Incentivising software updates

Daniel R. Thomas and Alastair R. Beresford

Cybercrime Centre, Computer Laboratory, University of Cambridge

Abstract

Internet of Things devices will need software updates to fix the security vulnerabilities that are found after they are sold. Even when using development practices that make producing updates easy, there is still a cost to doing so. Hence, incentives to provide updates are required. This incentive can only be provided by monitoring the relative performance of different companies at supplying software updates to their customers.

1 Introduction

Internet connected devices are always subject to external probing and attacks. Software always has bugs and some of those are security vulnerabilities. Hence, there will always be a need for software updates to fix security vulnerabilities because providing secure software is hard.

There are tools and techniques for reducing the number of vulnerabilities such as using static analyzers to find them, using languages that enforce type safety and prevent some classes of vulnerability, using secure libraries to perform security critical tasks like sanitising input, using operating systems that provide useful primitives and strong compartmentalisation. There are also mitigation strategies that reduce the severity of vulnerabilities; for instance using compartmentalisation/containment technologies such as Capsicum in the operating system [8] or CHERI in hardware [9]; or using stateless firewalls that hide devices from known bad addresses. Authenticated encryption of connections to and from devices can make it harder to exploit vulnerabilities and projects like Let's Encrypt [1] facilitate that.

However, even with all these strategies, updates are still required, so this needs to be easy to do. Upgrading can be made easier through side-by-side installation of updates, such as is used in Chrome [2], because this means updates are easily applied on the next restart, without prolonged interruptions in service for the user. That is only possible if there is sufficient room on the device's storage for two copies of the software plus working space, and so devices need to be provisioned with room to expand into. Proper package management that tracks dependencies and separates out all the component parts, such as the one used in Debian derived systems, makes producing security updates much easier because after fixing one library all its reverse-dependencies get the fix. However it does require coordinated maintainers for all the components. This contrasts with the package management used in Android and iOS where libraries get bundled with the apps making fixing one library require fixing many apps that include it. If software is open source then 'anyone' can write the patch to fix the security vulnerability, thus the manufacturer might be spared the cost of implementing the fix (though they still need to test it and deploy it). Even with all these and other techniques there still is a cost to producing updates, so incentives are required, or companies will often decide it is not worth doing for discontinued products. Even when an update is available, until the fix has reached almost all devices the vulnerability can still get exploited. Previously we have shown that it takes years for updates to be fully deployed on Android after a vulnerability is discovered and fixed [6], and so ensuring deployment is important.

Incentives can come from a variety of sources. If the business model is rent based (SAAS / Cloud / subscription) and switching providers is easy then the provider has an incentive to ensure that all the equipment that they own and operate inside customers property works correctly so that they retain the customer. However, for devices bought on a one-off basis the customer needs to believe at the time of purchase that the device will keep on working, thus the reputation of the company providing the device matters. This provides an opportunity to incentivise businesses to act correctly by providing customers and regulators with comparative information on the security offered by different providers.

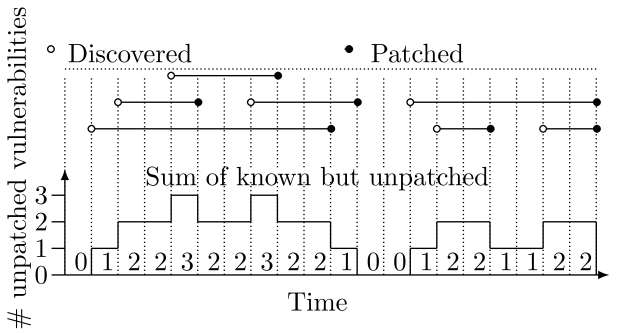

Figure 1: As vulnerabilities are discovered and patched the sum of known but unpatched vulnerabilities each day varies. From this m can be calculated: $m = (0\times3+1\times5+2\times10+3\times2)/20 = 1.55$. Example based on the one given by Wright [10].

2 Comparing providers

This section describes a scheme for assigning a device a score between zero and ten that should correlate with the actual security of the devices. We originally developed this scheme to compare Android manufacturers and models but it can also be applied to Internet of Things (IoT) devices [5]. To compare providers two kinds of information is required: 1) information on the deployed versions of all software running on devices; 2) information on the vulnerabilities affecting different versions of that software.

The FUM score is computed from three components:

free f The proportion of running devices free from critical vulnerabilities over time.

update u The proportion of devices that run the latest version of the software shipped to any device produced by that device manufacturer. This is a measure of internal updatedness, so a low score would mean many devices are left behind. This assumes that newer versions are better with stronger security. This is usually the case though sometimes updates introduce new vulnerabilities.

mean m The mean number of outstanding vulnerabilities affecting devices not fixed on any device shipped by the device manufacturer. An example is given in Figure 1.

These three metrics f, u and m, together measure the security of a platform with respect to updates for known vulnerabilities. The value f is a key measure of the direct risk to users as a known, unfixed, vulnerability means devices are vulnerable. However, it does not capture the increased risk caused by multiple known vulnerabilities, which gives an attacker more opportunities and increases the likelihood of a piece of malware having a matching exploit. This is captured by the m score, which measures the size of the device manufacturers queue of outstanding vulnerabilities. The m score does not take into account the update process or measure actual end user security. Neither of these metrics capture whether devices are left behind and not kept up-to-date with the most recent (and hopefully most secure) version, which is captured by u.

A score out of 10 is provided as this is easy for purchasers to understand, because many ratings are given as a score out of 10. Since f is the most important metric it is weighted more highly. Since m is an unbounded positive real number, it is mapped into the range (0–1]. This gives us the FUM score with a range of (0–10]:

$$\text{FUM score} = 4 \cdot f + 3 \cdot u + 3 \cdot \frac{2}{1 + e^m}$$

2.1 Security scores for Android

Using data collected by an Android app called Device Analyzer [7], which collects information on the deployed versions of Android on running devices; and from our AndroidVulnerabilities.org website [4], which collates information on known vulnerabilities in Android we applied the FUM scoring metric to Android.

On average, between July 2011 and March 2016, we found 12.4% of devices to be free from known vulnerabilities, 5.67% of devices to run the most recent version of Android and 0.661 outstanding vulnerabilities not fixed on any device. This gives a security score of 2.71 out of 10. However, there are a wide variety of scores depending on the source of the device. There is anecdotal evidence that Google's Nexus devices are better at getting updates than other Android devices because Google makes the original updates and ships them to its devices [3]. Table 1 shows that this is indeed the case with Nexus devices getting much better scores than non-Nexus devices. Different device manufacturers have very different scores, Table 1 shows the scores for several device manufacturers with a significant presence in the data, with *LG* (4.28 out of 10) scoring highest.

Name	f	u	m	**score** (out of 10)
Nexus	0.53	0.50	0.69	5.63 ± 0.02
non-Nexus	0.09	0.02	0.74	2.35 ± 0.00
LG	0.34	0.33	0.74	4.28 ± 0.02
Motorola	0.26	0.14	0.65	3.50 ± 0.02
HTC	0.13	0.09	0.87	2.59 ± 0.02
Sony	0.13	0.18	1.09	2.57 ± 0.02
Asus	0.23	0.46	5.61	2.29 ± 0.02
Samsung	0.11	0.06	0.99	2.24 ± 0.00
oneplus	0.02	0.31	7.85	1.00 ± 0.02

Table 1: FUM scores for Nexus and manufacturers

2.2 Gaming the score

If the comparative scoring metric given here is used to influence purchasing decisions then providers might try to game the score rather than genuinely improve security. The value of f is hard to game without providing good security but it does not get any worse if there is already one known vulnerability and another is found. A high value of u could be achieved by only having one version but that would give low values for f and m (and not be attractive to customers). A high value of m could be achieved by on only supporting one device at a time and ensuring that it gets updates, but that would lower f and u. One way to influence the scores would be to attack the data collection system either by providing false data or preventing correct data from being collected. Therefore, the score is secure against passive gaming attacks that change the measured distribution, but would require active defence against active gaming attacks, which target the measurement devices.

3 Outlook

Security updates will always be required because writing software without vulnerabilities is hard. There will always be some cost to producing and deploying updates, hence incentives are required. In order to apply pressure, regulators, consumers and corporate purchasers need data on the comparative performance of alternative providers. The FUM scoring metric is one attempt at providing that data. It requires third parties to a have visibility into the versions of running software on devices so that they can produce aggregate statistics. However, advertising this information to the whole world would make it easy for script kiddies to find vulnerable devices.

References

[1] Barnes, R., Hoffman-Andrews, J., and Kasten, J. *Automatic Certificate Management Environment (ACME).* Tech. rep. IETF, 03/2016. URL: https://datatracker.ietf.org/doc/draft-ietf-acme-acme/.

[2] Duebendorfer, T., and Frei, S. *Why silent updates boost security.* Tech. rep. April. ETH Zurich, 2009. URL: ftp://129.132.2.249/pub/publications/TIK-Report-302.pdf.

[3] Hoffman, C. *HTG Explains: Why Android Geeks Buy Nexus Devices.* 05/2013. URL: http://www.howtogeek.com/139391/htg-explains-why-android-geeks-buy-nexus-devices/ – https://archive.is/iGxrZ (visited on 2015-09-17).

[4] Thomas, D. R., and Beresford, A. R. *AndroidVulnerabilities.org.* 2015. URL: http://androidvulnerabilities.org/ – https://archive.is/VdiPu.

[5] Thomas, D. R., Beresford, A. R., and Rice, A. Security metrics for the Android ecosystem. In: *ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM).* ACM, Denver, Colorado, USA, 10/2015. ISBN: 978-1-4503-3819-6.

[6] Thomas, D. R., Beresford, A. R., Coudray, T., Sutcliffe, T., and Taylor, A. The lifetime of Android API vulnerabilities: case study on the JavaScript-to-Java interface. In: *Security Protocols XXIII.* Springer, 03/2015.

[7] Wagner, D. T., Rice, A., and Beresford, A. R. Device Analyzer: Understanding smartphone usage. *International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MOBIQUITOUS)* (2013), 1–12. ISSN: 1867-8211. URL: http://link.springer.com/chapter/10.1007/978-3-319-11569-6_16.

[8] Watson, R. N. M., Anderson, J., Kennaway, K., and Laurie, B. Capsicum: practical capabilities for UNIX. In: *USENIX Security Symposium.* Vol. 46. 2. USENIX Association, 08/2010, 29–46. URL: http://www.usenix.org/events/sec10/tech/full_papers/Watson.pdf.

[9] Watson, R. N. M., et al. CHERI: A hybrid capability-system architecture for scalable software compartmentalization. *Symposium on Security and Privacy* (2015), 20–37. ISSN: 10816011.

[10] Wright, J. L. Software vulnerabilities: lifespans, metrics, and case study. PhD thesis. University of Idaho, 2014. URL: http://www.thought.net/papers/thesis.pdf.