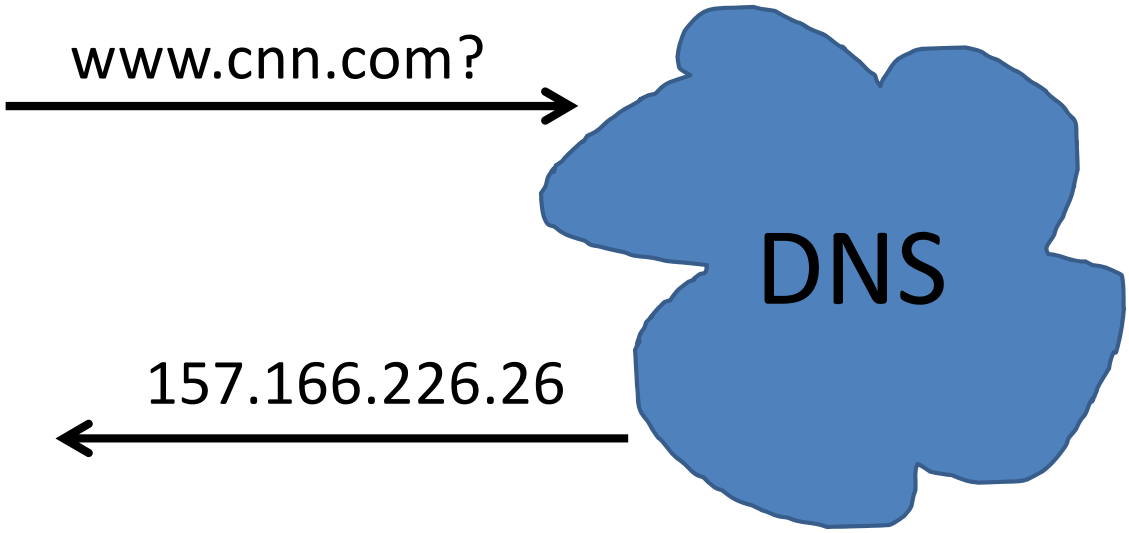
Provably Secure DNS: A Case Study in Reliable Software

Barry Fagin
Martin Carlisle

Overview

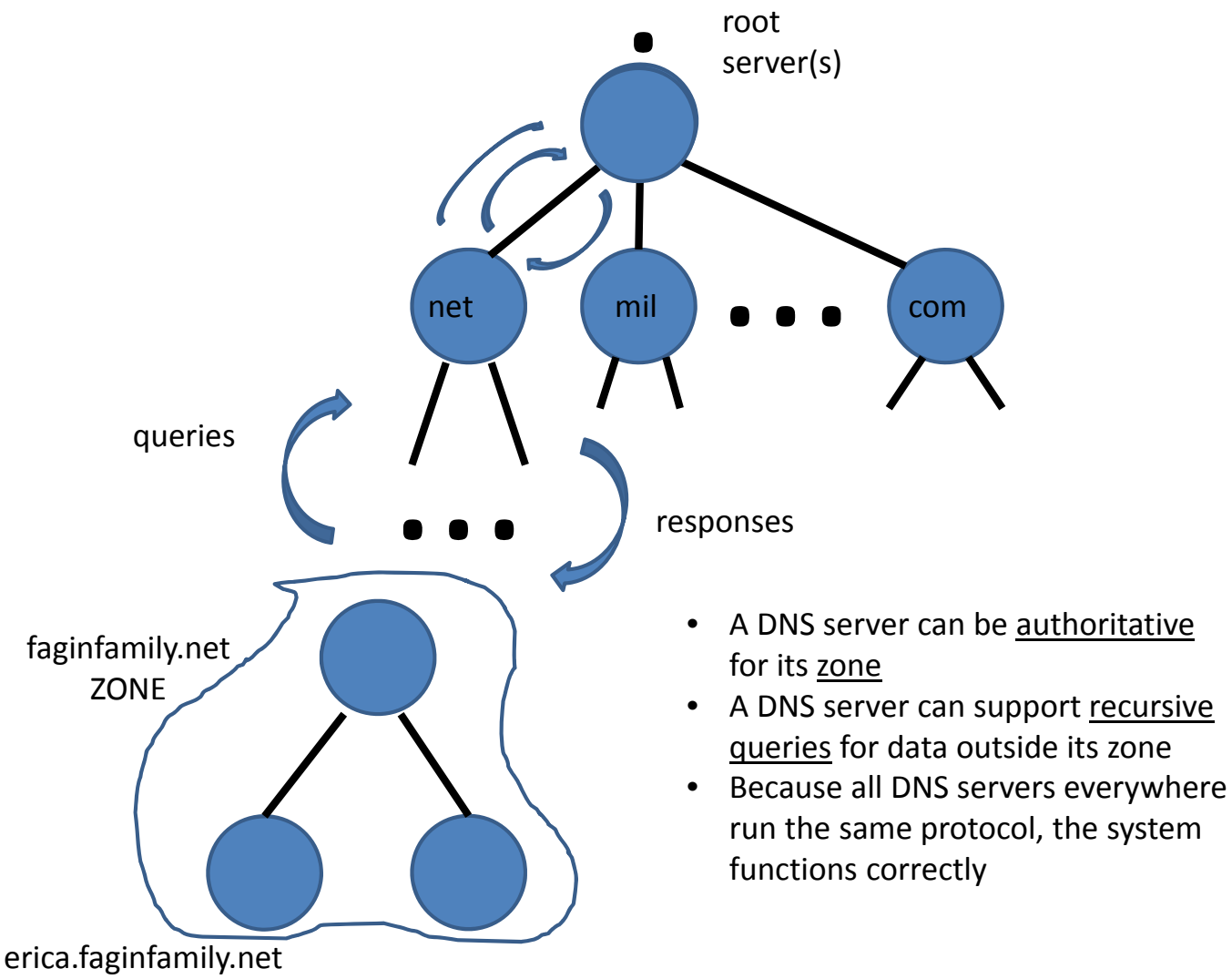
- Intro –DNS, security problems
- SPARK
- IRONSIDES
- Experimental results (previous, new)
- Lessons in humility
- Hitting the sweet spot
- Conclusions, future work

- DNS (internet Domain Name System) is the protocol for associating host names with IP addresses. Dates from 1983 [Mockapetris].



Key features: Distributed, Hierarchical

- DNS protocol implemented with a DNS server (refers both to the software and the machine running it)
- Distributed: Hundreds of thousands running simulatenously
- Hierarchical: Distributed database in a tree structure. Server only responsible for small subset of data, whatever it doesn't know it can query up the tree. Can store answers that come back (caching).



- A DNS server can be authoritative for its zone
- A DNS server can support recursive queries for data outside its zone
- Because all DNS servers everywhere run the same protocol, the system functions correctly

- DNS runs everywhere, have to have it if you connect to the internet.
- Most common: freeware Berkeley Internet Name Domain server (BIND), various bundled implementations with Windows.
- Market share estimates 75-90%, depending on source.

- These (and other) implementations of DNS have significant security problems:
- 51 vulnerabilities in BIND (Internet Systems Consortium)
- 8 security bulletins from Microsoft in last five years relating to Windows DNS
- Buffer overflows, DOS caused by bad packets, remote code execution vulnerabilities …
- Note: None of these due to protocol itself, there are other exploits that attack protocol weaknesses.

- Why so many security holes?

- Implemented in old languages,
- open source makes vulnerability detection easier,
- developed in early days of software engineering

- Later releases more secure, but usual issues with patches, compatibility, etc

- Time seems ripe to attack the problem at the source

SPARK: A tool for creating provably correct programs

- Formal methods and language design have come a long way in thirty years
- SPARK language/toolset used to create software with provable correctness & security properties
- Subset of Ada, augmented with special preprocessor annotations

```
PROCEDURE Add(Msg: IN Dns_Types.DNS_Packet;
        MsgLengthInBytes : in DNS_Types.Packet_Length_Range;
        Requester : IN Dns_Network.Network_Address_And_Port;
        WhereAdded : out buffer_pkg.BufferIndexType;
                     Failure : out boolean)
     --# global in out buffer, NumMsgs, BufferNotEmpty;
     --#    in Name;
     --# derives buffer from buffer, NumMsgs, Msg &
     --#    NumMsgs from buffer, NumMsgs &
```

```
 while Response_Counter <= NumFound and Integer(Current_Byte)
< DNS_Types.Packet_Size-(16+DNS_Types.Header_Bits/8) loop
   --# assert Response_Counter >=1
   --# and Response_Counter <= NumFound
   --# and Integer(Start_Byte) <= DNS_Types.Packet_Size
   --# and Answer_Count = Answer_Count~
```

- Preprocessor used to check certain characteristics, generate verification conditions (VCs)
- VCs and other conditions discharged via a theorem prover
- When all VC's discharged, proven that no array overflows, all loops terminate, all information flows as specified, etc.

- We believed we could use SPARK to build a DNS server with provable security properties.

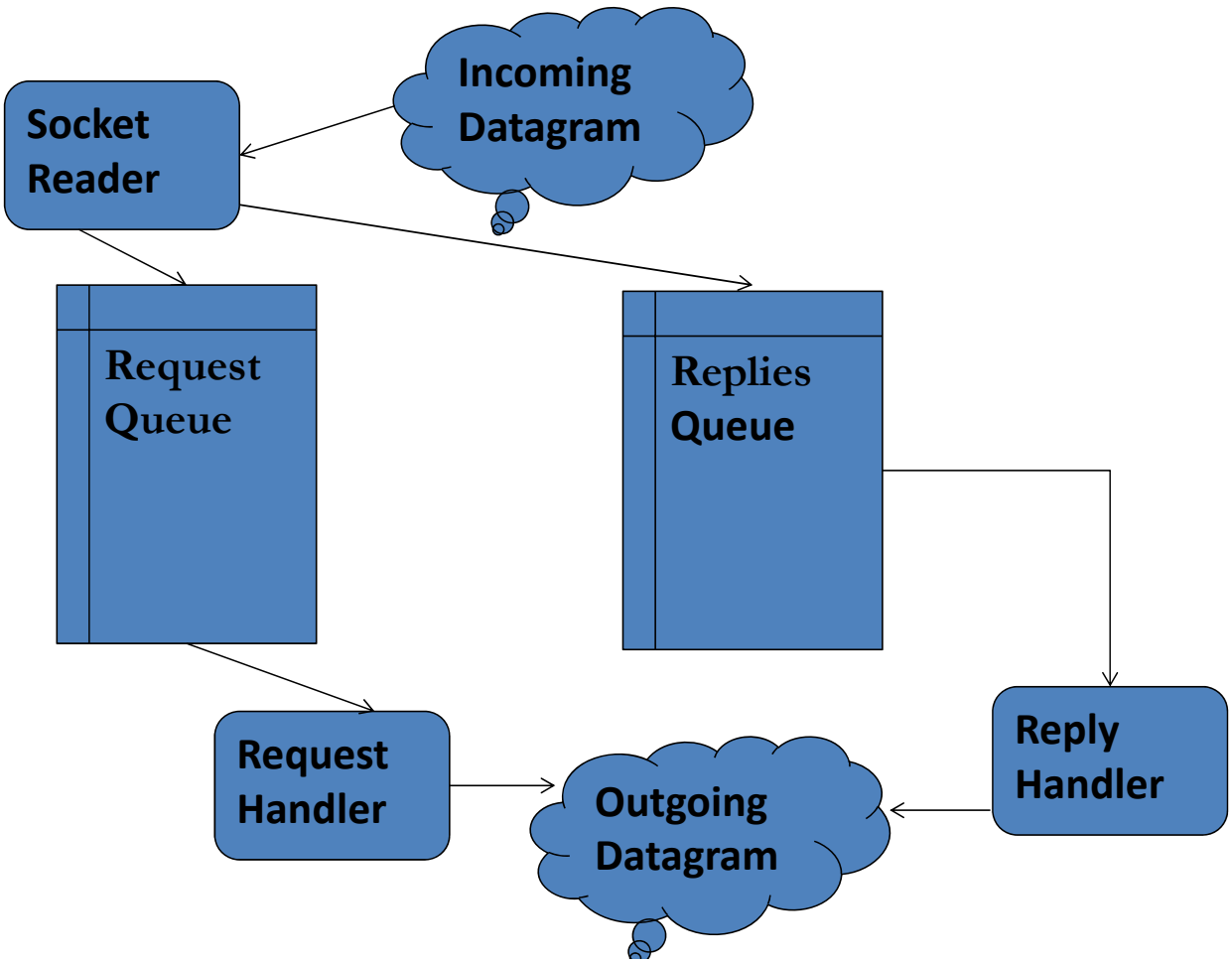
- Open source, freely available.

- Investigate tradeoff (?) between performance and security. How well would it perform compared to BIND, Windows DNS? Comparable feature support?

IRONSIDES

- IRONSIDES is an implementation of the DNS protocol (a DNS server) written in SPARK/Ada.
- First implemented as authoritative only [Carlisle, Fagin GLOBECOM 2012]
- Now supports DNSSEC, recursive queries and caching for A, AAAA, MX record types.
- Description of latest version, results

Recursive Queries



Proof requirements for latest version of IRONSIDES

	Total	Examiner	Simplifier	Victor
Assert/Post	5027	3181	1837	9
Precondition	268	0	241	27
Runtime check	3304	0	3256	44
Refinem. VCs	54	54	0	0
Totals:	8653	3235	5334	80
%Totals:		37%	62%	1%

Code size?

Total Lines: 11598

Blank Lines: 871

Non-blank non-comment lines: 7543

Lines of SPARK annotations: 1133 (< 10%)

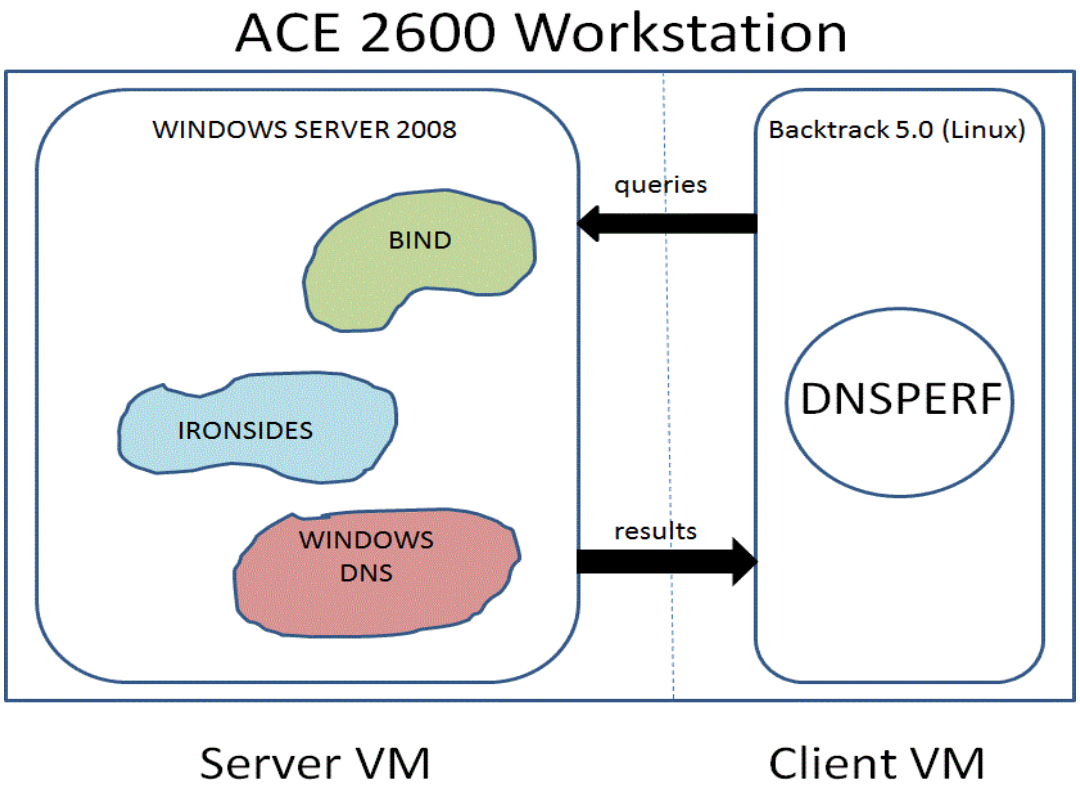
Semicolons: 5403

Performance results

Server	Authoritative	Recursive	Recursion ACL	Slave mode	Caching	DNSSEC	TSIG	IPv6	Wildcard	Free Software	split horizon
BIND	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Windows DNS	Y	Y	N	Y	Y	Y	Y	Y	Y	N	N
IRONSIDES	Y*	in progress	N	N	in progress	offline-signed	N	Y	N	Y	N

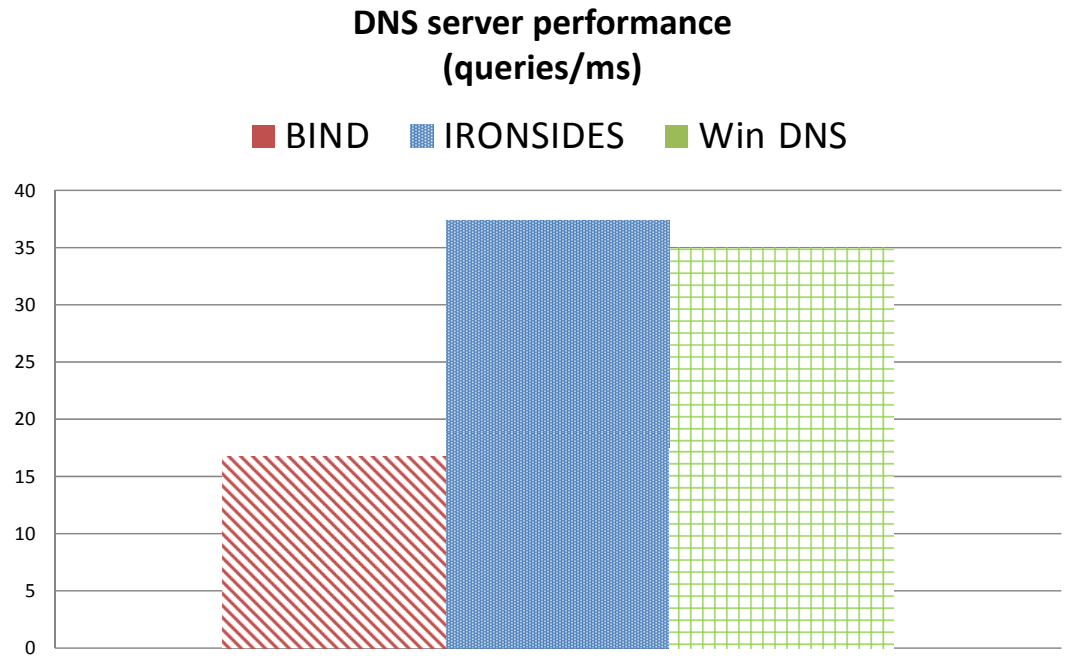
Feature comparison of BIND, Windows DNS, IRONSIDES

*The following resource record types are currently supported: A, AAAA, CNAME, DNSKEY, MX, NS, NSEC, PTR, RRSIG, SOA.



BIND			IRONSIDES			Win DNS		
16478.3	16667.9	17020.0	37329.1	37814.6	37024.4	34188.0	35676.1	35089.3

(queries per second, three test runs)



Lessons in humility

Authors are experienced software engineers (for academics, anyway ☺).

40 years combined CS teaching experience, consulting.

Still got bit on things that the tools caught.

1) The use in a zone file of a domain name consisting of a single character:

```
--SPARK caught possible exception if length=1, modified
--by adding "length > 1 and then"
if Name(1) = '.' or Name(1) = '-' or (length > 1 and then
(Name(Length-1) = '.' or Name(Length-1) = '-')) then
  RetVal := False;
```

2) A resource record of length equal to the maximum line length allowed:

```
--endIdx might be the maximum value possible, so must
catch last character here.  Caught by SPARK.
 if Ctr = EndIdx and numSeparators <= REQ_NUM_SEPARATORS
then
```

3) Failure to account for erroneous input:

```
if Query_Class /= IN_CLASS then   …
elsif Query_Type = A then   …
end if;
--Forgot else to handle erroneous input! Caught by SPARK.
```

Hitting the sweet spot

Proving correctness for non-trivial programs is hard, often harder than writing the original program

But tools have come a long way

Focus on proving security properties instead

Can prove these properties:

1. No buffer overflow
2. No incorrect calculation of buffer size
3. No improper initialization
4. No ineffective statements
5. No integer overflow/wraparound
6. No information leakage
7. All input validated
8. No allocation w/o limits (no resource exhaustion)

9. No improper array indexing
10. No null pointer dereferencing
11. No expired pointer dereferencing (use after free)
12. No type confusion
13. No race conditions
14. No incorrect conversions
15. No uncontrolled format strings
16. All loops guaranteed to terminate

BIND and Windows DNS have problems with all these vulnerabilities.

Conclusions, future work

Formal methods have advanced considerably in the past few decades, “sweet spot” of opportunity.

Produced provably exception-free DNS server

Not proven correct, responses correctly formatted, invulnerable to packet flooding

No need to sacrifice performance for security. Quite the contrary!

Conclusions, future work

Continue to add functionality: Online zone signing, more record types, full recursive query support, continue testing under load as we add more features. GUI/web interface? Other OS? More ports?

Where is next sweet spot, next target of opportunity?

IRONSIDES in public domain, freely available at http://ironsides.martincarlisle.com.