The leaking battery

A privacy analysis of the HTML5 Battery Status API

Łukasz Olejnik[1], Gunes Acar[2], Claude Castelluccia[1], and Claudia Diaz[2]

[1] INRIA Privatics, Grenoble, France
{lukasz.olejnik,claude.castelluccia}@inria.fr
[2] KU Leuven, ESAT/COSIC and iMinds, Leuven, Belgium
{gunes.acar,claudia.diaz}@esat.kuleuven.be

Abstract. We highlight privacy risks associated with the HTML5 Battery Status API. We put special focus on its implementation in the Firefox browser. Our study shows that websites can discover the capacity of users' batteries by exploiting the high precision readouts provided by Firefox on Linux. The capacity of the battery, as well as its level, expose a fingerprintable surface that can be used to track web users in short time intervals. Our analysis shows that the risk is much higher for old or used batteries with reduced capacities, as the battery capacity may potentially serve as a tracking identifier. The fingerprintable surface of the API could be drastically reduced without any loss in the API's functionality by reducing the precision of the readings. We propose minor modifications to Battery Status API and its implementation in the Firefox browser to address the privacy issues presented in the study. Our bug report for Firefox was accepted and a fix is deployed.

1 Introduction

HTML5 Battery Status API enables websites to access the battery state of a mobile device or a laptop. Using the API, websites can check the battery level of a device and use this information to switch between energy-saving or high-performance modes. All the information exposed by the Battery Status API is available without users' permission or awareness.

The *"Security and privacy considerations"* section of the W3C specification that describes the Battery Status API, states the following: *"The information disclosed has minimal impact on privacy or fingerprinting, and therefore is exposed without permission grants"* [14]. Our findings, however, show that the API, as implemented by the Firefox browser on GNU/Linux operating system, enables fingerprinting and tracking of devices with batteries in short time intervals.

As of June 2015, Firefox, Chrome and Opera are the only three browsers that supported the Battery Status API [3]. Although the potential privacy problems of the Battery Status API were discussed by Mozilla and Tor Browser developers as early as in 2012 [2, 1, 22], neither the API, nor the Firefox implementation, has undergone a major revision. We hope to draw attention to this privacy issue by demonstrating the ways to abuse the API for fingerprinting and tracking.

We present an analysis of Battery Status API as implemented by Firefox on GNU/Linux. Our analysis indicate that seemingly innocuous information provided by the Battery Status API can serve as a tracking identifier when implemented incorrectly.

The core contributions of this work are:

1. *We present a new device fingerprinting vector based on the Battery Status API.* We show that the Firefox's implementation of the Battery Status API allows the discovery of battery's capacity, provides short-term identifiers that facilitates tracking and potentially can be used for reinstantiating identifiers (*respawning*).

2. *We propose a solution that reduces the Battery Status API's fingerprintable surface* by rounding the level readings provided by the API. Our fix does not cause any loss in the effective functionality of the API. We filed a bug report for Mozilla Firefox to communicate the problem and the proposed solution [20]. The fix was quickly implemented and deployed by Mozilla engineers in response to our bug report.

2 Related work

The Panopticlick [9] study by Eckersley demonstrated the feasibility of browser fingerprinting for online tracking by measuring the entropy present in the browser properties such as screen size, list of system fonts and browser plugins. Other researchers demonstrated the many ways browsers can be fingerprinted using different properties, such as clock skew [13], font metrics [10], network protocol characteristics [7], JavaScript engine performance [16], WebGL and canvas rendering [17].

Recently, studies measured the prevalence of the browser fingerprinting on the Web [19, 5, 4], suggesting that questionable practices such as proxy circumvention or stealthy techniques to exercise browser fingerprinting are commonly used by the websites.

In a similar vein, researchers studied *zombie cookie* (or *evercookie*) which is another tracking mechanism that can be used to reconstruct tracking identifiers - even if the user decides to clear her history [12] - with the use of Flash cookies [21], ETags [6] and other vectors.

A recent work, indepentend to ours, includes a very short note about the possible use of Battery API as a potential privacy risk vector [18]. The problem is not further described or analyzed, and the authors only mention potential risks due to monitoring of charging and discharging rates. In essence, our analysis is more extensive and detailed. Moreover, we describe a clear risk in relation to Firefox browser and study it in detail.

3 Background

3.1 Battery Status API

World Wide Web Consortium's (W3C) Battery Status API allows the reading of battery status data. Among the offered information are the current battery level and predicted time to charge or discharge. The respective properties `level`, `chargingTime` and `dischargingTime` can be accessed in JavaScript by first calling the `navigator.getBattery()` method [3] to get a `BatteryManager` object which then exposes these properties.

The API does not require user permission to read the battery information, any website or third-party scripts included on them, can use the API. The API also does not require browsers to notify users when the battery information is accessed. That allows website and third-party scripts to access the battery information transparently - without users' awareness.

The Battery Status API also provides JavaScript event handlers that allow the monitoring of updates to battery status. The API defines the `level` property as a double-precision floating-point number, taking values between 0 (depleted) and 1.0 (full) [14].

[3] Firefox does not implement `navigator.getBattery()` method, instead, it exposes a `navigator.battery` object.

3.2 Power information under Linux

In our exploratory survey of the Battery Status API implementations, we observed that the battery level reported by the Firefox browser on GNU/Linux was presented to Web scripts with *double* precision. An example battery level value observed in our study was 0.9301929625425652. We found that on Windows, Mac OS X and Android, the battery level reported by Firefox has just two significant digits (e.g. 0.32).

Analyzing the Firefox source code, we found out that the battery level is read from *UPower*, a Linux tool allowing the access to the UPower daemon [11]. The UPower daemon provides access to comprehensive power-management data about the device. Specifically, it enables the access to detailed information about the battery status such as capacity, level, voltage and provides estimates about the discharge and charge times.

Analyzing the UPower source code (`linux/up-device-supply.c`) to understand how it computes the battery level, we compiled the following equations:

$$BatteryLevel = 0.01 \times Percentage \tag{1a}$$

$$Percentage = 100.0 \times \frac{Energy}{EnergyFull} \tag{1b}$$

$$Energy = \frac{ChargeNow}{1,000,000} \times DesignVoltage \tag{1c}$$

$$EnergyFull = \frac{ChargeFull}{1,000,000} \times DesignVoltage \tag{1d}$$

The *Energy* is the current amount of energy present in the battery and measured in *watt-hours*. *EnergyFull* is also measured in *watt-hours* and represents the maximum possible amount of energy that can be stored in the battery. The *ChargeNow* and *ChargeFull* are measured in μAh and represent the current and maximum charge capacities of the battery respectively. Note that, due to the aging of the battery, *EnergyFull* tend to be lower than the design capacity of the battery, moreover, it can also change after a discharge, followed by a full charge – possibly for calibration purposes. Although many batteries share the same design capacities (e.g. 48.84 Wh or 62.16 Wh), as they age in time, their capacities may be reduced in different amounts, resulting in a diverse number of possible *EnergyFull* values, which are internally stored with four decimal places (e.g. 42.1678).

Since Firefox browser under Linux is accessing the UPower-provided data, it reads the *Percentage* value in 64 bit double precision floating point format and multiplies it by 0.01 to obtain the battery *level* as shown in 1(a). The *level* value is then exposed to website scripts through the Battery Status API in double precision.

As noted above, the *EnergyFull* value may change, as the battery capacity degrades. The UPower daemon updates the current capacity by comparing the *EnergyFull* to the latest value stored when the battery is fully charged.

4 Tracking with the Battery Status API

We measure the extent to which it is possible to link (and track) a device with battery using the battery level and charge/discharge time readouts. We observe how it could be leveraged for fingerprinting and tracking across sites. Moreover, we present a method to recover the battery's effective capacity (*EnergyFull*) using the precise battery level readouts provided by Firefox on Linux.

4.1 Tracking across sites

In this section, we discuss several potential fingerprinting and tracking scenarios. A third-party script that is present across multiple websites can link users' visits in a short time interval by exploiting the battery information provided to Web scripts. In order to do that, scripts can use the values of battery level, dischargingTime and chargingTime. The readings will be consistent on each of the sites, because of the fact that the update intervals (and their times) are identical. This could enable the third-party script to link these concurrent visits. Moreover, in case the user leaves these sites but then, shortly afterwards, visits another site with the same third-party script, the readings would likely be utilized to help in linking the current visit with the preceding ones.

Below we analyze more specific cases.

Frequency of battery status changes We analyzed the update rates under different computing loads (such as watching a movie, simply browsing the Web, etc).

We tested the rate of these changes by setting up a simple page and registering JavaScript event handlers for battery status changes; we monitored JavaScript readouts of level and dischargingTime, as well as the timestamps of these events. We analyzed the collected data for relative time differences between level, chargeTime and dischargeTime changes. The results indicated that for about 30 s, battery status may serve as a static identifier, allowing (e.g.) a third-party script to link visits from the same computer in short time intervals.

Number of possible identifiers In our test setting, the lowest indication of dischargeTime we observed was 355 (in seconds), and highest 40277 s. Assuming all the values spanning a range $(355, 40277)$ are possible, this gives 39922 numbers. We can also assume that users seeing a near-drained battery generally connect their notebooks to AC power. Assuming users start to charge their devices when the battery level is 0.1, this leaves 90 available battery level states (0.11 to 1.0). The number of potential levels denoted by a tuple $(level, dischargeTime)$ would then be a simple multiplication 90×39922 and the final number of possible states would be 3592980, which only accounts for the discharging state. Using the information about the battery charge ($chargingTime$) could effectively double the number of possible states. The probability of a $(level, dischargeTime)$ collision (between different users, and assuming a uniform distribution) is therefore low and for a short time frame this would effectively be a unique identifier.

However, we emphasize that the dischargeTime levels can be subject to frequent changes, in response to change in the users' computer use patterns. This means that, in practice, the risk of long-term tracking with this information may be negligible. Moreover, depending on the battery level, some chargeTime or dischargeTime values may not be observed in practice [4]. Yet, the available combinations could be used to distinguish users behind a NAT (Network Address Translation). In such a setting, the computers may have similar fingerprints [9] and often identical public IP addresses. The readouts from the battery may allow distinguishing these users.

Reconstructing user identifiers in short-time intervals Users who try to re-visit a website with a new identity may use browsers' private mode or clear cookies and other client side identifiers. When consecutive visits are made within a short interval, the website can link users' new and old identities by exploiting battery level and charge/discharge times. The website can then reinstantiate users' cookies and other client side identifiers, a method known as *respawning* [21].

[4] For instance, 355 s dischargeTime may be too short for a full battery or, 40277 s dischargeTime may be too long for a battery with level 0.1.

Note that, although this method of exploiting battery data as a linking identifier would only work for short time intervals, it may be used against power users who can not only clear their cookies but can go to great lenghts to clear their evercookies.

5 Detecting battery capacity

In addition to using battery level and charge/discharge times for linking visits in short time intervals, Battery Status API can be used to infer the current battery capacity (*EnergyFull*) of a device if it allows high precision level readouts. In this section, we analyze the possibility of fingerprinting a device by exploiting high precision battery level readouts provided by the Firefox on Linux operating system.

We found that using the 64-bit double precision floating point battery level readouts from Firefox on Linux, it is possible to discover the value of *EnergyFull*, which signifies the actual battery capacity. We emphasize that our method only works for UPower and Firefox on Linux, and during our study we encountered some computers for which we cannot recover the capacity with our method. This can be due to the differences in how processors handle floating point calculations [5] or measurement errors in UPower.

The attack works by using the equations *1a-1d* by reading the battery level and finding candidate *Energy*, *EnergyFull* and *Voltage* levels which may give this floating point number reading. In order to do this, attacker may either brute-force the candidate values by testing all possible values or precompute a lookup table.

5.1 Test method

Assuming a uniform space of EnergyFull values (X, Y), we tested all the hypothetical level readouts to detect the possible identifiers. It is obvious that, for a given level reading, several possibilities for EnergyFull level may exist. However, if the attacker has access to multiple battery level readouts, the number of collisions becomes significantly smaller. We analyzed the number of potential EnergyFull candidates as a function of the battery level readouts.

In other words, we computed the number of collisions for one battery level readout, $State_1$. For each such possible readout level, we simulated another different readout level (different than the one in preceding state), $State_2$ (battery levels in $State_1$ and $State_2$ are different). We compared the candidate EnergyFull values in $State_1$ and $State_2$ and intersecting the sets of possible EnergyFull levels, we effectively decreased the number of candidate EnergyFull values. The number of EnergyFull candidates for a total of 1559 battery level readings are displayed on Figure 1. Figure 2, on the other hand, shows the reduction of EnergyFull candidates when a script is able read the battery level multiple times from the same device. The figure is based on 1559 battery level readings collected from a laptop running Ubuntu 12.04 operating system. We highlight that such analysis is made possible due to the fixed space a floating-point value can represent, and relatively limited capacities of batteries used in practice [6].

[5] See, for example, [8, 15] on the "floating-point determinism problem."

[6] Observe that, possible capacities in this calculations include the reduced battery capacities (e.g. not limited to battery capacities on the market). Still, we could find the candidate capacities on a off-the-shelf computer without a significant computation overhead. We believe, an adversary with moderate storage resources can easily build a lookup table to further reduce the computation time.

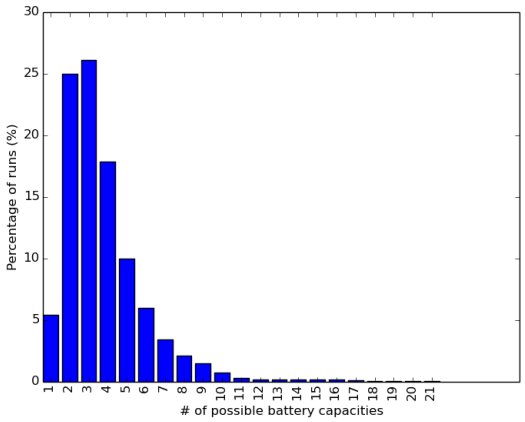

Fig. 1: Distribution of number of candidate battery EnergyFull values for a total of 1559 battery level readings (runs). In 5% of the cases the attacker can detect the battery capacity with just one reading.

6 Defense

In the following subsections we outline possible defenses against the exploitation of the Battery Status API for fingerprinting and tracking.

6.1 Limiting the precision of level readouts

In order to limit the tracking and fingerprinting potential of the Battery Status API, the implementations should avoid providing high-precision values. By simply rounding the `level` value of the battery, the threat would be minimized, without losing any functionality of the API. This comment especially applies to platforms where the OS provides high-precision read-outs about the battery.

We filed an appropriate bug report to Firefox implementation, pointing out the inconsistency of level reporting across different platforms [20]. The fix was implemented and deployed as of June 2015.

Moreover, we believe the Battery Status API could mention the risk of exposing high precision readouts in the "Security and privacy considerations" section of the standard. We plan to communicate the results of the study to the editors of the API.

6.2 Asking for user permission to access the Battery Status API

We also discussed potential scenarios where even the reduced precision of the level readout and charge/discharge times could constitute a tracking identifier in short time intervals. In these scenarios, the exposed battery information may allow an attacker to reinstantiate tracking identifiers in a manner similar to evercookies.

In order to prevent this, browser vendors might require user permissions for accesing the Battery Status API. Although this has been suggested by some concerned Mozilla developers [2], final decision was to make the API available without permissions. We believe, as a minimum,

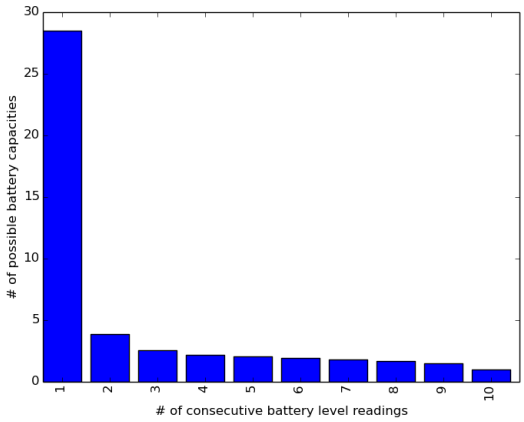

Fig. 2: Average number of candidate battery EnergyFull values as a function of consecutive battery level readings. Attacker can significantly reduce the number of candidate battery capacities if he can read the battery level multiple times.

users should be able to choose to be asked for battery access by Web scripts. As an alternative, browsers can enforce the user permission requirement in their private browsing modes.

To the best of our knowledge, the only browser that has a strong defense against fingerprinting by the Battery Status API is Tor Browser. Tor Browser completely disables the API [22] to thwart possible fingerprinting attempts.

Finally, the information on the API use could be made available to the user to aid transparency. We are advocating for streamlining the information to users, either directly via the browser's user interfaces, or at least by allowing to read the respective information by custom-made browser extensions. In this way, software could allow the users to learn and be aware about the use of the battery information on devices they own.

7 Conclusion

We analyzed the privacy implications of the Battery Status API, with a focus on its implementation in Firefox for Linux operating system. Our analysis shows that the high precision battery level readings provided by Firefox can lead to an unexpected fingerprinting surface: the detection of battery capacity.

In short time intervals, Battery Status API can be used to reinstantiate tracking identifiers of users, similar to evercookies. Moreover, battery information can be used in cases where a user can go to great lenghts to clear her evercookies. In a corporate setting, where devices share similar characteristics and IP addresses, the battery information can be used to distinguish devices behind a NAT, of traditional tracking mechanisms do not work.

The analysis of Web standards, APIs and their implementations can reveal unexpected Web privacy problems by studying the information exposed to Web pages. The complex and sizable nature of the new Web APIs and their deeper integration with devices make it hard to defend against such threats. Privacy researchers and engineers can help addressing the risks imposed by these APIs by analysing the standards and their implementations for their effect on Web privacy and tracking. This may not only provide an actionable feedback to API designers and browser manufactureres, but can also improve the transparency around these new technologies.

References

1. Proposal for a smaller battery API . https://groups.google.com/forum/#!searchin/mozilla.dev.webapi/Why$20is$20the$20battery$20API$20exposed$20to$20unprivileged$20content$3F/mozilla.dev.webapi/6gLD78z6ASI/Sz1DH2gWN9wJ, 2012. Accessed: 24.6.14.
2. Why is the battery API exposed to unprivileged content? https://groups.google.com/forum/#!topic/mozilla.dev.webapi/V361K7c0olQ/discussion, 2012. Accessed: 26.3.14.
3. Battery Status API - Can I use... Support tables for HTML5, CSS3, etc, 2014.
4. G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz. The Web never forgets: Persistent tracking mechanisms in the wild. In 21st ACM Conference on Computer and Communications Security (CCS), pages 674–689. ACM, 2014.
5. G. Acar, M. Juarez, N. Nikiforakis, C. Diaz, S. Gürses, F. Piessens, and B. Preneel. FPDetective: Dusting the Web for fingerprinters. In 20th ACM Conference on Computer and Communications Security (CCS), pages 1129–1140. ACM, 2013.
6. M. Ayenson, D. J. Wambach, A. Soltani, N. Good, and C. J. Hoofnagle. Flash cookies and privacy II: Now with HTML5 and ETag respawning. World Wide Web Internet And Web Information Systems, 2011.
7. Y.-C. Chen, Y. Liao, M. Baldi, S.-J. Lee, and L. Qiu. OS Fingerprinting and Tethering Detection in Mobile Networks. pages 173–179, 2014.
8. B. Dawson. FloatingPoint Determinism — Random ASCII. https://randomascii.wordpress.com/2013/07/16/floating-point-determinism/, 2013. Accessed: 31.8.15.
9. P. Eckersley. How unique is your web browser? In Privacy Enhancing Technologies, pages 1–18. Springer, 2010.
10. D. Fifield and S. Egelman. Fingerprinting Web Users through Font Metrics. In Financial Cryptography and Data Security (FC). Springer-Verlag, 2015.
11. R. Hughes. UPower Reference Manual. http://upower.freedesktop.org/docs/, 2010. Accessed: 22.6.14.
12. S. Kamkar. Evercookie. http://samy.pl/evercookie, 2010. Accessed: 24.6.14.
13. T. Kohno, A. Broido, and K. C. Claffy. Remote physical device fingerprinting. IEEE Transactions on Dependable and Secure Computing, 2(2):93–108, 2005.
14. A. Kostiainen and M. Lamouri. Battery Status API, 2012.
15. D. Monniaux. The pitfalls of verifying floating-point computations. ACM Transactions on Programming Languages and Systems (TOPLAS), 30(3):12, 2008.
16. K. Mowery, D. Bogenreif, S. Yilek, and H. Shacham. Fingerprinting information in JavaScript implementations. In Web 2.0 Workshop on Security and Privacy (W2SP), volume 2. IEEE, 2011.
17. K. Mowery and H. Shacham. Pixel perfect: Fingerprinting canvas in HTML5. In Web 2.0 Workshop on Security and Privacy (W2SP). IEEE, 2012.
18. G. Nakibly, G. Shelef, and S. Yudilevich. Hardware fingerprinting using HTML5. CoRR, abs/1503.01408, 2015.
19. N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In Security and Privacy (SP), 2013 IEEE Symposium on, pages 541–555. IEEE, 2013.
20. L. Olejnik. Bug 1124127 - Round Off Navigator Battery Level on Linux. https://bugzilla.mozilla.org/show_bug.cgi?id=1124127, 2015. Accessed: 30.2.15.
21. A. Soltani, S. Canty, Q. Mayo, L. Thomas, and C. J. Hoofnagle. Flash cookies and privacy. In AAAI Spring Symposium: Intelligent Information Privacy Management, 2010.
22. Tor Bugs: TorBrowser Bundle. #5293 Hook charging+discharching rates in Battery API . https://trac.torproject.org/projects/tor/ticket/5293, 2012. Accessed: 24.6.14.