1 Outline

1. Intro
   - ASN.1 intro and example
   - Mention some common problems with ASN.1 parsers
2. Approaches to verification of parsers
   - Functional parsers (Narcissus)
   - Extraction to C, partial correctness (Everparse, Galois)
3. Example of strtoimax of asn1c
   - Operational semantics proof (CompCert, C semantics)
   - VST proof
   - Bugs discovered
   - Lessons learned
4. Current work

2 Intro

ASN.1 intro and simple example

Some common problems with ASN.1 parsers

The Computer Vulnerabilities and Exposures (CVE) database [7a] lists critical ASN.1-related bugs found each year in the existing systems, and there have already been noteworthy exposures [8] that although not as dire to security as first feared [9] clearly spell out a first awareness of the vast risk and exposure. We analyzed the last 4 years of ASN.1-related issues reported in

Computer Vulnerabilities and Exposures (CVE) database [7b]. Among vulnerabilities studied were CVEs for various software and hardware products and vendors, including Apple, axTLS, Botan, Bounty Castle, librcrypto++, libtasn, LibTomCrypt, Linux Kernel, MatrixSSL, Mozilla NSS (Firefox), Objective Systems, OpenSSL, PolarSSL, RSA BSAFE, Samba, Samsung, Snapdragon, strongSwan, and Wireshark. It was found that 39 out of 52 problems analyzed were related to memory safety, 6 related to stack and heap bounds checking, and 3 related to issues caused by applications accepting not well-formed ASN.1 input. Having proved just six formal properties would have prevented 49 out of 52 vulnerabilities, that is, more than 90

3 Approaches to verification of parsers

Functional parsers (Narcissus)

Extraction to C (Everparse, Galois)

Project Everest [19] is our most direct thematic competition, and some of the high-level Project Everest documentation makes passing mention to ASN.1. The Project Everest stewards adopted a different approach based on F* [4], and the Project Everest ASN.1 work appears at best still in far distant plans.

It is also noted that Galois did some work on ASN.1 verification in the past (circa 2012) [11]. It appears that Galois abandoned [12] the goal of full ASN.1 verification that we pursue in our project with our more pragmatic approach; Galois is now only exploring a limited subset ASN.1 verification adequate for the vehicle-to-vehicle (V2V) market [13], but that particular subset has limited broader applicability and the Galois effort appears encumbered by aspects of an unsuccessful approach. It is noted that although our goal is eventually to verify all ASN.1, we decided to start with a different (X.509-related) subset of ASN.1; this subset is reasonably small, but X.509 is so widely used that our initial verified implementation of our chosen subset will have a large volume and wide range of immediate commercial applications.

4 Our approach

Our technology and approach will provide an entirely new level of software and protocol verification employing recently-emergent provable full-functional correctness methods. Systems and methods to date either (a) automatically tested but not formally verified (b) use verification approach which rely on automatic extraction from executable specifications (for example involving network stack synthesis [31], optimizing compilers [3], cryptographic libraries [6], and encoder/decoders [32]), or (c) apply a form of formal verification which only proves partial correctness properties (partial verification of NAT stack only proving parts of DPDK are specification compliant [33], partial verification of Linux kernel TCP implementation with 55% line coverage and 92% protocol coverage [34]). Consequently, (a) and (c) do not provide sufficient correctness guarantees, while (c) is often impractical due to poor performance and compatibility limitations. In contrast, we pursue a far deeper and comprehensive verification approach to performance and portability and seek to prove actual industrial-level C-code implementation.

4.1 Veryfing a function of asn1c: strtoimax

- Example of strtoimax of asn1c
- Operational semantics proof (CompCert, C semantics)
- VST proof
- Lessons learned

Future work Formalize the X.509 part of the ASN.1 standard.

Refactor ASN1C code to make it suitable for verification.

Establish correctness of encoders/decoders for primitive types.

Establish correctness of encoders/decoders for constructed types.

Prove high-level properties.

Experiment with OCaml code extraction from Executable Specifications.

Establish metrics and evaluate code bases to estimate the effort required to prove the remainder of ASN.1 stack.

Produce the final ASN1C code and associated documentation in the form of commercial product that could be sold.