Human-Computability Boundaries

Vijay Kothari[1], Prashant Anantharaman[1], Ira Ray Jenkins[1],
Michael C. Millian[1], J. Peter Brady[1], Sameed Ali[1],
Sergey Bratus[1], Jim Blythe[2], Ross Koppel[3], and
Sean W. Smith[1]

[1] Dartmouth College, Hanover, NH, USA
[2] Information Sciences Institute, University of Southern California,
Los Angeles, CA, USA
[3] Sociology Department, University of Pennsylvania, Philadelphia, PA, USA

Abstract. Human understanding of protocols is central to protocol security. The security of a protocol rests on its designers, implementors, and, in some cases, its end users correctly conceptualizing how it should work, understanding how it actually works, and predicting how others will think it works. Ensuring these conceptualizations are correct is difficult. However, a complementary field provides some inspiration on how to proceed: the field of language-theoretic security (LangSec) promotes the adoption of a secure design-and-development methodology that emphasizes the existence of certain computability boundaries that must never be crossed during parser and protocol construction to ensure correctness of design and implementation. We propose supplementing this work on theoretical computability boundaries with exploration of human-computability boundaries. Historically, computability research has focused on understanding what problems can be solved by machines or *idealized* humans—that is, computational models that behave like humans in principle but that are not subject to the natural limitations that humans face in practice. Humans may not have inexhaustible auxiliary resources, and they are often subject to a variety of deficiencies, e.g., finite memories, reduced attention spans, limited information, misperceptions, and cognitive biases. We argue that these realities must be taken into consideration if we are to be serious about securing protocols. A corollary is that while the traditional computational models and hierarchies built using them (e.g., the Chomsky hierarchy) are useful for securing protocols and parser, they alone are *inadequate* as they neglect human-computability boundaries that define what humans can do in practice. In this position paper, we introduce the notion of human-computability, we advocate for the exploration and discovery of human-computability boundaries, and we outline steps moving forward.

1 Introduction

Humans are integral to the conception of protocols, laying out the initial vision, creating the specification, implementing the protocol, and wittingly or unwittingly making use of it. Due to humans' close and varied interactions with protocols during design, development, and operation, we must—if we want such interactions to persist and also want to secure protocols—account for humans' intrinsic limitations in understanding protocols.[4]

The genesis of a protocol vulnerability often lies in some human failure or deficiency, e.g., the copy-and-paste blunder that produced the recent Apple *goto fail* vulnerability [5]. The designer may introduce mistakes or create the specification under incorrect assumptions. Or the implementor may fail to correctly conceptualize the specification, e.g., due to cognitive constraints. Or perhaps the user may misunderstand the protocol, driving them toward behaviors that jeopardize security. (While some may not consider the previous example to be a protocol vulnerability, it has the same form as one; it is a predictable failure of the protocol design-and-development process, which can be used as a reliable conduit for attack.)

Our thesis is that a whole class of vulnerabilities could be averted if we better understood human limits to computability and took a principled approach to protocol design and development grounded in such an understanding.

In the remainder of this paper, we provide a brief primer on the field of language-theoretic security (LangSec), suggest work to complement the LangSec approach with a human-computability approach, discuss how human deficiencies give rise to bugs, propose methods for understanding these human limits, and finally conclude.

2 LangSec and Computational Models

Language-theoretic security (LangSec) [2] incorporates the theoretical insights offered by language theory, automata theory, and computability theory into a design-and-development methodology that averts common pitfalls responsible for producing numerous protocol and parser vulnerabilities. It advocates separating the parser from the execution environment, modeling the parser as a formal grammar, ensuring the grammar does not exceed certain computability boundaries on an extended version of the Chomsky hierarchy, and ensuring that the parser is a *recognizer* or more precisely a *decider*, i.e., it rejects all bad inputs and accepts all good inputs. In essence, LangSec tells us how to design protocols and parsers based on our understanding of the limitations of machines. That is not to say that LangSec does not acknowledge or address human causes of protocol and parser vulnerabilities. On the contrary, Bratus et al. in their discussion of exploit programming [3], note that many exploits are manifestations of incorrect computability assumptions. LangSec aims to rectify these assumptions within the design-and-development process. Furthermore, successful application of LangSec principles *requires* reducing human error. For example, the parser

[4] While the discussion in this paper focuses on protocols, the general notion of human computability we present is also pertinent to software security (and perhaps broader domains).

combinator toolkit Hammer [6] helps eliminate user error by assisting the implementor in creating a parser that matches the specification grammar. We contend that, while LangSec is vital and has made great strides toward securing protocols, it alone is insufficient. Specifically, there is a limit to what can be achieved by considering traditional computability boundaries alone. (Of course, one might argue this would not be a problem if we could eliminate the human from all parts of the protocol life cycle—including design, development, and use; as far as we can tell, we're not quite there yet.)

We propose supplementing the field of LangSec with work that explores human-computability limits. Common computational models, such as the Turing machine are excellent for capturing what machines can do; however, they are, in general, not well-suited to what actual humans can do with and especially without aids. In practice, humans have finite memories—and often inadequate foundational knowledge to understand protocol workings in comparison to machines. They have short attention spans. They are subject to cognitive biases and often make reasoning mistakes in predictable ways. These deficiencies manifest in coding bugs and user error, both of which endanger security.

We argue that we must acknowledge these human deficiencies, understand why and how they occur, develop solutions to begin addressing them, and finally we must update our protocol and parser design-and-development processes in accordance with such findings. This is an initial position paper, but we believe further inquiry along will go a long way in securing protocols and parsers.

3 Human-Computability Boundaries

Using an extended version of the Chomsky Hierarchy that differentiates between non-deterministic and deterministic pushdown automata, LangSec recommends staying within either the boundary of Turing-decidability (linear-bounded automata) or the stricter boundary of parser-equivalence decidability (deterministic pushdown automata), depending on the problem at hand. The *exact* class boundaries for these decision problems are not part of the 5-class extended Chomsky hierarchy, e.g., the Turing-decidability boundary lies at recursive languages. However, the extended Chomsky hierarchy is natural for humans to interpret and allows sufficient expressiveness to still be useful in the design and development of parsers and protocols.

Human-computability boundaries—the boundaries that specify what *actual humans*, not "ideal" humans or machines, can do without some aid—are a different beast altogether. Fitting human-computability boundaries to an extended Chomsky hierarchy is futile as there exist grammars within the class of regular grammars—i.e., grammars that can be expressed with finite state automata—that humans, in general, fail to conceptualize correctly. We do not know exactly where these human-computability boundaries lie, but the discovery of them may be instrumental in securing protocols and parsers. This observation is captured in Figure 1. The ovals correspond to classes of grammars/automata/languages in the 5-class extended Chomsky hierarchy. LangSec boundaries are drawn at linear-bounded automata and deterministic pushdown automata, whereas the oddly-shaped blob corresponds to a single idealized human-computability boundary. ***If this boundary were representative of reality, we would want to constrain ourselves to the intersection of the blob and the appropriate LangSec computability boundaries during protocol and parser construction.***

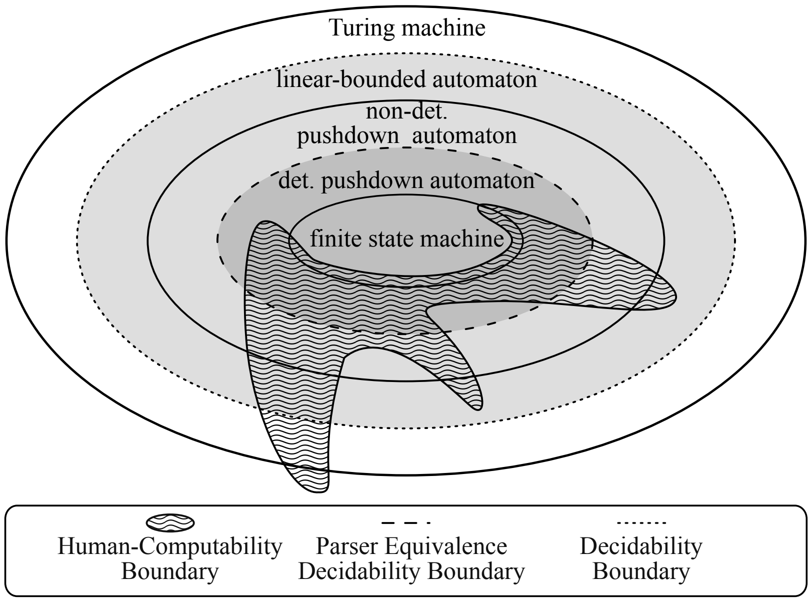

Fig. 1. Human-Computatability and LangSec Boundaries.

In practice, however, things are more complex. We can imagine different human-computability boundaries corresponding to different human roles and protocol interactions. We can also imagine fuzzy boundaries where the uncertainty comes from the variance of human attributes over a sub-population. We might consider human deficiencies of a probabilistic nature and aim to have, say, 99% of users be unsusceptible to a given flavor of attack based on protocol misconceptions; then, we may design and develop the protocol around this aim. If we know *a priori* what tools the various actors have at their disposal, the model we choose and boundaries we choose should take this into account. In short, the model used to express human-computability boundaries should be rooted in the protocol at hand, as well as the relevant sub-population and its capabilities.

4 What's Next?

This inquiry provides preliminary insights that warrant further exploration, along with other directions for future research. In this section, we briefly touch on these threads.

4.1 Determinants of Human-Computability Boundaries

There are myriad factors that determine human-computability boundaries, e.g., memory, attention span, dual-process model of cognition, and bounded rationality [4,7]. However, some will have a larger impact than others and some information will be easier to use in addressing vulnerabilities that arise from human deficiencies. That is, pragmatically speaking, the utility of exploring a determinant rests on its *salience* to human-computability boundaries and whether the information we can acquire about the determinant is *actionable*. For example, one approach might be to extend the compliance budget work of Beautement et al. [1] to a cognitive budget.

4.2 Usability Studies

Identifying the determinants of human-computability boundaries is insufficient. We must also conduct usability studies to understand the interplay between these determinants, human-computability boundaries, and security. Of course, this is not a one-way process; usability studies also help with identifying new determinants, which in turn guide new usability studies.

For example, an interesting genre of usability studies involves collecting concrete metrics for code complexity. Two classes of metrics are based on: (a) what the programmer can readily observe in the code and (b) what is represented in the abstract syntax tree (AST) for the program inputs in computer memory. As we mentioned earlier, program inputs are handled by code called parsers. Examples of metrics of the first type include lines of parser code and complexity per line of parser code, e.g., how many atomic structures such as combinators are used or represented in each line of code (on average or in the worst line). Examples of metrics of the second type include AST depth, number of branches, and tree balance.

4.3 Understanding Roles

In this paper, we considered three types of humans: the designer, the implementor, and the user. While we believe it is important to understand human-computability boundaries for each of these roles, the tools afforded by the role, the ideal "good" behavior associated with each role, and the importance of each role in securing protocols, we have yet to adequately answer any of these questions. Additionally, it may be valuable to consider other roles or more nuanced sub-roles.

4.4 A Theoretical Approach to Human-Computability Boundaries

While it may be infeasible to draw perfect or even close-to-perfect boundaries for human-computability, understanding *some* limitations can go a long way in addressing vulnerabilities in these spaces. In particular, it may be useful to develop traditional computational models to capture human limitations regarding finite-and-small states and/or finite-and-small memories predicated on usability studies. While even these models will not neatly fit within the extended Chomsky containment hierarchy used in LangSec, they would still be rooted in automata theory, which is certainly convenient. After all, understanding the commonality of two models of one type is generally easier than understanding the commonality of two models of different types.

4.5 How Feature Creep—Beyond the Traditional Bug

In security, we commonly hear of unnecessary feature expansion as being the cause of many software and hardware bugs—phenomena so widely recognized that terms like feature creep and software bloat have been developed to describe them. However, when one speaks of, say, feature creep, the focus is usually on a software (or perhaps hardware) bug, not on vulnerabilities rooted in human deficiencies. We contend that feature creep, by adding complexity to a protocol or a program that makes it exceedingly difficult for the human to understand, introduces predictable failures that can readily be exploited by the adversary. That is, the complexity introduced with feature creep creates a "bug" in the human that ultimately makes the software less secure. Therefore, we believe analyzing feature creep through the lens of human-computability boundaries or perhaps mismorphisms [7]—mismatches between interpretations— may be illuminating.

5 Conclusion

We argued that security rests, in large part, on acknowledging and accounting for human deficiencies in the design and development of network protocols. Existing LangSec work defines theoretical computability boundaries along the extended Chomsky hierarchy for which the decidability and parser equivalence decidability problems are solvable. Accordingly, recommendations to stay within these computability boundaries along with tools and other LangSec developments are valuable in guiding secure protocol and parser construction. However, they alone are insufficient. In this paper, we introduced the notion of human-computability boundaries, highlighted the difficulty in understanding and defining them, motivated the need for further exploration, and suggested threads for future work.

Acknowledgement

This material is based upon work supported by the United States Air Force and DARPA under Contract No. FA8750-16-C-0179 and Department of Energy under Award Number DE-OE0000780.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force, DARPA, United States Government or any agency thereof.

References

1. Beautement, A., Sasse, M.A., Wonham, M.: The Compliance Budget: Managing Security Behaviour in Organisations. In: Proceedings of the 2008 New Security Paradigms Workshop. pp. 47–58. ACM (2009)
2. Bratus, S.: LANGSEC: Language-theoretic Security: "The View from the Tower of Babel", `http://langsec.org`, [Online; accessed 2-January-2019]
3. Bratus, S., Locasto, M., Patterson, M., Sassaman, L., Shubina, A.: Exploit programming: From buffer overflows to weird machines and theory of computation. {USENIX; login:} (2011)
4. Herley, C.: So Long, And No Thanks for the Externalities: The Rational Rejection of Security Advice by Users. In: Proceedings of the 2009 workshop on New Security Paradigms Workshop. pp. 133–144. ACM (2009)
5. Naked Security, Sophos: Anatomy of a "goto fail" Apples SSL bug explained, plus an unofficial patch for OS X! (February 2014), `https://nakedsecurity.sophos.com/2014/02/24/anatomy-of-a-goto-fail-apples-ssl-bug-explained-plus-an-unofficial-patch/`, [Online; accessed 3-January-2019]
6. Patterson, M.: Parser combinators for binary formats, in C, `https://github.com/UpstandingHackers/hammer`, [Online; accessed 4-January-2019]
7. Smith, S.W.: Security and Cognitive Bias: Exploring the Role of the Mind. IEEE Security & Privacy **10**(5), 75–78 (2012)