

## **Protecting Domain Name Systems (DNS)**

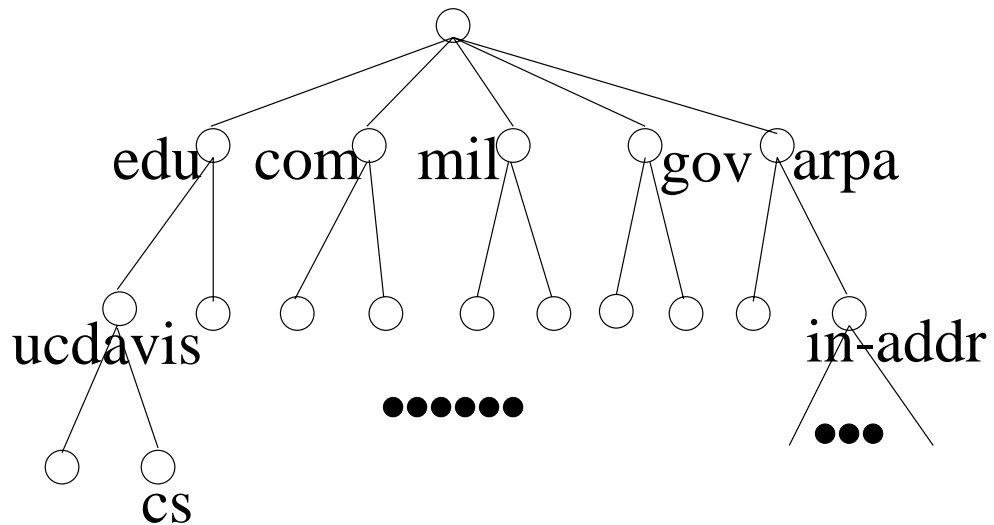
Steven Cheung

*Computer Security Lab.  
University of California, Davis.*

May 1999

## What is DNS?

- Distributed database indexed by “names”  
e.g., host names  $\leftrightarrow$  IP addresses.
- Hierarchical structure:



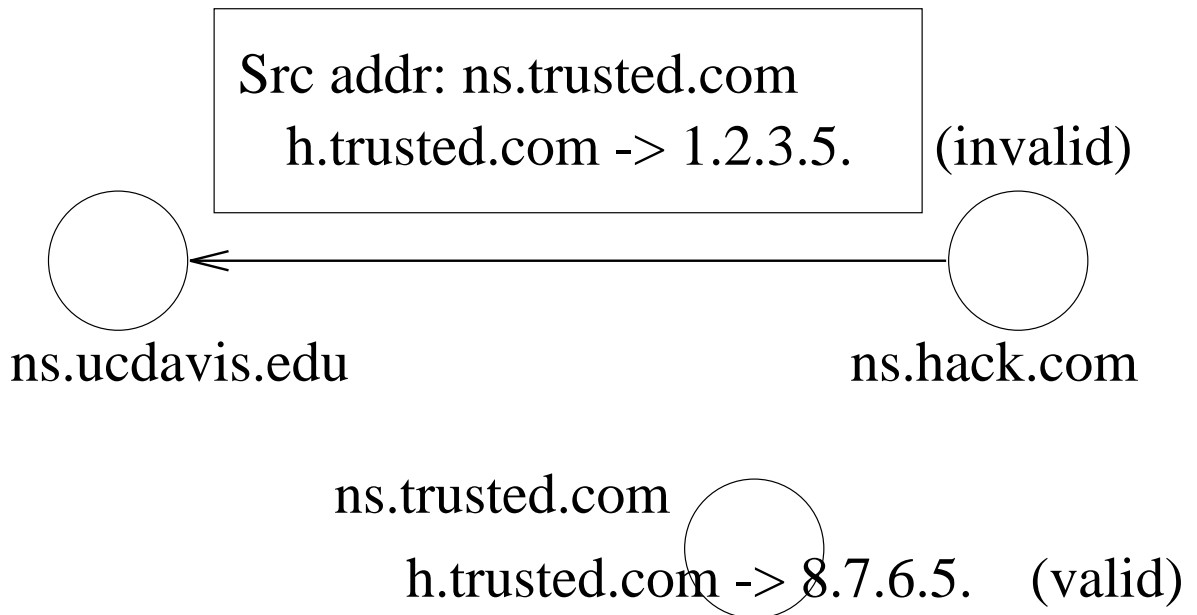
- Name servers resolve queries and manages segments of the database.
- A name server is authoritative for a datum if it is responsible for the corresponding segment of the database.
- Caching—storing results for future references.

## Two Security Problems of DNS

- Cache poisoning



- Lack of data authentication



## **Our Approach: Detection-Response**

- Declare threats—Using DNS messages to cache poison a server. Masquerading as certain name servers. (Note: Message authentication issues cannot be solved completely by our approach. Strong crypto such as DNSSEC is needed.)
- Declare goals—A protected DNS server has the same view as that of the corresponding authoritative servers.
- Develop DNS model—Each DNS server/client maintains a view of the database, which can be changed by receiving DNS messages.
- Design wrapper that verifies DNS messages w.r.t. our security goal and confirms with auth servers (if needed). Drop and log those messages that could violate our security goal.
- Use formal specification (written in VDM) to describe DNS components and our wrapper.

**Wrapper Specification (in VDM)****value**

Db: DbMap  
 ZoneData: Zone  $\rightarrow$  DbMap  
 AuthServer: Zone  $\rightarrow$  Server-set

**type**

RRType = {A, PTR, NS, MX, SOA, HINFO, ...}  
 RRClass = {IN, ...}  
 TTL =  $\mathcal{N}$   
 DName = ...  
 Idx :: dname: DName  
       type: RRType  
       class: RRClass  
 RR :: dname: DName  
       type: RRType  
       class: RRClass  
       ttl: TTL  
       rdata: RData  
 DbMap: Idx  $\rightarrow$  RR-set  
 Zone = ...  
 Process = Server  $\cup$  Resolver  
 Msg = Query  $\cup$  Resp  
 Msg :: Hdr: HeaderSection  
       Q: QuestionSection  
       Ans: AnswerSection  
       Auth: AuthSection  
       Add: AdditionalSection

...

## Wrapper Specification (cont.)

### functions

SubDomain: Domain  $\rightarrow$  Domain-set

SubZone: Zone  $\rightarrow$  Zone-set

Fresh: DbMap  $\rightarrow$  DbMap

ZoneDelegated: Server  $\rightarrow$  Zone-set

AuthAnswer: Idx  $\rightarrow$  RR-set

AuthAnswer(i) =  
 let  $z \in \text{Zone} \wedge i \in \text{dom}(\text{ZoneData}(z))$  in  
 ZoneData(z)(i)

Authoritative: RR-set  $\rightarrow$  Boolean

Authoritative(rrs) =  
 if  $\forall rr \in \text{rrs} \cdot$   
    $rr \in \text{AuthAnswer}(\text{dname}(rr), \text{type}(rr), \text{class}(rr))$   
 then true  
 else false

### state

$view_{auth}(w)$ : DbMap

$view_{cache}(w)$ : DbMap

xtab: QID  $\times$  Query  $\rightarrow$  QID

completed: Query  $\times$  QID

init  $mk\text{-Wrapper}(view_{auth}(w), view_{cache}(w), \text{xtab}, \text{completed}) ::$   
 $view_{auth}(w) = view_{auth}(s) \wedge view_{cache}(w) = \text{null} \wedge \text{xtab}$   
 $= \text{null} \wedge \text{completed} = \text{null} /* s is the protected server */$

inv  $mk\text{-Wrapper}(view_{auth}(w), view_{cache}(w), \text{xtab}, \text{completed})$   
 $view_{auth}(w) = view_{auth}(s) \wedge \text{Authoritative}(view(s))$

end

**Wrapper Specification (cont.)**

SendQuery (q: Query) p: Query

ext wr: xtab: QID  $\times$  Query  $\rightarrow$  QID

pre: true

post: /\* sending out a new query \*/

 $(\text{id}(q), Q(q)) \notin \text{dom } \overline{xtab} \Rightarrow$  $(\text{id}(p) = \text{rand}() \wedge \text{HdrMinusId}(p) = \text{HdrMinusId}(q) \wedge$  $Q(p) = Q(q) \wedge \text{xtab} = \overline{xtab} \uparrow \{(\text{id}(q), Q(q)) \rightarrow \text{id}(p)\})$  $\wedge$   
/\* re-sending a pending query \*/ $(\text{id}(q), Q(q)) \in \text{dom } \overline{xtab} \Rightarrow$  $(\text{id}(p) = \overline{xtab}(\text{id}(q), Q(q)) \wedge \text{HdrMinusId}(p) =$  $\text{HdrMinusId}(q) \wedge Q(p) = Q(q) \wedge \text{xtab} = \overline{xtab})$ 

RecQuery (p: Query) q: Query

pre: true

post:

 $(\text{wellformed}(p) \Rightarrow q = p) \wedge$  $(\text{not wellformed}(p) \Rightarrow q = \text{null})$

**Wrapper Specification (cont.)**

WrapResp (m: Resp) violation: Boolean

ext rd:  $view_{auth}(w)$ : DbMap

ext wr:

$view_{cache}(w)$ : DbMap

xtab: QID  $\times$  Query  $\rightarrow$  QID

completed: Query  $\times$  QID

pre: true

WrapResp1

;

WrapResp2

post: Authoritative(rng  $view_{cache}(w)$ )  $\wedge$

(Authoritative(RRof(m))  $\Rightarrow$  violation = false  $\vee$

not(Authoritative(RRof(m)))  $\Rightarrow$  violation=true) ...

**Wrapper Specification (cont.)**

WrapResp1 (m: Resp) r: Resp

**ext wr:**x<sub>tab</sub>: QID × Query → QID

completed: Query × QID

**pre: true****post:** Authoritative(*rng view<sub>cache</sub>(w)*) ∧

/\* m is a response for a pending query \*/

(∃ q ∈ Query ·

(id(q), Q(q)) ∈ **dom** x<sub>tab</sub> ∧ x<sub>tab</sub>(id(q), Q(q)) = id(m)

∧ Q(m) = Q(q) ⇒

((id(q), Q(q)) ∈ x<sub>tab</sub> ∧ completed =completed ∪ (q, id(m)) ∧

id(r) = id(q) ∧ MsgMinusId(r) = MsgMinusId(m))

∧

/\* m corr. to a query that has been responded already \*/

(∃ q ∈ Query, i ∈ QID ·

Q(m) = Q(q) ∧ id(m) = i ∧ (q, i) ∈ completed ⇒

id(r) = id(q) ∧ MsgMinusId(r) = MsgMinusId(m))

∧

(otherwise r = null)

**Wrapper Specification (cont.)**

WrapResp2 (m: Resp) violation: Boolean

**ext** wr:  $view_{cache}(w)$ : DbMap

**var**: addset: DbMap

**pre**:  $m \neq \text{null} \wedge \text{Authoritative}(\text{rng } view_{cache}(w))$

violation = **false**;

addset =  $\emptyset$ ;

**foreach**  $rr \in \text{RRof}(m)$  {

**if not** AuthVerified(rr,addset) **then** violation = true;

**if** (dname(rr),type(rr),class(rr))  $\in \text{dom}(\text{addset})$

**then** addset(dname(rr),type(rr),class(rr)) =

    addset(dname(rr),type(rr),class(rr))  $\cup \{rr\}$ ;

**else** addset(dname(rr),type(rr),class(rr)) =  $\{rr\}$ ;

}

**if not** violation  $\wedge$  **not** tc(m)

**then**  $view_{cache}(w) = \text{addset} \dagger \overline{view_{cache}(w)}$

**post**:  $\text{Authoritative}(\text{rng } view_{cache}(w)) \wedge$

( $\text{Authoritative}(\text{RRof}(m)) \Rightarrow \text{violation} = \text{false} \vee$

$\text{not}(\text{Authoritative}(\text{RRof}(m))) \Rightarrow \text{violation} = \text{true}$ )

**Wrapper Specification (cont.)**

```

AuthVerified (rr: RR, from: Process, knownNS: DbMap)
  auth: Boolean
  /* returns true if rr agrees with authoritative data from an
  authoritative server; returns false otherwise. knownNS con-
  tains verified NS resource records in the same response as rr
  does. */
var: ans: Server, q: Query
  auth = false;
  if ((from ∈ KnownAuthServer(dname(rr),class(rr),View(w)))
    /* from is an auth server for rr */ ∨
    (∃ z ∈ Zone ·
      from ∈ KnownAuthServer(ZtoD(z),class(rr),View(w))
      /* from gives info of delegated zones */ ∧
      ((dname(rr) ∈ SubZone(z) ∧ type(rr) = NS) ∨
      (∃ cz ∈ Zone, nsr ∈ U(rng knownNS) ·
        cz ∈ SubZone(z) ∧ type(rr) = A ∧
        dname(nsr) = cz ∧ type(nsr) = NS ∧
        dname(rr) = rdata(nsr) ∧ dname(rr) ∈
        SubDomain(z)))) /* rr is in z's domain */ ∨
      (rr ∈ U(rng View(w)))) /* rr is known to be auth */
  then auth = true
  else
    nz = NearestZoneKnown(w,rr);
    /* find nearest zone that w knows its name servers */
    auth = rr ∈ Ans(CheckAuthServer(rr,ZtoD(nz),nz,1))
pre: Authoritative(rng View(w))
post: auth = Authoritative({rr})

```

## Experiments

- Implemented a DNS wrapper for BIND release 4.9.5.
- Goal: Evaluate response time, false positive rate, false negative rate, and computational overhead.
- Experimental procedure:
  1. Collected all DNS queries generated by a user for two days (1340 queries).
  2. Ran a wrapped name server.
  3. Used a modified *nslookup* to send those queries to the name server and recorded response times.
  4. Recorded CPU time used by the wrapped server.
  5. Examined security log generated by wrapper for false positives.
- Results (based on 33 runs):
  - Avg. (per query) response time: 0.08s  $\rightarrow$  0.12s.
  - Avg. CPU time: 9.33s  $\rightarrow$  11.29s.
  - Avg. num. false positives: 5.85.

## False Negatives

- Incorrect resource records for a remote domain name to the victim name server.
- Incorrect RR that conflict with zone data for which the victim is authoritative.
- RR that correspond to a non-existing domain name that lives in the victim's zone.
- First query to trigger the victim to send a query (QID= $i$ ) to the compromised name server. Second query to triggers the victim to query the compromised name server again. Instead of using the query ID found in the second query, the attacker uses  $i + 1$  as the query ID in the second response.