IRONSIDES: DNS With No Single-Packet Denial of Service or Remote Code Execution Vulnerabilities

Martin Carlisle
Barry Fagin

Department of Computer Science
US Air Force Academy
USAFA, CO USA
{martin.carlisle, barry.fagin}@usafa.edu

***Abstract*—We describe the development of IRONSIDES, an implementation of DNS that is provably invulnerable to remote code execution exploits and single-packet denial of service attacks. Our experimental results show it to be over three times as fast as BIND, the most common implementation of DNS.**

Keywords-DNS, computer security, network security, denial of service, remote execution, buffer overflow

I. Introduction

The Internet Domain Name System, or DNS, is an essential component of internet infrastructure. Responsible for turning names into IP addresses, its protocols are running on hundreds of thousands of computers all over the world. Designed originally to solve a problem of scalability during the early days and rapid growth of the ARPAnet, it has by any standard been an incredible success.

This success, however, has come with a price. Because DNS originates in the earliest days of the internet, before security issues were well understood, its most popular implementations are rife with security vulnerabilities. DNS servers around the world can be crashed by hackers, or even worse, taken over by them.

To address this problem, we have developed IRONSIDES, a DNS server that is provably invulnerable to many of the problems that plague other servers. It achieves this property through the use of formal methods in its design, in particular the language Ada and the SPARK formal methods tool set. Code validated in this way is provably exception-free, contains no data flow errors, and terminates only in the ways that its programmers explicitly say that it can. These are very desirable properties from a computer security perspective.

IRONSIDES is not only stronger from a security perspective, it also runs faster than its leading competitors. It provides one data point showing that one need not trade off reliability for performance in software design.

We begin with an overview of the problem IRONSIDES is designed to solve. We then discuss why we believe the time is ripe for formal methods to play a role in improving the security posture of internet infrastructure software. We describe IRONSIDES, our experimental results, and conclude with a summary and directions for future work.

II. The Nature of the Problem

A. What is BIND?

BIND stands for the Berkeley Internet Name Domain server. Originally written in 1984, it has been ported to a number of systems and compilers, and has been maintained in the public domain since its inception.

According to one source, in 2003 BIND handled about 85% of all the internet DNS requests [1]. A more recent survey of over 1 million sampled domains showed that over 75% are running BIND [2]. Thus security problems in BIND would seem to merit the most attention, since they would have the greatest impact. Hackers, of course, know this quite well, and would naturally focus their efforts on BIND. They have found and exploited numerous flaws.

B. DNS Security Vulnerabilities

Virtually all DNS servers running today contain significant security vulnerabilities. The problem is particularly acute in BIND, due to three factors: a) BIND was the first implementation of DNS, b) BIND is written in C, a language where it is easy to make mistakes that cause security vulnerabilities, and c) BIND is open-source, which means security holes can quickly be identified and exploited. To deal with these problems, the latest release of BIND (v9) is a complete rewrite from scratch. It is a significant improvement, but flaws are still being found.

Security problems, however, are by no means limited to BIND. Windows DNS was designed and implemented much later than BIND, and is proprietary software, but still has numerous security vulnerabilities.

C. Types of Vulnerabilities

Although there are dozens of security flaws in DNS software, the known ones can be classified into a few distinct types:

- DOS: Denial of Service. This flaw means the server can be crashed by a client sending it a specially formatted query.
- RCE: Remote Code Execution. The attacking program sends a non-standard query that diverts execution flow to malicious code, giving the attacker control of DNS on the target machine.
- Spoofing/cache poisoning: Attackers inject incorrect information into DNS to misdirect traffic from its correct destination to one selected by the attacker.
- Protocol weaknesses: These exploit security defects inherent in the DNS protocol or algorithms themselves.

D. *Current Status and Impact of DNS Security Vulnerabilities*

Just how serious are DNS security flaws? For BIND, as of this writing an analysis of the security advisories at http://www.isc.org/ shows a total of 45 vulnerabilities. Twenty-four of these are remote denial of service attacks, nine are remote exploitation/execution. The remaining twelve are weaknesses in cryptographic algorithms, the DNS protocol itself, or similar problems with the algorithms themselves rather than their implementation.

The statistics for Windows DNS security flaws are more difficult to determine. However, our examination of published Microsoft Security bulletins from the past five years found eight related to Windows DNS. In reverse chronological order, they are:

- MS11-058 – two vulnerabilities, one denial of service and one remote exploit
- MS11-030 – remote code execution
- MS 09-008 – spoofing vulnerability
- MS 08-037 – spoofing vulnerability (2)
- MS 08-020 – spoofing vulnerability
- MS 07-062 – spoofing vulnerability
- MS 07-029 – remote exploitation
- MS 06-041 – remote exploitation vulnerabilities (4)

This gives a total of thirteen known security flaws in various versions of Windows DNS.

In principle, these vulnerabilities and those of BIND can be removed by applying the appropriate patches/code updates. In practice, however, updates are not always applied in timely fashion, if ever. More importantly, since there is no claim of formal validation for either of these implementations, it is likely that both contain further as yet unknown security holes for hackers to exploit.

From this point forward, we concern ourselves only with the security and performance of BIND. For a comparison of IRONSIDES with Windows DNS, the reader is referred to [3].

III. Formal Methods as a Solution

One reason security flaws exist in software is because our ability to reason about software has lagged far behind our ability to write it. It has long been known that in principle it is possible to prove correctness and security properties of computer programs. But in practice, the difficulties in doing so efficiently have proven extremely challenging.

Attempts to use proof techniques from mathematics in software design belong to the computer science discipline known as *formal methods.* For most of computing since the days of the internet, the use of formal methods for all but the most trivial of programs has been either impossible or grossly cost-ineffective. The result has meant that most errors are discovered and removed from software via pre-release testing. Further errors are either discovered by users or exploited by the hacker community, with the results being repaired in post-release patches. This describes the current state of most modern security vulnerabilities, and DNS vulnerabilities in particular.

A. *Progress in Formal Methods*

Fortunately, our ability to automatically prove program correctness has improved significantly. As tools have become more cost-effective and user friendly, the scope and power of software for which formal methods can be applied has grown dramatically.

For example, research from Microsoft's SLAM project was incorporated into releases of Windows Vista. Based on a formal methods approach, Vista's device driver validation module detects if drivers linked to it violate certain interface rules [4]. The next year, Heitmeyer and Jeffords [5] reported the successful use of the SCR requirements model in the Deep Impact probe and International Space Station software. Implementing the Common Criteria for Information Technology Security Evaluation has proven fertile ground for formal methods, as shown by Heitmeyer and others [6].

In 2008, Airbus described its successful use of the SCADE system to automatically generate code from formal specifications in its A340-500/600 aircraft [7]. Sony developed the firmware for its new contactless IC cards using the VDM++ and VDMTools relying on formal methods [8]. More recently still, the Verisoft project, funded by the German Federal Ministry of Education and Research, published claims of proof of correctness of a real-time operating system known as OLOS, designed for automotive applications [9].

We have taken advantage of the recent progress in formal methods to construct and eventually release IRONSIDES, an open-source, provably exception-free implementation of DNS. We now turn to a discussion of the source language and tool set used, and then discuss its functionality, performance, and security properties.

B. *SPARK: A Tool for Creating Provably Correct Programs*

The SPARK language and toolset from Praxis Critical Systems Limited is used in the creation of software systems with provable correctness and security properties [10]. SPARK is a subset of Ada, augmented with special annotations. These annotations appear as ordinary comments to Ada compilers, but are visible to SPARK's pre-processing tools used to verify the software. SPARK is a fairly mature technology and has been used on several projects [11], [12], [13]. Accordingly, given our prior institutional experience with Ada [14], we chose SPARK and Ada as the platform for constructing DNS software that would not be subject to most of the vulnerabilities that afflict DNS implementations currently deployed around the globe.

IV. IRONSIDES: Formal Methods and DNS

IRONSIDES is an Ada/SPARK implementation of the DNS protocols. Currently, it supports only authoritative name service, but future versions are expected to support recursive queries. We are also in the process of adding support for DNSSEC, the protocol that adds encryption to DNS transaction to further reduce vulnerability to spoofing and other attacks [15].

The architecture of IRONSIDES is shown in Figure 1. IRONSIDES was written from the "ground up" in Ada/SPARK, using the relevant RFC's and other descriptions of DNS as a guide [16-19].

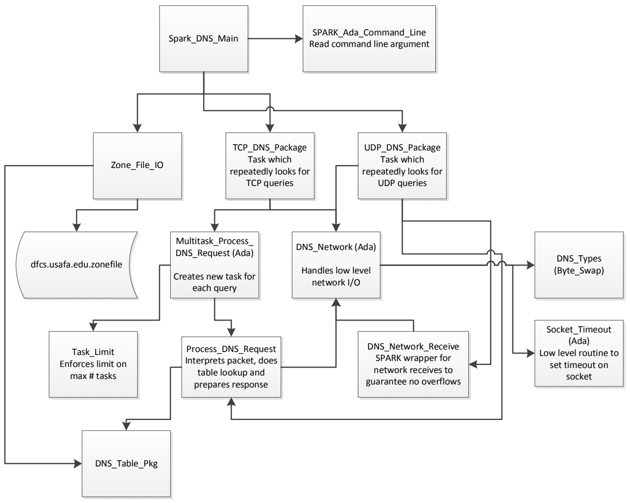
Spark_DNS_Main
SPARK_Ada_Command_Line Read command line argument
Zone_File_IO
TCP_DNS_Package Task which repeatedly looks for TCP queries
UDP_DNS_Package Task which repeatedly looks for UDP queries
dfcs.usafa.edu.zonefile
Multitask_Process_DNS_Request (Ada) Creates new task for each query
DNS_Network (Ada) Handles low level network I/O
DNS_Types (Byte_Swap)
Task_Limit Enforces limit on max # tasks
Process_DNS_Request Interprets packet, does table lookup and prepares response
DNS_Network_Receive SPARK wrapper for network receives to guarantee no overflows
Socket_Timeout (Ada) Low level routine to set timeout on socket
DNS_Table_Pkg

Figure 1. IRONSIDES System Archtiecture

As of this writing, verification of IRONSIDES requires the generation and proof of 6,183 verification conditions, or VCs. These include assertions that variables always remain in type, array bounds are never exceeded, specific pre- and post-conditions of procedures are always true, and so forth. When a VC is proved, it is said to be discharged. Discharge of a VC is accomplished through a multi-stage process using the SPARK automatic theorem proving tools. For the VCs in IRONSIDES, 2,086 were proved by the first stage of the tools, and 4,033 by the second, almost 99%. The remaining 1% of VCs were sufficiently complex to require the use of the Alt-Ergo theorem prover [20], built into SPARK as an option to discharge VCs that the other tools cannot.

Software verification is not considered complete until all VCs are discharged. For IRONSIDES, the complete verification of code through the discharge of all VCs, from their initial generation to the final summary report, takes 3 minutes 6 seconds on an IBM ThinkPad X220 Tablet PC with 8GB of memory.

As a result of this process, IRONSIDES code is known to be free of uninitialized values, data flow errors (e.g. writes that are never read or values derived from incorrect sources), array bounds errors, and all runtime exceptions. This renders it invulnerable to single-packet denial of service attacks and all remote execution exploits. If IRONSIDES is properly compiled and configured, it cannot be taken over as a result of any external input, no matter when the input arrives and no matter how it is formatted. Also, it cannot be crashed and all its loops are guaranteed to terminate, which renders it invulnerable to denial of service attacks that rely on badly formatted packets. It is, as far as we know, the only DNS server to make these claims.

A. *Experimental Results*

In this paper we compare the performance of IRONSIDES with BIND using the DNS stress testing tool 'dnsperf' [21]. Because IRONSIDES is still in its early stages of development, it does not have all of BIND's features. Any comparison thus needs to take these differences into account. Following the style of [22], we show a comparison of IRONSIDES and BIND in Table I below. Footnotes and parenthetical comments for BIND are omitted to save space.

TABLE I. IRONSIDES and BIND Feature Comparison

Server	**Authoritative**	**Recursive**	**Recursion ACL**	**Slave mode**	**Caching**	**DNSSEC**	**TSIG**	**IPv6**	**Wildcard**	**Free Software**	**Interface**	**split horizon**
BIND	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Web, command line	Y
IRONSIDES	Y*	N	N	N	N	in progress	N	Y	N	Y	command line	N

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

IRONSIDES, for example, does not yet support recursive queries and slave mode operation. Caching will be added once recursive queries are supported, and DNSSEC/TSIG are in progress. We do not currently have plans to support wildcarding or split-horizon DNS. Readers interested in learning more about these terms are referred to [22].

Our experimental test bed for comparing BIND and IRONSIDES is shown in Figure 2:

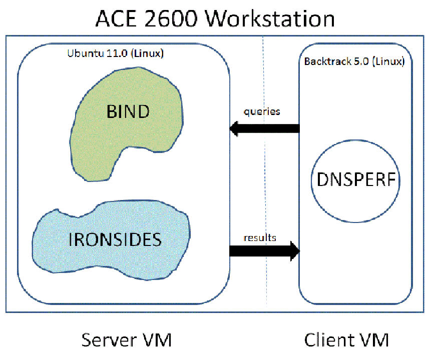

Figure 2. Test Bed for Comparing DNS Implementations

'dnsperf' runs on a Backtrack 5.0 client virtual machine. For the server VM we used Ubuntu 11.0. Testing is done by starting up the DNS server to be tested and then running dnsperf. Only one DNS server, server VM, and client VM are active at any one time.

Since the purpose of the experiment is to measure the computational performance of the server, all VMs are loaded on the same computer, in this case an ACE 2600 Workstation with 8GB of RAM. Using the same computer for client and server eliminates the effect of network latency. 'dnsperf' issues queries over the standard DNS port to whichever server is listening. The server in turn responds as appropriate. At the end of a run, the tool generates a performance report.

For each server, we performed three test runs and averaged the results. (For all tests, the standard deviation was never higher than 2.1% of the mean, so we believed three test runs were adequate). The performance of BIND and IRONSIDES is shown below:

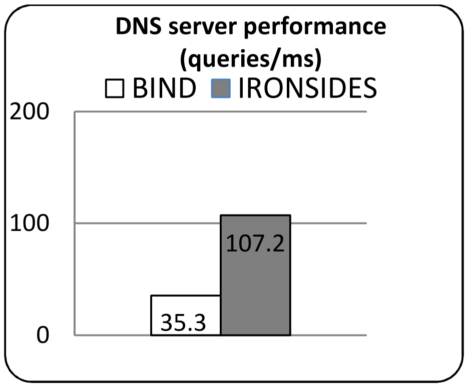

Figure 3. BIND and IRONSIDES Performance

B. Analysis

IRONSIDES is over three times faster than BIND on Linux. Given IRONSIDES' superior security posture, we find these results significant. They show that one need not sacrifice security for performance in software design.

In fact, it should not be that surprising that there are at least some instances in which the use of formal methods can improve performance. Data flow analysis, for example, can identify redundant or ineffective statements that generate unnecessary code. Code that has been proven exception-free no longer needs run-time bounds checking, so that code can be eliminated as well.

On the other hand, there are also cases where total reliance on formal methods negatively impacts performance. Given the current state of the art in formal methods tools, it continues to be appropriate to allow programmers the flexibility to override warnings of unproven properties when they believe the tradeoffs are worth it. This is especially true if they themselves can see that certain properties hold, even when tools cannot make that determination.

For example, DNS queries return data of varying size. The use of dynamically allocated data structures significantly complicates formal analysis, and renders the ability to bound the maximum storage required for a program impossible. For this reason, SPARK requires all data structures to be statically allocated [9]. Thus routines that return varying amounts of data use an output data structure of fixed size, defined at compile-time as a known upper limit.

Data flow analysis of such structures, however, requires that all such storage be explicitly initialized, to ensure that stale data is never passed back and that undefined values are never used. Arrays in SPARK in particular are treated as entire variables. Thus initializing an array in a loop with a statement like A(I) := 0 is considered a dataflow error, because only part of A is set. Instead, the use of Ada aggregates is required to eliminate dataflow errors.

This "precautionary principle" can have significant performance consequences. If only one entry of an array is filled and returned by a procedure, but 128 entries must be explicitly initialized, this is inefficient and wasteful. Thus in a few cases throughout the code where such things matter, we removed aggregate initialization, with explicit instructions to the tools to ignore all related dataflow errors. We then manually inspected the code to ensure this did not introduce any security vulnerabilities. For example, by keeping a simple index variable to indicate the upper limits of useful data, it can be shown by inspection that undefined or stale data is never read. Employing this optimization improved the performance of IRONSIDES by 29%.

Our experience indicates that allowing users to override formal proof requirements when appropriate is an important feature that current formal methods tools should always support. Since such overriding is optional, users in environments where manual verification of source code is deemed too risky can revert to the original, formally verified source code at some cost in performance.

C. *Resistance to Denial of Service Attacks*

IRONSIDES is invulnerable to denial of service attacks caused by badly formatted packets that raise exceptions. But terminating a server is not the only way to deny service. If the server can be thrown into an infinite loop, service is just as effectively denied. IRONSIDES is invulnerable to this form of service denial as well, because the tools employed help prove that all of its 85 loops terminate. This is accomplished by using loop invariant assertions to show that loop variables monotonically increase and have an upper bound. This is not accomplished automatically by SPARK, but with appropriate loop assertion annotations added by the programmer SPARK can assist in showing these properties to be true.

For example, consider the code below:

```
-- Amount_Trimmed is used to guarantee we don't end up in an infinite loop
while Answer_Count=0 and Amount_Trimmed<RR_Type.WireStringType'Last and
      Natural(Character'Pos(Current_Name(Current_Name'First)))/=0 and
      Current_Qname_Location <= DNS_Types.QNAME_PTR_RANGE(Output_Bytes) loop
   --# assert Answer_Count=0 and Amount_Trimmed>=0 and
                Amount_Trimmed<RR_Type.WireStringType'Last
   --# and Output_Bytes <= DNS_Types.Packet_Length_Range'Last
   --# and Current_Qname_Location <= DNS_Types.QNAME_PTR_RANGE(Output_Bytes);
   Trim_Name(
      Domainname         => Current_Name,
      Trimmed_Name       => Trimmed_Name,
      Qname_Location     => Current_Qname_Location,
      New_Qname_Location => New_Qname_Location);
   Create_Response_SOA(
      Start_Byte      => Start_Byte,
      Domainname      => Trimmed_name,
      Qname_Location  => New_Qname_Location,
      Output_Packet   => Output_Packet,
      Answer_Count    => Answer_Count,
      Output_Bytes    => Output_Bytes);
   Current_Name := Trimmed_Name;
   Current_Qname_Location := New_Qname_Location;
   Amount_Trimmed := Amount_Trimmed +
         Natural(Character'Pos(Domainname(Domainname'First))+1);
end loop;
```

Figure 4. Using loop invariants to prove termination

SPARK annotations begin with "--#". Here the annotations are loop invariants that serve as both a postcondition for one part of the loop and as preconditions for the next. In this case the tools prove that Amount_Trimmed is at all times both non-negative and below a constant upper bound. They also show that Amount_Trimmed is not modified elsewhere in the loop. Given these properties and the last line of the loop, we can conclude that Amount_Trimmed is monotonically increasing, therefore the loop terminates.

Note that without the use of this variable and the proof annotations, we could not prove loop termination. This would leave open the possibility for the other termination conditions to never be reached, something that could be exploited under the right circumstances to deny service through an infinite loop.

While IRONSIDES is not completely resistant to packet flooding, neither is any other program. Since it performs significantly better than BIND, however, at a minimum it can handle as much or more flooding. Additionally, IRONSIDES contains two features not related to formal methods designed to make it more resistant to flooding-based denial of service attacks. First, the number of simultaneous TCP connections is limited by a user-tunable parameter. Additionally, IRONSIDES enforces a socket timeout, to prevent an attacker from holding a connection open for a long period of time.

D. *Lessons in Humility*

Despite the use of formal proofs in the determination of IRONSIDES security properties, a cautionary tale remains in order. It is always worth remembering that the quality of software written in a high level language is only as good as the quality of the compiler that generates code from it. For at least one combination of operating system, compiler, and optimization level, we were able to replicate a case where a fully validated version of IRONSIDES still crashed with an exception, due to a code generation error in the Ada compiler (Free Software Foundation GNAT) shipped with Ubuntu.

Clearly blame for bugs of this nature cannot be laid at the feet of the tools vendors, since they are not responsible for public domain compilers. Nonetheless, the mere existence of such errors is somewhat disturbing. Until formal methods have progressed sufficiently to prove the correctness of compilers themselves, practitioners should continue to exercise caution in how they compile and test verified software.

It remains true that the use of formal methods and the SPARK tools in particular produced results that are both impressive and humbling. Both the authors are experienced software engineers, having written compilers, introductory programming environments, circuit emulators, and other non-trivial software systems. In addition to over 40 years combined computer science teaching experience, we have consulted for both industry and government.

But despite all our experience and credentials, the formal methods tools we employed caught boundary conditions and potential problems that we should have detected on our own but did not. These include array subscript overruns, unusual but possible boundary conditions like single-letter domain names, and so forth. Had the tools not flagged these as possible run-time errors, they could have become security flaws similar to those in BIND.

V. Conclusions and Future Work

We began this project concerned about the large number of security vulnerabilities in a vital component of internet infrastructure, the Domain Name System. We had reason to believe that formal methods had progressed to the point where they could now be employed to produce, for the first time, a public-domain version of DNS with proven security properties. We believe we were successful in this objective.

In the course of developing IRONSIDES, we discovered that certain tradeoffs can be made to improve performance at the expense of formal verification assurance. But in general, our work clearly shows that it is absolutely possible to obtain both improved performance and improved reliability in software design, *provided security considerations and formal methods are incorporated at the very beginning of the process.* IRONSIDES runs over three times faster than BIND under the version of Linux we tested. Unlike BIND, it cannot be subverted or crashed through bad packets.

It might be argued that since BIND is open source and extremely widely deployed, it is going to be widely targeted by hackers and therefore our comparison is not strictly fair. However, we would argue that hackers can target IRONSIDES all they want (the distribution will be open source). IRONSIDES is provably exception-free. None of the vulnerabilities we have described exist for hackers to target.

For a truly fair comparison with BIND, however, we need to incorporate more features into IRONSIDES. In addition to support for DNSSEC, future plans include support for recursive queries, GUI and web interfaces (IRONSIDES is currently command line only) and other more advanced features. Future work could also include testing under more operating systems and testing under actual network loading. Finally, we believe other implementations of internet protocols that suffer from security flaws could benefit from the approach described here.

IRONSIDES is in the public domain, and will be distributed free of charge.

Acknowledgment

This work was funded by the US Defense Advanced Research Projects Agency, whose support is gratefully acknowledged. We also thank AdaCore Technologies and Altran Praxis for providing technical support on using their tools. We also wish to thank the USAFA Department of Computer Science, the Academy's Director of Research, and the Academy Center for Cyberspace Research.

References

[1] http://www.pugetsoundtechnology.com/training/bind-postfix-apache/ (2003 web server training)

[2] Infoblox survey, http://dns.measurement-factory.com/surveyssum1.html, 2011

[3] B. Fagin and M. Carlisle, "Provably exception-free DNS: a case study in formal methods," ISSRE 2012 (submitted).

[4] B. Beckert et al., "Intelligent systems and formal methods in software engineering", *IEEE Intelligent Systems,* 21(6):73–85, 2006.

[5] C. Heitmeyer and R. Jeffords., "Applying a formal requirements method to three NASA systems: Lessons learned", Naval Research Laboratory Research Report #07-1226-0320, *Proceedings of the 2007 IEEE Aerospace Conference*.

[6] C. Heitmeyer, M. Archer, E. Leonard and J. McLean, "Applying formal methods to a certifiably secure software system", IEEE Transactions on Software Engineering, Vol 34 No 1, Jan-Feb 2008, pp 82-98

[7] G. Berry, "Synchronous design and verification of critical embedded systems using SCADE and Esterel", *Proceedings of the Formal Methods for Industrial Critical Systems*, Lecture Notes in Computer Science, vol. 4916. Springer-Verlag, Berlin, Heidelberg Germany.

[8] J. Fitzgerald, P. G. Larsen and S. Sahara, "VDMTools: Advances in support for formal modeling in VDM", SIGPLAN Notices 43, 2 (Feb.), 3–11.

[9] M. Daum, N. Schirmer and M. Schmidt, "From operating-system correctness to pervasively verified applications", Integrated Formal Methods - IFM 2010 6396 (2010) 105-120.

[10] J. Barnes, High Integrity Software: The SPARK Approach to Safety and Security. Addison-Wesley Publishing, 0-321-13616-0, © 2003.

[11] http://www.adacore.com/2010/08/16/spark-skein/

[12] J. Barnes and R. Chapman, "Engineering the tokeneer enclave protection software", Proceedings of the 1st IEEE Symposium on Secure Software Engineering (2006).

[13] J. Woodcock et al.: Formal methods: Practice and experience. ACM Comput. Surv. 41, 4, Article 19 (October 2009), 36 pages.

[14] R. Sward, M. Carlisle, B. Fagin and D. Gibson, "The case for Ada at the USAF Academy", Proceedings of the ACM SIGAda International Conference on Ada pp 68-70 (2003).

[15] DNSSEC – The DNS Security Extensions, http:// http://www.dnssec.net/

[16] P. Mockapetris, "Domain Names – Concepts and Facilities", RFC 1034, November 1987.

[17] P. Mockapetris, "Domain Names –Implementation and Specification", RFC 1035, November 1987.

[18] P. Vixie, "Extension Mechanisms for DNS (EDNS0)", RFC 2671, August 1999.

[19] Zytrax Inc., "DNS for Rocket Scientists", http://www.zytrax.com/books/dns/.

[20] S. Conchon, E. Contejean and J. Kanig, "Ergo : A theorem prover for polymorphic first-order logic modulo theories", 2006. Available from alt-ergo web site at http://alt-ergo.lri.fr/papers/ergo.html.

[21] Nominum, Inc. How to Measure the Performance of a Caching DNS Server. Available online at http://www.nominum.com/wp-content/uploads/2010/08/caching-performance.pdf.

[22] Comparison of DNS Server Software, http://en.wikipedia.org/wiki/