2015

SPE: Security and Privacy Enhancement Framework for Mobile Devices

Brian Krupp
Baldwin Wallace University, bkrupp@bw.edu

Nigamanth Sridhar
Cleveland State University, n.sridhar1@csuohio.edu

Wenbing Zhao
Cleveland State University, w.zhao1@csuohio.edu

Original Citation

B. Krupp, N. Sridhar and W. Zhao, "SPE: Security and Privacy Enhancement Framework for Mobile Devices," Dependable and Secure Computing, IEEE Transactions on, 2015.

SPE: Security and Privacy Enhancement Framework for Mobile Devices

Brian Krupp, *Member, IEEE,* Nigamanth Sridhar, *Member, IEEE,* Wenbing Zhao, *Senior Member, IEEE*

Abstract—In this paper, we present a security and privacy enhancement (SPE) framework for unmodified mobile operating systems. SPE introduces a new layer between the application and the operating system and does not require a device be jailbroken or utilize a custom operating system. We utilize an existing ontology designed for enforcing security and privacy policies on mobile devices to build a policy that is customizable. Based on this policy, SPE provides enhancements to native controls that currently exist on the platform for privacy and security sensitive components. SPE allows access to these components in a way that allows the framework to ensure the application is truthful in its declared intent and ensure that the user's policy is enforced. In our evaluation we verify the correctness of the framework and the computing impact on the device. Additionally, we discovered security and privacy issues in several open source applications by utilizing the SPE Framework. From our findings, if SPE is adopted by mobile operating systems producers, it would provide consumers and businesses the additional privacy and security controls they demand and allow users to be more aware of security and privacy issues with applications on their devices.

Index Terms—Mobile Security, Mobile Privacy, Sensing, Encryption, iOS, Android

1 Introduction

Mobile computing devices are quickly becoming the platform of choice for consumers and businesses. Given that mobile devices started to outsell PCs in 2011 [1] and mobile applications are freely available in marketplaces, consumers are more likely to focus on mobile devices as their primary personal computing platforms. Additionally, users perform many of the same tasks that were performed previously with traditional computers on their mobile devices.

Most devices are equipped with numerous sensors such as cameras, microphones, GPS, accelerometers, and gyroscopes where users can share data about their environment or habits quickly, but also unknowingly. Here are a few examples of such unintentional sharing: (a) *Facebook* leaked the phone number from a mobile device before the user logged into the application [2]; (b) *Angry Birds* collected user data, which was found to be used by the NSA to profile users [3]; (c) out of 25,976 Android applications, 969 applications leaked location data and 347 recorded audio without the user's permission [4] (d) *Path* was found to geotag photos even after a user disabled location services [5], and sent user's privacy data unknowingly by uploading a user's entire address book [6]. Even when a user permits an application to access data on the device, the user is not aware of what *else* the data is being used for, how often it is being accessed, and with whom it is being shared — there is no way to confirm that the application is truthful in how it states the information will be used.

Along with privacy concerns there are security concerns as well. Between 2011 and 2012 malware family count rose from 6 to 67 [1] and between 2010 to 2011 malware samples alone rose from 11,138 to 28,472, a 155% increase [7]. One misconception is that malware is typically found only on jailbroken devices or third party application stores, however in September and October of 2011, 322 samples of malware with zero-day vulnerabilities were found in *Google Play* [8].

The rapid growth in the mobile device ecosystem demands viable solutions security and privacy concerns. Even though mobile devices are becoming more powerful, there still exist constraints on computing power, memory capacity, and a virtual endless supply of energy that traditional computing platforms offer today. These constraints limit mobile devices from performing computationally expensive operations such as pattern-based intrusion detection or fuzzy checking of privacy leakage. Even if computing power on mobile devices were to increase, the effect on the device's battery would be unacceptable for a user. Additionally, the user experience may be affected if more computationally expensive operations are being executed while the user is interacting with the device.

Recent research in this area has introduced novel methods for providing additional security and privacy controls. However most of these methods require a modification to the operating system or the device to be jailbroken (we provide a survey in Section 6). In this paper we propose the Security and Privacy Enhancement (SPE) framework for mobile devices. The SPE Framework provides additional protections without requiring a modification to the OS or the device to be jailbroken, a problem that has not been solved in other research. While SPE does not require a device to be jailbroken or operating system to be modified, it does require modification to the application. However, the required modification to the application is completely machine-automated, where the developer does not have to do any work aside from passing the source code through the injection framework. The details of this process are

- *B. Krupp is with the Computer Science Department, Baldwin Wallace University. E-mail: bkrupp@bw.edu*
- *N. Sridhar and W. Zhao are are with the Department of Electrical Engineering and Computer Science at Cleveland State University. E-mail: n.sridhar1@csuohio.edu, w.zhao1@csuohio.edu*

described in more detail in Section 4.3. We believe this is a more sustainable approach and has several advantages over previous approaches that required an operating system to be modified or jailbroken:

- When an OS is updated, a consumer does not have to wait for other solutions to be incorporated into the OS since SPE is integrated within the application. The framework would only need to be updated when significant API changes are made.
- Users would not need to jailbreak their device in order to adopt SPE. Additionally, with each minor release of the OS a user would not have to reinstall a custom OS to have enhanced security and privacy controls; they would be able to install the latest OS version when it becomes available.
- SPE can be utilized on a stock device running a stock OS. This approach allows maximum penetration into the user base as users do not need a specific OS or device. In contrast, other solutions that have focused on modifying the OS such as Android can only be applied to devices that run pure Android such as Nexus devices.
- Since SPE intercepts calls to security and privacy sensitive resources, it can guarantee that a consumer's policy is enforced. Unlike other solutions, the framework does not attempt to recognize patterns where false positives and false negatives can be returned.
- A consumer will not lose the inherent trust with an OS or void their warranty by utilizing a modified OS or jailbroken device.

In this paper, we describe the following contributions of the SPE framework:

- SPE provides a verbose fine-grained policy model to allow users to define additional security and privacy controls on a mobile device. These policies include constraints on precision as well as temporal and spatial properties of sensing data, and more. These policies are derived from a rigorous ontology [9], which we briefly describe in Section 2.
- SPE implements a novel intent-based[1] validation engine; each application must specify its intent in order to obtain access to a protected resource. If an application wishes to share any data it is permitted to access, it must specify an additional "chained intent" stating such a purpose. This allows the framework to understand how the application intends to access data and what it intends to do with the data.
- SPE allows the user to understand an application's data needs, and to check if an application is truthful. To enforce this, SPE will verify that the application does not perform an action that is not explicitly permitted. SPE also helps in detecting *application poisoning* where a trusted application's execution is modified by configuration data.
- SPE will be released as an open source framework for iOS, an operating system that cannot be modified for additional security and privacy controls. Included in this release is a client-side policy application to allow consumers to manage their policies as well as a point-and-click *conversion assistant* to transform an application to utilize the SPE Framework.

1. Intents in SPE differ from Intents in Android. The distinction is described in Section 3.1

The research presented here is an expansion of our initial research in this domain reported in [10]. Since that beginning, we have defined a detailed ontology for describing security and privacy policies [9]. We use this ontology to ensure that the coverage afforded by the framework is complete. We also discuss our complete implementation of the framework across all security and privacy components that we identified. Using this complete implementation, we provide evaluation results from testing several representative iOS applications with enhanced security and privacy policies where we were able to detect several security and privacy concerns. We also evaluate the correctness of the framework to ensure that it enforces a user's given policy, and does so correctly. Lastly we evaluate the performance impact of enforcing these enhanced security and privacy policies as mobile devices are more constrained in resources than traditional platforms.

The remainder of this paper is organized as follows: Sections 2 and 3 describe the policies and the SPE Framework. In Section 4, we describe an implementation of the SPE Framework that enforces policies on iOS applications that users define using a client application. Section 5 evaluates SPE across several applications with a focus on correctness, detecting security and privacy concerns, and computational overhead. After describing some relevant related research in Section 6, we conclude with a summary of our contributions, and some pointers to future work in Section 8.

2 Policy Model

The policy model that is utilized by the SPE Framework is derived from an ontology for enforcing security and privacy policies on mobile devices, presented in [9]. In this section we describe how the policy model was derived and implemented for SPE. From the ontology, a policy can exist primarily in two categories: privacy and security. Additionally, a policy can be classified as a general policy that contains spatial and temporal restrictions, or can exist as a chainable policy that creates a relationship between a privacy and security policy.

2.1 Privacy

Today privacy components are generally granted an "all or nothing" access characteristic. For example, if a user permits an application to access their photos, the application has access to all photos on the device, there is no way to restrict specific photos. Or if a user grants access to location data, the user does not have the ability to set the level of precision the application can retrieve location data or specify any kind of temporal or spatial restrictions. These are the kind of policies we aim to enhance with SPE.

2.1.1 Sensors

In this paper we focus our discussion primarily on the location sensor, however these policies can be applied to other sensors such as accelerometers or sound recorders. With

each sensor, we allow a *Data Accuracy Allowed* parameter to be defined. This allows the user to specify precise data, anonymous data, generalized data, no data, or bogus data from a sensor. Three of these classifications with exception to generalized data were presented in TISSA for Android [11]. *Generalized* allows for a predictable generalization of the sensor data being retrieved. For example with location, a *generalized* sensor accuracy policy would return a consistent latitude/longitude for a particular region where *anonymous* could be any point in particular region. This type of generalized data would be useful for applications such as weather or radio streaming, where anonymity is useful for participatory sensing applications. The five types of classifications are enforced within the framework.

Users can also define spatial and temporal permissions for sensors, such as: *Time Restrictions* and *Location Restrictions*. To remove the risk of an application detecting a user's exact location by specifying small regions and detecting when a user enters those regions, we add an additional permission of *Permit Regional Monitoring With Restrictions*. This policy also ensures if regional monitoring occurs within a designed spatial or temporal restriction, that regional reporting will not occur. These policy settings allow a user to have more fine grained control on how their location data is accessed. Access to location data is a growing privacy concern where one study found that out of 25,976 Android applications, 969 applications leaked location data [4]. This data can be used to examine a user's daily patterns and predict their location given a time interval.

Additionally, with sound and camera sensors the needs for these privacy controls is accentuated. For example, a user may be at a sensitive location such as a courtroom or a government classified location. Utilizing location restrictions, a policy could be created to ensure that audio is not recorded in these locations or that access to the camera is permitted.

2.1.2 Multimedia

The policies for multimedia focus on files stored on the device such as video and pictures. Access to multimedia is typically granted "all or nothing" with both read and write permissions. To provide more control, a user can specify a policy that separately defines read and write permissions. Multimedia can also contain metadata that describes where and when the element was captured. A user can specify additional policies to protect these attributes by specifying *Permit Read Photo Date Attribute* and *Permit Read Photo Location Attribute*. An example of where this data was misused was with the application Path where the application was found to geotag photos after a user disabled location services [5].

Spatial and temporal restrictions can be placed on multimedia as well. For example, a user can specify a date range of photos that an application can not access. Additionally, a user can specify restricted regions where if a photo was captured in the region, an application cannot access the photo. A user can also specify specific elements within multimedia that an application cannot access, so if a user wants to restrict specific photos, they are able to. These additional policy attributes allow a user to still provide an application access to their multimedia to take advantage of an application's features, while limiting what the application can access and what operations can be performed.

2.1.3 Contacts & Calendar

Like multimedia, contacts and calendar data is typically granted access to the entire library with read and write permissions. This is a concern as this data can be leaked: the social networking application *Path* was found to upload a user's entire address book without the user knowing [6]. Additionally, contacts and calendar data contain specific attributes that an application may not require access to such as first name, last name, address, phone numbers, and email addresses. To protect against misuse of this data, a user can specify *Permit Read* and *Permit Write*, *Restrict Address Access*, *Restrict Phone Number Access*, and *Restrict Email Address Access*. As an example of how these enhanced policy attributes could be used, if a user was using an application to make phone calls, the user could set *Permit Write* = false, *Restrict Address Access* = true, *Restrict Email Address* = true. As with other privacy data, calendar data allows for general read/write access. Additionally, with calendar data, a user can specify *Restricted Dates*. For both contacts and calendar, a user can restrict access to specific contacts or events.

2.2 Security

SPE focuses on application-centric security concerns, primarily focusing on enhancing the confidentiality of data that exists on the device. This includes detecting and preventing the leakage of data from being persisted on the device or sent on the network. Preventing data being leaked to the device may not seem important for sandboxed applications. However, if a device is lost or stolen, the consumer may want to ensure that their data is not stored in an unencrypted state. Additionally, a user may want to ensure that data is not automatically synchronized to a cloud service where that data could be further exposed. This type of control would have prevented the recent iCloud breach [12]. The security policies are defined under the categories of *Communication*, *Data Persistence*, and *Credentials*. SPE does not focus on OS level security such as address space layout randomization, application sandboxing, antivirus, and intrusion detection systems.

2.2.1 Communication

In the *Communication* category we focus on network communication where a user can define a *Domain Policy* that has the following attributes: *Permit Data*, *Require SSL*, *Permitted Credentials*, and *Credentials Require SSL*. These attributes specify if a domain can receive data, if the connection requires SSL, the credentials that can be transmitted, and if SSL is required to send credentials. The *Require SSL* option is used even though credentials may be sent over SSL, the session token in the form of a cookie may be sent over a non-SSL channel which can be read by an eavesdropper and used to impersonate the user in an attack known as "sidejacking" [13]. Each domain policy is specified for a particular domain but can also contain wildcards to cover multiple domains, for example: **.google.com*.

2.2.2 Data Persistence

With *Data Persistence* we focus on providing policies that specify what data can be persisted, if it should be encrypted, and if it can be synchronized to a cloud service. These

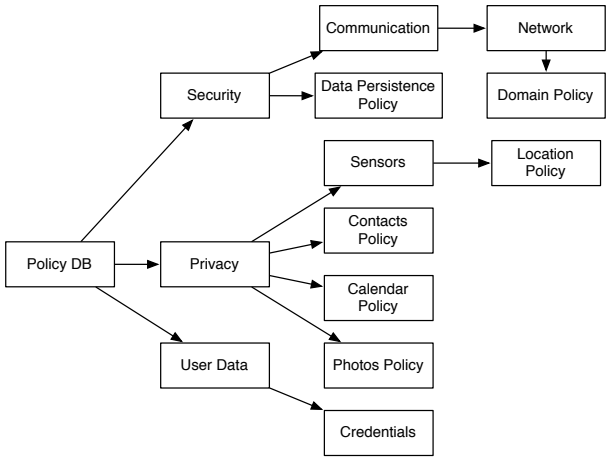

Fig. 1. Relationship of policy elements within policy model.

policies are specified with *Can Persist to Device*, *Data Encryption Level*, and *Can Persist to Cloud*. While both iOS and Android offer disk encryption [14], [15], not all applications may utilize encryption. The goal with this policy is to allow consumers the ability to specify how data should be encrypted. The *Data Encryption Level* is dependent on the platform where for iOS, it utilizes Data Protection classes specified in Apple's iOS security document [16]. This policy also mitigates the risk of data being retrieved from the device if stored unencrypted. A malicious user can utilize a filesystem browser, such as iExplorer, and open data stored on the device without unlocking it if left unprotected. By enforcing encryption, a user can ensure that if their device is ever lost or stolen that the data is secure.

2.2.3 Credentials

With credentials we specify a basic permission: *Access Allowed*. This permission allows the user to specify a global policy on whether a particular credential can be used by another application. This credential could be a username/-password combination to their email, banking account or any other service that would require authentication.

2.3 Chainable Policies

The ontology we utilize also specifies that policies can be *chained* to one another [9]. This is implemented within the SPE Framework to provide more fine grained control of when an application gains access to privacy elements, how and where that data can be transmitted or persisted to disk. For example, a user may specify that for an application, communication to a set of domains is permitted, however sensitive photos may not be sent to that domain unless they are encrypted. By attaching a domain policy to their photo policy they can achieve this more granular level of protection. Figure 1 shows the relationship between policies.

2.4 Policy Definition

The ontology for policies in SPE is comprehensive, and the complete ontology is described in [9]. In this paper we do not present every permutation. However to provide an example, a user can define a policy such as this: "Facebook can have my general location with exception between 7am and 5pm, but not my home or work location and can only share this data to domain facebook.com over SSL".

To define this policy, we perform the following steps:

- Define Location Policy
  - Set *Data Accuracy Allowed* = Generalized
  - Add *Time Restriction* = {700,1700}
  - Add *Location Restriction* = Home
  - Add *Location Restriction* = Work
- Define Domain Policy
  - Set *Permit Data* = TRUE
  - Set *Require SSL* = TRUE
  - Add *Permitted Domain* = "*facebook.com"
- *Location Policy* attach *Domain Policy*

A user would create this policy through the *SPE Policy* application. *SPE Policy* allows the user to define a policy and make it accessible to applications. *SPE Policy* is described in more detail in Section 4.4. With SPE, consumers have much greater flexibility in defining the security and privacy policies for an application.

3 SPE Framework

Here we discuss the design of the Security and Privacy Enhanced (SPE) framework and focus on the core components including `SPEIntents`, *class wrappers*, and the validation process. In the validation process we discuss how a `SPEIntent` is validated for both truthfulness and against a user defined policy.

3.1 SPEIntents

SPE uses a construct called `SPEIntent` to define an application's intent with a user-private resource. SPEIntents are structured analogous to the policy model. As an example of how a `SPEIntent` would be used, consider the following: An application intends to communicate with a particular set of URLs, the communication will be over a secure channel, and a set of credentials will be sent. In this example, the application would utilize a `SPENetworkIntent`. This type of `SPEIntent` has a tight correlation of policies defined in `SPENetworkDomainPolicy`. In this case, a `SPEIntent` has two primary purposes: (i) It defines what operations an application intends to perform so that at install time the user can clearly see what these operations are, and (ii) It detects the trustworthiness of an application by identifying any attempts to violate intents by performing operations that are not defined in the intent. The potential violation of the trust can be intentional or can be caused from malware, but in either case the user is able to realize the violation.

All SPE intents are subclasses of the main `SPEIntent` class where internal properties and validation methods are defined. `SPEIntent` also defines a delegate interface for the consuming application to implement so that the application can be notified of results of intent validation and handle it appropriately for their application.

Each subclass of `SPEIntent` contains more specific methods and properties based on the type of intent. For intents that have common methods and attributes, there is a

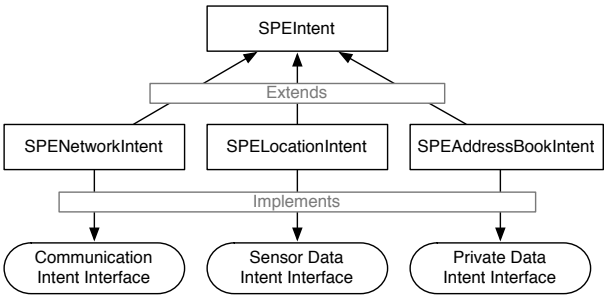

Fig. 2. `SPEIntent` relationships shown for intents subclassing from main `SPEIntent` class and common interfaces for similar intents

defined interface for the intent to implement. For example, all intents that relate to sensor data should implement methods that allow the application to specify if fuzzing, general, or anonymous data is acceptable and the accuracy level the application desires. Figure 2 shows this relationship between the `SPEIntent` class, its subclasses, and protocols that define common attributes across related intents.

When a `SPEIntent` is validated, either a protected object is returned or an operation on a protected object is performed. Since the application may have the same intent for multiple object retrievals or operations, an intent can be re-used across multiple protected objects. For example, in an application that performs consistent network communication to a particular set of domains, a consuming application can define the intent and reuse it for each new object that is used to communicate over the network.

3.1.1 Chained Intents

`SPEIntent`s can have a one-to-many relationship with protected objects. However, a protected object can only have one `SPEIntent`. A single object can be related to a number of intents: a `SPEIntent` can be chained to another `SPEIntent` depending on its type, much like how a policy can be attached to another policy. This chaining is accomplished using the Decorator design pattern. For example, an application could create a `SPELocationIntent` that can be used to retrieve the location of the user. To transmit the location data over the network, they would then chain a `SPENetworkIntent` to the `SPELocationIntent` which would further define the intent of the application in how it intends on sending the location data over the network. Chained intents provide flexibility in the framework so that the application can attach both a `SPENetworkIntent` and a `SPEDataPersistIntent` without bloating the implementation of the `SPELocationIntent`. It also allows reuse within the application and has a tight correlation to an attached policy where the chained intent will be validated against an attached policy if one exist, otherwise it will default to the user's global policy.

3.2 Class Wrappers

The SPE Framework includes class wrappers for all of the security- and privacy-sensitive classes within iOS. These class wrappers are utilized to retrieve a protected object or perform an operation on a protected object. These objects can then only be instantiated and used through the SPE Framework, meaning the objects returned are immutable. To minimize changes required in the application utilizing the framework, the wrapper method's implementation are as close as possible to the original implementation for the protected object. For example, in Objective-C if the consuming application wanted to create a `NSURLRequest` object, the changes are highlighted below:

Original Method:

```
NSURLRequest *request =
    [NSURLRequest requestWithURL:url]
```

Wrapper Method:

```
NSURLRequest *request =
    [SPEFramework requestWithURL:url
                      withIntent:speNetworkIntent];
```

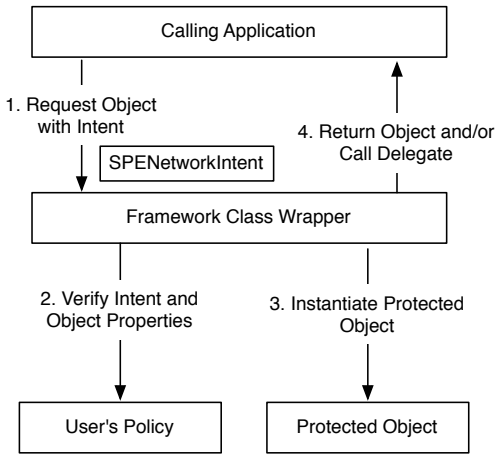

Fig. 3. Request of protected object using framework class wrapper with object being returned only after successful validation.

The class wrappers reduce the complexity of verifying the application's use of the framework by checking to ensure that each protected object is not directly instantiated or used. Upon validation, the application will always be returned an immutable object — any changes to the object's state have to be made through the framework. We recognize that making these changes throughout an application may not be feasible or impractical. To assist developers, we describe a *SPE Conversion Assistant* in Section 4.3 that will automatically enforce the SPE Framework on an application so that manual modification of the source code is not needed.

3.3 Validation

The SPE Framework ensures the application is truthful by examining the `SPEIntent` and the action being performed against the user's defined policy. If the properties within the object being requested do not match what is described in the intent (truthfulness) or if the intent violates the user's policy (policy violation) then no object is returned and the delegate that the consuming application defined is notified (Figure 3). If no policy violation occurred and the intent was truthful, then the object is returned as well as a notification to the delegate that validation was successful.

When an intent is validated, validation is performed within the `SPEIntent` object using the reflection API to

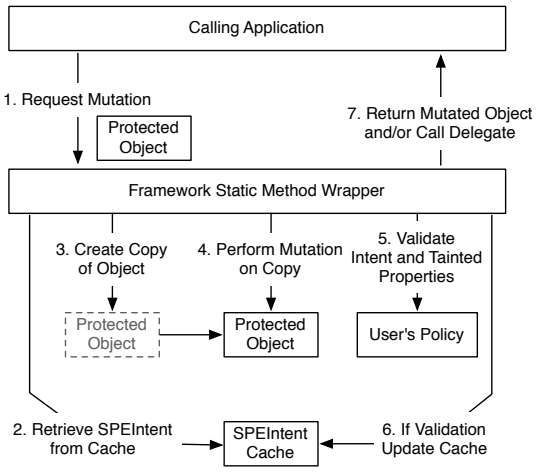

Fig. 4. Request of mutation of previously acquired protected object, mutation is created on copy, validated with cached intent, and returned as immutable.

determine the type of intent it is to validate itself appropriately. When an application needs to make a change to the protected immutable object that it received, it again goes through the framework so validation can be performed. The application invokes the corresponding operation on the framework with the immutable copy of the protected object. This way, the framework can perform the operation in a manner that can be validated.

To reduce the burden of requiring the application to pass with each call the protected object's related intent, the SPE Framework contains an *Intent Cache* which maps each protected object to its related intent. Each wrapper method in the SPE Framework will mark properties of the protected object that will be mutated as "tainted" by utilizing a *Taint Descriptor*. As part of validation, the framework will create a mutable copy of the protected object, perform the mutation on the copy, and then perform validation with the properties that were tainted via the *Taint Descriptor*. If validation succeeds, the framework will then change the mutable object to immutable, update the *Intent Cache* by replacing the previous object and its related intent with the new immutable object. The object is then returned with the mutation performed, or if there is an error the application is notified. This process is depicted in Figure 4.

When validation is performed, the process can be complex for properties that hold variable data. For example, consider an `HTTP POST` request where the body of the request can hold a variable amount of privacy data. Validation in this scenario is more complex and will consume more resources as they have to be checked for any security or privacy sensitive data. To perform this validation, SPE delegates the responsibility of checking this variable data to the *Data Inspector*. The *Data Inspector*'s responsibility is to examine chained intents on the protected object for any privacy or security data being utilized and then validate both the object and the intent to ensure that the intent is being truthful. To improve the efficiency of this process, the *Data Inspector* examines the *Intent Cache* for sensitive data that has been released to the application and check for that data to exist. This reduces resource consumption by not having to check each privacy and security sensitive component on the mobile device and only check those that have been accessed through the SPE Framework.

4 Implementation

Thus far, we have described the general concepts that define SPE. In this section, we present some salient details of our implementation of SPE for iOS. Most related research is targeted at Android, since that source code base is open to modification. Our goal, however, is a framework that is immediately accessible without OS modifications.

4.1 SPEIntents

The SPE Framework has a base `SPEIntent` class that it derives from. This class is used for validating itself against defined policies whether they be global policies or attached policies. To group like `SPEIntent`s together, we created several protocols. Some of these to note are: `SPEChainableIntentProtocol`, which allows for chained intents, and `SPESensorDataIntentProtocol`, which allows the application to set desired and minimum data accuracy requirements. For example, a navigation application would need to have precise location; if it just had the user's general location it would not be able to function properly. If the minimum sensor accuracy is not met, SPE validation will succeed; however, SPE will let the application respond appropriately and perhaps make a better case to the user why they need more precise location. These general protocols allow SPE to perform similar validation across similar `SPEIntent`s.

The class hierarchy of `SPEIntent` and its subclasses are analogous to the policies described in Section 2. For example, with a `SPEPhotoIntent` an application can specify that it intends to modify photos, that it will require the date attribute, and that it will require the location attribute. The corresponding policy allows the user to set restricted dates, restricted locations, and if the application can read the date and location attributes as well as write to the library. Within a policy a user can permit the application to read photos. However within a `SPEIntent`, read access is not specified as it is implied a read operation would occur.

4.2 Validation

All validation is performed within the `SPEIntent`; the intent will use introspection to first find what protocol the intent conforms to and from there deduce the correct subclass of `SPEIntent`. As mentioned earlier, this allows for maximum re-use of validation within the intent. When performing the validation there are two areas that SPE focuses on: (1) Is the `SPEIntent` *truthful* with its action, and (2) Does the `SPEIntent` *comply* with the user defined policy. To check if the `SPEIntent` is truthful, we improve the efficiency of how this is performed by tainting properties and operations of a protected object using a class wrapper. Because all mutations and instantiations of an object occur through the class wrappers provided by the SPE Framework, portions of the `SPEIntent` that have previously been

validated need not be revalidated. To validate a policy intent, we use an internal representation of the policy database in memory for the application and check the operations and instantiations of the protected object.

To validate attached data policies and chained `SPEIntents`, we created a `SPEShareableDataPolicy` protocol that allows a `SPEPolicyDataPersist` and/or `SPEPolicyNetworkBranch` to be attached to a privacy `SPEIntent`. For chained intents, we created a `SPEChainableIntentProtocol` which allows intents to be chained to a privacy related intent. When we validate the intent, we first validate the privacy intent against the privacy policy. If this passes, we then iterate through each chained intent and pass the shareable data policy which will have potentially an attached network or data persist policy. So when we validate chained intents, we are validating against the relevant attached policy if it exists, otherwise we validate the chained policy against the global network or data persistence policy for the application.

Here is the general validation algorithm:

validation = **true**
if Validate Privacy Intent to Privacy Policy **then**
  for all *Chained Intents* as *Chained Intent* **do**
    if not Validate *Chained Intent* to *Attached Policy* **then**
      validation = **false**
      break
    end if
  end for
else
  validation = **false**
end if

When we cache protected objects, we use a `NSDictionary` to store a mapping to a singleton copy of the validated object during the life of the application. If a validated object was a singleton itself, we store an identifier in the cache to the singleton so that it can be retrieved.

4.2.1 Sensor Accuracy and Privacy

With sensor accuracy and privacy, our goal is not to create a better privacy-preserving algorithm for location (that is the focus in other research), rather our focus is on providing broad methods of preserving privacy and show its effectiveness and effect on the OS. With the location policy, there are several levels that a user can define in their policy: precise, anonymous, generalized, no data, or bogus data.

Consider a coordinate location, say (-29.6404955, 144.7000729). If the policy setting is *anonymous*, we utilize a random number generator to generate a random number between -.04 and .04 and add it to the latitude and longitude (approximately a 2 mile radius around a given location). To compute the *generalized* location, we take only 1 significant digit for latitude and 2 significant digits for longitude which provides a consistent generalized location for a user given a specified area they exist in. If the policy setting is *bogus*, we generate a random location by generating random latitude and longitude values, and if the policy setting is no data we simply return nothing. For the *heading*, if it is *generalized* we set it to 0 (North), if it is *anonymous* or *bogus* we pick a random direction to return to the application.

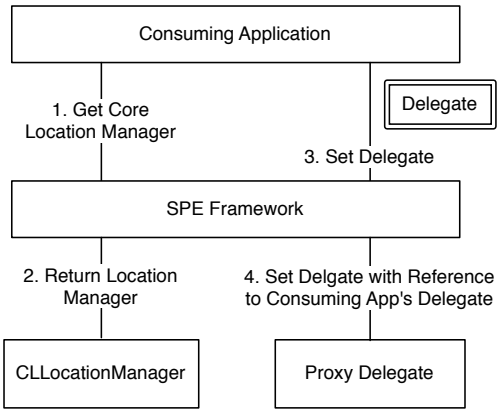

Fig. 5. Depicts proxy delegate utilized in SPE Framework to enforce location scrubbing and policies.

Implementing a class wrapper for location is more interesting; in iOS, a delegate is used when retrieving locations, so we needed to proxy the data being retrieved from the device. To accomplish this, we implement a proxy delegate that receives location updates, heading updates, and region updates. It will then notify the delegate within the application if validation passes of each location update (Figure 5). The proxy delegate also performs fuzzing and scrubbing based on the accuracy set. So if a user set their permitted location accuracy to *generalized*, the proxy delegate will capture the data and fuzz it before it is returned to the application. By utilizing a proxy delegate, we minimize the changes an application would need to make to gather locations from a user. Further, validation and scrubbing is done when the application attempts to directly access location and heading.

4.2.2 Proxying Privacy Data

Several libraries require that privacy data is enumerated using a block in Objective-C. This adds complexity for SPE to perform validation on the data being returned as the block can be called asynchronously. However much like a proxy delegate, we implement a proxy block that will check the data being sent back from the library being consumed and perform validation and scrubbing as necessary. An example of this is when accessing the photo library, `ALAssets` are returned through a block. SPE will utilize the proxy block to wrap the original block passed in from the application to proxy the request and perform the necessary validation. This again allows us to check against the policy and intent for both truthfulness and policy violations. Additionally, the proxy block allows the framework to scrub data such as removing date and location attributes from a photo if a user's policy did not permit access to those elements.

4.2.3 Temporal and Spatial Restrictions

When a user defines a location restriction, she can specify a point and the radius around that point to specify a restricted region. This type of spatial restriction is represented as a `CLCircularRegion` in iOS. These restrictions can be applied to policies that have spatial restrictions, *e.g.*, an application attempting to retrieve the user's current location

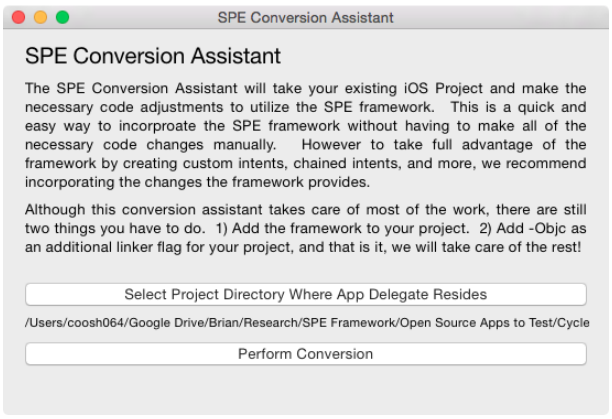

Fig. 6. Screenshot of SPE Conversion Assistant converting an open source application.

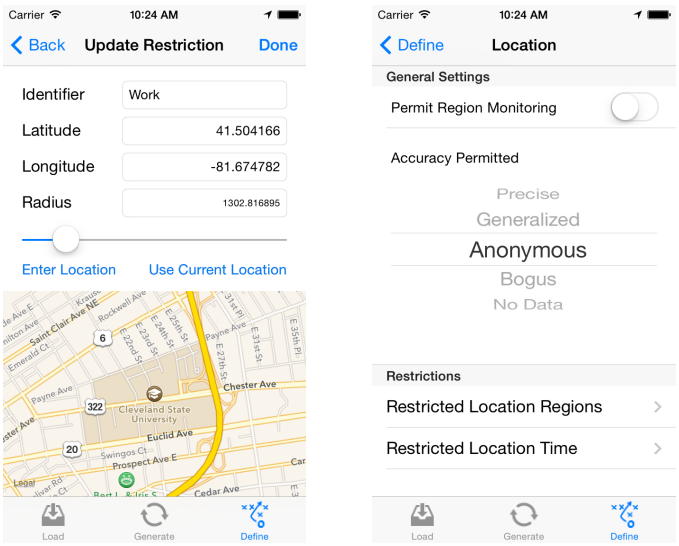

Fig. 7. Screenshots of SPE Policy application showing a restricted location being specified and general location policies.

or an application attempting to access a photo taken at a restricted location. To test if a location falls within one of these restricted regions, we utilize the iOS convenience method in the `CLCircularRegion` class:

```
1 - (BOOL)containsCoordinate:
2     (CLLocationCoordinate2D)coordinate;
```

When a user defines a temporal restriction, we provide a simple policy where the user can specify the time of day. Temporal restrictions can be applied to location as well as other policies including photos. The combination of these temporal and spatial policies allow a user to define with more granular access to their privacy data.

4.2.4 Notifications

SPE supports two types of notifications. To ensure that the application implementing the framework can handle different policy settings, the application can implement the `SPEResponseDelegate` protocol, which includes two methods: `didReceiveSuccessful-Validation` will be called with the successful intent, and `didReceiveDeniedValidation` will be called with an `NSError` object and the `SPEIntent` that was denied. In order to ensure that the user is informed of any kind of policy violations and truthfulness mismatches with the intent, the framework sends a `UILocalNotitification` that also shows a `UIAlertView` and allows the user to dismiss future alerts. If the user dismisses future alerts, they can still see the notifications in the *Notification Center*. If an application attempts to validate several intents without checking whether they were successful or not, we display an aggregate `UIAlertView` letting the user know that several violations occurred.

4.3 SPE Conversion Assistant

The class wrappers included in SPE attempt to stay as close as possible to the the original Objective-C implementations of protected methods. However it may be infeasible or impractical for an application to adopt these methods in place of existing methods. To address this situation we created the *SPE Conversion Assistant* which is a point-and-click solution that will inject the SPE Framework into an application's source code. It does this by injecting code into the application delegate that swaps out implementations of protected methods with SPE's own implementation. Once the call is intercepted, SPE performs the validation just as if the call was made to it directly through one of the class wrappers. Part of this approach is known to the iOS development community as *method swizzling*, which takes advantage of the Objective-C runtime to exchange method implementations at runtime. Figure 6 shows the *SPE Conversion Assistant* interface to perform a conversion of one of the open source applications we evaluated. A developer does not need access to the SPE framework's source code in order to integrate SPE into their application.

The SPE Framework is compiled into a static library that is linked to the application when it is built. The source code of the implementation of the SPE Framework is not open for modification when integrating within an application.

4.4 Policy Management

To allow a user to manage their policies, we created the *SPE Policy* application. Using the *SPE Policy* application a user can generate a policy and store it on a dedicated web server or a cloud service of their choosing. Because the policy file can be publicly accessible, it is encrypted using a credential-based key using the PBKDF2 algorithm [17].

When an SPE-converted application is first launched, SPE will prompt the user for the location of the policy file and the credentials to decrypt it. It will then download the file and install it for utilization by the SPE Framework. If the application is restarted, SPE will check if a policy file was installed and prompt the user to unlock the existing policy file using their credentials or it will allow the user to download a new policy file. Figure 7 shows the *SPE Policy* application with location restrictions being configured.

5 Evaluation

The primary goal of the SPE Framework is to protect the user from apps that misuse personal data. Such misuse may

App Name	Description	Nature of Violation
AlienBlue	Commercial open source Reddit client for iOS	No violations reported.
CamLingual	Read and translate signs using OCR	Location/Unencrypted Write to Disk
Congress	Application for tracking activity in congress	Block Google Analytics Data/Location
CycleStreets	Plan a cycle journey within the UK	Location/Unencrypted Write to Disk/Network
Doppio	Allow you to find closest Starbucks location	Location
FoxBrowser	Full fledged browser with Firefox Sync support	Block Google Analytics Data/Location/Unencrypted Write to Disk
Hacker News Client	Social news website on computer hacking and startup companies	No violations reported.
IRCCloud	IRC Chat client	Insecure Network
Plain Note	Simple note taking client with ability to synchronize notes	Insecure Sync of Notes on Network
Scanvine	News Aggregator	Unencrypted Write to Disk
Sol	Weather application	Location
Spika	Instant messaging application	Unencrypted Write to Disk/Network/Location
The White House	Shell application that White House utilizes	Block Google Analytics Data
Wikipedia	Popular crowdsourced internet encyclopedia	Location

TABLE 1
List of apps used for SPE Framework evaluation

either stem from malicious design on the part of the app developer, or it may be "accidental": a developer may use a third-party library, which in turn misuses personal data. The SPE Framework provides a way for app developers to be open and clear about their intentions of how personal data from a user is being used.

The SPE Framework, in its current state, requires the source code of the application that is being monitored. As such, in order to fully evaluate the capabilities of the framework, we need access to app source code. In [10], we presented a preliminary evaluation of the framework's capabilities by testing it against apps that we created for the purpose of testing. These apps did make data misuse violations, and the SPE Framework correctly identified those violations. For the evaluation in this paper, the *SPE Conversion Assistant* was utilized as it allows an application to be easily integrated with the SPE Framework.

5.1 Security and Privacy Enhancements Evaluation

For the purpose of this evaluation presented here, we did not use any applications we wrote ourselves. Instead, we identified apps written by other developers. Our evaluation is performed on a set of open-source iOS apps that are available for download from the iTunes App Store. The source code for these apps are all available for download from the GitHub source code repository. We found a total of **35** apps that were available both on GitHub and the App Store. However, not all of these apps were suitable for use in the evaluation. Out of these apps, the source code for **15** of them were not in a state where it would compile and build. A further **6** apps would not actually run on an iOS device or simulator. We finally arrived at the list of **14** apps (listed in Table 1 that would work for the evaluation. Out of the 14 apps that we tested, only **2** did not report a violation.

For this evaluation we built a policy using the *SPE Policy* application that would only allow network communication to the required domains for the targeted application. The policy did not allow access to privacy information on the device, which allowed us to monitor what privacy elements were accessed by the application as the framework would alert violations to the policy. The application has access to the policy file (Sec. 4.4). Once this file is generated, it was then stored on Google Drive and made publicly accessible via a dedicated URL. We used the *SPE Conversion Assistant* to inject the SPE Framework into each application . We then ran each application on an iPhone 5 or an iPad 3 depending on what devices the application supported.

When opening the application the SPE Framework prompts for both a URL to the policy file and the credentials to decrypt the policy file. After the policy was downloaded to the device and installed, we then ran the application and performed any required account setup that may be required to utilize the application. Using the policy we defined, we were able to determine what violations the SPE Framework were able to find, and adjust the policy as needed to allow the application to perform its expected operations.

5.1.1 Location Violations

Several of the applications we tested attempted to use the user's location. The applications that had a legitimate use case for the user's location were CycleStreets, Doppio, Sol, and Wikipedia. However there were cases where the location data was not yet required for the application to function, or the precise location of the user was not required.

With CycleStreets we found that the location was being requested before the user set up a route based on their current location. With Doppio and Wikipedia, a user may want to provide an anonymous location within their region so that they can view information around their current location without providing their exact location. For example, if a user is using Doppio to find Starbucks in their region, they would not have to provide their exact location to find Starbucks in the area. Instead, this location could be anonymized or generalized using the SPE Framework. Likewise with Wikipedia, if a user is looking for information about a city they are in, they again would not need to provide the exact location. Additionally, with the weather application Sol, an exact location is not required to get the forecast for the area. In these scenarios, SPE can be utilized to enforce generalized or anonymous location data.

While the above applications had legitimate use cases to request the user's location, Camlingual and Spika did not have legitimate use cases. Camlingual's sole functionality is to provide OCR capabilities to photos that a user takes using their mobile device; access to the user's location is

not required. Additionally, Spika requested the location of the user even though it was not required for the application to function. Lastly the Congress application does not need access to the user's location, however if the user wanted to provide it it could also be generalized or anonymized to provide the same functionality.

FoxBrowser was an exception to applications requesting access to the user's location. FoxBrowser did not explicitly request the user's location, however if the user visited a web page that requested the user's location, the request would be made. Because the SPE Framework utilizes method swizzling to intercept calls to protected resources, even though the location request was not explicitly defined in FoxBrowser, SPE was still able to enforce a location policy.

5.1.2 Network Violations

In several apps the SPE Framework was able to block data leaving the device to unauthorized endpoints, over untrusted channels, or to the Google Analytics service. In CycleStreets, the SPE Framework was able to block network requests that were not destined to the `openstreetmap.org` domain. In Plain Note, the framework was able to block notes from being synchronized over a non-SSL channel. If a user had sensitive data in a note that they created, they would not want it to be insecurely transferred to a cloud service or in general restrict it from being transmitted off the device. In IRCCloud, the framework was able to block communication to non-SSL endpoints. This is particularly useful in this application as available networks for the user to connect to was returned in a non-SSL request and could be spoofed. With a spoofed response, a user's credentials could be sent to an endpoint the user did not intend to send to. In Congress, FoxBrowser, and the White House, the framework was able to block data being transmitted to the Google Analytics service. By ensuring that network data only goes to required endpoints, consumer's can limit private date from being transmitted to advertising services.

5.1.3 Disk Violations

There were several apps (CamLingual, CycleStreets, Scanvine, and Spika) that attempted to write data to disk without explicitly requiring the data to be encrypted. This is a concern as this data could contain user information and if the device is ever lost or stolen, a malicious user could gain access to this data without unlocking the device.

5.1.4 Discussion

AlienBlue and Hacker News Client were the only applications that did not report a violation. The SPE Framework still was able to enforce a user's defined policy, however the applications did not access privacy data or send data over the network unencrypted or write data to disk unencrypted.

From this evaluation we are able to demonstrate the effectiveness of the SPE Framework. The evaluation presented here can be performed against any application where the source code is available. With our evaluation we did not specifically seek applications that appeared to be malicious, had previous reports of leaking privacy data, or had security concerns. Instead our evaluation included any application for which we could obtain the source code, build, and run the application in the simulator or device. With all apps that fit this criteria, the SPE Framework is able to be successfully integrate within the application using the *SPE Conversion Assistant*. Because the SPE Framework intercepts requests to methods that request access to security and privacy sensitive elements, the SPE Framework is able to guarantee enforcement of the user's defined policy. This enforcement is also guaranteed against third-party libraries, which was shown by blocking requests to the Google Analytics service. In comparison to other previously proposed methods, SPE does not attempt to identify patterns of malicious behavior which can report false positives and false negatives. SPE also does not attempt to perform taint analysis of privacy data which is also subject to reporting false positives [18].

5.2 Performance Evaluation

While our main goal is to look at policy enforcement, we also evaluate the performance overhead of using the SPE framework. To evaluate the overhead we created an application that focused on individual units of the framework. All test cases were repeated one hundred times on an iPhone 5 and the results were averaged. The performance tests were focused on individual components of the framework to clearly identify where more overhead was observed.

5.2.1 Network Access

To evaluate network access we utilized a policy that allowed communication to a specific domain, required SSL, and permitted a credential to be sent over SSL. In Test A the application's intent was truthful and it followed the user-defined policy. In Test B the application attempted to send a network request to a domain it did not intend to, simulating an intent being untruthful. In Test C, the intent was truthful, however it did not abide to the policy. Finally in Test D, the intent was truthful however it attempted to inject credentials in the request it was not permitted to use from the user policy (Figure 8).

While the observed computing overhead seems significant, this test did not actually send the network request onto the network. We did not send the request as the response time could be inconsistent and mocking the network would only provide a constant delay to both tests. From observing the overhead, it may seem counter-intuitive that the truthful case took longer to execute. However since the truthfulness of the intent is checked first and passed, the request was then validated against the policy which took longer to perform. Additionally, this overhead difference was expected as with each network request, the *Data Inspector* described earlier in the paper scans the data in the request to ensure that none of the data that has been requested by the framework is being leaked in the network request without intent.

5.2.2 Data Persistence

To evaluate data persistence, we wrote a medium sized image (378 KB) to the local filesystem (Figure 9). In Test A, the intent was truthful and the policy was followed. In Test B, the intent was untruthful but the policy was followed. In Test C, and D, we mirrored Test A and B respectively with requiring encryption. In contrast to our network evaluation, the difference here is smaller between operations that use

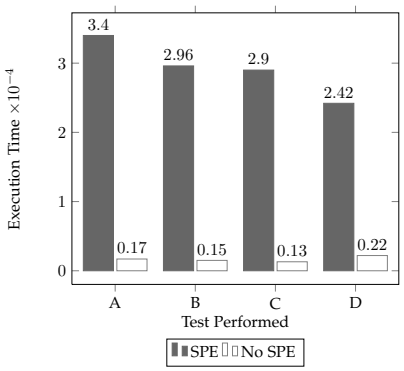

Fig. 8. Network Testing with NSURLRequest - Intent Truthfulness, Abiding Policy, Credential Injection

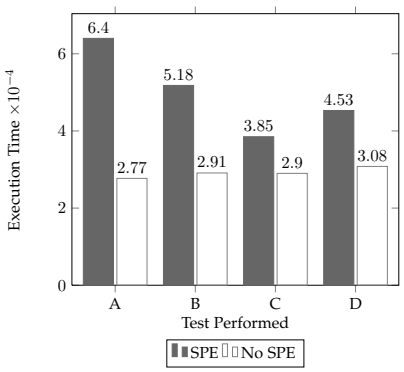

Fig. 9. Data Persistence Testing with NSData - Intent Truthfulness and Encryption

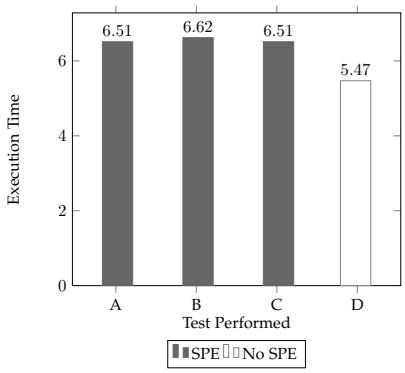

Fig. 10. Photo Library Access with Scrubbing and Intent Truthfulness

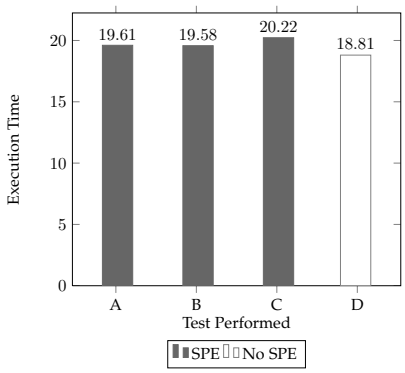

Fig. 11. Location Testing with Anonymization, Spatial and Temporal Restrictions

SPE and those that do not. This can be attributed to the actual data persist operation being performed whereas in the other case, the request is never sent over the network. We also observed that when not using encryption, the validation and execution of tasks using the SPE Framework contains less of an overhead.

5.2.3 Photo Library

To evaluate photos, we enumerated through 25 photos on the device, retrieved date and location attributes, and created a UIImage object from the photo (Figure 10). In Test A, SPE was utilized to scrub date and location attributes of the photo with a truthful intent. Test B was similar to Test A, however with the intent not being truthful. Test C did not scrub date and location attributes and the intent was truthful. Finally in Test D, SPE was not used at all.

From our observations, utilizing SPE showed a marginal overhead even when attributes were scrubbed. This is in stark contrast to our network evaluation, but unlike the network evaluation we were enforcing policies against properties of a photo and not examining data for potential privacy leakage. With our photo evaluation, we only included one non-SPE test as there is only primarily one method to access the photo library.

5.2.4 Location Evaluation

Evaluation of location sensing is interesting as there is an inherent delay in receiving location updates. We tested receiving 25 location updates five times to get an average response time (Figure 11). Test A evaluated a spatial restriction that the device did not violate and likewise Test B utilized a temporal restriction that the device did not violate. Test C used anonymization to "scrub" the location data. In Test D, the SPE framework was not used to enforce policies or use the proxy delegate described earlier.

In this evaluation we did not include region monitoring as that would require the device to enter and exit regions which we believe would not provide much value. From our observations, the computing overhead appeared small compared to the overall operation of receiving location data.

5.2.5 Performance Evaluation Discussion

Our performance evaluation examined several different permutations of using the SPE Framework. Only in the network evaluation did we observe a significant proportional overhead. This overhead was expected as the data within a network request is inspected for any privacy data that was requested and may have been leaked. Future work can focus on improving the detection of any privacy data being leaked in locations such as the body of an HTTP request. However efficient methods for doing this was not the focus of our research. Additionally, during our evaluation of the several iOS open source applications that we tested against, we did not perceive a noticeable delay.

The policy that the user defines will also have an impact on performance. For example, the more restricted locations a user specifies, the longer it would take for the framework

to enforce the policy. The focus of this evaluation was to examine the core concepts of the framework against a policy.

In our evaluation, we focused primarily on execution time to measure the performance overhead of SPE and to ensure that there was no delay in user experience. CPU and energy is more difficult to measure in the targeted evaluation we created that evaluated different aspects of utilizing SPE. However both CPU and energy can be expressed as functions of execution time, at least to a limited extent where they would be proportional.

To measure the memory footprint we used the *Instruments* application that is available with the *Xcode* IDE and analyzed memory consumption across several of the open source apps we tested. With each application we utilized a policy that was fairly restrictive and could be considered a moderate size policy. Overall we found the memory footprint to be small. For example, a *SPENetworkIntent* consumed 128 bytes and a *SPEPhotoIntent* consumed 112 bytes. A user's policy in memory had an overhead of approximately 1.2 KB. Lastly the *SPEMethodRegistry*, which is responsible for holding the runtime implementation of methods SPE is protecting, only consumed 64 bytes. Overall we found the average memory consumption of SPE to be approximately 1.7 KB while running the applications. The larger a policy is and the more complex rules that SPE would have to enforce could increase memory consumption, however based on our findings utilizing a moderate size policy file, memory overhead is less of a concern.

6 Related Work

There has been a good amount of work focusing on enhancing the security and privacy controls on mobile devices [19]. In this section we review how some of that work relates to the work described in this paper.

Taming Information Stealing Smartphone Applications (TISSA) provides lightweight protection, application transparency, and is built on top of existing Android security mechanisms. The implementation required less than one thousand lines of code and had a low performance overhead [11]. TISSA focused on four types of user data: contacts, phone identity, call logs, and location data. They provide the option to return for these values one of three return value types: none for no data being returned, anonymous for an anonymous version of the data, and bogus to provide a fake result [11]. TISSA required modification to the Android OS.

IdentiDroid focuses on anonymity by proposing a custom Android OS that ensures applications cannot identify a user [21]. IdentiDroid takes a unique approach by shadowing data that identifies the user and block runtime permissions that lead access to identifying data. Their evaluation demonstrated that their solution is highly effective with minimal impact to the applications on the device.

The *Android Runtime Security Policy Enforcement Framework* (SEAF) focuses on dynamic behavior of the application and validates applications by exercising permission patterns [22]. This allows SEAF to define certain patterns that could be malicious such as *READ CONTACTS, INTERNET*, which can indicate misuse by sending user's contacts over the Internet [22]. This framework also exhibits low overhead and requires a modification to the Android OS.

TaintDroid uses dynamic taint analysis to monitor variables, method calls, and data on the filesystem to determine if data is being leaked [18]. TaintDroid showed success by only incurring a 14% overhead from 30 applications that it monitored and was able to identify 68 instances of misuse [18]. However it was acknowledged that dynamic context-based privacy sensitive information such as latitude and longitude coordinates could be hard to detect as they could be just random floating point numbers [18].

AndroidLeaks take a different approach as the previously mentioned work. Instead of modifying the OS, they propose a static analysis framework for Android by using WALA and performing reachability analysis [4]. They create a set of mappings from Android APIs that can leak data, and they also look at popular advertisement libraries. They then use ded and dex2jar to convert applications into Java source code or byte code and if an application has one source and one sink, they perform static taint analysis to determine if it reaches the sink. To check for privacy leakage on callbacks, they taint data so that when it comes back via a callback, they can see if that data is being used.

PSiOS focuses on iOS, however it does require the device to be jailbroken as it requires including a shared library with each application [23]. With their approach, the enforcement framework is applied to all applications and provides fine grained privacy controls. In their research they evaluated several popular applications to prove its effectiveness. While their solution does not require modification to the source code, the inherent trust in the OS is lost with the device being jailbroken.

Other related work includes *RecDroid* where they take a novel approach by recognizing that permissions granted when the application is first run is not effective, and their approach utilizes export recommendations to modify permissions at runtime on Android [24]. Papamartzivanos et al. [25] proposed a cloud solution that utilizes crowdsourcing to identify privacy leaks in mobile applications. Their solution uses this information to alert users and the community of application misbehaviors. Chin et al. [26] performed a study across 60 smartphone users to find out what tasks users feel comfortable performing on their smartphone, why applications are selected by users, and from this study they provide recommendations for smartphone platforms. In another study Liu et al. [27] analyzed the permissions users granted to mobile applications on Android and realized that the permission model is to complex and could be reduced. The platform they propose requires the device to be rooted and allows the user to choose between three different settings where settings can be changed at a later time dynamically. Beresford et al. [28] proposed *MockDroid* where a user could mock an application's access to a resource and allow the user to determine if the application could function without having access to a particular resource. They rely on the idea that an application has to be resilient enough to where if access to a resource is not available, that the application could still function. This allows a user to be selective on what resources an application can access while still being able to function. As with other previously mentioned work, Mockdroid required a modified Android OS. The MOSES framework for Android is a policy based framework that focuses on isolating software and data on the device [20]. With

MOSES, security profiles can be defined to effectively build virtual environments within the operating system to isolate applications. This is done through dynamic switching between different security profiles. MOSES showed minimal overhead in both latency and battery.

Finally, *ProtectMyPrivacy* (PMP) [29] is another solution that helps mitigate and detect privacy leaks on iOS. Their approach is significantly different: they utilize a crowd-sourced engine that cannot be significantly impacted by one user and they were able to recommend protection settings for over 97.1% of the 10,000 most popular apps. Their solution requires a jailbroken iOS device, however, but they have had success in the Cydia app store where at the time of publication they had 90,621 users.

Much of the recent related work presented here has focused on jailbreaking the device or modifying the operating system to provide the consumer with additional security and privacy controls. To modify the operating system, the OS needs to be open source (only Android as of now). Even with only Android being open source, most devices carry a modified version of the OS that is not open source but maintained by the device manufacturer. So most of these proposals can only be applied to those devices that run 100% pure Android such as Nexus devices.

7 Discussion

While SPE provides a way for developers to be open and clear about their intentions, developers may require an incentive to adopt the framework. Demand for adoption of SPE can be driven by consumers and businesses, similar to businesses demanding applications to adopt MDM frameworks. The advantage with SPE is the change in the application is small and automated. Additionally, developers that adopt SPE do not have to consider the framework during development since the framework is injected at runtime. However, applications supporting SPE can earn the trust of consumers by adopting the framework.

Using the runtime modification approach to inject SPE into an application, future enhancements can be made to strengthen the detection capabilities in the *Data Inspector*. Currently the *Data Inspector* can detect data that has not been modified by being encoded, encrypted, or some other transformation. Using the Objective-C runtime, SPE can perform data flow analysis to detect when data is being included in a request that leaks data from the device. Additional future work includes verifying that an application is using an unmodified version of the SPE Framework. By using SPE's interception implementation, verification can be built into the static library to ensure that the framework has not been modified and is enforcing the user's policy. Additionally, an external entity can verify that an application implementing SPE is using an unmodified version of the framework through a challenge response mechanism and comparing a generated signature of the static library with existing signatures.

We also recognize that maintaining the policy for the SPE Framework can be cumbersome for users. We are currently working on a follow up framework that dynamically constructs a policy for the user based on user choices and machine learning algorithms.

8 Conclusion

In this paper we have presented the *Security and Privacy Enhanced* (SPE) framework. We described the policy model it utilizes, the core design of the framework, and details on an implementation that allows a consumer or business to effectively ensure that security and privacy policies are enforced. Additionally, we proposed a novel approach that uses intents to describe to the user how the application will use their data and enforce these intents. Compared to recent research that has focused on modifying open mobile operating systems or jailbreaking closed-source operating systems like iOS, the SPE Framework takes a different approach. While the SPE Framework does require modification to the application, it does not require modification to the OS or for a device to be jailbroken or rooted. We believe this is a more sustainable approach as OS updates do not impact the SPE Framework unless there are significant API changes. Frequent updates to mobile operating systems have led to fragmentation, with modifications to Android by both carriers and device manufacturers. Additionally, a consumer does not need to compromise the built-in security of their device by jail breaking or rooting the device; with SPE they add another layer of protection. Lastly with SPE a consumer can use a stock device with a stock operating system. Based on the results of our evaluation, SPE is highly effective and prevents several privacy and security concerns from several iOS applications. In the near future we plan on releasing the SPE Framework, SPE Conversion Assistant, and SPE Policy application as open source projects. From this, an external entity can be created for developers to retrieve the SPE Framework to incorporate within their application or the framework could be tied into the workflow for application submission.

9 Acknowledgments

This work is partially supported by a NSF CAREER award (*CNS-0746632*) and by a CSU Doctoral Research Award. A preliminary version of this article was presented at the Workshop on Mobile Cloud and Social Computing [10].

References

[1] Symantec, "Internet security threat report volume 19," Symantec, Mountain View, Tech. Rep., 2014.

[2] ——, "Norton mobile insight discovers facebook privacy leak," February 2014. [Online]. Available: http://www.symantec.com/connect/blogs/norton-mobile-insight-discovers-facebook-privacy-leak

[3] J. Ball, "Angry birds and 'leaky' phone apps targeted by nsa and gchq for user data," February 2014. [Online]. Available: http://www.theguardian.com/world/2014/jan/27/nsa-gchq-smartphone-app-angry-birds-personal-data

[4] C. Gibler, J. Crussell, J. Erickson, and H. Chen, "Androidleaks: automatically detecting potential privacy leaks in android applications on a large scale," in *Proceedings of the 5th international conference on Trust and Trustworthy Computing*, ser. TRUST'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 291–307. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-30921-2_17

[5] C. Welch, "Path allegedly geotagging photo posts even after users disable location services (update)," February 2013. [Online]. Available: http://www.theverge.com/2013/2/1/3941554/path-allegedly-geotagging-posts-when-location-services-disabled

[6] K. Bostic, "Path app again accused of unacceptable address book access." [Online]. Available: http://appleinsider.com/articles/13/04/30/path-app-again-accused-of-unacceptable-address-book-access

[7] Juniper, "2011 mobile threats report," Juniper, Sunnyvale, Tech. Rep., 2012.

[8] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "Riskranker: scalable and accurate zero-day android malware detection," in *Proceedings of the 10th international conference on Mobile systems, applications, and services*, ser. MobiSys '12. New York, NY, USA: ACM, 2012, pp. 281–294. [Online]. Available: http://doi.acm.org/10.1145/2307636.2307663

[9] W. Z. Brian Krupp, Nigamanth Sridhar, "An ontology for enforcing security and privacy policies on mobile devices," in *Proceedings of the 6th International Conference on Knowledge Engineering and Ontology Development (KEOD'14)*, 2014.

[10] B. Krupp, N. Sridhar, and W. Zhao, "A framework for enhancing security and privacy on unmodified mobile mobile operating systems," in *The First International Workshop on Mobile Cloud and Social Computing*, 2013.

[11] Y. Zhou, X. Zhang, X. Jiang, and V. W. Freeh, "Taming information-stealing smartphone applications (on android)," in *Proceedings of the 4th international conference on Trust and trustworthy computing*, ser. TRUST'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 93–107. [Online]. Available: http://dl.acm.org/citation.cfm?id=2022245.2022255

[12] "Hundreds of intimate celebrity pictures leaked online following alleged iCloud breach," Sep. 2014. [Online]. Available: http://bit.ly/1vEPD60

[13] B. Adida, "Sessionlock: securing web sessions against eavesdropping," in *Proceedings of the 17th international conference on World Wide Web*, ser. WWW '08. New York, NY, USA: ACM, 2008, pp. 517–524. [Online]. Available: http://doi.acm.org/10.1145/1367497.1367568

[14] Z. Wang, R. Murmuria, and A. Stavrou, "Implementing and optimizing an encryption filesystem on android," in *Proceedings of the 2012 IEEE 13th International Conference on Mobile Data Management (mdm 2012)*, ser. MDM '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 52–62. [Online]. Available: http://dx.doi.org/10.1109/MDM.2012.31

[15] C. Miller, D. Blazakis, D. DaiZovi, S. Esser, V. Iozzo, and R. Weinmann, *iOS Hacker's Handbook*. Wiley, 2012. [Online]. Available: http://books.google.com/books?id=KmAMKpWhOwUC

[16] Apple, "ios security," October 2014. [Online]. Available: https://www.apple.com/iphone/business/docs/iOS_Security_Feb14.pdf

[17] B. Kaliski, "Pkcs 5: Password-based cryptography specification," 2000. [Online]. Available: https://www.ietf.org/rfc/rfc2898.txt

[18] W. Enck, P. Gilbert, B.-G. 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 *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–6. [Online]. Available: http://dl.acm.org/citation.cfm?id=1924943.1924971

[19] M. La Polla, F. Martinelli, and D. Sgandurra, "A survey on security for mobile devices," *Communications Surveys Tutorials, IEEE*, vol. 15, no. 1, pp. 446–471, First 2013.

[20] G. Russello, M. Conti, B. Crispo, and E. Fernandes, "Moses: Supporting operation modes on smartphones," in *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies*, ser. SACMAT '12. New York, NY, USA: ACM, 2012, pp. 3–12. [Online]. Available: http://doi.acm.org/10.1145/2295136.2295140

[21] B. Shebaro, O. Oluwatimi, D. Midi, and E. Bertino, "Identidroid: Android can finally wear its anonymous suit," *Trans. Data Privacy*, vol. 7, no. 1, pp. 27–50, Apr. 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2612163.2612165

[22] H. Banuri, M. Alam, S. Khan, J. Manzoor, B. Ali, Y. Khan, M. Yaseen, M. N. Tahir, T. Ali, Q. Alam, and X. Zhang, "An android runtime security policy enforcement framework," *Personal Ubiquitous Comput.*, vol. 16, no. 6, pp. 631–641, Aug. 2012. [Online]. Available: http://dx.doi.org/10.1007/s00779-011-0437-6

[23] T. Werthmann, R. Hund, L. Davi, A.-R. Sadeghi, and T. Holz, "Psios: Bring your own privacy & security to ios devices," in *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ser. ASIA CCS '13. New York, NY, USA: ACM, 2013, pp. 13–24. [Online]. Available: http://doi.acm.org/10.1145/2484313.2484316

[24] B. Rashidi, C. Fung, and T. Vu, "Recdroid: A resource access permission control portal and recommendation service for smartphone users," in *Proceedings of the ACM MobiCom Workshop on Security and Privacy in Mobile Environments*, ser. SPME '14. New York, NY, USA: ACM, 2014, pp. 13–18. [Online]. Available: http://doi.acm.org/10.1145/2646584.2646586

[25] D. Papamartzivanos, D. Damopoulos, and G. Kambourakis, "A cloud-based architecture to crowdsource mobile app privacy leaks," in *Proceedings of the 18th Panhellenic Conference on Informatics*, ser. PCI '14. New York, NY, USA: ACM, 2014, pp. 59:1–59:6. [Online]. Available: http://doi.acm.org/10.1145/2645791.2645799

[26] E. Chin, A. P. Felt, V. Sekar, and D. Wagner, "Measuring user confidence in smartphone security and privacy," in *Proceedings of the Eighth Symposium on Usable Privacy and Security*, ser. SOUPS '12. New York, NY, USA: ACM, 2012, pp. 1:1–1:16. [Online]. Available: http://doi.acm.org/10.1145/2335356.2335358

[27] B. Liu, J. Lin, and N. Sadeh, "Reconciling mobile app privacy and usability on smartphones: Could user privacy profiles help?" in *Proceedings of the 23rd International Conference on World Wide Web*, ser. WWW '14. New York, NY, USA: ACM, 2014, pp. 201–212. [Online]. Available: http://doi.acm.org/10.1145/2566486.2568035

[28] A. R. Beresford, A. Rice, N. Skehin, and R. Sohan, "Mockdroid: Trading privacy for application functionality on smartphones," in *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications*, ser. HotMobile '11. New York, NY, USA: ACM, 2011, pp. 49–54. [Online]. Available: http://doi.acm.org/10.1145/2184489.2184500

[29] Y. Agarwal and M. Hall, "Protectmyprivacy: detecting and mitigating privacy leaks on ios devices using crowdsourcing," in *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, ser. MobiSys '13. New York, NY, USA: ACM, 2013, pp. 97–110. [Online]. Available: http://doi.acm.org/10.1145/2462456.2464460