Whack-A-Mole Security: Incentivising the Production, Delivery and Installation of Security Updates

Alastair R. Beresford
Computer Laboratory, University of Cambridge
arb33@cam.ac.uk

Abstract

Writing vulnerability-free code is currently impossible. The best we can hope for is *whack-a-mole security*. In other words, fixing bugs and updating Internet-enabled devices before remote exploitation occurs. Unfortunately, security updates do not always happen in a timely fashion, or at all. The root cause of this problem is the lack of incentives, something which we must fix.

Introduction

Many computers today, from servers to laptops, and from tablets to smartphones are connected to the Internet. Connectivity is an essential feature of these platforms. Over the next few years, the rise of the Internet of Things (IoT) will see many more embedded computers connected to the Internet too. Everything from heating controllers to lightbulbs will be online.

Unfortunately, connecting computers to the Internet does not just provide essential functionality, but also enables remote attack. Yet we rely on many of our Internet-enabled devices to operate correctly. This state of affairs occurs because we cannot write vulnerability-free software. We cannot even provide reasonable guarantees of software correctness when developing complex, feature-rich software for a reasonable price. What, then, can we do? Current best practice is *whack-a-mole security*: in other words, devices are secured by patching latent vulnerabilities in software after their discovery, but before exploitation.

In: D. Aspinall, L. Cavallaro, M. N. Seghir, M. Volkamer (eds.): Proceedings of the Workshop on Innovations in Mobile Privacy and Security IMPS at ESSoS'16, London, UK, 06-April-2016, published at http://ceur-ws.org

Whack-a-mole security works when vulnerabilities are discovered by good netizens, such as penetration testers, anti-virus vendors and academics, who report the vulnerabilities to the appropriate software vendors. Software vendors then produce an update and send it to all affected devices before a potential attacker discovers the vulnerability and uses it for malice.

Failures occur for two reasons: firstly, attackers exploit vulnerabilities before good netizens find them (zero-day exploits); secondly, software updates from vendors do not arrive before attackers exploit a vulnerability found by a good netizen.

There is good academic literature on exploits, both on how they are found and how they work. There is also excellent work looking at reducing the likelihood of introducing vulnerabilities in the first place. There is much less work on whether software vendors actually patch vulnerabilities however. As a research community we should do more in this space.

An example: the Android ecosystem

We have looked at the Android ecosystem in order to determine if Android smartphones receive security updates. What we found was worrying—from July 2011 until July 2015, 87.7% of Android devices were vulnerable to at least one of 11 critical vulnerabilities [TBR15]. In this case, the problem occurs because handset manufacturers are not producing security updates. Analysis of the Android Open Source Project, and Google Nexus devices reveals that Google does regularly produce fixes for disclosed vulnerabilities. When updates do appear for devices, data from our Device Analyzer [WRB13] project shows that updates are installed by users across many different mobile network operators, suggesting that the network operators are providing the updates, and users are installing them.

Note that in this analysis we have quantified the proportion of devices which are at *risk* of attack. We have not yet measured the *harm* which might follow

through exploitation.

For the Android ecosystem, the cost of producing the fix is non-trivial and manufacturers are economically incentivised to spend valuable engineering time on producing new models, not fixing older ones which generate little or no future revenue. To address this, we need to develop better measures of both risk and harm. And we need to provide better incentives to manufacturers through accreditation, through regulation, and through awareness raising with consumers and corporate buyers.

The anatomy of an update

It's not just operating systems which require updates. All layers of the stack, including apps themselves, require updates to fix vulnerabilities. For example, Fahl et al. found that 8% of Android apps examined contained SSL/TLS code which was vulnerable to man-in-the-middle attacks [FHM+12]. As part of an undergraduate project, we found the same vulnerabilities in many apps on the Google Play store two years later. Incentivising developers to fix apps is, in some ways, harder than fixing operating systems because there are many more app developers than smartphone handset manufacturers, many of whom are individuals or small companies with limited resources and less expertise.

The issue of managing updates extends beyond software, since the security of Internet-connected devices has a basis in cryptography, which in turn requires managing updates to key material. Indeed, the integrity of updates to operating systems and apps is protected by cryptography. At the extreme, in the case of JavaScript running in a browser, an update to the software may be performed on every page load. Similarly, in the case of TLS, a server may present a new certificate on every connection establishment, or even during the lifetime of a single connection.

Since we are unable to write vulnerability-free software, keys can, and are, compromised. Therefore devices will need to securely update keys too.

The solution space

The rate at which security vulnerabilities are found in a stable code base may reduce over time [OS06], which offers some hope for IoT systems whose feature set might be small and does not necessarily need to change. This is unlikely to be a popular solution for smartphones or laptops since a vibrant ecosystem demands new apps, new hardware and new operating system features regularly. (So called *feature creep* may turn out to be a requirement in the IoT space too.)

Auditing of software by independent third-parties is useful. Google Play and Apple's App Store do this for apps, although unfortunately they do not often publish detailed results. This makes it hard for consumers or regulators to make informed choices based on data. In the cryptographic sphere, Certificate Transparency [Lau14] shows great promise, because it produces an auditable public log. Auditable public logs can be applied to software updates too. This would allow us to track the progress (or not) of updates from software vendors, through network operators and on towards installation by individuals. Note that there is a potential privacy conflict here—the update status of devices may implicitly leak some personal data.

It is also important to distinguish between risk and harm. Estimating the potential future harm from risk is a difficult process which deserves more attention.

Finally, quantifying the ability of software vendors to whack their security moles is not enough. We need to improve incentives. This can take a variety of forms, including presenting appropriate data to the public, as well as to corporations and governments. Better regulation is also likely to be important.

Acknowledgements

Huge thanks to my collaborators Daniel R. Thomas and Andrew Rice on Android vulnerabilities, and Graham Edgecombe for his work analysing Android apps.

References

[FHM+12] Sascha Fahl, Marian Harbach, Thomas Muders, Lars Baumgärtner, Bernd Freisleben, and Matthew Smith. Why Eve and Mallory love Android: An analysis of Android SSL (in) security. In *Proceedings of the ACM conference on Computer and Communications Security*, pages 50–61. ACM, 2012.

[Lau14] Ben Laurie. Certificate Transparency. *Queue*, 12(8):10, 2014.

[OS06] Andy Ozment and Stuart E. Schechter. Milk or wine: does software security improve with age? In *Usenix Security*, 2006.

[TBR15] Daniel R. Thomas, Alastair R. Beresford, and Andrew Rice. Security metrics for the android ecosystem. In *Proceedings of the ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, pages 87–98. ACM, 2015.

[WRB13] Daniel T. Wagner, Andrew Rice, and Alastair R. Beresford. Device analyzer: Understanding smartphone usage. In *Mobile and Ubiquitous Systems: Computing, Networking, and Services*, pages 195–208. Springer, 2013.