The Effectiveness of Install-Time Permission Systems for Third-Party Applications

Adrienne Porter Felt
Kate Greenwood
David Wagner

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2010-143
http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-143.html

December 3, 2010

The Effectiveness of Install-Time Permission Systems for Third-Party Applications

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

ABSTRACT

In many modern development platforms, application permissions control third-party access to sensitive parts of the API (e.g., the camera or microphone). We study *install-time permissions*, which the user grants to applications during installation; different applications can receive different install-time permissions. Install-time permissions offer several advantages over traditional user-based permissions, which assign the user's full privileges to all applications. However, 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 install-time permission.

We perform case studies on two systems with install-time permissions for third-party applications, the Google Chrome extension platform 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 install-time permissions are effective at protecting users. Our results indicate that install-time application permissions have a strong positive impact on system security, but a number of changes could further improve their utility.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection; D.2 [**Software**]: Software Engineering

General Terms

Security

Keywords

Permissions, smartphones, browser extensions, Android

*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.

1. INTRODUCTION

Modern 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 [4, 15], and third-party code can introduce vulnerabilities because the authors of third-party applications usually are not security experts [11, 20].

In order to protect users from the threats associated with third-party code, many major platforms use application permissions to control access to security- and privacy-relevant parts of the platform's API. Users decide whether to grant individual applications permissions for sensitive resources like the camera. Per-application permissions differ from the traditional user model, which associates permissions with users; in practice, user-based permissions lead to all applications running with the system's full permissions [12].

We consider systems that implement the application permission model with *install-time permissions*. Systems with install-time permissions ask developers to declare their applications' required permissions so that users can grant the requested permissions during installation. Applications are permanently limited to their declared permissions. We measure the security role of install-time permissions in two case studies, the Google Chrome extension system and Android application platform.

We focus on three possible advantages of an install-time permission system over the traditional user model:

- **User Consent:** Security-conscious users may be hesitant to install applications that ask for dangerous permissions without justification. If so, this will decrease the likelihood that the user installs malware.
- **Defense in Depth:** The impact of a vulnerability in a third-party application will be limited to the vulnerable application's privileges (i.e., a subset of the API).
- **Review Triaging:** It facilitates central review of third-party applications because security reviewers can ignore low-privilege applications and focus their attention on applications with dangerous permissions. This may decrease the average review time.

These advantages depend on the assumption that most applications request fewer than the maximum set of permissions. We evaluate this assumption by performing a large-scale study of permission usage in Google Chrome extensions and Android applications.

Our measurement study quantifies the permission usage of 1000 Google Chrome extensions and 956 Android applications. We provide detailed data on the permission requirements of applications in the wild. We consider which and how many permissions developers ask for. From this data, we assess whether the potential benefits of install-time permissions are being realized in these platforms.

We find that almost all applications ask for fewer than maximum permissions: only 24 of 1000 extensions request the most dangerous privileges, and all Android applications ask for less than half of the available dangerous permissions. This significantly decreases the impact of an application vulnerability and simplifies review. However, users are presented with at least one dangerous permission request during the installation of almost every extension and application. We hypothesize that this high frequency of permission requests desensitizes users to dangerous permissions, although we do not perform user studies.

Even though our primary study indicates that developers use less than maximum permissions, dangerous permission usage is widespread. To address this, we examine the influence of developer error, wildcard permissions, and permission granularity on permission usage. We find that more than 10% of applications erroneously include permissions, and we suggest error detection tools. Our results also show that many developers are willing to make use of fine-grained permissions, motivating a fine-grained permission design.

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 benefits of install-time permissions. Our results should guide the design of future permission systems, and we provide concrete suggestions to help Google Chrome and Android approach their full potential.

2. BACKGROUND

We define install-time application permissions and contrast them with other types of permissions. The Google Chrome extension system and Android application platform feature install-time permissions.

2.1 Permission Systems

Traditional operating systems associate permissions with users. When a user installs an application, the application receives all of the user's privileges. All applications must be trusted equally, and an application vulnerability puts all of the user's data at risk. Users can choose to use a low-privilege account, but most operate as a full-privilege administrator for daily tasks [12].

Several modern systems associate permissions with applications rather than users. Users can decide whether to grant specific permissions to individual applications. We focus on *install-time* permissions, which are presented to the user as part of the installation process. Install-time permissions must be declared by the developer (e.g., in a manifest file). Once granted, an application is restricted to its install-time permissions. Examples of install-time permission systems include the Android OS, Google Chrome extension system, and Facebook Platform.

Alternately, systems like Apple iOS and BlackBerry use *time-of-use* permissions. They prompt the user when a running application attempts to access restricted functionality.

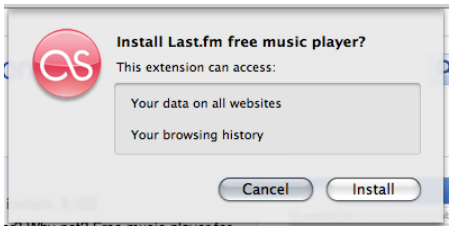

Figure 1: Google Chrome extension installation.

Time-of-use permissions do not need to be declared. However, a time-of-use permission system could hypothetically require developers to declare a maximum set of possible permissions (i.e., user prompts would only appear for previously declared permissions). This would make a time-of-use permission system eligible for the benefits of review and defense in depth, similar to install-time permission systems.

2.2 Google Chrome Extensions

Browser extension platforms allow third-party applications to run as part of the browser environment. *Extensions* change the user's browsing experience by editing web sites and extending browser behavior. All extensions are free to install from the official Google Chrome extension gallery. We explore the benefits of install-time permissions in the context of Google Chrome extensions:

- *User Consent.* Malicious extensions for other browsers have been caught spying on users and installing persistent malware [21, 3]. A user can cancel an extension's installation if she is not comfortable with its permissions. Figure 1 shows an installation warning.

- *Defense in Depth.* When a user navigates to a malicious web site, the malicious web site might try to trick a vulnerable extension into yielding a privileged reference or performing a privileged action [11, 20]. A buggy extension might leak privileges that web sites should not receive. Limiting the scope of extension vulnerabilities was a primary motivating factor behind the extension system's design [2].

- *Review Triaging.* Google formally reviews extensions with the most dangerous permissions. Mozilla editors also review all Firefox extensions prior to being publicly listed in the Mozilla add-on directory. The length of the non-triaged Mozilla review process is often a concern for developers [13].

Permissions. Google Chrome extensions can have three types of components: core extensions, content scripts, and plug-ins. Core extensions are the main, persistent JavaScript 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. Plug-ins are native executables. Three types of privileges are available to extensions:

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 (e.g., notifications) also exist, but they do not prompt a warning.

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 `XMLHttpRequest`s (XHRs) and inject additional content scripts into the domains it has permissions for. Content script and core extension domain permissions are listed separately.

All-domain access is the most broad web permission. If either the content scripts or the core extension have all-domain access, the browser warning states that the extension "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. Additionally, a domain list for a content script can include the `file` protocol. This enables read-only access to files when they are loaded in the browser by the user. When this happens, the content script can read arbitrary other local documents (except in the developer release of the Google Chrome browser). Requesting access to the `file` protocol generates the all-domain warning, and these extensions are reviewed.

2.3 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. In the context of Android, the benefits of install-time permissions are:

- *User Consent.* Smartphones are increasingly becoming targets of Trojans [16, 10]. Android's permission system informs users about an application's potential for harm. (See Figure 2.)
- *Defense in Depth.* Applications that handle content from untrusted sources can also be vulnerable to external attacks [9].
- *Review Triaging.* Android applications do not need to be reviewed prior to inclusion in the Android Market, but other phone vendors like RIM and Apple maintain official review processes. External researchers can also use permissions to guide application analyses [6].

Although Android applications can communicate, we assume they do not collude.

Permissions. 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. There are 134 permissions in API 8 (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.

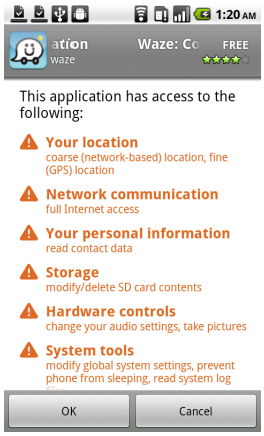

Figure 2: Android application installation.

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 connecting to a Bluetooth device, opening a network socket, recording audio, and using the camera.

Signature. Especially sensitive operations are protected with Signature or SignatureOrSystem permissions. Android permissions are only granted to applications that have been signed with the device manufacturer's certificate or installed into the `/system/app` folder. Advanced users who have rooted their phones [8] can manually install applications into this folder, but the official Market installation process will ignore the permissions. Market applications are only eligible for Signature or SignatureOrSystem permissions if they are updates to applications that were pre-installed by the device manufacturer (e.g., Google Maps and Twitter are pre-installed on some phones). Examples include the ability to call emergency numbers without user confirmation and brick the phone (i.e., render it inoperable).

Warnings for Dangerous permissions are grouped into functionality categories. E.g., 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 permissions are not displayed to users at all. Applications can define their own extra permissions, but we only consider permissions defined by the Android OS.

3. PERMISSION PREVALENCE

We examine the frequency of permission requests in Google Chrome extensions and Android applications. Based on these results, we evaluate how often users face permission warnings, calculate the potential impact of a typical third-party vulnerability, and estimate how many applications would be eligible for a hypothetical review process that triages applications according to their permissions.

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

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

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 %
1+ privacy-related manager	74.8 %	43.4 %

Figure 3: 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.

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 that are submitted to the directory.

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]. Figure 3 provides an overview.

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

Privacy-related 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. Access to bookmarks and geolocation is 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. 2% of the 500 request access to the `file` protocol, 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 non-privacy-related managers.

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 the sorting algorithm weakens. Figure 4 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.

[2]We discovered that current versions of Google Chrome sometimes fail to generate a warning for history access. We reported the bug, and it will be fixed for new versions [7]. Our analysis assumes that all requests for history access correctly generate a warning. The bug affects 3 of the 500 most popular extensions and 2 of the 500 less popular extensions.

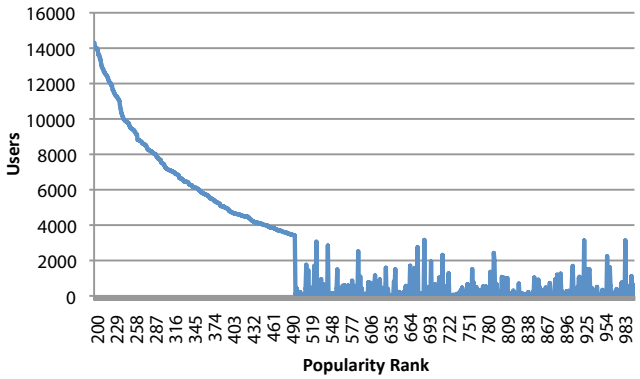

Figure 4: Users per extension. We omit the first 200 so that the bottom 500 can be distinguished; the most popular extension has 1.3M users.

Figure 3 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. All of the differences are significant at a 99% confidence level. We hypothesize that this is because less popular extensions offer less functionality.

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 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 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 or `file` permissions, 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. The permission system prevents half of popular extensions from having unnecessary web privileges.

Review Triaging. Of the 1000 extensions in our study, only 2.4% require review under current Google Chrome review triaging procedures. Alternate triaging procedures are possible. If the review process were modified to include ex-

tensions with all-domain access, our study of unpopular extensions indicates that only 26.4% of extensions would need review. Reviewers could ignore 28.4% of submitted extensions, based on the number of less-popular extensions with no security-relevant permissions. These results suggest that the burden of the Firefox extension review process would be significantly reduced if Mozilla were to adopt a similar permission system.

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; we observe no differences between popular and recently added free applications, so we present them together.

3.2.1 Dangerous Permissions

We are primarily concerned with the prevalence of Dangerous permissions. Dangerous permissions 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 warning.

Android permissions are grouped into functionality categories, and Figure 5(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 warning, so Figure 5(a) also indicates how often users see each type of warning.

A small number of permissions are requested very frequently. Figure 5(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. Enck et al. found that some free applications leak personal data [6]; this may explain the difference in `ACCESS_COARSE_LOCATION` requests.

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, we find that 99%, 94%, and 78% of the 306, 149, and 14 respective applications that request `ACCESS_COARSE_LOCATION`, `READ_CONTACTS`, and `READ_CALENDAR` have the `INTERNET` permission.

Although many applications ask for at least one Dangerous permission, the total number of permission requests is typically low. Even the most highly privileged application in our set asks for less than half of the available 56 Dangerous permissions. Figure 6 shows the distribution of Dangerous permission requests. Paid applications use an average of 3.99 Dangerous permissions (5.83 average total permissions); free applications use an average of 3.46 Dangerous permissions (4.71 total permissions).

[3]The applications were collected in October 2010.

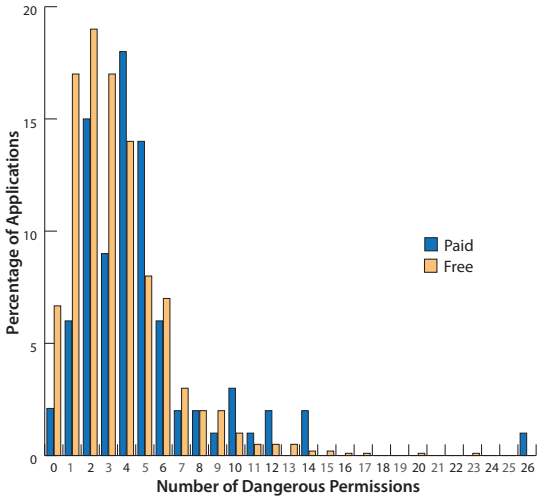

Figure 6: Dangerous permissions per application.

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). Applications installed through the Android Market can only obtain Signature or System permissions if they are upgrades to the pre-installed applications. It is therefore pointless for a typical application to request these permissions because the permission requests will be ignored.

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 user device. Requests for unobtainable permissions may be leftover from testing or caused by developer error.

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 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.

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

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 %

(a) Applications with at least one Dangerous permission in each category.

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 %

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

Figure 5: Survey of 856 free and 100 paid Android applications. We indicate significant difference between the free and paid applications at 99% (**) and 95% (*) confidence levels.

Defense in Depth. Given the prevalence of Dangerous permissions, an application vulnerability is likely to yield at least one Dangerous permission. However, no application requests more than half of the available Dangerous permissions, and the majority ask for less than seven. Only 10% of applications, if vulnerable, could yield access to functionality that costs the user money. This is a significant improvement over the traditional OS full-privilege approach.

Review Triaging. Reviewing only applications that request a Dangerous permission (and exempting the rest) would not reduce reviewer workload much. 18% of paid applications would be exempt from review, but only 7% of free applications lack a Dangerous permission. Excluding `INTERNET` permissions could reduce the review load to 78% of paid and 79% of free applications. An application with only the `INTERNET` permission cannot leak sensitive personal information without a second Dangerous permission (e.g., it would need `READ_CONTACTS`).

4. REDUCING PRIVILEGES

Our survey of extensions and applications indicates that a large number of applications ask for dangerous permissions. We investigate factors that influence permission requirements and present corresponding suggestions for reducing the frequency of highly privileged applications. The principle of *least privilege* states that an application should run with as few privileges as possible [14].

4.1 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.

4.1.1 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 extensions and found no evidence of developers doing this.

Domains. We also review fifty randomly selected extensions for excessive domain access. 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 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. "PBTweet+" requests web access for a nonexistent core extension. "Send using Gmail (no button)" requests both specific domain access and all-domain access for the same component; the all-domain access is unnecessary. "iBood" and "Castle Age Autoplayer" request access to all domains, although they only interact with iBOOD and Facebook, respectively. "Add to Google Calendar" and "Orkut Chrome Extension" each ask for all subdomains of a site, but they only need access to specific pages; "Add to Google Calendar" also requests permissions for a website that it never accesses. "Pendule" unnecessarily runs a content script on all pages, even though its core extension determines when to run a script on a page.

Developers sometimes request access to all and specific domains in the same list, as "Send using Gmail (no button)" demonstrates. We find that 27 of the 1000 popularity-ranked extensions 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`.

4.1.2 Errors in Android Applications

We manually review the top free and top paid application from eighteen Android Market categories. 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.

4.1.3 *Tools for Error Reduction*

Development platforms could provide tools that detect and warn about unnecessary permissions. The tool could run whenever a 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 would be prompted to remove them.

As shown in Section 4.1.1, a simple JavaScript text search is sufficient to remove unnecessary browser manager permissions from 147 of the 1000 popularity-ranked extensions. A similar tool could be built for developer use. A text search may have 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. Detecting overly broad domain requests is a challenging open problem for future research, but redundant wildcard errors are detectable. If a developer includes a wildcard alongside a more specific domain, the tool could ask the developer to remove the broad wildcard in favor of the more specific domain. The developer could then decide whether the wildcard is really necessary.

An Android tool could analyze applications to find all references to Android API calls, and from that deduce what permissions are necessary for the implementation of the applications. 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 other applications with certain permissions.) Unfortunately, incomplete documentation prevents this tool from being built at this time; exactly which API calls require which permissions is currently undocumented.

4.2 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. Writing a list of specific domains may also require the use of wildcards to capture multiple subdomains; it may not be feasible for a developer to list all possible subdomains. A developer might choose to use a wildcard even though it includes more privileges than the application requires. Wildcards are a high-privilege default because developers can avoid specifying exact permissions.

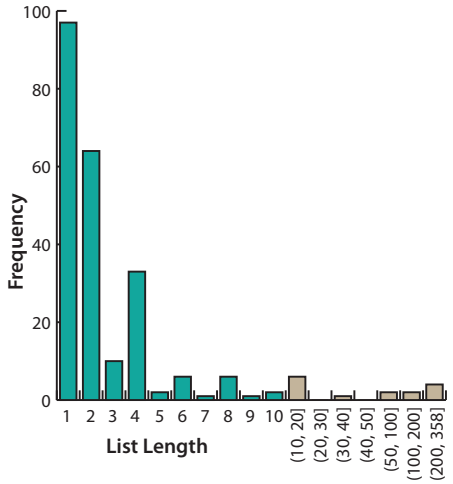

Figure 7: The lengths of specific domain lists, as written for content scripts.

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 at least one 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 7 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 are the result of developers internationalizing their extensions by repeating the same domains with different international suffixes; wildcards cannot be used to represent suffixes.

Noncompliance. Section 4.1 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, 12% 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 the 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 12% of extensions with web access use wildcards improperly.

4.3 Permission Granularity

If a single permission is applied to 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.

4.3.1 Google Chrome Browser Managers

In the Google Chrome extension system, permissions are at the granularity of a browser manager: there is one permission per entire browser manager. This poses a problem for the window manager, which provides indirect access to history via the location property of loaded windows along with other functionality. Using the window manager generates history warnings regardless of whether the developer accesses the location property of windows.

The fact that the window manager causes a history warning is confusing to both users and developers. Consider this disclaimer placed in the description for *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.

Separating history from the window manager would let extensions use the window manager's other functionality without becoming overprivileged. Accessing the location property of windows could require both the window and history permissions. Currently, 588 of 1000 extensions generate history warnings because of the window manager; in comparison, the history manager is only requested by 5 extensions. Given this, we surmise that separating history and window management could potentially remove a significant number of security warnings.

4.3.2 Fine-Grained Android Permissions

In some cases, Android separates functionality into fine-grained permissions. We evaluate whether these fine-grained permissions are an improvement over a hypothetical coarse-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 how 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.

In conclusion, Android facilitates least privilege with its fine-grained permissions. Future permission systems should consider adopting similar fine-grained permissions.

4.3.3 Android Internet Access

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 4.1.2) 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 benefit from 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 WARNINGS

Our study in Section 3 demonstrates that almost all extensions and applications request dangerous permissions and trigger warnings when installed. 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. In order to preserve the significance of truly important warnings, one possibility is to de-emphasize or remove less important 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., access to 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.

5.2 Android

Android has a permission threat hierarchy, 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 (i.e., send text messages and place phone calls), 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 one has a clear security or privacy 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, and the user has no way of knowing why an application requires SD card access. 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 [6] indicate that free applications commonly use the Internet to contact advertisers. Section 4.3.3 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 an 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.

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 `INTERNET` permission would indeed be satisfied with access to a list of specific domains.

Enck et al. [6] apply taint tracking techniques to a random sample of 30 free applications to determine whether free applications are using the `INTERNET` permission to leak user information. They find that 20 send user information to content or advertisement servers. In our study, we do not differentiate between legitimate and illegitimate uses of privileges. However, we find that significantly more free than paid applications request Internet access and location data. This is possibly indicative of widespread leakage of information to advertisers in free applications.

Researchers at SMobile present a survey of the permissions requested by 48,694 Android applications [19]. 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.

User Warnings. We consider whether installation warnings are of value to hypothetical security-conscious users. Other researchers have examined the best way to visually display installation permissions to users [18], but they do not consider the effect of the frequency of warnings. Future work needs to address this issue. Other researchers have shown that browser warnings for phishing sites and invalid SSL certificates are ignored by most users ([5, 17]); it is possible that even infrequent permission installation warnings will be ignored.

7. CONCLUSION

This study contributes evidence in support of install-time 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 24 of 1000 Google Chrome extensions use native code or request access to the `file` protocol, which are 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 the majority request less than 3.

These findings indicate that install-time permissions 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 significant number of applications could be eligible for an expedited review process. These results can be extended to time-of-use permission systems if the time-of-use 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. As a consequence, installation security warnings may not be an effective malware prevention tool, even for alert users. Future work should examine the influence of warning frequency on security and identify which permission warnings are useful to users.

8. REFERENCES

[1] D. Barrera, H. G. Kayacik, P. C. van Oorschot, and A. Somayaji. A Methodology for Empirical Analysis of Permission-Based Security Models and its Application to Android. In *ACM CCS*, 2010.

[2] A. Barth, A. P. Felt, P. Saxena, and A. Boodman. Protecting Browsers from Extension Vulnerabilities. In *NDSS*, 2010.

[3] D. Caraig. Firefox Add-On Spies on Google Search Results. http://blog.trendmicro.com/firfox-addo-spies-on-google-search-results.

[4] G. Cluley. 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/.

[5] S. Egelman, L. F. Cranor, and J. Hong. You've Been Warned: An Empirical Study of the Effectiveness of Web Browser Phishing Warnings. In *CHI*, 2008.

[6] W. Enck, P. Gilbert, B. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *OSDI*, 2010.

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

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

[9] D. Lenoe. Security Advisory for Adobe Flash Player. http://blogs.adobe.com/psirt/2010/09/security-advisory-for-adobe-flash-player-apsa10-03.html.

[10] J. Leopando. TrendLabs Malware Blog: New Symbian Malware on the Scene. http://blog.trendmicro.com/new-symbian-malware-on-the-scene, June 2010.

[11] R. S. Liverani and N. Freeman. Abusing Firefox Extensions. Defcon17, July 2009.

[12] S. Motiee, K. Hawkey, and K. Beznosov. 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] J. H. Saltzer. Protection and the Control of Information Sharing in Multics. In *CACM*, volume 17, 1974.

[15] N. Seriot. iPhone Privacy. *Black Hat DC*, 2010.

[16] J. Shah. McAfee Labs Blog: Windows Mobile trojan sends unauthorized information and leaves devices vulnerable. http://www.avertlabs.com/research/blog/index.php/2008/02/26/windows-mobile-trojan-sends-unauthorized-information-and-leaves-device-vulnerable, February 2008.

[17] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, and L. F. Cranor. Crying Wolf: An Empirical Study of SSL Warning Effectiveness. In *USENIX Security Symposium*, 2009.

[18] J. Tam, R. W. Reeder, and S. Schechter. I'm Allowing What? Disclosing the authority applications demand of users as a condition of installation. Technical Report MSR-TR-2010-54, Microsoft Research, 2010.

[19] T. Vennon and D. Stroop. Threat Analysis of the Android Market. Technical report, SMobile Systems, 2010.

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

[21] C. Wuest and E. Florio. Firefox and Malware: When Browsers Attack. http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/firefox_and_malware.pdf, 2009.

APPENDIX

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