The Effectiveness of Application Permissions

Adrienne Porter Felt*, Kate Greenwood, David Wagner
University of California, Berkeley
apf, kate_eli, daw@cs.berkeley.edu

Abstract

Traditional user-based permission systems assign the user's full privileges to all applications. Modern platforms are transitioning to a new model, in which each application has a different set of permissions based on its requirements. Application permissions offer several advantages over traditional user-based permissions, but these benefits rely on the assumption that applications generally require less than full privileges. We explore whether that assumption is realistic, which provides insight into the value of application permissions.

We perform case studies on two platforms with application permissions, the Google Chrome extension system and the Android OS. We collect the permission requirements of a large set of Google Chrome extensions and Android applications. From this data, we evaluate whether application permissions are effective at protecting users. Our results indicate that application permissions can have a positive impact on system security when applications' permission requirements are declared up-front by the developer, but can be improved.

1 Introduction

Browsers and smartphone operating systems provide development platforms that support thriving markets for third-party applications. However, third-party code creates risks for the user. Some third-party authors are malicious [3, 14], and third-party code can introduce vulnerabilities because the authors of third-party applications usually are not security experts [10, 19].

In order to protect users from the threats associated with third-party code, modern platforms use application permissions to control access to security- and privacy-relevant parts of their APIs. Users decide whether to allow individual applications to access these sensitive resources. *Time-of-use* systems prompt users to approve permissions as needed by applications at runtime, and *install-time* systems ask developers to declare their applications' permission requirements up-front so that users can grant them during installation.

Traditional user-based permission systems assign the user's full privileges to all of the user's applications. In the application permission model, however, each application can have a customized set of permissions based on its individual privilege requirements. If most applications can be satisfied with less than the user's full privileges, then three advantages of application permissions over the traditional user-based model are possible:

- **User Consent:** Security-conscious users may be hesitant to grant access to dangerous permissions without justification. For install-time systems, this might alert some users to malware at installation; for time-of-use systems, this can prevent an installed application from accessing sensitive content.
- **Defense in Depth:** For install-time systems, the impact of a vulnerability in an application will be limited to the vulnerable application's declared privileges. This could also be true for a time-of-use system in which developers declare their applications' maximum possible permissions up-front.
- **Review Triaging:** Up-front application permission declarations facilitate central review because security reviewers can ignore low-privilege applications and focus on applications with dangerous permissions. This may decrease the average review time.

The real-world impact of these potential advantages depends on low application permission requirements. We evaluate the practical benefits of application permissions. by performing a large-scale study of Google Chrome extensions and Android applications.

We perform a measurement study that quantifies the permission use of 1000 Google Chrome extensions and 956 Android applications. Both platforms use install-time permissions. Our study provides detailed data on the permission requirements of applications in the wild. From this data, we assess whether the platforms are realizing the potential benefits of application permissions.

We find that almost all applications ask for fewer than maximum permissions. Only 14 of 1000 extensions request the most dangerous privileges, and the average Android application requests fewer than 4 of 56 available

*This material is based upon work supported under a National Science Foundation Graduate Research Fellowship. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

dangerous permissions. In comparison, all typical desktop applications receive the equivalent of all 56 Android permissions. These results indicate that application permission systems with up-front permission declarations can decrease the impact of application vulnerabilities and simplify review. This supports the adoption of install-time permission systems. Current time-of-use platforms do not require up-front permission declarations, which means that they do not provide the same defense in depth or review triaging benefits. However, time-of-use platforms could gain these advantages by requiring developers to state the maximum set of permissions that the application will require at runtime.

Although developers use fewer than full permissions, Google Chrome and Android users are presented with at least one dangerous permission request during the installation of almost every extension and application. Warning science literature indicates that frequent warnings desensitize users, especially if most warnings do not lead to negative consequences [15, 11]. Users are therefore not likely to pay attention to or gain information from install-time permission prompts in these systems. Changes to these permission systems are necessary to reduce the number of permission warnings shown to users.

We examine the effects of developer incentives, developer error, wildcard permissions, and permission granularity on permission usage. We find that more than 10% of applications request unneeded permissions. Our results show that developers are willing to make use of fine-grained permissions, motivating a fine-grained permission design. We suggest error detection tools and other platform changes to reduce permission requests.

We view the Google Chrome and Android permission systems as case studies for the future of application permissions. Our primary contribution is a large-scale study that demonstrates the defense in depth and review triaging benefits of application permissions with up-front declarations. This motivates the addition of up-front developer permission declarations to time-of-use permission systems. We also discuss tools and changes that would improve the effectiveness of these systems. Our results should guide the design of in-progress and future permission systems, and we also provide concrete suggestions for Google Chrome and Android.

2 Background

Popular platforms with application permissions include Apple iOS, the Safari extension system, and Facebook. In-progress platforms with application permissions include Mozilla Jetpack, the W3C device API for web applications, and the Google installable web application platform. In this paper, we focus on the Google Chrome extension system and Android application platform, which feature install-time permissions.

2.1 Google Chrome Extensions

Browser extension platforms allow third-party code to run as part of the browser environment. *Extensions* change the user's browsing experience by editing web sites and changing browser behavior. All extensions are free from the official Google Chrome extension gallery.

Google Chrome extensions can have three types of components: core extensions, content scripts, and plug-ins. A core extension comprises the main, persistent portion of an extension. Content scripts are injected into web sites; when the page loads, the content script's JavaScript executes in the context of the site. A content script has full access to its host site's page. Core extensions and content scripts are written in JavaScript, whereas plug-ins are native executables.

The extension gallery prompts the user with a warning (e.g., Figure 1) that indicates what privileges the extension has requested:

Plug-ins. Plug-ins are native executables, so a plug-in grants the extension full permissions to the user's machine. The installation warning for an extension with a plug-in says the extension "can access all data on your computer." Extensions with plug-ins are reviewed.

Browser managers. Core extensions can access the extension API, which is a set of browser managers. Each manager is controlled with one permission. The managers include history, bookmarks, and geolocation. The browser warns that extensions with these permissions can access "your browsing history," "bookmarks," and "your physical location," respectively. Non-security relevant browser managers also exist, but their use does not prompt a warning so we do not consider them.

Web access. The developer must specify web permissions for content scripts and core extensions. Content script web permissions determine which domains they are installed on by default. A core extension can send `XMLHttpRequests` (XHRs) and inject code into the domains it has permissions for. Content script and core extension domain permissions are listed separately.

All-domain access is the broadest web permission. If either the content scripts or the core extension have all-domain access, the browser warning states that the exten-

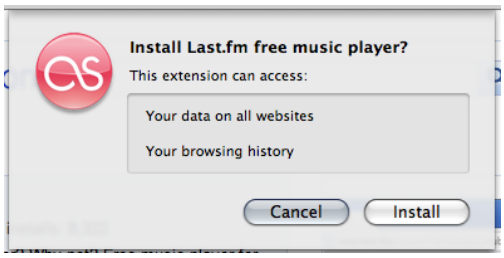

Figure 1: Google Chrome extension installation.

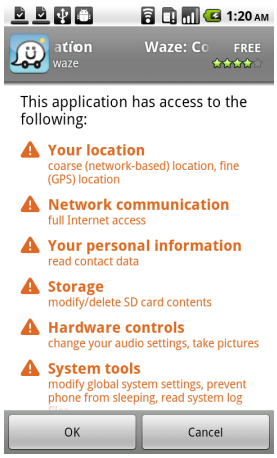

Figure 2: Android application installation.

sion "can access your data on all web sites." Alternately, a developer can list specific domains, using wildcards for protocols or subdomains (e.g., `*://*.bar.com`). The installation warning will list the requested domains.

2.2 Android Applications

The Android smartphone operating system supports third-party Java applications, which can be installed by users through the Android Market. Some third-party applications are free, and some are paid. Android applications do not need to be reviewed prior to inclusion in the Android Market, although other phone vendors like RIM and Apple maintain official review processes.

Android's API provides access to phones' cameras, microphones, GPS, text messages, WiFi, Bluetooth, etc. Most device access is controlled by permissions, although large parts of the overall API are not protected by permissions. Applications can define their own extra permissions, but we only consider permissions defined by the Android OS. There are 134 permissions in Android 2.2. Permissions are categorized into threat levels:

Normal. API calls with annoying but not harmful consequences are protected with Normal permissions. Examples include accessing information about available WiFi networks, vibrating the phone, and setting the wallpaper.

Dangerous. API calls with potentially harmful consequences are protected with Dangerous permissions. These include actions that could cost the user money or leak private information. Example permissions are the ones used to protect opening a network socket, recording audio, and using the camera.

Signature. The most sensitive operations are protected with Signature permissions. These permissions are only granted to applications that have been signed with the device manufacturer's certificate. Market applications are only eligible for Signature permissions if they are updates to applications that were pre-installed and signed by the device manufacturer. Requests for Signature permissions by other applications will be ignored. An example is the ability to inject user events.

SignatureOrSystem. This category includes signed applications and applications that are installed into the `/system/app` folder. Typically, this only includes pre-installed applications. Advanced users who have rooted their phones [9] can manually install applications into this folder, but the official Market installation process will not do so. Requests for SignatureOrSystem permissions by other applications will be ignored. For example, these permissions protect the ability to turn off the phone.

The Android Market displays a permission prompt to the user during installation for Dangerous permissions (e.g., Figure 2). Warnings for Dangerous permissions are grouped into functionality categories. For example, all Dangerous permissions related to location are displayed as part of the same location warning. Normal permissions can be viewed once the application is installed but are hidden behind a collapsed drop-down menu. Signature/System permissions are not displayed to users at all.

3 Permission Prevalence

We examine the frequency of permission requests in Google Chrome extensions and Android applications. These results should be compared to traditional systems that grant all applications full privileges.

3.1 Chrome Extensions

We study the 1000 "most popular" extensions, as ranked in the official Google Chrome extension gallery[1]. Of these, the 500 most popular extensions are relevant to user consent and application vulnerabilities because they comprise the majority of user downloads. The 500 less popular extensions are installed in very few browsers, but they are relevant to reviewers because reviewers would need to examine all extensions in the directory. Table 1 provides an overview of our results.

3.1.1 Popular Extensions

Of the 500 most popular extensions, 91.4% ask for at least one security-relevant permission. This indicates that nearly every installation of an extension generates at least one security warning[2].

[1] We crawled the directory on August 27, 2010.

[2] We discovered that Google Chrome sometimes fails to generate a warning for history access. The bug has been fixed for new versions [7]. Our analysis assumes that all requests for history access correctly generate a warning. The bug affects 5 of extensions in our set.

Permission	Popular	Unpopular
Plug-ins	2.80 %	0.00 %
Web access	82.0 %	60.8 %
All domains	51.6 %	21.8 %
Specific domains	30.4 %	39.0 %
Browser manager(s)	74.8 %	43.4 %

Table 1: We measure the prevalence of permissions in 1000 Google Chrome extensions, split into the 500 most popular and 500 less popular. For web access, we report the highest permission of either the content script or core extension.

Plug-ins. Only 14 of the 500 extensions include plug-ins.

Browser managers. The majority of security warnings are caused by the window manager, which is requested by almost 75% of the 500 extensions. Requesting access to the window manager generates a warning about history access because history is indirectly available through the window manager. The bookmark and geolocation managers are requested infrequently: 44 times and once, respectively.

All domains. Half of the 500 extensions request all-domain access for either content scripts or the core extension. 52% request access to all `http` sites, and 42% ask for all `https` sites.

Specific domains. One-third of extensions only request a set of specific domains. This reduces the attack surface and removes the possibility that an extension is snooping on sensitive web data.

No warning. Only 43 of the 500 extensions do not request access to a security-relevant permission. 38 do not ask for any permissions at all; they load normal web sites into their extension windows or apply "themes" to the user interface. The remainder use browser managers that are not relevant to privacy or security.

3.1.2 Unpopular Extensions

Not all of the extensions listed in the "most popular" directory ranking are popular. After approximately the first 500 of 1000 popularity-ranked extensions, the number of users per extension abruptly decreases, and applications are no longer ranked solely according to the number of users. (Although the ranking algorithm is private, we believe it incorporates time.) Figure 3 shows the transition. 16.2% of the bottom 500 extensions have fewer than ten users. These 500 low-ranked extensions are of uneven quality. E.g., two of them are unaltered versions of the example extension on the developer web site.

Table 1 presents the results of our survey of the 500 less popular extensions. 71.6% of the less popular extensions have at least one security-relevant permission. When compared to the top 500 extensions, the unpopular extensions request far fewer permissions than popular extensions. We hypothesize that this is because less popular extensions offer less functionality. All of the differences are significant at a 1% significance level.

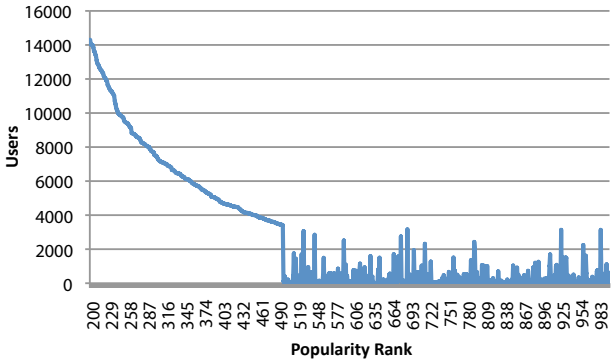

Figure 3: Users per extension. We omit the first 200 for graph clarity; the most popular extension has 1.3M users.

Unranked extensions are strictly less popular than the unpopular extensions in our data set. If one were to review the remaining 5,696 unranked Google Chrome extensions, we expect their permission requirements would be equivalent to or less than the permission requirements of these 500 unpopular applications. We note with caution that future studies on permissions need to consider the effect of popularity. E.g., a study that looks at the full set of 6,696 extensions to evaluate warning frequency would would likely underestimate the number of warnings that users see in practice by approximately 20%.

3.1.3 Evaluation

User Consent. Nearly all popular extensions (91% of the top 500) generate at least one security warning, which decreases the value of the warnings. History and all-domain permissions are requested by more than half of extensions; users have no reason to be suspicious of extensions with these permissions because they are not anomalous. However, warnings about plug-ins are rare and therefore potentially notable.

Defense in Depth. This study shows that the permission system dramatically reduces the scope of potential extension vulnerabilities. A negligible number of extensions include plug-ins, which means that the typical extension vulnerability cannot yield access to the local machine. This is a significant improvement over the Firefox and Internet Explorer extension systems, which provide *all* extensions with access to the local file system. We also find that all-domain access is frequent but not universal: 18% of popular extensions need no web access, and 30.4% only need limited web access. This means that the permission system prevents *half* of popular extensions from having unnecessary web privileges.

Review Triaging. Of the 1000 extensions in our study, only 1.4% require review under current Google Chrome review triaging procedures. (It should be noted, however, that we do not know how many extensions requiring review were submitted to the directory.) These results suggest that the Firefox extension review process could be significantly streamlined if Mozilla were to adopt a similar permission system. Reviewers could indisputably ignore 28% of submitted extensions regardless of the exact triaging criteria, based on the number of less-popular extensions with no security-relevant permissions.

3.2 Android Applications

We survey 100 paid and 856 free applications from the Android Market[3]. For the paid applications, we selected the 100 most popular. The free set is comprised of the 756 most popular and 100 most recently added applications. Unlike Google Chrome extensions, we observe no differences between popular and recently added free applications, so we present them together. It is possible that we do not see a popularity bias in Android applications because of differences in the developer communities and entrance barriers. We do not compare applications based on their categories in the Android Market; the categories are loosely defined and include a wide variety of different functionality [1]. Although Android applications written by the same developer could collude, we consider each application's permissions independently. Legitimate developers have no incentive to hide communication and circumvent permission warnings.

3.2.1 Dangerous Permissions

We are primarily concerned with the prevalence of Dangerous permissions, which are displayed as a warning to users during installation and can have serious security ramifications if abused. We find that 93% of free and 82% of paid applications have at least one Dangerous permission, i.e., generate at least one permission prompt.

Android permissions are grouped into functionality categories, and Table 1(a) shows how many applications use at least one Dangerous permission from each given category. This provides a relative measure of which parts of the protected API are used by applications. All of the permissions in a category display the same permission prompt, so Table 1(a) also indicates how often users see each type of permission request.

A small number of permissions are requested very frequently. Table 1(b) shows the most popular Dangerous permissions. In particular, the `INTERNET` permission is heavily used. We find that 14% of free and 4% of paid applications request `INTERNET` as their only Dangerous permission. Barrera et al. hypothesize that free applications often need the `INTERNET` permission only to load advertisements [1]. The disparity in `INTERNET` use between free and paid applications supports this hypothesis, although it is still the most popular permission for paid applications.

[3]The applications were collected in October 2010.

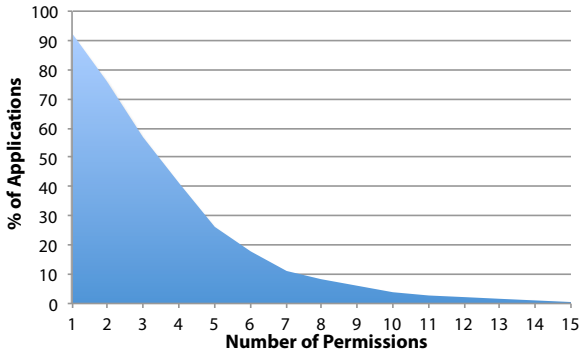

Figure 4: Percentage of paid and free applications with *at least* the given number of Dangerous permissions.

The prevalence of the `INTERNET` permission means that most applications with access to personal information also have the ability to leak it. For example, 97% of the 225 applications that ask for `ACCESS_FINE_LOCATION` also request the `INTERNET` permission. Similarly, 94% and 78% of the respective applications that request `READ_CONTACTS` and `READ_CALENDAR` also have the `INTERNET` permission. We find that significantly more free than paid applications request both Internet access and location data, which possibly indicates widespread leakage of location information to advertisers in free applications. This corroborates a previous study that found that 20 of 30 randomly selected free applications send user information to content or advertisement servers [5].

Although many applications ask for at least one Dangerous permission, the number of permissions required by an application is typically low. Even the most highly privileged application in our set asks for 26 permissions, which is less than half of the available 56 Dangerous permissions. Figure 4 shows the distribution of Dangerous permission requests. Paid applications use an average of 3.99 Dangerous permissions, and free applications use an average of 3.46 Dangerous permissions.

3.2.2 Signature and System Permissions

Applications can request Signature and SignatureOrSystem permissions, but the operating system will not grant the request unless the application has been signed by the device manufacturer (Signature) or installed in the `/system/app` folder (System). It is pointless for a typical application to request these permissions because the permission requests will be ignored.

(a) Prevalence of Dangerous permissions, by category.

Category	Free	Paid
NETWORK**	87.3 %	66 %
SYSTEM_TOOLS	39.7 %	50 %
STORAGE**	34.1 %	50 %
LOCATION**	38.9 %	25 %
PHONE_CALLS	32.5 %	35 %
PERSONAL_INFO	18.4 %	13 %
HARDWARE_CONTROLS	12.5 %	17 %
COST_MONEY	10.6 %	9 %
MESSAGES	3.7 %	5 %
ACCOUNTS	2.6 %	2 %
DEVELOPMENT_TOOLS	0.35 %	0 %

(b) The most frequent Dangerous permissions and their categories.

Permission (Category)	Free	Paid
INTERNET** (NETWORK)	86.6 %	65 %
WRITE_EXTERNAL_STORAGE** (STORAGE)	34.1 %	50 %
ACCESS_COARSE_LOCATION** (LOCATION)	33.4 %	20 %
READ_PHONE_STATE (PHONE_CALLS)	32.1 %	35 %
WAKE_LOCK** (SYSTEM_TOOLS)	24.2 %	40 %
ACCESS_FINE_LOCATION (LOCATION)	23.4 %	24 %
READ_CONTACTS (PERSONAL_INFO)	16.1 %	11 %
WRITE_SETTINGS (SYSTEM_TOOLS)	13.4 %	18 %
GET_TASKS* (SYSTEM_TOOLS)	4.4 %	11 %

Table 2: Survey of 856 free and 100 paid Android applications. We indicate significant difference between the free and paid applications at 1% (**) and 5% (*) significance levels.

As far as we are aware, none of the paid applications in our data set are signed or distributed by device manufacturers. Three of the paid applications request Signature permissions, and five request SignatureOrSystem permissions. Of the free applications, 25 request Signature permissions, 30 request SignatureOrSystem permissions, and four request both. We have found four of the aforementioned free applications pre-installed on phones; the remainder will not receive the permissions on a typical device. Requests for unobtainable permissions may be developer error or leftover from testing.

3.2.3 Evaluation

User Consent. Nearly all applications (93% of free and 82% of paid) ask for at least one Dangerous permission, which indicates that users are accustomed to installing applications with Dangerous permissions. The INTERNET permission is so widely requested that users cannot consider its warning anomalous. Security guidelines or anti-virus programs that warn against installing applications with access to both the Internet and personal information are likely to fail because almost all applications with personal information also have INTERNET.

Several important categories are requested relatively infrequently, which is a positive finding. Permissions in the PERSONAL_INFO and COST_MONEY categories are only requested by a fifth and a tenth of applications, respectively. The PERSONAL_INFO category includes permissions associated with the user's contacts, calendar, etc.; COST_MONEY permissions let applications send text messages or make phone calls without user confirmation[4]. Users have reason to be suspicious of applications that ask for permissions in these categories. However, users may not notice these rare warnings because the overall rate is so high.

[4]The separate PHONE_CALLS category contains permissions that modify telephony state but do not cost the user money.

Defense in Depth. Given the prevalence of Dangerous permissions, an application vulnerability is likely to occur in an application with at least one Dangerous permission. However, the average Android application is much less privileged than a traditional operating system program. Every desktop Windows application has full privileges, whereas no Android application in our set requests more than half of the available Dangerous permissions. A majority of the Android applications ask for less than seven, and only 10% have access to functionality that costs the user money. This is a significant improvement over the traditional full-privilege, user-based approach.

Review Triaging. A hypothetical review process could exempt applications that do not have Dangerous permissions. Unfortunately, this alone would not reduce reviewer workload much. Only 18% of paid and 7% of free applications would be exempt from review. To improve this, a review process could also exclude applications whose only Dangerous permission is INTERNET. An application with only the INTERNET permission cannot leak sensitive personal information because reading user data requires a second permission. This would increase the number of exempt applications to 22% of paid and 21% of free applications.

4 Reducing Application Privileges

Our application survey indicates that up-front permission declarations can promote defense in depth security and provide moderate review triaging advantages. However, a large number of applications still ask for dangerous permissions. Decreasing the number of privileges that applications require to function will improve the utility of permissions. We investigate factors that influence permission requirements and present corresponding suggestions for reducing the frequency of permission usage.

4.1 Developer Incentives

Developer incentives can encourage or discourage permission requests. Current incentives include the length of the review process, how the automatic update system treats additional permissions, and pressure from users.

Review Process. Formal review can delay an application's entrance into the directory. Developers are often concerned about the length of the review process [13]. If dangerous permissions increase the review time (and a lack of dangerous permissions decreases it), then developers have an incentive to use as few permissions as necessary. Google Chrome extensions have to undergo a review process if they include plug-ins, which incentivizes developers to not use plug-ins. Other platforms could adopt similar review systems or institute a timed delay for applications with more permissions.

Pressure From Users. The ultimate goal of a developer is to reach as many users as possible. If users are hesitant to install applications with certain permissions, then developers are motivated to avoid those permissions. Users can express their dislike of permission requests in application comments and e-mails to the developer.

We read the user comments for 50 randomly selected Google Chrome extensions with at least one permission. Of the 50 extensions, 8 (15%) have at least one comment questioning the extension's use of permissions. The percentage of comments pertaining to permissions ranges widely, from 1 of 2 to 5 of 984. A majority of the permission comments refer to the extension's ability to access browsing history. Several commenters state that the permission requests are preventing them from installing an application, e.g., "Really would like to give it a try. ... But why does it need access to my history? I hope you got a plausible answer because I really would like to try it out." These comments indicate that a small number of users are pressuring developers to use fewer permissions.

Additionally, developers of 3 of the 50 extensions descriptions include an explanation of their permission usage. This indicates that these developers are concerned about user reactions to permission requests.

Automatic Updates. Android and Google Chrome automatically update applications as they become available, according to user preferences. However, automatic updates do not proceed for applications whose updates request more permissions. Instead, the user needs to manually install the update and approve the new permissions; in Android, this amounts to several additional screens. This incentivizes developers to request unnecessary permissions in in case later versions require the permissions. If update UIs were improved to minimize the user effort required to update applications with new permissions, this disincentive might be eliminated.

4.2 Developer Error

Developers may ask for unnecessary permissions due to confusion or forgetfulness. We explore the prevalence of developer error. Tools that help developers select correct permissions could reduce application privileges without requiring any changes to the permission system itself.

4.2.1 Errors in Google Chrome Extensions

Browser Managers. We count the extensions that request browser managers but do not use them. About half of the extensions in our set of 1000 "popular" extensions request access to security-relevant browser managers. We search their source code (including remotely sourced scripts) for references to their requested browser managers. 14.7% of the 1000 extensions are overprivileged by this measure because they request access to managers that they never use. It is possible for an extension to name a browser manager without explicitly including the name as a string (e.g., `"book"+"marks"`); we examined a random sample of 15 overprivileged extensions and found no evidence of developers doing this.

Domains. We also review fifty randomly selected extensions for excessive domain access (see Appendix A). For each extension, we compare the permissions it requests with the domains needed to implement its functionality, which we determine by manually exercising the user interface and consulting its source code when necessary. We find that 41 of the 50 extensions request access to web data, and 7 of those are overprivileged: 5 request too many domain permissions for their core extensions, and 2 install content scripts on unnecessary domains.

The reasons for overprivilege are diverse. One example is "PBTweet+", which requests web access for a nonexistent core extension; other examples are "iBood" and "Castle Age Autoplayer", which request access to all domains even though they only interact with iBOOD and Facebook, respectively.

"Send using Gmail (no button)" demonstrates a common error, which is that developers sometimes request access to all and specific domains in the same list. We find that an additional 27 of the 1000 popularity-ranked extensions also make this mistake. This is a conservative measure of wildcard-induced error; subdomain wildcards can feature the same mistake, like asking for both `http://www.example.com` and `http://*.example.com`.

4.2.2 Errors in Android Applications

We manually review the top free and top paid application from eighteen Android Market categories (see Appendix A for a list). For each of the applications, we compare

its functionality to the permissions it requests. To determine an application's functionality requirements, we exercise the user interface. Android's permission documentation is incomplete; when we were unable to determine whether functionality requires permissions, we conservatively assumed it does.

Of the 36 applications, 4 are overprivileged. Unnecessary `INTERNET` permissions account for three of the overprivileged applications. One of the developers may have done this with the mistaken belief that launching the browser requires the `INTERNET` permission, since that is how the application interacts with the Internet. The fourth overprivileged application requests `ACCESS_FINE_LOCATION` unnecessarily.

In addition to the four overprivileged applications, another four could re-implement the same functionality without the `INTERNET` permission. For example, "DocsToGo" provides the ability to update the application over the Internet even though that functionality is already provided by the Android Market, and "Jesus Hates Zombies" could store its small set of static resources locally.

4.2.3 Tools for Error Reduction

As far as we are aware, none of the prominent platforms with install-time permissions provide developer tools to detect unnecessary permissions. We recommend that future platforms provide developers with tools to guide the writing of permission declarations. Such a tool could help reduce privileges by aiding developers in correct permission selection. The tool could run whenever an application is submitted to the directory, or it could be provided to developers as part of the development or packaging process. If unnecessary permissions are found, the developer could be prompted to remove them.

Our Google Chrome extension overprivilege detection tool is simple but sufficient to find some types of errors. As shown in Section 4.2.1, a JavaScript text search is sufficient to remove unnecessary browser manager permissions from 147 of the 1000 popularity-ranked extensions. Our text search has a small number of false positives; e.g., we found three extensions that only contain references to browser managers in remotely sourced scripts. However, a developer can disregard a warning if she feels it is incorrect. Our tool also detects simple redundant wildcard errors and asks the developer to remove the broad wildcard in favor of the more specific domain. Detecting the larger problem of overly broad domain requests is a challenging open problem for future research in JavaScript program analysis.

A similar Android tool could analyze applications to find all Android API calls, and from that deduce what permissions the applications need. The tool could ask the developer to discard permissions that are not required by any of the API calls. The tool cannot completely replace developers; developers must still edit their permission requirements if they want to include additional permissions for inter-application interactions. (Applications can choose to only accept messages from applications with certain permissions.) Unfortunately, incomplete documentation currently prevents this tool from being built; the documentation does not completely state which API calls require which permissions. Experimentally determining the permission-API relationship is an active area of future research.

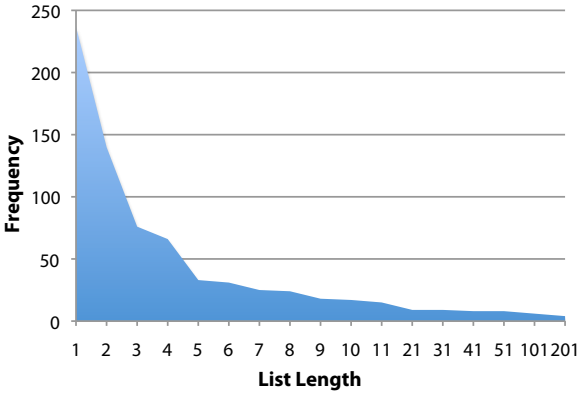

Figure 5: The number of content script specific domain lists with *at least* a given length. Note the non-linear x-axis.

4.3 Wildcards

Domain access in the Google Chrome extension system relies on wildcards. A developer can write `<all_urls>` or `*://*/*` and gain access to all domains, or she can define a list of specific domains. When it is not feasible for a developer to list all possible subdomains, she can use wildcards to capture multiple subdomains. However, a developer might choose to use a wildcard even though it includes more privileges than the application requires.

Compliance. To determine whether developers are willing to write specific domain lists when they can more easily request access to all domains, we evaluate the prevalence of specific domain lists in the 1000 popularity-ranked extensions. Of the 714 extensions that need access to web data, 428 use a specific domain list for either a content script or core extension. This is a surprising and positive finding: 60% of developers whose extensions need web access choose to opt in to domain restrictions for at least one component. However, 367 extensions also have at least one component that requests full domain access. (An extension with multiple content scripts might request full domain access for some scripts but place restrictions on others.)

Developer Effort. We suspect that developers will default to requesting all-domain access if the number of specific domains in the list grows too high. To examine this further, we consider the 237 content scripts that use specific domain lists. The lists are short: only 31 are longer than five. Figure 5 presents the distribution. This indicates that most developers either request a very small number of domains or opt to request full domain access, with few in-between. However, six developers wrote eight lists that are longer than fifty domains. These outliers result from developers internationalizing their extensions by repeating the same domains with different suffixes; wildcards cannot be used to represent suffixes because the domains may have different owners.

Noncompliance. Section 4.2 describes a manual analysis of fifty extensions. Five of those extensions are overprivileged due to improper wildcard use. Two of the developers choose to request all-domain access rather than write specific domain lists, two write specific domain lists but unnecessarily use wildcards for subdomains, and one incorrectly requests all-domain access alongside specific domains. In other words, 10% of the extensions with web access request excessive permissions because their developers are unable or unwilling to write sufficiently specific domain lists.

In summary, our findings are twofold. We show that 60% of extension developers write at least one specific domain list. This demonstrates that the option to write specific domain lists is a worthwhile part of a declarative permission system. On the other hand, 40% of developers whose extensions need web access do not write any specific domain lists. Furthermore, our manual analysis indicates that 10% of extensions with web access use wildcards improperly.

4.4 Permission Granularity

If a single permission protects a diverse set of API calls, then an application seeking to use only a subset of that functionality will be overprivileged. Separating a coarse permission into multiple permissions can improve the correlation between permissions and application requirements. On the other hand, excessively fine-grained permissions would burden developers with a large list of permissions required to perform simple actions.

4.4.1 Google Chrome Browser Managers

At the time of our study, Google Chrome extension permissions were at the granularity of a browser manager: one permission per entire browser manager. This posed a problem for the window manager, which includes some methods that provide indirect access to history via the location property of loaded windows. Using the window manager generated history warnings, regardless of whether the extension used any of the methods that provide access to the location property.

The fact that the window manager caused a history warning was confusing to users and developers. Consider this quote from the developer of *Neat Bookmarks*:

> Installing this extension will ask for permission to access your browsing history, which is totally useless, not used and not stored by the extension at all. Not really sure why 'History' is part of 'Bookmarks' in the Chrome browser.

The developer is so confused by the history warning that he or she believes it is caused by the extension's use of the bookmark manager, rather than the window manager.

Since the time of our study, the window manager has been changed so that certain methods do not require any permission. Consequently, developers can access some non-history-related functionality without acquiring a permission that shows users the history warning.

4.4.2 Fine-Grained Android Permissions

We evaluate whether Android's fine-grained permissions are an improvement over a coarser-grained alternative.

Categories. Android permission categories are high-level functionality groups. Categories are comprised of multiple permissions, which developers must request individually. A coarse-grained permission system might simply have one permission per category, but Android subdivides each category into multiple finer-grained permissions. We find that no application (out of 956) requires all of the permissions in any category except `STORAGE`, a category with only one permission. This demonstrates that coarse-grained permissions at the category level would overprivilege all extensions.

Read/Write. Android controls access to data with separate read and write permissions. For example, access to contacts is governed by `READ_CONTACTS` and `WRITE_CONTACTS`. We find that 149 applications request one of the contacts permissions, but none requests both. 10 of 19 applications with calendar access request both read and write permissions. Text messages are controlled by three primary permissions; only 6 of the 53 applications with text message permissions request all three. These results demonstrate that separate read and write permissions reflect application requirements better than coalesced permissions would.

Location. Location is separated into "fine" and "coarse" permissions, referring to the precision of the location measurement. `ACCESS_FINE_LOCATION` governs GPS location, and cell location is controlled

by ACCESS_COARSE_LOCATION. 358 applications request at least one of the location permissions; 133 request only ACCESS_COARSE_LOCATION. This indicates that 37% of applications that need to know the user's location are satisfied with a "coarse" location metric, which benefits user privacy.

Future permission systems should consider adopting similar fine-grained permissions.

4.4.3 Coarse-Grained Android Permissions

Not all of Android's permissions are fine-grained. The INTERNET permission lets an application send HTTP(S) requests to all domains, load any web site into an embedded browser window ("WebView"), and connect to arbitrary destinations and ports. The granularity of the INTERNET permission is important because 86.6% of free and 65% of paid applications in our large-scale study use it.

We find that 27 of the 36 Android applications in our manual review (Section 4.2.2) have the INTERNET permission. Of those, 13 only use the Internet to make HTTP(S) requests to specific domains. These Android applications rely on backend servers for content, much like web applications. A fourteenth application additionally uses the INTERNET permission to support Google AdSense, which displays advertisements from a single domain in a WebView.

These results indicate that many applications would tolerate a limited Internet permission that only permits HTTP(S) or WebView access to a specific list of domains, similar to what Google Chrome offers extensions. This hypothetical limited permission would be sufficient for 52% of the 27 applications that use INTERNET.

5 Reducing User Prompts

Our study in Section 3 demonstrates that almost all extensions and applications trigger prompts for dangerous permissions during installation. The high rate of permission warnings makes it unlikely that even an alert, security-conscious user would pay special attention to an application with several dangerous privileges.

Possible solutions to this problem depend on the intended role of permission prompts. If permission prompts are only intended to inform the user and decrease platform liability, then perhaps their presentation and frequency do not matter. If a prompt is supposed to warn or alert the user, however, then increasing user attention will improve its efficacy. In order to preserve the significance of truly important warnings, one possibility is to de-emphasize or remove lesser warnings.

5.1 Google Chrome

Google Chrome currently presents all permissions equally. Critical extension privileges (e.g., including a plug-in) should always be prominently displayed as part of the installation process, but less significant permissions (e.g., bookmarks) could be omitted from the installation warning and simply listed on the download page.

Not all Internet access needs to be displayed to users. Web sites with private information (e.g., financial, commercial, and e-mail sites) use TLS to protect users from man-in-the-middle attacks. We assume that HTTP-only sites are not concerned about eavesdropping. If Google Chrome were to only show warnings for extensions with access to HTTPS sites, 148 of the 500 most popular extensions would no longer trigger web access warnings. 102 extensions would no longer prompt a warning at all, reducing the number of extensions with at least one warning from 91.4% to 71% of the 500 most popular extensions. Users would be at risk of man-in-the-middle attacks on HTTP-only sites, but they already are at risk of this on their networks.

5.2 Android

Android ranks permissions by threat level, and only Dangerous permissions are displayed to users. However, there is still great variance within Dangerous permissions. Dangerous permissions let an application perform actions that cost the user money (e.g., send text messages), pertain to private information (e.g., location, contacts, and the calendar), and eavesdrop on phone calls. On the other hand, Dangerous permissions also guard the ability to connect to paired Bluetooth devices, modify audio settings, and get the list of currently running applications. Users may not care about Dangerous permissions that cannot cause direct harm to the user or phone. De-emphasizing the less-threatening Dangerous permissions could reduce the number of user warnings.

WAKE_LOCK and WRITE_EXTERNAL_STORAGE are two of the most popular Dangerous permissions, and neither has a clear implication for users. The WAKE_LOCK permission lets an application perform actions that keep the phone awake without user interaction. Playing music, for example, requires this permission. Although the permission could be used to slowly drain the battery, it does not pose a serious privacy or security threat. 26% of the 956 applications have the WAKE_LOCK permission. The WRITE_EXTERNAL_STORAGE permission controls access to the SD card, which could be used to access other applications' files that are on the SD card. However, the user has no way of differentiating between legitimate and illegitimate access to the SD card. It seems reasonable for all applications to store data, and only the developer

knows whether to use internal or external storage. 35.7% of the 956 applications have this Dangerous permission.

INTERNET is the most popular permission. The higher prevalence of the INTERNET permission in free applications and past work [5] indicate that free applications commonly use the Internet to contact advertisers. Section 4.4.3 suggests enabling applications to request access to a specific list of web domains. Accordingly, the Android Market could display a less severe warning for applications with limited Internet access than for applications with the full INTERNET. The warning could further notify the user if a known advertising domain is included in the specific domain list.

6 Related Work

Google Chrome Extensions. When Barth et al. introduced the Google Chrome extension permission system, they conducted a motivating analysis of 25 Google Chrome extensions [2]. However, their sample set is too limited to be definitive. Google employees authored 9 of the 25 extensions, and the extension platform had only been public for a few weeks prior to their study. The results of our large-scale evaluation of Google Chrome extensions show that their small-scale study overestimated the prevalence of extension privileges. Guha et al. [8] performed a concurrent, short study of the permissions used by Google Chrome extensions, although they do not study the effect of popularity. We provide a significantly more detailed discussion of extension privileges.

Android Applications. Barrera et al. [1] analyze the permissions requested by 1,100 free Android applications. They primarily focus on the structure of the permission system; they group applications together using a neural network and look for patterns in permission group requests. They note that 62% of the applications collected in December 2009 use the INTERNET permission. Significantly more applications in our data set use the INTERNET permission, which is possibly due to changes in applications over time. We also provide data that can be used to evaluate two of their proposals for changes to Android permissions. First, they suggest that applications should be able to simultaneously request multiple permissions with wildcards (e.g., android.permission.SMS.*). Our Google Chrome survey shows that developers often use wildcards to request excessive privileges, and our Android study shows that the majority of applications do not need access to all permissions in a group. Next, they propose that the INTERNET permission should support specific domain lists. A manual review finds that 14 of 27 applications with the INTERNET permission would indeed be satisfied with access to a list of specific domains.

Researchers at SMobile present a survey of the permissions requested by 48,694 Android applications [18]. They do not state whether their sample set is composed of free applications, paid applications, or a combination. They report that 68% of the applications in their sample set request enough permissions to be considered "suspicious." We similarly find that applications have high privilege requests. They also report with alarm that 9 applications request access to the BRICK permission, which can be used to make a phone non-operational. However, this is a Signature permission; it is only available to a very small number of applications signed by the device manufacturer. We find that a surprising number of applications request Signature and SignatureOrSystem permissions, given that most applications are unable to actually use these permissions.

Kirin [6] is a tool that evaluates the security of an Android application. It compares the application's requested permissions to a set of permission rules. They propose several rules and test them against 311 applications. Their rules are specific enough to only flag a small number of the applications in our set, but we did not check to see whether the applications are malicious.

User Warnings. We consider whether installation warnings are of value to security-conscious users. Other researchers have examined the best way to visually display installation permissions to users [17] but not examined the frequency of prompts in install-time permission systems. Warning science literature indicates that frequent exposure to specific warnings, especially if the warnings do not lead to negative consequences, drastically reduce the warnings' effectiveness [11, 15]. Other researchers have shown that browser warnings for phishing sites and invalid SSL certificates are ignored by most users [4, 16]; it is possible that even infrequent permission installation warnings will be ignored.

LUA. Windows users can reduce application privileges by running Windows as a low-privileged user account (LUA). While in LUA mode, all applications have reduced privileges. When an application wants to perform a task that requires administrative privileges, Windows presents the user with a prompt for approval. Unlike the application permission model discussed in this paper, only two security modes are available (user or administrative). Furthermore, in practice, users run in administrative mode all the time, thereby granting the system's full privileges to applications [12].

7 Conclusion

This study contributes evidence in support of application permission systems. Our large-scale analysis of Google Chrome extensions and Android applications finds that real applications ask for significantly fewer than the maximum set of permissions. Only 14 of 1000 Google Chrome extensions use native code, which is the most dangerous privileges. Approximately 30% of extension developers restrict their extensions' web access to a small set of domains. All Android applications ask for less than half of the available set of 56 Dangerous permissions, and a majority request less than 4.

These findings indicate that permission systems with up-front permission declarations have two advantages over the traditional user permission model: the impact of a potential third-party vulnerability is greatly reduced when compared to a full-privilege system, and a number of applications could be eligible for expedited review. These results can be extended to time-of-use permission systems if the system requires developers to declare a set of maximum permissions.

However, our study shows that users are frequently presented with requests for dangerous permissions during application installation in install-time systems. As a consequence, installation security warnings may not be an effective malware prevention tool, even for alert users. Future work should identify which permission warnings are useful to users and consider alternate methods of presenting permissions to users.

References

[1] BARRERA, D., KAYACIK, H. G., VAN OORSCHOT, P. C., AND SOMAYAJI, A. A Methodology for Empirical Analysis of Permission-Based Security Models and its Application to Android. In *ACM CCS* (2010).

[2] BARTH, A., FELT, A. P., SAXENA, P., AND BOODMAN, A. Protecting Browsers from Extension Vulnerabilities. In *NDSS* (2010).

[3] CLULEY, G. Windows Mobile Terdial Trojan makes expensive phone calls. http://www.sophos.com/ blogs/gc/g/2010/04/10/windows-mobile-terdial-trojan-expensive-phone-calls/.

[4] EGELMAN, S., CRANOR, L. F., AND HONG, J. You've Been Warned: An Empirical Study of the Effectiveness of Web Browser Phishing Warnings. In *CHI* (2008).

[5] ENCK, W., GILBERT, P., CHUN, B., COX, L. P., JUNG, J., MCDANIEL, P., AND SHETH, A. N. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *OSDI* (2010).

[6] ENCK, W., ONGTANG, M., AND MCDANIEL, P. D. On Lightweight Mobile Phone Application Certification. In *ACM CCS* (2009).

[7] FELT, A. P. Issue 54006: Security: Extension history permission does not generate a warning. http://code.google.com/p/chromium/issues/ detail?id=54006, August 2010.

[8] GUHA, A., FREDRIKSON, M., LIVSHITS, B., AND SWAMY, N. Verified Security for Browser Extensions. In *IEEE Security and Privacy* (2011).

[9] IBRAHIM, S. Universal 1-Click Root App for Android Devices. http://androidspin.com/2010/08/10/ universal-1-click-root-app-for-android-devices/, August 2010.

[10] LIVERANI, R. S., AND FREEMAN, N. Abusing Firefox Extensions. Defcon17, July 2009.

[11] MAGAT, W., VISCUSI, W. K., AND HUBER, J. Consumer processing of hazard warning information. *Journal of Risk and Uncertainty* (1988).

[12] MOTIEE, S., HAWKEY, K., AND BEZNOSOV, K. Do Windows Users Follow the Principle of Least Privilege? Investigating User Account Control Practices. In *SOUPS* (2010).

[13] MOZILLA ADD-ONS BLOG. The Add-on Review Process and You. http://blog.mozilla.com/addons/2010/02/ 15/the-add-on-review-process-and-you.

[14] SERIOT, N. iPhone Privacy. *Black Hat DC* (2010).

[15] STEWART, D. W., AND MARTIN, I. M. Intended and Unintended Consequences of Warning Messages: A Review and Synthesis of Empirical Research. *Journal of Public Policy Marketing 13*, 1 (1994).

[16] SUNSHINE, J., EGELMAN, S., ALMUHIMEDI, H., ATRI, N., AND CRANOR, L. F. Crying Wolf: An Empirical Study of SSL Warning Effectiveness. In *USENIX Security Symposium* (2009).

[17] TAM, J., REEDER, R. W., AND SCHECHTER, S. I'm Allowing What? Disclosing the authority applications demand of users as a condition of installation. Tech. Rep. MSR-TR-2010-54, Microsoft Research, 2010.

[18] VENNON, T., AND STROOP, D. Threat Analysis of the Android Market. Tech. rep., SMobile Systems, 2010.

[19] WILLISON, S. Understanding the Greasemonkey vulnerability. http://simonwillison.net/2005/Jul/ 20/vulnerability/.

A Manual Review

Android Applications. Jesus Hates Zombies, Compass, Aquarium Live Wallpaper, Movies, Mobile Banking, Calorie Counter by FatSecret, Daily Horoscope, Pandora Radio, The Weather Channel, Advanced Task Killer, Google Sky Map, Barcode Scanner, Facebook for Android, NFL Mobile Aquarium, Live Wallpaper, weird facts, Google Maps Screen Crack, screen krack, twidroyd for twitter, touch to talk, open home, pageonce pro, personal finance, baby esp, gentle alarm, picsay pro, beautiful widgets, iQuran Pro, Grocery King, Touitor Premium, MLB.com at Bat 2010, myBackupPro, London Journey, BeyondPod Unlock Key, Text to Speech Extended, DocumentsToGo Full

Google Chrome Extensions. Orkut Chrome Extension, Google Similar Pages beta (by Google), Proxy Switchy!, AutoPager Chrome, Send using Gmail (no button), Blog this! (by Google), Fbsof, Diigo Web Highlighter and Bookmark, Woot!, Pendule, Inline Search & Look Up, YouTube Middle-Click Extension, Send to Google Docs, [Non-English Title], PBTweet+, Search Center, Yahoo Mail Widget for Google Chrome, Google Reader Compact, Chromed Movilnet, Ubuntu light-themes scrollbars, Persian Jalali Calender, Intersect, deviantART Message Notifier, Expand, Castle Age Autoplayer Alpha Patched, Patr Pats Flickr App, Better HN, Mark the visited links, Chrome Realtime Search, Gtalk, SpeedyLinks, Slick RSS, Yahoo Avatar, Demotivation.ru ads remover, [Non-English Title], PPTSearch Edu Sites, Page2RSS, Good Habits, VeryDou, Wikidot Extender, Close Left, iBood, Facebook Colored, eBay Espana (eBay.es) Busqueda avanzada, Keep Last Two Tabs, Google Transliteration Service, Ohio State University Library Proxy Extension, Add to Google Calendar, Rocky, Short Youtube