







TRUST IN SOFT

Applying formal methods to existing software: what can you expect?

Benjamin Monate

Co-founder of TrustInSoft and CTO

6/27/2018

Sound Static Analysis
aka formal methods to prove properties of software

Works for Safety Critical Software

- Consequences of failures are analyzed from the beginning
- Regulation is strong: standards and associated liability
- Adapted development process: specific languages/dev. cycles
- Software errors mitigated with system architecture
  → considering probability of failure

Sound Static Analysis for Security Critical Software?

- Meanings of failure probability?
  → adversary defies standard distributions of the software input
- One single error → arbitrary corruption
- Confidentiality: secrets must not escape software
- Software already deployed in production: barely tested for security
  - Because testing security is hard: looking for behaviors that have undefined consequences but are **most of the time invisible**
  - Observing a **data leak is difficult**: where shall it be observed? How shall one recognize that some bits are part of a secret?

TRUST IN SOFT pragmatic and incremental security

Confidence Level	Property	Tool	Guaranteed properties
Level 1	Each compilation unit compiles	Compilers with warnings	Static typing and syntactic compliance
Level 2	Integrity of link	Sound Source Linker	Consistency of compilation units (ODR/static inline/weak)
Level 3	Only defined behaviors	Sound Static Analyzer	Absence of undefined behaviors/Integrity Compilers optimizations makes the consequences more and more dangerous
Level 4	Dataflow integrity	Sound Static Analyzer	Absence of unwanted data flows/Confidentiality
Level 5	Functional correctness	Sound Functional Verification	Program fulfills its functional specification

Each level requires some previous ones to be meaningful
TrustInSoft Analyzer addresses Level 2 up to Level 5

How to reach these levels on legacy code?

- **Level 1** for free
- **Level 2** automatic with TrustInSoft Analyzer: just provide all source files

  May be detected by modern binary-level LTO

  Reduced example from Xen

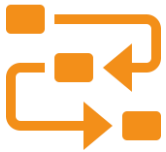
```
file1.c: int GlobalConfig[255] = { 0 };
file2.c: extern int *Globalconfig;
```

- **Level 3** not easy to get because

  - Soundness: false alarms, **not the most important problem**

  - **Programs contain bugs**: must be fixed to give semantics

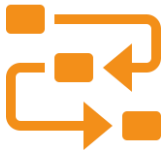
Methodology toward Level 3 security



Do not explore all execution paths at once, but

- **Explore** simple path: rely on existing test-suite
- **Fix** all discovered bugs
  - Unlike testing: detect invisible undefined behaviors
    → Invisible but may hide security bugs thanks to compiler optimization/platform specificities
  - No need to be fully deterministic: external functions/hardware are stubbed
  - Use an existing test to reach some difficult-to-reach program points (after SSL certificate validation) and then invent new tests by mutating the input data that do not change the initial paths (fuzzing, manual testing)
  - Generalize the tests progressively → **fix bugs one after the other**
  - Maybe one reaches a state where all behaviors are covered
    → But if one does not, the security is still vastly improved, **step-by-step**

How long does it take to get a proof of absence of undefined behaviors?



- Major industrial question: ROI, Time To Market, Total Cost of Ownership

- Important but flawed question:
  - It takes the time that one needs to fix all the discovered bugs
  - No one knows how to evaluate this soundly

- Cyber-security is incremental
  - **Soundness does not mean**: "all questions **answered**"
  - **Soundness does mean: "some questions answered definitively"**

Not necessarily "the" whole question

Examples: tooling funded for zero false positive and zero false negative source code analysis

THE LINUX FOUNDATION

CORE INFRASTRUCTURE INITIATIVE

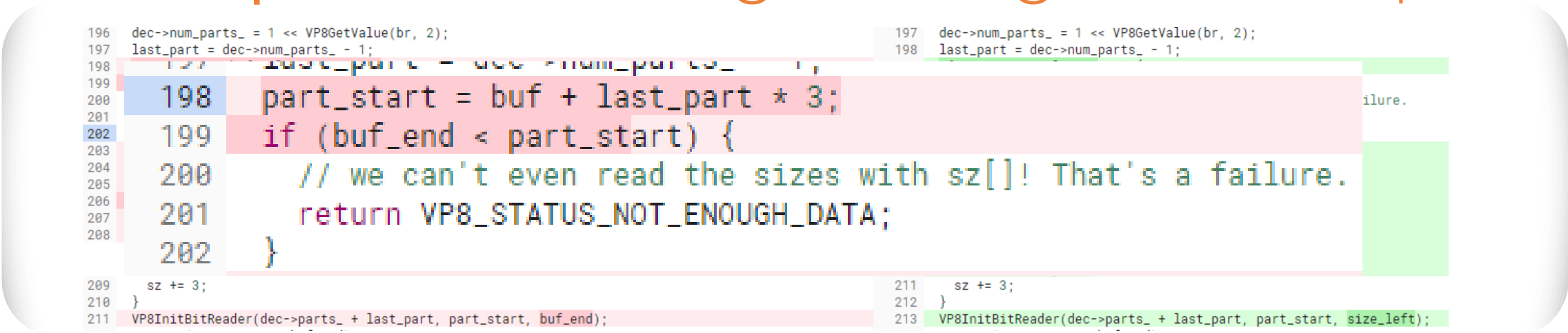
OpenSSL, Amazon S2N, Google Libwebp, expat, libpng, SQLite, musl, libjpg, libsodium, LibreSSL, tiny ssh, libxml, zlib, ntpd, libbzip2, dpdk, nova, libksba

Hundreds of security bugs discovered: most of them fixed upstream

- Initial analysis: existing test-suites
- Further analysis: AFL fuzzing
- Next steps: generalized input to reach more behaviors

Invalid memory accesses, signed overflows, uninitialized data, double free, strict aliasing violations, constant execution time...

Examples: subtle bug in Google's libwebp



```
196  dec->num_parts_ = 1 << VP8GetValue(br, 2);
197  last_part = dec->num_parts_ - 1;
198  part_start = buf + last_part * 3;
199  if (buf_end < part_start) {
200    // we can't even read the sizes with sz[]! That's a failure.
201    return VP8_STATUS_NOT_ENOUGH_DATA;
202  }
209  sz += 3;
210  }
211  VP8InitBitReader(dec->parts_ + last_part, part_start, buf_end);
```

```
197  dec->num_parts_ = 1 << VP8GetValue(br, 2);
198  last_part = dec->num_parts_ - 1;
                                                  ilure.
211    sz += 3;
212  }
213  VP8InitBitReader(dec->parts_ + last_part, part_start, size_left);
```

- Invalid pointer computation: invisible UB
- Followed by invalid pointer comparison
  - result depends on memory layout
  - If the result is wrong, out-of-bound access occurs

- LLVM ASan statistically uses the memory layout without consequences

- TrustInSoft Analyzer's soundness means: all memory layouts are explored

Full Level 3 is reachable

arm MBED POLARSSL

Proof of absence of UBs for some configurations of mbed TLS

Read the full technical report at https: //trust-in-soft.com/polarssl-verification-kit/

Good news for cyber-security

- These examples are the **most difficult** software to analyze
  - Huge legacy, multi-purpose code bases
  - No developer was involved in the analysis: only bug reports
  - Time to convince developers/maintainers that fixing issues is important

- **And still: it works!**
  Security is improved: **fewer bugs** and **unmodified dev. process**

- In the industry, **this is much simpler**!

Level 4 : Dataflow integrity

Example: look for the sources of random numbers in OpenSSL

Explicit security property: **Random generators seeds are acceptable**

- Customer knows what "acceptable" means
- Tools can extract the origin of the data: sound means **exhaustive**

  Findings: a **dozen of sources are used**, including the private certificate

  Customer conclusion: we **must configure the stack** to avoid this

Classical security analysis: define attack surfaces and implement proper mitigations

This works on the source code, if security expressed in terms of programs behaviors

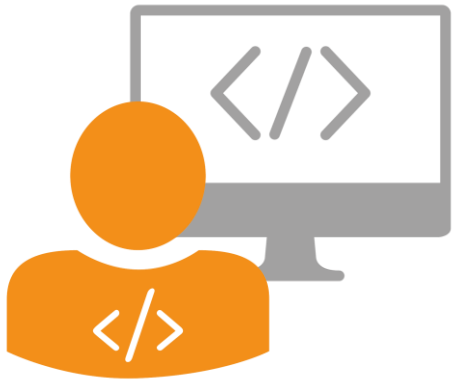
Level 5: full specifications for all functions

Kind of a **Grail** for program correctness

- We support this usage
- Impacts the dev. Cycle:
  - Produce specifications
  - Check specifications

  → **software developed to make it provable** maybe with dedicated languages/methodology
- Adopted only for very specific parts of **very specific safety/security critical software**

Conclusion



- **Soundness** of tools is a definitive improvement for security
- Do not try to reach the highest integrity levels **instantly**
- Stopping in the middle of any level is worth it

  → one **reduces** its hidden technical debt
- Difference with unsound tools

  → each step is a **definitive improvement for security**

  →**When it is done, it is for real**

Thank you

Benjamin.Monate@trust-in-soft.com

TRUST IN SOFT

Mathematical Guarantees Eliminate Software Risk