USING STRONGLY TYPED NETWORKING TO ARCHITECT FOR TUSSLE

Chitra Muthukrishnan (UW-Madison)
Vern Paxson (ICSI/UC Berkeley)
Mark Allman (ICSI)
Aditya Akella (UW-Madison)

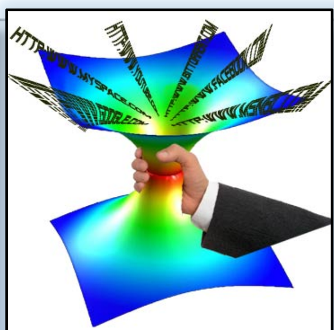
Tussles

Cogent-Telia Peering Dispute Widely Felt

March 18th, 2008 : Rich Miller

The ongoing peering dispute between Cogent and Telia has left man

Metallica Rips Napster



Internet architecture must accommodate them (Clark et al, Sigcomm'02)

Design for variation in outcome

Allow tussles to play out within the design

Providers v. Users

Network elements:
- discriminate/control traffic
  - Attack resistance, monitoring, AUPs, competition

Users desire free access
- Some try to evade by cloaking or encrypting traffic

Arm race… Evasion schemes ⟷ Layers of messy protocols/controls

… Ending in: **Heavy handed policies** (“block all encrypted traffic”); **Opaque, difficult-to-debug** network

Return to a completely neutral network?

Maybe not…

- Any definition unlikely to be universally binding
- Private networks will not be neutral
- Traffic distinction also useful to improve performance

Tussle will not go away.

Non-technical net neutrality approaches not viable.

How to accommodate this tussle space?

Thought exercise: design guidelines and core functions to allow tussle to play out

Ignore practical issues, e.g., overhead and efficiency

Architecting for Tussle

Design guidelines

Core functions

Extensions and conclusion

Guideline 1.

Transparency

- Network elements (NEs): know the exact semantics of traffic they carry
- Users: know network policies (e.g., restrictions, transformations)

Guideline 2.

Choice

- Users: choose which msg. parts can be inspected, what must remain private; able to switch paths
- Providers: express level of visibility desired

Core functions

Strong typing
- Include semantic information within communication

Dialog
- Negotiate and agree upon communication rules

Selective encryption
- Choose paths, enforce access controls

Verification
- Communication proceeds as agreed

Strong Typing

Prog. languages: Every group of bits has semantic context, enforceable at run-time

Networking: All messages carry *type* information

Governs how receiver will interpret msg. components

(1) **Extensive**: Atomic values (IP address, status codes) to aggregate objects (MIME, HTTP structs)

(2) **Exhaustive**: Every single part of communication is typed

Typing provides transparency

Extends Blumenthal & Clark's labeling approach

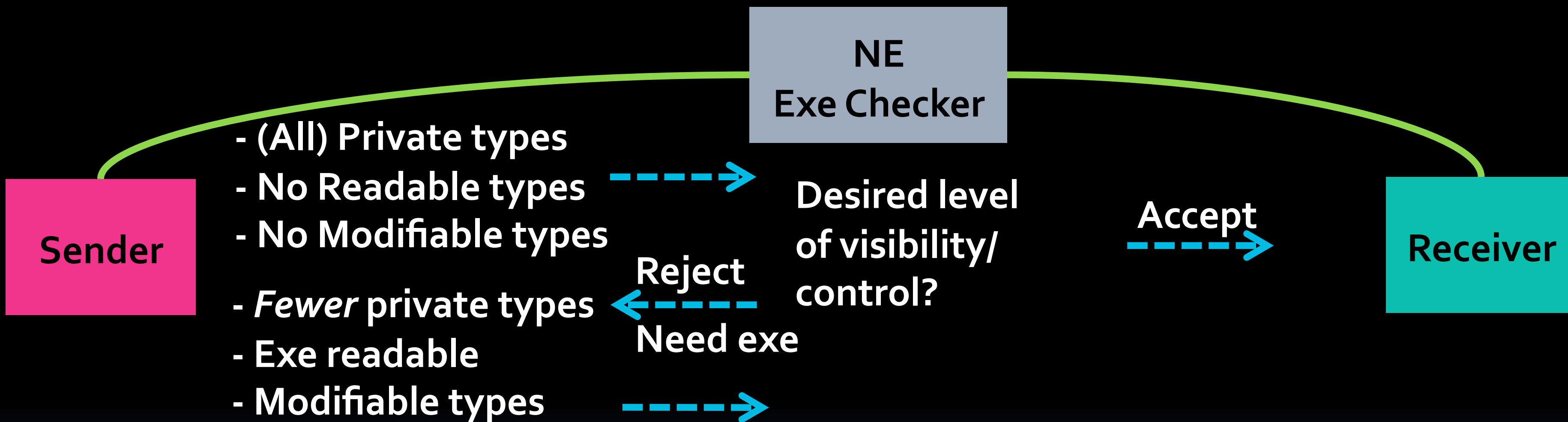
Semantic-focus; enforceable

One possibility: use XML.
Tag content in a hierarchical fashion

```
<http>
   <reply>
         <status> 200 OK </status>
         <content>
               <exe>
                   <data> [exe data] </data>
               </exe>
         </content>
   </reply>
</http>
```

Dialog

Typing paves way for *dialog* to negotiate communication properties



Pre-connection or in-band

Sender may choose an alternate path.
Fail if no such path → *reason in full view*

Network has upper hand, but visibility limits collateral damage

Selective Encryption

Helps enforce access controls

<http, EncryptedData>

<KeyName> K1 </KeyName> → Sender exchanged K1 with exe checker NE (not discussed)

<CipherData, reply>
<EncryptedData>
<KeyName> Krcvr </KeyName>
<CipherData>

→ Encrypted using K1

<status> 200 OK </status>
<set-cookie> … </set-cookie>
<MessageIntegrity> … </MessageIntegrity>

→ Encrypted using Krcvr

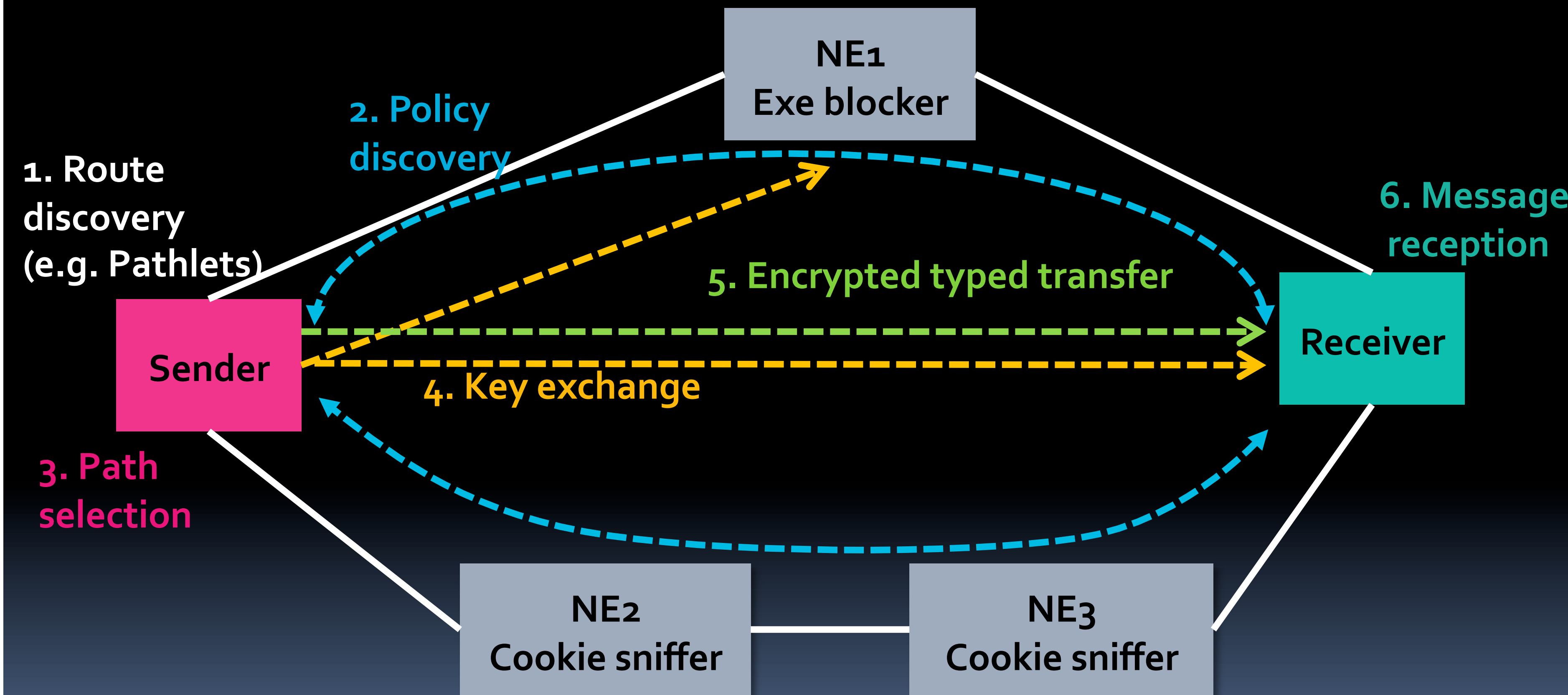
</CipherData, EncryptedData>
<content, exe>
[exe data]
<MessageIntegrity> … </MessageIntegrity>
</exe, content>
</reply, CipherData>

→ Encrypted using K1
Cksum prevents illegal mods

Integrity key: If NE has permission, it can change exe. Checksum updated with K1

</EncryptedData, http>

Core Functions in Action

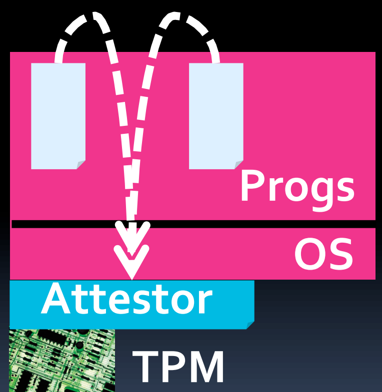


Verification

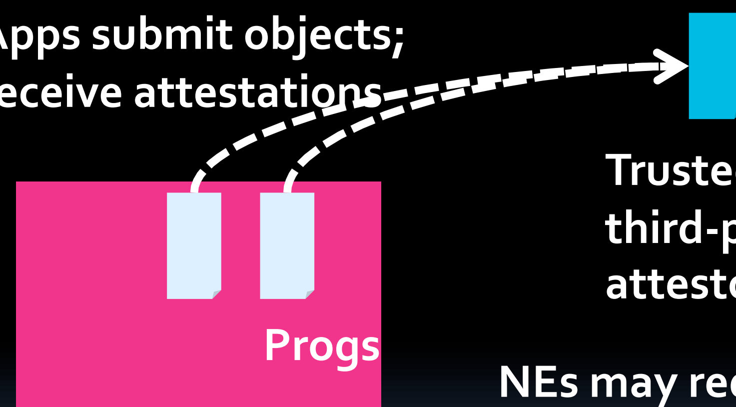
Rely on trusted receivers to enforce types: type assertions

Inherent validation in other cases using attesters

Apps submit objects;
receive attestations



Apps submit objects;
receive attestations



NEs may require TTP attestations as part of policy

Not effective against steganography

This may be the best outcome we can hope for

Extensions

Routing changes: no context at new NEs

- Type safe handoffs using periodic certificates

Transport properties: transfer rate, #connections etc.

- Not exactly “type” information, but can be fit in

Cooperative scenarios: NEs can serve better if they know precise traffic semantics

- Transcoders for mobile phones, application-specific caches/compression engines

A thought exercise on architecting for
  providers v. users tussle

Transparency and **user choice** key guidelines

Strong typing is the primary building block
  Dialog, selective encryption, verification

Many practical hurdles (crypto inefficiency, key management, typing overhead, …)

Practical Issues

- Routing changes: re-establishing transfer?
  - Type safe handoffs?
- Overhead of typing: processing/network (mobile devices?)
- Crypto: Key management, encryption/decryption overhead

Key Establishment

Sender shares requisite keys with NEs and receiver

Keys can be applied to all flows to receiver, assuming route stationarity

<Key exchange>
    <Key name> $PubK_{NE}$ </Key name>
    <Cipher data>
        [K1 encrypted with $PubK_{NE}$]
    </Cipher data>
    <Carried key name> K1 </Carried key name>
    <Integrity checksum> .. <Integrity checksum>
</Key exchange>

Impending Mess: Mis-decisions, Entanglement, Brittleness

- Networks: crude, hidden mechanisms to identify/ control traffic
- Users: can't resolve why some activities fail ("*why is my connection slow?*")

Arms race: evasion schemes

↑ ↓

layers of messy protocols

- Heavy-handed policies: "Block all encrypted traffic"
- Network becomes more opaque, difficult to debug

Backup: related work

- Treating middleboxes as first class entities
- Traversal, transparency and negotiation in isolation and for specific purposes
  - Our work combines these themes
  - We look at it from the perspective of a tussle
- Labeling
  - Typing is an instance of labeling
    - Semantically clear
    - Improves enforceability