The right to secure software: Whitepaper on the social aspect of software security

Abstract

It happens way too often, that people's devices, company services or federal infrastructure gets hacked from "the outside" (from a remote network location). The potential and actual negative impact of such a hack is often disastrous for the affected ones.

A big imbalance in the software sector nowadays is that there is often not enough need for a profit-oriented software company to seriously invest into the inner security and hardening of their products, while the arising dangers from it are hitting someone else.

Every software vendor has its own philosophy when it comes to security. Some take better care, others handle it more sloppy. A software vendor in the end might only suffer from a decline in sales when a successful hack was made public. Software gets hacked every day!

This paper proposes a balanced right for secure software.

Why do we need more than high-end security tooling and brilliant tech expertise to get software secure?

We live in a time, where our reality is more and more interwoven with and partially shaped by all kinds of software around us. Software is used in so many areas. It serves us in a wide range from being directly visible to being almost unobservable in the background. It comes in form of daily personal tools, working environments, essential federal or corporate services and infrastructure as well as knowledge bases for everyone – like for the playing and learning kid, vehicle drivers, business decision maker, scientists and many, many more.

So in general our satisfaction and our safety as users of products of the information age is based on the technical quality of its algorithms as well as on the fact that is has not been infiltrated or manipulated by anyone in ways it was not designed for.

The most sensitive area here – because it is the potentially most harmful one – is the security of the software, which we intend to use. No one wants to be spied on, blackmailed or manipulated.

High-end security tooling and expert security consultancy is available, but of course, they mean substantial extra costs.

In a capitalist market order we need an effective financial incentive, so that software manufacturers and software providers really do their job properly for our all good.

Background

When we deal with complex software, we always deal with all sorts of unintended behavior too – let's call it bugs. In this paper we address the potentially most dangerous kind of it: **Remotely exploitable vulnerabilities**

Remote infiltration of a device or service is technically possible, when:

- one of the software components, running on that device, contains a remotely exploitable gap, and
- it is reachable for an attacker via a network (eventually everything is, directly or indirectly)

This paper primarily addresses the consequences for vendor and provider of a software product in case an "authorized user" (which might be a person or a company as a whole) get's compromised from a remote network location because a remotely exploitable security hole in that software allowed it. We are not going too far when we say that this is seen as the potentially most dangerous scenario in the field of software security and also in the much wider field of software quality by most people who are familiar with computer science.

That's why we need to address it also from it's social perspective. The potential negative impact of remotely compromised software is just too high to handle it only on the technological level. But what can be done more than setting our trust in reliable companies, managers with moral principles and skilled developers and admins?

Let's recall the key finding from above

A big imbalance nowadays is, that there is not enough need for a (profit-oriented) software company to seriously invest into the ongoing security of their products as long as they would just suffer from a decline in sales when a successful hack was made public.

Which software vendors are we talking about?

Potentially all of them as long as they sell it as a product. To get a clearer picture here are some examples of major relevant vendors:

- Operating System vendors
- PC software and App vendors
- Cloud service provider (liable for their pure cloud services, not for third-party software running on it)
- Car manufacturers (which will be seen as software providers of their car's own software - even when the software was produced by a supplier)
- Manufacturers of industry control systems
- Public service software vendors, etc.

For the weakest market members (e.g. single developers and small startups in the first 3 years) we should discuss special rules in order to allow healthy growth.

Furthermore, we should keep in mind, that also the user(s) (sometimes the provider) of a device or software service has responsibilities, which are essential to keep the device or service secure.

So we see, that even though we restrict the topic exclusively on secure (like "unbreakable") software, we already deal with a quite complex matter. Nevertheless, this is a suitable scope, which should be addressed in our society properly as a whole.

Solution: Liable Vendors

This paper proposes a balanced right for secure software – balanced because the negative consequence for the vendor/provider of a specific software product will be measured according to the negative impact of each actually happened remote hack of their software (theoretically if no insurance would cover).

The user (respectively the service provider) has the following responsibilities:

- being aware what software is in use
- make sure each software is used in its environment, were it is designed to run
- and most important: the software is kept up-to-date

For all new issued software products, including all of it's updates, the vendor of that product (short: **vendor**) will be fully responsible for the harm caused by an actually conducted remotely exploited gap in the software he delivers with his product. The vendor will be liable only, when the software was **up-to-date**, no matter what the root cause of that software gap was (bug, mistake by design, oversight, gap in third party library, etc.)

Up-to-dateness and outdated software

Up-to-date means here, that a provided automatic update mechanism and schedule from the vendor (respectively of his software supplier) was not blocked or massively delayed by the user recently, so that he was causing the outdated software state.

An outdated software state is measured by the time period beginning from the time of the availability of an automatic update for the users device or service (respectively the affected software component). Here we should consider enough time for the vendor to reach the user's device as well as the typical usage pattern, the expected uninterrupted usage of the device or service as well as the potential harm of a gap left open.

We recommend to define it commonly per product class and severity level of fixed bugs. It should be reviewed and adjusted on a yearly basis. According to common sense it should be typically something between a day and a month. Initially, before all categories are defined, a default time period of two weeks could be used instead.

Insurance

Insurance companies could help to cover major parts of the risks for software / product vendors here.

How to deal with existing software when these rules would be active?

This is a delicate topic. It's flavor will be a major factor driving the success of the whole proposition. We need to discuss this.

Here is a simple proposal:

Because it takes a lot of effort to establish high-end security checks as part of an existing development pipeline and find + fix potential bugs in existing products, there should be a transitional period of (lets say) three years until they fall under these rules. In this time the vendor is only liable for properly fixing the addressed kind of security holes quickly as soon as it gets discovered – lets say in less than three weeks.

After three years the vendor will be liable for older software too. He had have enough time to harden or replace it.

Conclusion

This paper offers a strategic solution to reduce the spread of insecure software. When the described solution is put into regional or worldwide law, over the time we expect to see a substantially higher level of robustness against external bad actors for all sorts of devices and services used in our daily live.

Let's design our future more secure!

References

- Schneier, Bruce. (2003). Liability changes everything https://www.schneier.com/essays/archives/2003/11/l
- Ryan Dewhurs. Static Code Analysis https://owasp.org/www-community/controls/Static_Code_Analysis
- Wikipedia. Open-source software https://en.wikipedia.org/wiki/Open-source_software
- CCC/erdgeist. (2021). CCC veröffentlicht Formulierungshilfe für Digitales im neuen Regierungsprogramm https://www.ccc.de/de/updates/2021/ccc-formulierungshilfe-regierungsprogramm

Final thoughts

Software and their devices can be compromised even easier (and also harder to harden against it), when an attacker has direct access to an internal secure network or the device hardware itself. Even though it's a related topic, this paper does not discuss that area directly, because that are special circumstances which can be seen like an insider-attack, which has a complete different quality and demands a different perspective and handling.

And what about Open Source Software?

That serves a different purpose. Please consider these arguments.

- Open Source Software is not made public with the purpose to sell a product
- Open Source Software is publicly available in it's source code, so one can review it or let it review by someone else.
- Anyone with the appropriate skill can fix it
- Furthermore, high-end static code analysis technology (SAST; Taint Analysis) becomes more and more available for major programming languages. These tools are capable of finding major parts of remotely exploitable software gaps.

Software quality

The reader might argue, that we should address software quality in general in such a way – and he is not wrong with that. But lets solve one problem at a time. This topic is way too complex to be handled as a whole, so we're forced to focus on the potentially most harmful area first, which is software security.

At last we won't forget to mention, that the general idea, to make vendors liable for insecure software was already published 2003 by the internationally renowned security technologist Bruce Schneier.

Version 1.1, Sep. 30, 2023, bitmagier