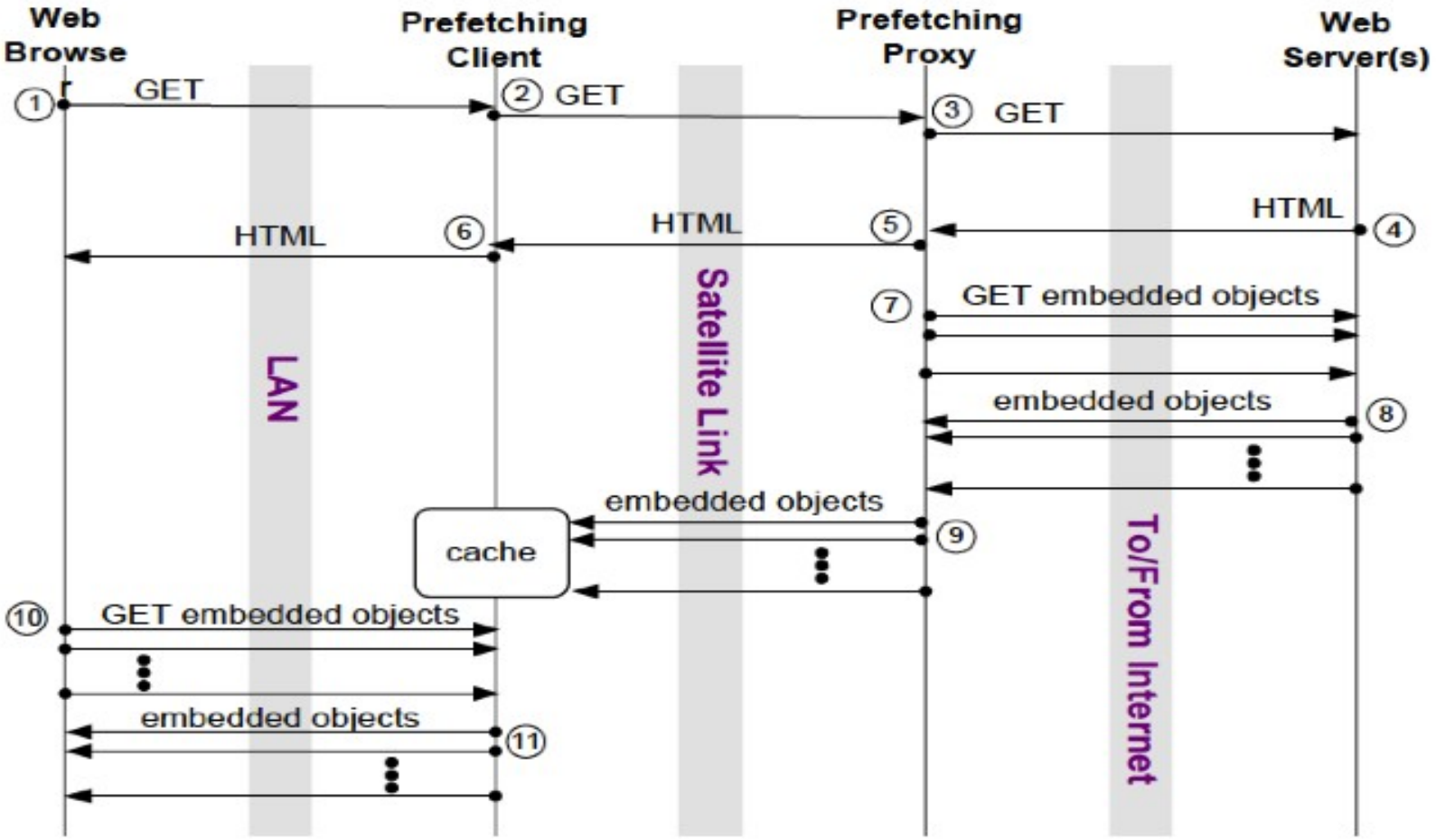
Prefetching Proxy

Brian Jones
CS 259
3/9/2011

Brief Overview

- HughesNet wants to decrease noticeable latency to the end user.
- They hope to accomplish this by prefetching embedded content in webpages.

System Model



Security Properties

- Want to prevent an adversary from monitoring traffic through the prefetching proxy.
  - Nonsensitive data should be shared between headless browsers, but no browser should know the intended user of the data.
- Malware downloaded to prefetching proxy should be prevented from being shared and should not reach the end users.
- HTTPS transfers should be trusted to pass through the prefetching proxy securely.

Alloy Modeling

- Perfect for describing an abstract system.
- Modeled basic system as proposed.
  - Included shared cache
- Assumptions:
  - Users and Prefetching Clients one-to-one
  - One headless browser per user
  - Shared cache holds HTML responses and embedded objects
  - Two different GETs with same content go to same website.
  - Two different GETs with different content go to different websites.

Alloy Code

```
sig PrefetchingClient extends Entity{
   user : one User,
   cache : one Cache
}
{
   cache not in PrefetchingProxy.sharedCache
}

one sig PrefetchingProxy extends Entity{
   sharedCache : one Cache,
   headlessBrowser : set HeadlessBrowser
}
{
   all hb : HeadlessBrowser { hb in headlessBrowser }
   sharedCache not in PrefetchingClient.cache
   sharedCache.data = HTMLResponse.content + embeddedObject.content
}

sig HeadlessBrowser extends Entity{
   hbCache : one Cache
}
{
   hbCache = PrefetchingProxy.sharedCache
}
```

Alloy Code

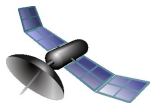
```
abstract sig Message{
   to : Entity,
   from : Entity,
   content : Data,
   u : one User,
   pc : one PrefetchingClient,
   w : one WebServer
}
{
   from != to
   u.pClient = pc
   pc.user = u
   to in User => { to = u }
   from in User => { from = u }
   to in PrefetchingClient => { to = pc }
   from in PrefetchingClient => { from = pc }
   to in WebServer => { to = w }
   from in WebServer => { from = w }
}
```

```
sig embeddedObject extends Message{}
{
   from not in PrefetchingClient + SatelliteLink + PrefetchingProxy +
      PrefetchingProxy.headlessBrowser + InternetLink + WebServer
      iff to = none
   from in PrefetchingClient iff to = u
   from in SatelliteLink iff to = pc
   from in PrefetchingProxy iff to = SatelliteLink
   from in PrefetchingProxy.headlessBrowser iff to = PrefetchingProxy
   from in InternetLink iff to in PrefetchingProxy.headlessBrowser
   from in WebServer iff to = InternetLink

   w not in AttackerWebServer => {
      all a : AttackerWebServer {
         content != a.malware
      }
   } else {
      some a : AttackerWebServer {
         content = a.malware
      }
   }

   content in (pc.cache).data
}

sig GETembedded extends Message{}
{
   from not in User + PrefetchingProxy.headlessBrowser + InternetLink iff to = none
   from in User iff to = pc
   from in PrefetchingProxy.headlessBrowser iff to = InternetLink
   from in InternetLink iff to = w

   content in (pc.cache).data
}
```

Threat Analysis

- Analysis via Alloy proved to be very difficult given the large number of applicable protocols.
- No single protocol was proposed by Hughes for this system.
- System is meant to aid current protocols.
- Analysis primarily by inspection.
- Threat model confirmed in Alloy.

Ideal System



User / PClient Cache

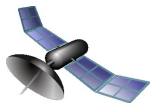
PProxy / Shared Cache

WebServer

HTTP Request → Request Cached → Request Received

Response Received ← Response Cached ← HTTP Response w/ Data

Eavesdropping Attack



User / PClient Cache

PProxy / Shared Cache

WebServer

HTTP Request

Request Cached

Request Received

Response Received

Response Cached

HTTP Response w/ Data

Attacker

HTTP Request

Request found in Cache

Response Received

HTTP Response w/ Data

Eavesdropping Attack

- Attacker can sift through the cache by brute force.
- Can prevent the attacker's requests from polluting the cache by setting **cache-control = only-if-cached**.

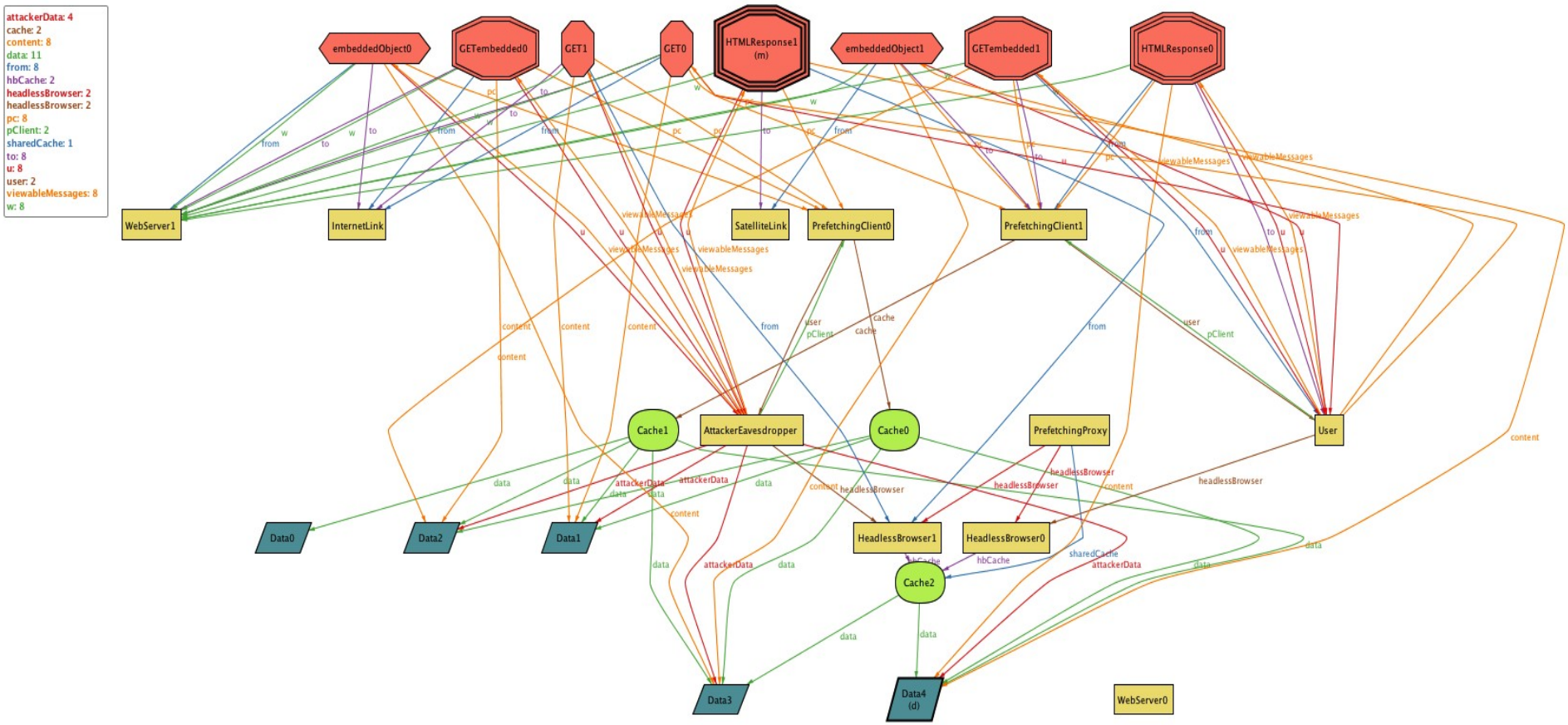
Eavesdropping Attack Prevention

- Modify selected HTTP header fields for requests and responses at the prefetching proxy.
- Treat all HTTP requests the same and all HTTP responses the same.
- Attacker won't know if data from cache was polluted by itself.

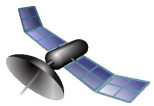
Eavesdropping Attack

```
pred eavesdroppingAttack {
   one attacker : AttackerEavesdropper, user : User {
      one g1, g2 : GET {
         g1.content = g2.content and
         g1.u = attacker and
         g2.u = user and
         g1.w = g2.w => {
            attacker.attackerData != none
            attacker.attackerData = ((user.pClient).cache).data
         } else {
            attacker.attackerData = none
         }
      }
   }
}

run eavesdroppingAttack
for 11 Entity, exactly 2 User, exactly 1 AttackerEavesdropper,
exactly 2 PrefetchingClient, exactly 2 HeadlessBrowser, exactly 2 WebServer,
exactly 3 Cache, 8 Data,
exactly 8 Message, 2 GETembedded, 2 embeddedObject, 2 GET, 2 HTMLResponse
```

Eavesdropping Attack



Ideal System



User / PClient Cache

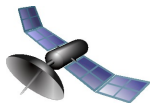
PProxy / Shared Cache

WebServer

HTTP Request → Request Cached → Request Received

Response Received ← Response Cached ← HTTP Response w/ Data

Malware Infection



User / PClient Cache

PProxy / Shared Cache

Attacker

HTTP Request → Request Cached → Request Received

User Infected ← Malware Infection ← HTTP Response w/ Malware

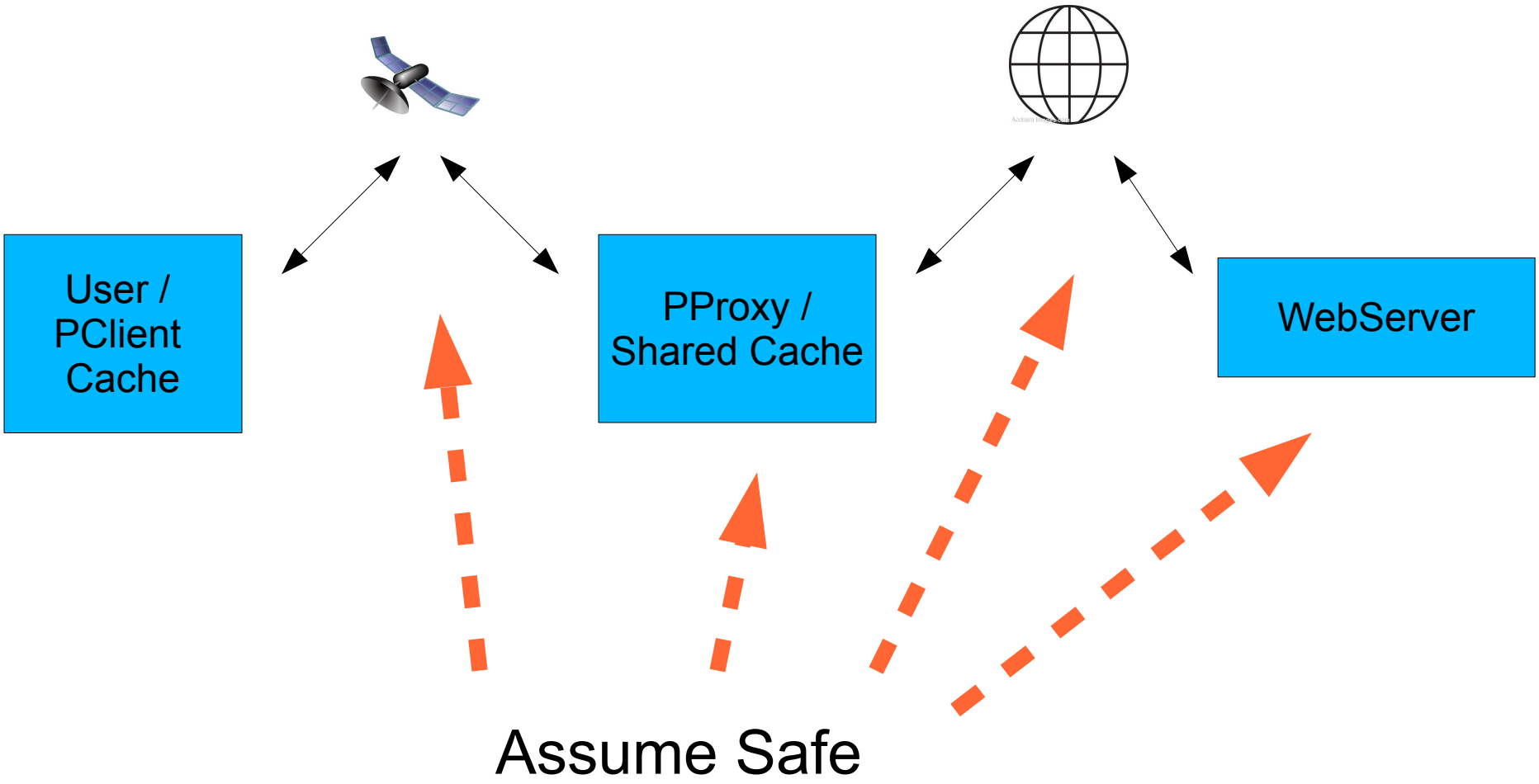
Malware Prevention

- Install anti-malware software on the prefetching proxy.
- Or, don't share cache at prefetching proxy.

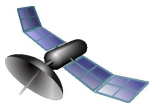
HTTPS

- Need SSL Bridging.
- End user needs to import the certificate of the SSL bridging server (prefetching proxy).
- Prefetching Proxy still vulnerable to a SSL_Strip attack.

HTTPS SSL_Strip



HTTPS SSL_Strip



User / PClient Cache

Attacker

PProxy / Shared Cache

HTTPS Request → HTTP Request → Saves in Cache

HTTP Response ← HTTP Response

HTTP Request → HTTPS Request → Saves in Cache

HTTPS Response ← HTTPS Response ← HTTPS Response

HTTPS SSL_Strip Prevention

- Prefetching Proxy can keep track of HTTP messages and HTTPS messages.
- If it sees a HTTP request before a HTTPS request for the same domain name, don't forward HTTPS request.

Conclusion

- Alloy modeling difficult without specific protocol.
- Eavesdropping threat threatens confidentiality.
- Malware threat threatens integrity due to vast infection.
- HTTPS SSL_Strip threatens privacy.