Input Handling Done Right: Building Hardened Parsers using Language-theoretic Security

Prashant Anantharaman*, Michael C. Millian*, Sergey Bratus
Dartmouth College
Hanover, NH 03755
{pa, mcm, sergey}@cs.dartmouth.edu

Meredith L. Patterson
Upstanding Hackers, Inc.
Brussels, Belgium
mlp@upstandinghackers.com

***Abstract*—Input-handling vulnerabilities have been a constant source of security problems for decades. Many famous recent bugs are in fact input-handling bugs. We argue that the techniques for writing parsers in its present form are insufficient, and hence we propose a new pattern. In this tutorial, we will show participants a new design pattern for designing and implementing parsers using this new method. Participants will witness how this new method leads to more readable code that is easier to audit - while also inherently preventing many input-handling mistakes and having a small CPU footprint.**

I. Introduction

Parsing input strings is one of the earliest-studied problems in computer science [10]. All modern software programs involve parsing input, and yet alarmingly little care is given to this task despite how much is known about the topic. Whenever someone writes a parser, they implicitly declare a formal language they want to accept. However, too often the intended language and the actual language differ. This input validation problem is the source of countless vulnerabilities, including HeartBleed [1], [5], [6]. The problem of insufficient input-handling has been studied in the Internet-of-Things [3], Android applications [11] and extensively in the DNP3 power grid protocol [4].

Language-theoretic Security (LangSec) identifies several anti-patterns ubiquitous in the software development process that directly cause classes of vulnerabilities [6]. These anti-patterns include:

- Shotgun Parsers - code in which parsers are interspersed with code that does processing of input.
- Overly Complex Parsers - where the computing power of the parser exceeds the complexity of the input language.
- Overly Complex Input Languages - input languages must be deterministic context-free to avoid parser differentials and ensure that parsers are equivalent.

A. Hammer parser-combinator toolkit

Hammer is a tool, written in C with bindings for other languages, that tackles the problem of writing correct parsers[8]. *Hammer* implements combinators taken from formal language theory such as character matching and kleene star. *Hammer* code looks like a grammar, so it is easy to check that the implementation matches the intent.

*Prashant Anantharaman and Michael C. Millian contributed equally to this paper, and will be leading the tutorial.

II. Summary of Tutorial

In this tutorial, participants will build secure parsers for several protocols using the *hammer* parser-combinator toolkit [8]. Users will then fuzz the parsers they have build using *libfuzzer* [9]. This process will demonstrate the resilience of the LangSec methodology. The protocols we cover come from a set including power grid protocols, like Modbus and ICCP, and others that are widely used, such as DNS, Base64, JSON, MQTT or XMPP.

We will follow this format for our tutorial:

- Provide an overview of LangSec and the *hammer* parser-combinator toolkit.
- Build a parser for one of the above mentioned protocols and fuzz-test it.
- Ask participants to build parsers on their own for another protocol, providing help as needed.
- As a conclusion, show pre-written parsers for the participant-chosen protocols and discuss parser differentials if any different implementations are seen.

III. Expected Audience and Learning Outcomes

The *hammer* parser-combinator has bindings in various languages. We intend to cover C, Python, and Ruby in our tutorial. The audience must be familiar with one of these languages. Further, knowledge of basic linux shell commands and a laptop with an *Ubuntu 14.04 – 64 bit VM* is required.

The audience will build parsers for protocols on their own from scratch. Then, they will fuzz their parsers with *libfuzzer* [9], to identify mistakes made in the implementation.

The audience will gain understanding and experience programming with a new design pattern. They will think critically about protocols and parsers for protocols. As a result, they will be able to better handle input by treating all input as formal languages.

IV. Prior Talks

Prashant has previously given two talks on the use of Language-theoretic Security. Prashant presented “Building Hardened Implementations of SCADA/ICS Protocols Using

Language-Theoretic Security" at the *CREDC Industry Workshop, Tempe, AZ* on the 28th of March, 2017 [2]. Prashant also presented his work on "Building Hardened Internet-of-Things Clients with Language-theoretic Security" at the *LangSec Workshop at IEEE Symposium on Security and Privacy* [3]. Both talks had 50-100 attendees and were well received.

V. Conclusion

We believe that this tutorial will be a good fit for the Secure Software Development community as the LangSec framework aims to eradicate categories of vulnerabilities arising from input-handling vulnerabilities. We hope to make use of this tutorial to train more people to write better parsers, which will prevent vulnerabilities from occurring in the future.

Acknowledgment

This material is based upon work supported by the Department of Energy under Award Number DE-OE0000780.

References

[1] Cve-2014-0160. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160, 2014.

[2] Prashant Anantharaman. Building Hardened Implementations of SCADA/ICS Protocols Using Language-Theoretic Security. https://publish.illinois.edu/2017credciw/agenda/showcase-credc-research/, March 2017. CREDC Industry Workshop.

[3] Prashant Anantharaman, Michael Locasto, Gabriela F Ciocarlie, and Ulf Lindqvist. Building Hardened Internet-of-Things Clients with Language-theoretic Security. In *IEEE Symposium on Security and Privacy Workshops (SPW)*, 2017.

[4] Sergey Bratus, Adam J Crain, Sven M Hallberg, Daniel P Hirsch, Meredith L Patterson, Maxwell Koo, and Sean W Smith. Implementing a vertically hardened dnp3 control stack for power applications. In *Proceedings of the 2nd Annual Industrial Control System Security Workshop*, pages 45–53. ACM, 2016.

[5] Zakir Durumeric, James Kasten, David Adrian, J Alex Halderman, Michael Bailey, Frank Li, Nicolas Weaver, Johanna Amann, Jethro Beekman, Mathias Payer, et al. The matter of heartbleed. In *Proceedings of the Conference on Internet Measurement Conference*, pages 475–488. ACM, 2014.

[6] Falcon D Momot, Sergey Bratus, Sven Hallberg, and Meredith L Patterson. The seven turrets of babel: A taxonomy of langsec errors and how to expunge them. In *IEEE Cybersecurity Development Conference (IEEE SecDev)*, 2016.

[7] Meredith Patterson, editor. *Hammer Tutorial*, May 2017. Hackathon preceding the LangSec Workshop at IEEE Security and Privacy.

[8] Meredith L. Patterson. Hammer. https://github.com/UpstandingHackers/hammer, 2017. Parser combinators for binary formats, in C.

[9] K. Serebryany. Continuous Fuzzing with libFuzzer and AddressSanitizer. In *IEEE Cybersecurity Development (SecDev)*, pages 157–157, Nov 2016.

[10] Alan Mathison Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London mathematical society*, 2(1):230–265, 1937.

[11] Katherine Underwood and Michael E Locasto. In Search of Shotgun Parsers in Android Applications. In *Security and Privacy Workshops (SPW), IEEE*, pages 140–155. IEEE, 2016.