The IRONSIDES Project: Final Report

Barry S. Fagin and Martin C. Carlisle

Dept of Computer Science, US Air Force Academy 80840 Tel: 1-719-333-3338; email: barry.fagin@usafa.edu.

Abstract

In a project intended to improve the security of internet software, the authors developed IRONSIDES: A DNS server written in Ada/SPARK. Our long-term goals were a) to show that a fully functional component of the internet software suite could be written with provably better security properties than existing alternatives, b) to show that it could be done within the relatively modest re-sources available for a research project at an undergraduate university, c) to determine the suitability of Ada/SPARK for such a project, and d) to compare the performance of the resulting software to existing alternatives and determine to what extent, if any, the addition of provable security properties affects performance. We report our conclusions from this multi-year project.

Keywords: Ada, DNS, formal methods, internet software, performance analysis, SPARK.

1 Introduction

The Domain Name System (DNS) is the internet protocol that transforms hostnames (e.g. cnn.com) into IP addresses (e.g. 151.101.0.73). Originally proposed by Mockapetris in [1], it is a distributed database protocol that uses the internet as a tree structure to manage records containing information about machine names and properties.

Software that implements this protocol is referred to as a DNS server. This term can also describe the machine that runs a DNS server. Servers responsible for resolving names in a single zone (typically a company, university, or similarly scoped institution) are called authoritative servers. If queried about names outside the zone for which they are responsible, authoritative servers reply with a failure message, the equivalent of “I don’t know”.

Servers capable of resolving names for any publicly visible machine on the internet are called recursive. They use a recursive process to travel up the distributed internet tree structure to determine the name of the machine in question. In modern DNS practice, most recursive solvers do not use full recursion to traverse the name tree. Instead, they refer queries to a publicly available fully recursive DNS server (for example, Google’s public DNS at 8.8.8.8), and then cache the result for future use.

DNS is a vital internet protocol. Unfortunately, because it dates from the early days of networking, it contains security flaws that require mitigation to prevent malicious actors from exploiting the system [2]. Additionally, most DNS software is written in older languages with inherent security problems. These languages do not lend themselves to rigorous software design and provable security properties. The two most popular DNS servers, BIND and WINDNS, have a large number of known security flaws, including crashing in response to the injection of bad data and bugs that permit remote execution [3], [4]. These are described in more detail in the sections that follow.

2 The IRONSIDES Project

The authors believed many of the security problems with DNS servers, web servers, and other internet software could be avoided with the use of better programming tools, such as the use of different programming languages and formal methods. They chose Ada and SPARK as an appropriate development environment to implement a provably secure DNS server from the ground up.

The SPARK language and toolset from Altran UK is used in the creation of software systems with provable correctness and security properties [5]. 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 validate software. SPARK is a mature technology and has been used on several projects, including an open-source OS kernel provably free from runtime errors [6], the British Air Traffic Control System [7], and multi-level security workstations [8]. Accordingly, given our prior institutional experience with Ada [9], 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 world.

The SPARK toolset generates verification conditions (VC’s) that it then attempts to verify. VCs include assertions that variables always remain in type, array bounds are never exceeded (a common source for buffer overflow vulnerabilities), pre- and post- conditions are always met, and so forth. When a VC has been proved by SPARK, it is said to be discharged.

2.1 Milestone 1: An authoritative server on Ubuntu

The first IRONSIDES milestone was achieved with the successful construction of an authoritative server, tested against BIND on Ubuntu [10]. The original test bed and performance results are shown in Figure 1 and Figure 2, respectively.

We were pleasantly pleased to discover that the authoritative IRONSIDES DNS server performed significantly better than BIND under Linux.

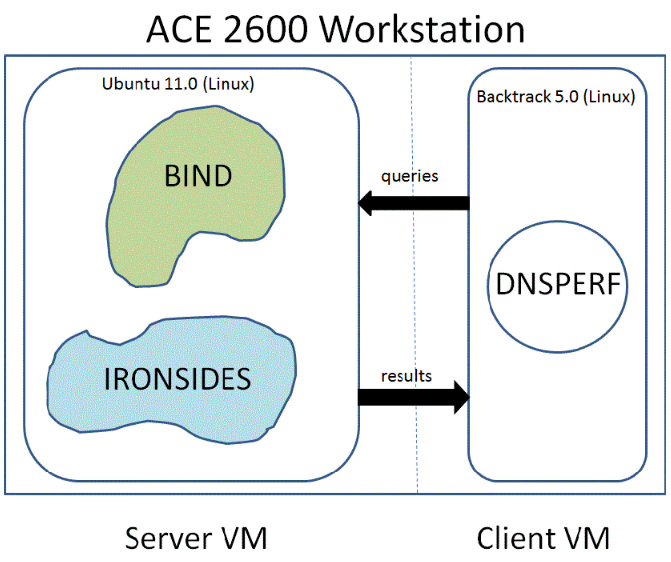

Figure 2 IRONSIDES original test bed for authoritative servers

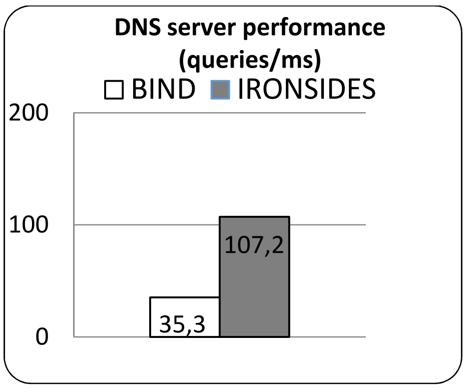

Figure 3 BIND and IRONSIDES performance under Linux

2.2 Milestone 2: An authoritative server on Windows

The next milestone was porting IRONSIDES to Windows, and testing it against both WinDNS and BIND [11]. The test bed was similar, except the virtual machine used ran Windows Server 2008. Performance results are shown below:

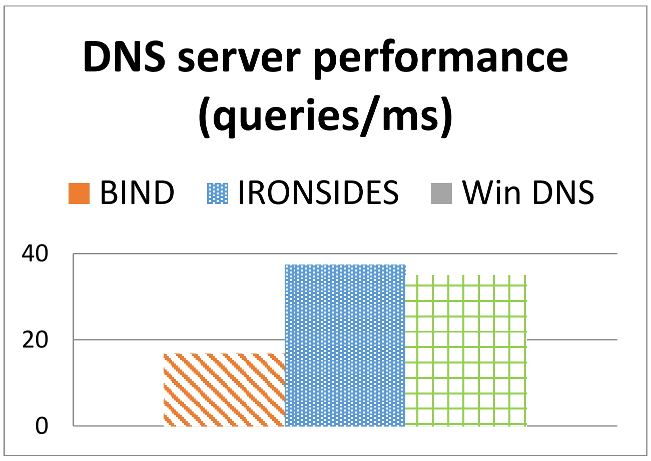

Figure 4 Performance comparison on Windows

We fully expected IRONSIDES to perform better than BIND, but were surprised to find it outperformed Windows DNS on its own native OS by 7%.

At this point, the proof requirements of IRONSIDES looked like this:

Table 1 Proof requirements of IRONSIDES authoritative

	Total	Simplifier	Victor	Examiner
Assert/Post	3106	2209	884	13
Precondition	561	0	532	29
Check stmt.	12	0	12	0
Runtime check	3750	0	3704	46
Refinement. VC s	44	42	2	0
Inherit. VCs	0	0	0	0
Totals:	7473	2251	5134	88
%Totals:		30%	69%	%

Victor invokes an optional theorem-prover to discharge VC's that the first two stages of the tools (the Examiner and the Simplifier) cannot.

2.3 Milestone 3: A recursive server and detailed performance comparisons

Recursive servers are more complex than authoritative ones, requiring more sophisticated data structures, cache management, and tasking. Building on our experience with the authoritative version, we next added recursive query functionality to IRONSIDES. The resulting basic structure (little changed to the present day) is shown in Figure 4.

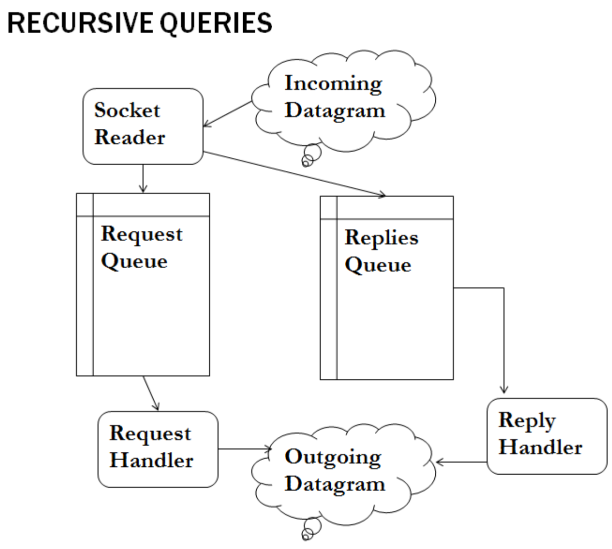

Figure 5. IRONSIDES recursive service structure

This structure was implemented with the modules and data dependency relationships shown in Figure 5.

Lines indicate a data dependency, transitive dependencies are implied. Their functions are:

- spark_dns_main: Top-level executable.
- udp_query_task: Concurrently executing task responsible for all incoming DNS traffic.

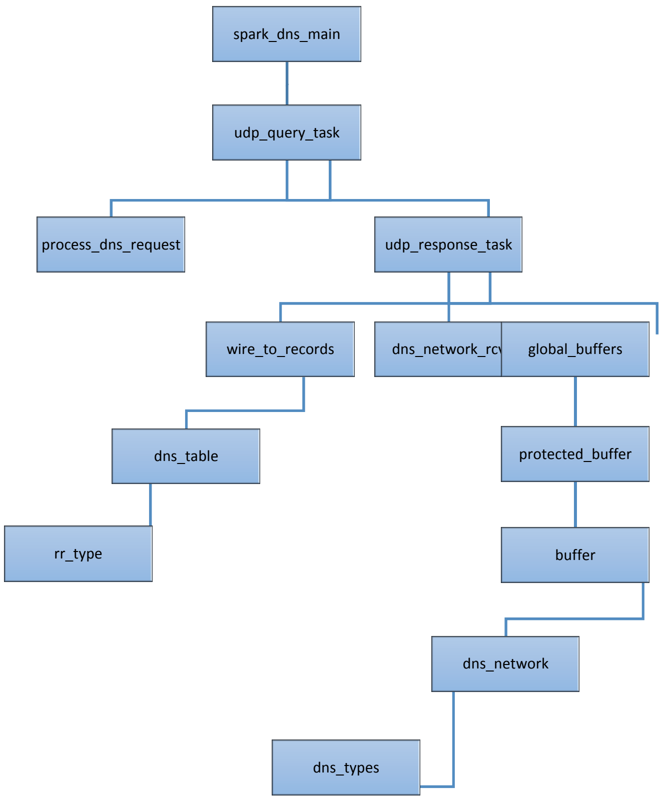

Figure 5. IRONSIDES recursive service structure implementation

- udp_response_task: Concurrently executing task responsible for managing all responses from upstream servers.
- process_dns_request: Interprets incoming packet, queries DNS table, queues query if answer not found.
- wire_to_records: Builds DNS resource records from DNS packets on the wire.
- dns_network_rcv: SPARK wrapper for network traffic to guarantee no overflows.
- global_buffers: Query and response queues.
- protected_buffer: ADT for the query and response queues.
- buffer_pkg: ADT for a queue.
- dns_table: Cache of DNS resource records.
- rr_type: Top-level package for all DNS resource record types.
- dns_network: Handles low-level network IO.
- dns_types: Data types for working with DNS packets.

The proof requirements for the recursive version were:

	Total	Exam.	Simp.	Victor
Assert/Post	3510	2248	1194	68
Precondition	641	0	609	32
Runtime check	9705	0	9502	203
Refinem. VCs	98	98	0	0
Totals:	13954	2346	11305	303
%Totals:		17%	81%	2%

For static code size, we measured the following:

Total Lines	14448
Blank Lines	1268
Comments	4142
SPARK Lines:	1713
Ada lines	9038
Ada statements	6917
SPARK statements	806

Once we had produced a validated recursive server, we were ready to do a detailed performance comparison with a variety of both open-source and proprietary DNS servers [12]. As shown in Figure 6, 2e expanded the test bed to include a virtual machine running each server/OS combination, a VM running the Resperf performance analyzer [13], and a VM running the network simulator INETSIM [14].

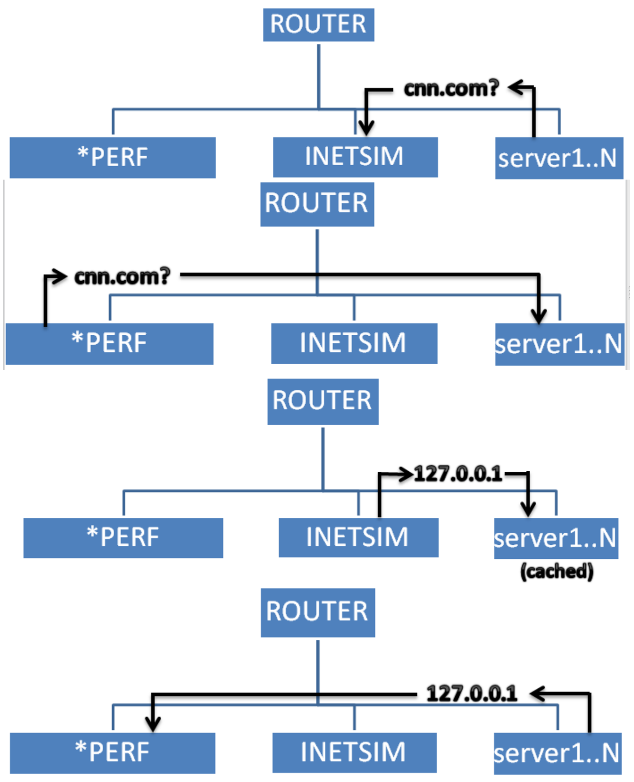

Figure 6 Recursive server test bed

When we ran the server in authoritative mode under Ubuntu, IRONSIDES continued to outperform BIND and others, although the gap had narrowed from about 3x to about 2x, as shown in Figure 7.

Figure 8 shows how under Windows, however, WinDNS now performed slightly better, perhaps due to improvements in later releases or the increased complexity of IRONSIDES required to support recursive queries.

The number of queries handled as a function of increasing requests per second for recursive servers is shown in the two charts included in Figure 9.

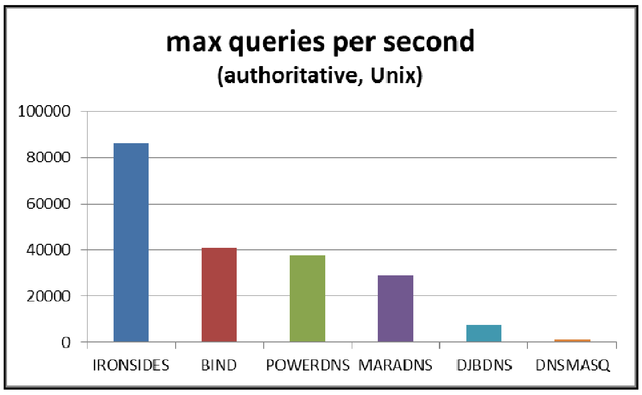

Figure 7 Performance comparison of authoritative DNS servers under Unix

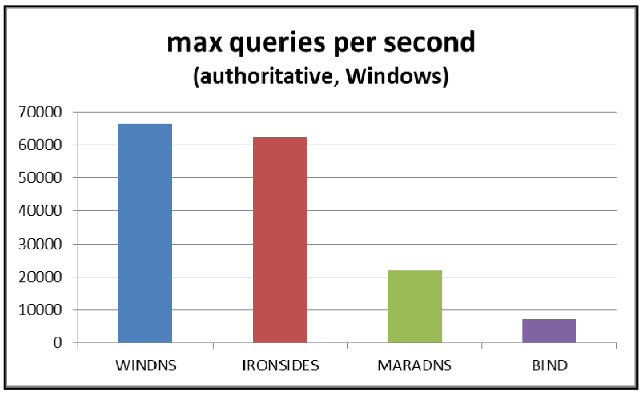

Figure 8 Performance comparison of authoritative servers under Windows

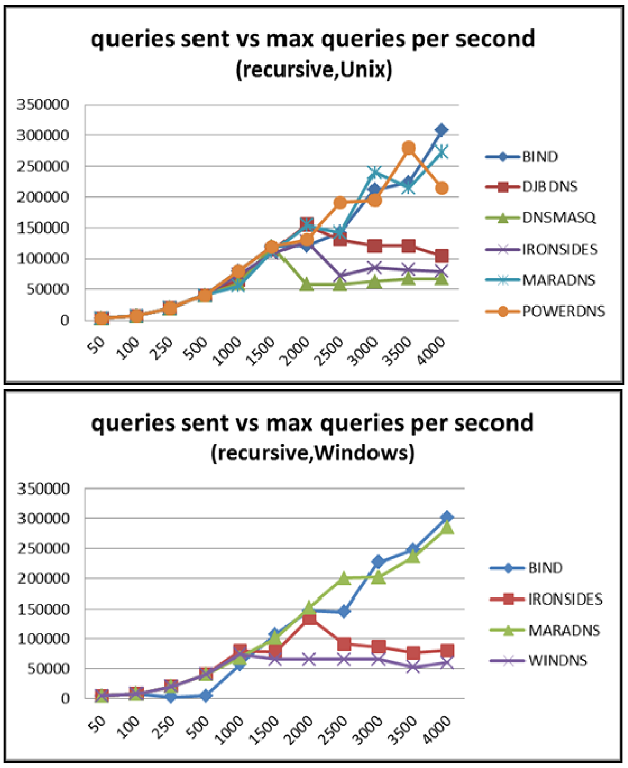

Figure 9 Performance of recursive servers

Up to 1500 queries per second, the performance of all the servers is essentially indistinguishable. At higher values, IRONSIDES, DNSMASQ and DJBDNS dropped off fairly rapidly. Surprisingly, under Windows, BIND also did the best.

On the other hand, in terms of queries lost, WinDNS and IRONSIDES performed best, as shown in Figure 10.

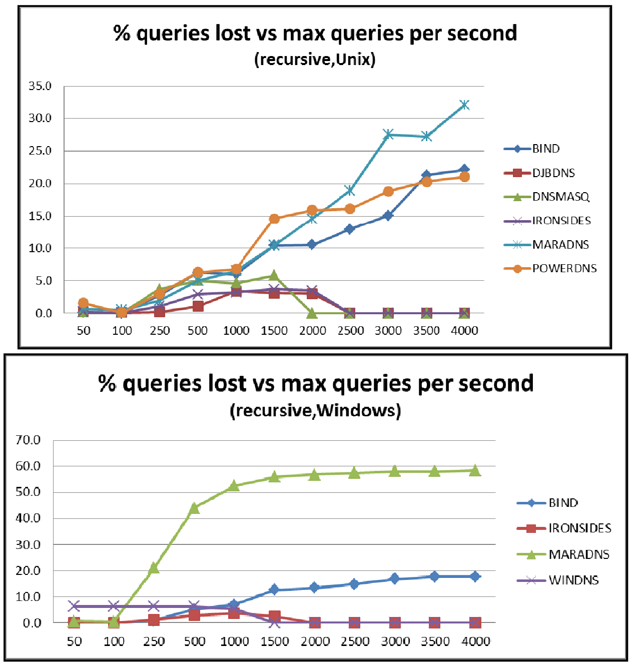

Figure 10 Queries lost by DNS servers

IRONSIDES had the second lowest latency for Unix DNS servers, but the longest latency for Windows servers. We believe this is due to latency being extremely important to Microsoft, and to IRONSIDES policy of trying to handle every query it can (BIND, by contrast, drops queries if it is too busy):

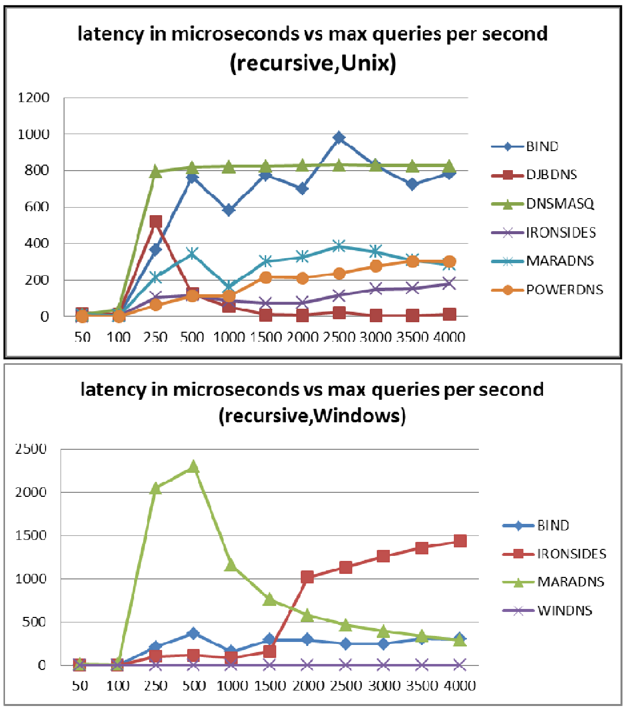

Figure 11 Latency of DNS servers

For further details, the reader is referred to the references.

3 Insights from experience

The results from the authoritative server design process gave our first hints that performance did not need to be sacrificed to improve security. In fact, there were clear examples in which the use of formal methods actually improved performance. For example, data flow analysis identified redundant or ineffective statements that in turn permit the removal of inefficient code. Code that has been proven exception-free no longer requires runtime bounds checking, so that can be eliminated as well.

We also learned, however, that there were cases were total reliance on formal methods and proof negatively impacted performance. Because SPARK requires all data structures to be statically allocated, data structures must be sized at the upper limits of expected use. Explicit initialization of such structures, while required for validation, is inefficient and wasteful. In those rare cases, we explicitly told the tools to relax that requirement. This improved IRONSIDES performance by almost 30%. Thus we believe allowing users to override formal proof requirements when appropriate is an important feature that formal methods tools should always support.

It is crucial to always remember the role of the compiler. Despite our confidence in the tools to help us produce crash-proof software, we found one combination of operating system, compiler and optimization level where a fully validated version of IRONSIDES crashed with an exception. This was due to a code generation error in the version of GNAT shipped with Ubuntu, long since corrected. Still, until formal methods have progressed sufficiently to the point where they can prove the correctness of compilers for a given target architecture and OS, programmers should continue to exercise healthy skepticism when compiling and testing verified software.

Our experience with the tools produced results we would describe as both impressive and humbling. Despite both of us having computer science PhD's, over 50 years of combined industry and academic experience, and an extensive knowledge of programming languages and software engineering practice, the tools still caught boundary conditions and potential problems that in principle we could have found but did not. This is the whole point of using formal systems, but the experience is nonetheless humbling. Perhaps it will become less so as formal methods and proof tools become a standard part of the software engineering process.

IRONSIDES has numerous provable security properties absent from all the other servers tested, including BIND and WinDNS. These include:

1) No classic 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

With all these advantages, we were pleased to discover that IRONSIDES also performs comparably to servers with security problems, including the industry standards of BIND and WinDNS. IRONSIDES offers comparable performance at nominal loads, trailing off only under maximal loading. This is particularly significant considering each server's respective development costs. BIND is produced with an industrial consortium. WinDNS is bundled with the flagship product of a multibillion dollar software company. IRONSIDES was written by the equivalent of a little more than one professor at an undergraduate university with near full-time teaching duties.

4 Conclusions and future work

The success of IRONSIDES indicates that formal methods can be used both improve the security properties of software without incurring significant performance penalties, and in some cases can actually improve performance. This was done in an environment with significantly fewer resources available than comparable products.

Why then are similar approaches not more widely adopted? Existing products have greater sunk costs. In all fairness to them, they also offer more functionality and a better support base, options that are not available to the time and resource constrained environments present in the academic development of prototype software. Programming with formal methods also requires the use of a relatively unfamiliar language (particularly in the United States) as well as comfort with mathematical logic and proof. Most software engineers are not yet trained to work this way. Perhaps, however, over time this will change as the advantages of formal methods and proof are shown to produce more reliable software that keeps a company's name out of the newspapers.

Since the latest stable release of IRONSIDES, Ada has added more features that meld it more tightly with SPARK. We hope to upgrade IRONSIDES in the future to support these features and to examine their effects.

We hope this work will be further extended to apply formal methods and performance analysis outside the DNS domain, in the hopes of continued confirmation that internet software can be made provably more secure without significant sacrifices in performance. Web servers, for example, suffer from similar security problems for similar reasons. ICS and SCADA systems are currently attractive targets for hacking, and formal methods have been used to improve their security [30], but the effect of formal methods on performance in this domain remains unknown. These are the subject of current work at the Academy Center for Cyberspace Research.

Acknowledgments

This work was supported in part by the US Air Force Office of Scientific Research under grant #1220961, the US Department of Defense Advanced Research Projects Agency, and the Academy Center for Cyberspace Research.

References

[1] https://tools.ietf.org/html/rfc882.

[2] Carnegie Mellon University Software Engineering Institute (2008), *Multiple DNS implementations vulnerable to cache poisoning*, available online at http://www.kb.cert.org/vuls/id/800113.

[3] Internet Security Consortium, BIND *9 Security Vulnerability Matrix*, available online at https://kb.isc.org/article/AA-00913/0/BIND-9-Security-Vulnerability-Matrix.html

[4] https://technet.microsoft.com/library/security/MS15-127

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

[6] R. Buerki and A. Rueegsegger (2013), *Muen - an x86/64 separation kernel for high assurance*, Technical Report, University of Applied Sciences Rapperswil (HSR), Switzerland. Available on line at http://people.cs.ksu.edu/~danielwang/Investigation/Formal_Verification/muen-report.pdf.

[7] AdaCore (2017), *GNAT Pro chosen for UK's next generation ATC system*, AdaCore Technologies press release, available online at http://www.adacore.com/customers/uks-next-generation-atc-system/.

[8] AdaCore (2010), *Spark PRO adopted by secunet*, AdaCore Technologies press release, available online at http://www.adacore.com/customers/multi-level-security-workstation/

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

[10] M. Carlisle and B. Fagin (2012), *IRONSIDES: DNS with no single-packet denial of service or remote code execution vulnerabilities*, GLOBECOMM 2012, Anaheim CA.

[11] B. Fagin and M. Carlisle (2013), *Provably secure DNS: A case study in reliable software,* 2013 International Conference on Reliable Software Technologies, Berlin, Germany pp 81-93.

[12] B. Fagin, M. Carlisle and B. Klanderman (2017), *Making DNS Servers Resistant to Cyber Attacks*, COMPSAC 2017, Turin, Italy pp 566-571.

[13] http://www.nominum.com/measurement-tools/

[14] Internet Services Simulation Suite, http://www.inetsim.org/

[15] J. Groote, A. Osaiweran and J. Wsesselius (2011), *Analyzing the effects of formal methods on the development of industrial control software*, 2011 IEEE Conference on Software Maintenance, Williamsburg VA, pp 467-472.

[16] H. Boulakhrif (2015), *Analysis of DNS resolver performance measurements*, Masters' Thesis, University of Amsterdam, available at https://www.nlnetlabs.nl/downloads/publications/os3-2015-rp2-hamza-boulakhrif.pdf.