Elisa Bertino
Purdue University

The Persistent Problem of Software Insecurity

Software is increasingly playing a key role in all infrastructure and application domains we may think of. One notable example is represented by the increased "softwarization" of computer networks, including wireless communications networks—see, for example, the various initiatives toward open radio access networks (RANs) to disaggregate the various components of a RAN and standardize the interface of these components to run them on the cloud. Or, there is the notion of "network programs"[1] as an approach to better control and manage networks and possibly even enhance their security.

Unfortunately, as we all know, software systems are still often insecure despite the fact that the "problem of software security" has been known to the industry and research communities for decades. One would have at least expected that, given the increased awareness of the need for better software security, recently developed applications would be more secure. However, this does not seem to always be the case. Take, for example, mobile applications. A relatively recent analysis of over 13,000 such applications using login and password authentication[2] has shown that about 18% of these applications did not correctly check the certificate sent by the server or did not even check the certificate at all. On the other hand, one might expect that applications would today be free of long-known defects that make them vulnerable to attacks, e.g., Structured Query Language (SQL) injection. However, the number of SQL vulnerabilities accepted as common vulnerabilities and exposures remains high even in recent years.

The landscape of software security is today further complicated by new application domains with their own specific vulnerabilities. One notable example is represented by control applications that typically have to configure combinations of large numbers of parameters. For example, control software for drones, such as ArduPilot and PX4, has to manage 247 configurable control parameters. Not only does the value assigned to each parameter have to be validated but even combinations of parameter values have to be checked, as certain parameter values combinations are unsafe, even when each parameter value by itself is correct. To all the preceding we need to add attacks to the software supply chain that target not only software by companies—an example of which is the SolarWinds attack—but also exploit flaws of open source software—an example of which is the recent case of Log4j. The risk with open source software is not only the vulnerabilities that a piece of code may have but also malicious changes that can be introduced into the code.

One obvious question is why software is still insecure. Reasons that are often mentioned include the lack of vendor liability, the lack of training of software engineers and developers, the use of insecure languages, and so on. However, it can also be argued that the benefit of software, even if insecure, outweighs its lack of security. After all, could we imagine our society today without software? On the other hand, software security is increasingly a critical need. So, what would it take to convince decision makers at various levels and organizations that software security must be a priority? As argued by Bruce Schneier in his "Last Word" column "What Will It Take?" in the May/June 2021 issue of *IEEE Security & Privacy*,[4] it is critical that decision makers and the public "not only need to believe that the present situation is intolerable, they also need to believe that an alternative is possible."

We need systematic approaches to software security. Such an approach would need to include a comprehensive taxonomy of software vulnerabilities, such as memory, cryptography and authentication, and

continued on p. 107

Digital Object Identifier 10.1109/MSEC.2022.3158841
Date of current version: 26 May 2022

Last Word *continued from p. 108*

corresponding automatic detection and patching techniques. For example, whereas fuzzing is useful for detecting memory vulnerabilities, it may not be suitable for detecting vulnerabilities such as a lack of proper certificate validation and input validation. We need techniques that are highly accurate and time efficient; to this end, one may combine different techniques and perhaps leverage artificial intelligence/machine learning techniques. Formal methods could also be useful tools for enhancing software security, provided, however, that they are scalable. In addition, monitoring tools for software would be invaluable, as they would allow one to check at runtime if software is behaving as expected. Such tools may not be easy to design, as software may behave in different ways depending on input parameters and context. However, I believe that this could be doable, at least for monitoring behavior with respect to specific sensitive actions (see Bossi et al.[3] for a simple early approach). And also, we need to work on effective solutions for secure software supply chains. ■

References

1. N. Foster, N. McKeown, J. Rexford, G. Parulkar, L. Peterson, and O. Sunay, "Using deep programmability to put network owners in control," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 50, no. 4, pp. 82–88, 2020, doi: 10.1145/3431832.3431842.
2. S. Ma *et al.*, "Finding flaws from password authentication code in Android apps," in *Proc. Eur. Symp. Res. Comput. Security*, 2019, pp. 619–637, doi: 10.1007/978-3-030-29959-0_30.
3. L. Bossi, E. Bertino, and S. R. Hussain, "A system for profiling and monitoring database access patterns by application programs for anomaly detection," *IEEE Trans. Softw. Eng.*, vol. 43, no. 5, pp. 415–431, 2017, doi: 10.1109/TSE.2016.2598336.
4. B. Schneier, "What will it take?," *IEEE Secur Priv*, vol. 19, no. 3, pp. 63–64, May/June 2021. doi: 10.1109/MSEC.2021.3063800.

Elisa Bertino is a professor with Purdue University, West Lafayette, Indiana, 47907, USA. Contact her at bertino@purdue.edu.

Digital Object Identifier 10.1109/MSEC.2022.3172000