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
vijayk@cs.dartmouth.edu
[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, its implementors, and, in some cases, its 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. A complementary field, however, 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 classical computability boundaries with exploration of human-computability boundaries. Classic computability research has focused on understanding what problems can be solved by machines or *idealized* human computers—that is, computational models that behave like humans carrying out rote computational tasks in principle but that are not subject to the natural limitations that humans face in practice. Humans are often subject to a variety of deficiencies, e.g., constrained working memories, short attention spans, misperceptions, and cognitive biases. We argue that such 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 parsers, they alone are *inadequate* as they neglect the human-computability boundaries that define what humans can do in practice. In this position paper, we advocate for the discovery of human-computability boundaries, present challenges with precisely and accurately finding those boundaries, and outline future paths of inquiry.

1 Introduction

Humans are integral to the conception and operation of protocols. They lay out the initial vision, create the specification, implement the protocol, and wittingly or unwittingly make use of it. Due to humans' close and varied interactions with protocols during their design, development, and operation, we must - if we want to secure protocols - account for humans' intrinsic limitations in understanding protocols.[1]

The genesis of a protocol vulnerability often lies in some human failure or deficiency, e.g., the copy-and-paste blunder that produced the Apple *goto fail* vulnerability [16]. 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 remaining sections of this paper, we: discuss Turing's notion of computability; provide a brief primer on the field of language-theoretic security (LangSec), which informs our work; present the idea of complementing LangSec with the incorporation of human-computability boundaries; discuss challenges in defining human-computability boundaries and follow-on work; discuss related work; and conclude.

2 The Human Computer

Today, Turing machines are often thought of as computational models for modern-day electronic computers; however, Turing very much had humans in mind during his conception of the Turing machine. As Jack B. Copeland points out in his discussion on the Church-Turing thesis:

> "Turing introduced his machines with the intention of providing an idealized description of a certain human activity, the tedious one of *numerical computation*. Until the advent of automatic computing machines, this was the occupation of many thousands of people in business, government, and research establishments. These human rote-workers were in fact called *computers*. Human computers used effective methods to carry out some aspects of the work nowadays done by electronic computers. The Church-Turing thesis is about computation *as this term was used in 1936*, viz. human computation[.]" [7].

[1] While the discussion in this paper focuses on protocols, the notion of human-computability boundaries is certainly applicable more broadly.

In Turing's seminal paper [22], in which he proved the Entscheidungsproblem is not, in general, solvable, he also introduced the Turing machine, along with the notion of computability. Turing wrote, in the paper, that: "Computing is normally done by writing certain symbols on paper. We may suppose this paper is divided into squares like a child's arithmetic book." In the same paper, Turing uses "the fact that the human memory is necessarily limited" as justification for the finite state property of Turing machines.[2]

Despite Turing's inspirations to model human computation, Turing machines are not adequate in fully capturing all aspects of human computation in protocol and program design, development, and use. It was never meant to do this. The Turing machine was a computational model that dealt with an ideal - a human in principle, not in practice. More importantly, human computation at the time was envisioned narrowly as rote processes carried out by humans. It was never intended to capture how humans design, develop, conceptualize, and use computer programs and protocols, in the fashion they do today. While we still have human computation in the present day, the role of humans and the tasks they perform are fundamentally different—and any computational models we use to capture human computation must reflect this reality.

3 LangSec and Computational Models

Language-theoretic security (LangSec) [5] 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 [6], 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 combinator toolkit Hammer [17] 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

[2] We note that not everyone held this view. For example, Shagrir provides discussion on Gödel's rejection of this assumption [19].

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 boundaries. Classical computational models, such as the Turing machine are excellent for capturing what machines can do; however, they are generally not well-suited for capturing what actual humans can do with and especially without aids. In practice, humans have finite memories—and often inadequate knowledge to understand protocol workings in comparison to machines. They have short attention spans. They are subject to cognitive biases and often make mistakes in reasoning in predictable ways. These deficiencies manifest in bugs during protocol and parser conceptualization, coding bugs, and user error, all 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. We hope this initial position paper will lay some groundwork for further inquiry that helps in securing protocols and parsers.

4 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 five-class extended Chomsky hierarchy, e.g., the Turing-decidability boundary lies at recursive languages. The extended Chomsky hierarchy, however, 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* can do with the capabilities they possess and the deficiencies they are subject to—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 Fig. 1. The ovals correspond to classes of grammars (or languages or automata) in the five-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.***

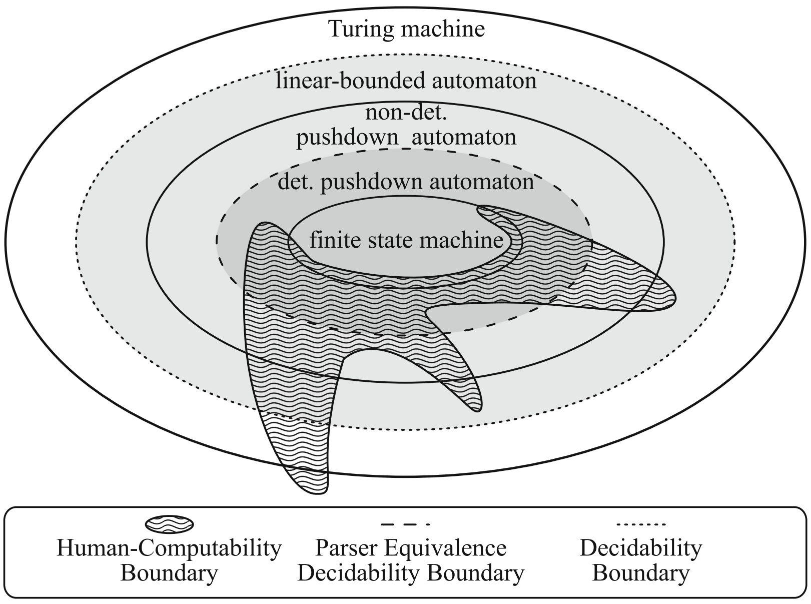

Fig. 1. Human-Computatability and LangSec Boundaries.

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 ensure most users are 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-populations and their capabilities.

5 Challenges and Future Work

In the previous section, we introduced the notion of human-computability boundaries and motivated the need for their discovery. However, there are a wide variety of challenges associated with accurately and precisely defining where these boundaries lie, developing models to capture them, and utilizing them in practice. In this section, we briefly touch on these threads and suggest directions for future research.

5.1 Determinants of Human-Computability Boundaries

There are many factors that determine where human-computability boundaries lie, e.g., memory, attention span, dual-process model of cognition, and bounded rationality [10, 21]. However, some of these determinants will have a larger impact than others and some information will be easier to attain and utilize in addressing vulnerabilities that arise from human deficiencies. That is, pragmatically speaking, the utility of exploring a determinant rests on its *salience* with respect to human-computability boundaries and whether the information we can acquire about the determinant is *actionable.* The effectiveness of the models that enable us to determine where human-computability boundaries follows directly from the determinants we choose.

5.2 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.

One example of a genre of usability studies we are interested in 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.

5.3 Understanding Roles

Drawing useful human-computability boundaries requires understanding which roles are pertinent, the goals associated with the roles, the tools afforded by each role, and the interplay between each role and the protocol. Such understanding must be reflective of the protocol at hand and the application domain. The protocol and application domain may warrant consideration of additional roles or sub-roles that we have not discussed.

5.4 Developing Models

We've discussed the importance of defining where and how the protocol is used, determining the roles of the various human actors, identifying the determinants of human-computability boundaries, and gathering the requisite data grounded in usability studies to draw human-computability boundaries.

The next step is then to incorporate these findings into a model that captures human-computability boundaries in a way that enables us to reason about the security of the protocol. It may be infeasible to draw perfect or even close-to-perfect boundaries for human computability. Understanding *some* limitations, however, can go a long way in addressing vulnerabilities.

The power of the model used to capture human-computability boundaries lies its utility in the design and development of safe protocols. While it may be infeasible to draw perfect or even close-to-perfect boundaries for human computability, all is not lost. Indeed, it may be better to capture a few limitations in a manner that enables us to design and develop safe protocols than many in a way that does not. As we discussed earlier, one inspiration for this paper was in developing human-computability boundaries that complement LangSec boundaries. In pursuit of this objective, we may wish to develop models similar to those of the classical automata, such as Turing machines, to capture these boundaries. 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.

We note that there has been some interesting, recent work on developing models for end users (e.g., [1,4,11]) that can assist in safe protocol and program construction. Another approach might be to extend the compliance budget work of Beautement et al. [2] to a cognitive budget for human agents.

6 Related Work

Jeanette M. Wing expounded on computational thinking as an essential mindset that everyone would benefit from, thereby providing a strong pedagogical basis for incorporating computational thinking into college and pre-college curricula [24]. She writes:

> "Stating the difficulty of a problem accounts for the underlying power of the machine—the computing device that will run the solution. We must consider the machine's instruction set, its resource constraints, and its operating environment." [24]

This mindset is crucial in efficiently solving problems on machines. In this paper, we argue for a parallel notion: Just as we must understand the computational capabilities of the machines that humans use, we must understand computational capabilities of humans as they interact protocols and programs, e.g., as they conceptualize and reason about code during development.

For completeness, we note that in recent years, human computation has developed into a field in its own right, e.g., [14,18,23]. The work in this field, however, is largely tangential to our work in this paper. Our interests are in developing an understanding of human-computability boundaries as they pertain to secure program and protocol design, development, and use. That said, in the past two decades, there has been some exciting research efforts to capture humans in protocol and parser design. Below, we touch on a few particularly relevant ones.

In 2007, Carl Ellison [8] presented the notion of ceremony [3] as a natural extension to the network protocol. A ceremony incorporates everything conventionally thought to be out-of-band to the protocol, e.g., UI interactions, human-human interactions, provisioning tasks. This holistic view of the protocol as a ceremony enables the security practitioner to better conceptualize and analyze protocol security. Since then, researchers have expanded on the idea of ceremonies. Notably, Bella and Coles-Kemp [3] pursued a formal model of security ceremonies with multiple layers: information, operating system, human-computer interaction, personal, and communal.

Johansen et al. [11] argued for the development of a new discipline, Behavioral Computer Science, lying at the intersection of behavioral sciences, ubiquitous computing and Internet of Things (IoT), and artificial intelligence. This discipline blends the study of HCI, modeling, and the notion of computational trust. The authors argue we must rethink the rational agent models often used for human behavior by acknowledging that: differences exist between humans' experienced utility, predicted utility, and remembered utility [13]; humans employ the dual-process model of cognition wherein they may invoke either a fast, knee-jerk, intuitive, and automated response or a slower, deliberate, rational response [12,20]; and humans are subject to all sorts of heuristics that affect their judgements [9]. The authors then discuss approaches to building models that capture this complexity, grounded in the Bella-Coles-Kemp model discussed earlier [3].

Basin et al. [1] studied the security of protocols in the presence of human error. They developed a formal model that includes human agents whose behavior may deviate from the behavior assumed by the protocol specification. They captured human error using two approaches: (1) a skilled human approach that begins with an infallible human agent who knows the protocol specification and modifies it to allow for a small number of mistakes; (2) a rule-based approach that begins with an untrained human that does not know the protocol specification and imposes a set of rules upon human agent behavior that dictate permissible behaviors. They then demonstrate how these two approaches can be used to formally model fallible humans with the Tamarin verification tool [15]. They also do a case study to show how this modeling approach can be used to discover human-based vulnerabilities in a protocol, and they use the model to compare different authentication protocols.

The most relevant work we've seen to our paper is by Blum and Vempala [4]. They proposed a model of human computation for end users in studying the security of protocols. They argued that traditional notions of computability cannot blindly be applied to humans and that, instead, human computational models must take into account the reality that human processing power is inferior to that of computers. They argued that human computation occurs in two dis-

[3] As noted by Carl Ellison: "The term 'ceremony' was coined for this purpose by Jesse Walker of Intel Corporation." [8].

tinct phases: a pre-processing phase and a processing phase. Accordingly, they developed a model for human computation—a variant of the Turing machine—and introduced the notion of a schema to be the human analog to a computer algorithm. Finally, they applied this model to different problems. Our paper certainly has some overlap with this work. However, we explore notions of human-computability boundaries more generally. We are also not solely concerned with users; we also focus on human designers and implementors. Last, we are interested in combining models of human computability with traditional computability models.

7 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 highlights 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, as we argue in this paper, they alone are insufficient. We discussed the notion of human-computability boundaries, highlighted the difficulty in understanding and defining them, and discussed open challenges 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. Basin, D., Radomirovic, S., Schmid, L.: Modeling human errors in security protocols. In: 2016 IEEE 29th Computer Security Foundations Symposium (CSF), pp. 325–340. IEEE (2016)
2. 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)
3. Bella, G., Coles-Kemp, L.: Layered analysis of security ceremonies. In: Gritzalis, D., Furnell, S., Theoharidou, M. (eds.) SEC 2012. IAICT, vol. 376, pp. 273–286. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30436-1_23
4. Blum, M., Vempala, S.: The complexity of human computation: a concrete model with an application to passwords. arXiv preprint arXiv:1707.01204 (2017)
5. Bratus, S.: LANGSEC: Language-theoretic security: "The View from the Tower of Babel". http://langsec.org. Accessed 2 Jan 2019

6. Bratus, S., Locasto, M., Patterson, M., Sassaman, L., Shubina, A.: Exploit programming: from buffer overflows to ”weird machines” and theory of computation. Login USENIX Mag. **36**(6), 13–21 (2011)
7. Copeland, B.J.: The Church-Turing Thesis. In: Zalta, E.N. (ed.) The Stanford Encyclopedia of Philosophy. Metaphysics Research Lab, Stanford University, Spring 2019 edn. (2019)
8. Ellison, C.: Ceremony design and analysis. Cryptology ePrint Archive, Report 2007/399 (2007). https://eprint.iacr.org/2007/399
9. Gilovich, T., Griffin, D., Kahneman, D.: Heuristics and Biases: The Psychology of Intuitive Judgment. Cambridge University Press, Cambridge (2002)
10. 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)
11. Johansen, C., Pedersen, T., Jøsang, A.: Towards behavioural computer science. In: Habib, S.M.M., Vassileva, J., Mauw, S., Mühlhäuser, M. (eds.) IFIPTM 2016. IAICT, vol. 473, pp. 154–163. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41354-9_12
12. Kahneman, D.: A perspective on judgment and choice: mapping bounded rationality. Am. Psycholog. **58**(9), 697 (2003)
13. Kahneman, D., Thaler, R.H.: Anomalies: utility maximization and experienced utility. J. Econ. Perspect. **20**(1), 221–234 (2006). https://doi.org/10.1257/089533006776526076
14. Law, E., Ahn, L.V.: Defining (Human) computation. Synth. Lect. Artif. Intell. Mach. Learn. **5**(3), 1–121 (2011)
15. Meier, S., Schmidt, B., Cremers, C., Basin, D.: The TAMARIN prover for the symbolic analysis of security protocols. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 696–701. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_48
16. Naked Security, Sophos: Anatomy of a “goto fail” - Apple’s 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/. Accessed 3 Jan 2019
17. Patterson, M.: Parser combinators for binary formats, in C. https://github.com/UpstandingHackers/hammer. Accessed 4 Jan 2019
18. Quinn, A.J., Bederson, B.B.: Human computation: a survey and taxonomy of a growing field. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 1403–1412. ACM (2011)
19. Shagrir, O.: Gödel on turing on computability. In: Olszewski, A., Wole’nski, J., Janusz, R. (eds.) Church’s Thesis After Seventy Years, pp. 393–419. Ontos Verlag (2006)
20. Sloman, S.A.: Two systems of reasoning. In: Gilovich, T., Griffin, D., Kahneman, D. (eds.) Heuristics and Biases: The Psychology of Intuitive Judgment. Cambridge University Press, Cambridge (2002)
21. Smith, S.W.: Security and cognitive bias: exploring the role of the mind. IEEE Secur. Privacy **10**(5), 75–78 (2012)
22. Turing, A.M.: On computable numbers, with an application to the entscheidungsproblem. Proc. London Math. Soc. **2**(1), 230–265 (1937)
23. Von Ahn, L.: Human computation. In: Proceedings of the 2008 IEEE 24th International Conference on Data Engineering, pp. 1–2. IEEE Computer Society (2008)
24. Wing, J.M.: Computational thinking. Commun. ACM **49**(3), 33–35 (2006)