integrity protection of cells prevent this

Crowds

- a crowd is a collection of users formed dynamically
- each user runs a process called *jondo* on his computer
- when the jondo is started it contacts a server called *blender* to request admittance to the crowd
- if admitted, the blender reports the current membership of the crowd and sends information necessary to join the crowd (keys)
- the user sets his browser to use his jondo as a web proxy
- when the jondo receives the first request from the browser, it initiates the establishment of a random path of jondos in the crowd
  - the jondo picks a jondo (possibly itself) in the crowd at random, and forwards the request to it (after sanitizing it)
  - when this jondo receives the request it forwards it with probability p_f (to a randomly selected jondo again) or submits the request to the destination server with probability $1 - p_f$
- subsequent requests follow the same path
- the server replies traverse the same path (in reverse direction)
- communication between jondos is encrypted

Examples



Degrees of anonymity

absolute privacy	beyond suspicion	probable innocence	possible innocence	exposed	provably exposed

- **beyond suspicion:**
  - **attacker can see evidence of a sent message, but ...**
  - **the sender appears no more likely to be the originator than any other potential sender in the system**
- **probable innocence:**
  - **the sender may be more likely the originator than any other potential sender, but**
  - **the sender appears no more likely to be the originator than to not be the originator**
- **possible innocence:**
  - **the sender appears more likely to be the originator than to not be the originator, but**
  - **there's still a non-trivial probability that the originator is someone else**

Types of attackers

- local eavesdropper
  - can observe communication to and from the users computer
- end server
  - the web server to which the transaction is directed
- collaborating crowd members
  - crowd members that can pool their information and deviate from the protocol

Security analysis – local eavesdropper

- a local eavesdropper can see that the user originated a request
  - it can observe an outgoing message without an incoming one
  - sender is exposed
- however, he typically cannot see the target of the request
  - requests are encrypted unless they are submitted to the target server
  - if request is encrypted, each end-server appears for the attacker equally likely to be the target of the request → beyond suspicion anonymity
  - if the user's own jondo submits the request, then the target is exposed; the probability of this is 1/n where n is the size of the crowd (see next slide)
  - Pr{ receiver / beyond suspicion } =
    Pr{ local eavesdropper sees only encrypted request } = 1 – 1/n
    → 1 as n → infinity

Security analysis – local eavesdropper

- α – originator of request
- ω – jondo that submits request to end server

Pr{ω = x | α = x} = ?

$Pr\{ x \rightarrow x \rightarrow SRV \} = (1/n)(1-p_f)$
$Pr\{ x \rightarrow i \rightarrow x \rightarrow SRV \} = \Sigma_i (1/n)p_f(1/n)(1-p_f) = (1/n)p_f(1-p_f)$
$Pr\{ x \rightarrow i \rightarrow j \rightarrow x \rightarrow SRV \} = \Sigma_i\Sigma_j (1/n)p_f(1/n)p_f(1/n)(1-p_f) = (1/n)p_f^2(1-p_f)$
...

$Pr\{\omega = x \mid \alpha = x\} =$
$Pr\{ x \rightarrow * \rightarrow x \rightarrow SRV \} =$
$(1/n)(1-p_f)\Sigma_{k=0}^{\infty} p_f^k = (1/n)(1-p_f)(1/(1-p_f)) = 1/n$

Security analysis – end server

- end-server is the target of the request
  – receiver anonymity is not possible

- anonymity for the originator is strong
  – user's jondo always forwards the request to a random member of the crowd (~ hides user identity with a one-time pad)
  → the end-server receives the request from any crowd member with equal probability
  – from the end-server perspective, each user is equally likely to be the originator → beyond suspicion sender anonymity is guaranteed

Security analysis – end server

$Pr\{ \alpha = x \mid \omega = y\} = ?$

$Pr\{ \alpha = x, \omega = y \} / Pr\{ \omega = y \} =$
$Pr\{ \omega = y \mid \alpha = x \}Pr\{ \alpha = x \} / \Sigma_z Pr\{ \omega = y \mid \alpha = z \}Pr\{ \alpha = z \} =$
$// Pr\{ \alpha = z \} = 1/n$
$Pr\{ \omega = y \mid \alpha = x \} / \Sigma_z Pr\{ \omega = y \mid \alpha = z \} =$
$Pr\{ x \rightarrow * \rightarrow y \} / n Pr\{ z \rightarrow * \rightarrow y \} =$
$(1/n) / n(1/n) = 1/n$

if user's jondo could submit the request to the server immediately:

$Pr\{ \omega = y \mid \alpha = x \} = ?$
if $y = x$, then $Pr\{ x \rightarrow SRV \} + Pr\{ x \rightarrow * \rightarrow x \rightarrow SRV \} = (1-p_f) + p_f(1/n)$
if $y \neq x$, then $Pr\{ x \rightarrow * \rightarrow y \rightarrow SRV \} = p_f(1/n)$

$Pr\{ \alpha = x \mid \omega = y\} = Pr\{ \omega = y \mid \alpha = x \} / \Sigma_z Pr\{ \omega = y \mid \alpha = z \} =$
$Pr\{ \omega = y \mid \alpha = x \} =$
if $x = y$, then $(1-p_f) + p_f(1/n)$
otherwise, $p_f(1/n)$

→ sender is more likely to be the jondo from which the request was received, than any other jondo !

Security analysis – collaborating jondos

- ω_C – jondo from which first collaborator on the path receives the request

$Pr\{ \omega_C = y \mid \alpha = x \} =$
if $y = x$, then $Pr\{ x \rightarrow C \} + Pr\{ x \rightarrow * \rightarrow x \rightarrow C \}$
if $y \neq x$, then $Pr\{ x \rightarrow * \rightarrow y \rightarrow C \}$

→ $Pr\{ \alpha = x \mid \omega_C = y \} < Pr\{ \alpha = y \mid \omega_C = y \}$

Security analysis – collaborating jondos

- notation
  - H_i – the event that the first collaborator on the path is in the i-th position
  - $H_{i+} = H_i \vee H_{i+1} \vee H_{i+2} \vee \ldots$
  - I – the event that the first collaborator on the path is immediately preceded on the path by the initiator
- definition
  - the path initiator has probable innocence if $P(I \mid H_{1+}) \leq 1/2$
- theorem
  - if $n \geq (c + 1)p_f / (p_f - 1/2)$, then the path initiator has probable innocence against c collaborators
- in addition, Pr{ absolute privacy } → 1 as n → infinity both for sender and receiver anonymity

Security analysis – collaborating jondos

- observation: I implies H_{1+}
- $Pr\{ I \mid H_{1+} \} = Pr\{ I, H_{1+}\} / Pr\{H_{1+}\} = Pr\{ I \} / Pr\{ H_{1+} \}$
- $Pr\{ H_k \} = [\, p_f (n-c)/n \,]^{k-1} (c/n)$
- $Pr\{ H_{1+} \} = \Sigma_{k=1}^{\infty} Pr\{ H_k \} = (c/n)(1 - p_f (n-c)/n)^{-1} = c / (n - (n-c)p_f)$
- $Pr\{ I \} = Pr\{ x \rightarrow C \} + Pr\{ x \rightarrow * \rightarrow x \rightarrow C \} =$

  $= (c/n) + [\, \Sigma_{k=0}^{\infty} (p_f (n-c)/n)^k \,] (1/n)\, p_f (c/n) =$

  $= (c/n) + (c/n)\, p_f / (n - (n-c)p_f)$

→ $Pr\{ I \mid H_{1+} \} = (n - p_f(n-c-1))/n \leq ½$

→ $n \geq (c + 1)p_f / (p_f - 1/2)$

Overview of security offered by Crowds

attacker	sender anonymity	receiver anonymity
local eavesdropper	exposed	Pr{ beyond suspicion } → 1
c collaborating crowd members	probable innocence Pr{ absolute privacy } → 1	Pr{ absolute privacy } → 1
end server	beyond suspicion	N/A

Timing attacks

- HTML pages can include URLs that are automatically fetched by the browser (e.g., images)
- first collaborating jondo on the path can measure the time between seeing a page and seeing a subsequent automatic request
- if the duration is short, then the predecessor on the route is likely to be the initiator
- solution:
  - last jondo on the path parses HTML pages and requests the URLs that the browser would request automatically
  - user's jondo on the path returns HTML page, doesn't forward automatic requests, rather waits for the last jondo to supply the results