Name: In-depth assessment – Google Apple Exposure Notification (**Apple**)

Contract nr: LC-01499045

Date of signature: 30-04-2020

Name of the deliverable: Deliverable 3.11

Authors: Joost Agterhoek, Michiel Leenaars (NLnet)

Level of distribution: Confidential, only for members of the facility (including the Commission Services)

Confidential: Yes

Radically Open Security

Source Code and Binary Audit

European Commission

V 1.2
Amsterdam, January 24th, 2023
Confidential

Document Properties

Client	European Commission
Title	Source code and binary audit
Targets	Source code - https://developer.apple.com/exposure-notification/ Binaries contained in iOS 15.4 firmware - https://updates.cdn-apple.com/2022FCSWinter/fullrestores/071-09661/51A3C446-9140-48F9-9190-3DFD963CA63D/iPhone10,3,iPhone10,6_15.4_19E241_Restore.ipsw
Version	1.2
Pentesters	Fabian Freyer, Daniel Attevelt
Authors	Daniel Attevelt, Marcus Bointon
Reviewed by	Sipke Mellema, Marcus Bointon
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
1.0	April 30th, 2022	Daniel Attevelt	Initial draft
1.1	January 4th, 2023	Marcus Bointon	Review
1.2	January 24th, 2023	Daniel Attevelt	Final

Contact

For more information about this document and its contents please contact Radically Open Security B.V.

Name	Melanie Rieback
Address	Science Park 608 1098 XH Amsterdam The Netherlands
Phone	+31 (0)20 2621 255
Email	info@radicallyopensecurity.com

Table of Contents

1 Executive Summary

1.1 Introduction

Between March 1, 2022 and April 30, 2022, Radically Open Security B.V. carried out a code audit and binary analysis for the European Commission (the EC).

This report contains our findings as well as detailed explanations of exactly how ROS performed the audit.

1.2 Scope of work

The scope of the penetration test was limited to the following targets:

- Source code - https://developer.apple.com/exposure-notification/
- Binaries contained in iOS 15.4 firmware - https://updates.cdn-apple.com/2022FCSWinter/fullrestores/071-09661/51A3C446-9140-48F9-9190-3DFD963CA63D/iPhone10,3,iPhone10,6_15.4_19E241_Restore.ipsw

1.3 Project objectives

ROS will perform a code audit and binary analysis of the selected targets with the EC in order to assess the security of the COVID-19 Exposure Notification framework, as well as the privacy claims made by Apple regarding said framework. To do so ROS will analyze the source code and appropriate binaries to ascertain the security properties of the system as well if sufficient safeguards regarding user privacy are in place.

1.4 Timeline

The security audit took place between March 1, 2022 and April 30, 2022.

1.5 Results In A Nutshell

As part of the EC Review Facility, we reviewed Apple's implementation of the Google / Apple Exposure Notification system. The Exposure Notification system is built into the iOS operating system. It serves as a framework which allows for COVID-19 related contact tracing using the iPhone's bluetooth technology.

We based our review on the source code Apple has Published (April 2022), alongside reverse engineering of the iOS implementations in iOS 15.4, since earlier published versions did not match well with the code running on iPhones.

The iOS sandboxing model is focused on rogue apps, however, it is not sufficient to protect against vulnerabilities in Apple's own code. Large parts of the exposure notification framework's supporting services are implemented within iOS daemons which handle potential attacker-controlled input such as `bluetoothd`, pose large external attack surfaces, and have had a number of vulnerabilities identified in them in the past.

In contrast to previously published source code, the source code published by Apple at the time of testing matches very well with parts of the implementations in iOS 15. However, large parts of the implementation have been omitted, including many APIs that are used between internal Apple services. The published source code suggests a much more limited functionality compared to the actual implementation. This is especially true for the Exposure Notifications Express mode.

It is clear that Apple has implemented more functionality related to Exposure Notification since the publication of the source code.

Security Model

At the core of the security model lies a privilege separation between the Regional/National Exposure Notification app, and the Exposure Notification implementation within the operating system:

The app acts only as a thin UI layer, setting configuration values used for calculating the risk score, and displaying the risk score. The app can also retrieve the following information from the system:

- Exposure windows, which correspond to a potential exposure during a 30-minute period.
- Temporary Exposure Keys (TEK)s (also referred to as Diagnosis Keys in this context) used to derive the Rolling Proximity Identifiers (RPI) which are part of the broadcast beacons submitted to a key server. This only includes keys that are over 24 hours old, and requires user consent. Authorization can be requested up to five days prior to it being used.
- Whether a user traveled to other regions within the last 14 days.

The app should not be able to receive other information from the operating system; especially not their (precise) location or RPIs seen by the device.

Per country or region, only a single, official, app available from the Apple App Store is authorized to retrieve this information. This is enforced through provision of an entitlement by Apple, which is tied to contractual obligations, to the authorized apps. Unauthorized apps with this entitlement are not allowed in the App Store by Apple. The security model of Exposure Notifications is therefore contingent on Apple's "walled garden" App Store model.

The operating system provides a set of services to the app. It operates the Exposure Notification Bluetooth Protocol, listening for and storing Bluetooth Low Energy (BLE) beacons as well as broadcasting beacons containing RPIs and the Associated Encrypted Metadata (AEMK) when Exposure Logging is active. On behalf of the app, it downloads datasets from key servers, matches them against the local database of received beacons, calculates a risk score and exposure windows, and passes these to the app. It enforces authentication and authorization of the app to ensure that only a single, official, national or regional app is allowed to access the exposure notification information.

Furthermore, exposure notification telemetry is collected and gathered online using Local Differential Privacy. This approach aims to "provide statistical, aggregate, and differentially private metrics to [Public Health Authorities (PHAs)] while protecting the individual contributions, and therefore the privacy of each participant" (Exposure Notification Privacy-preserving Analytics (ENPA) White Paper).

Two features

We have detected there are two NE systems present in the iOS operating system. The first, we will call the API, the second ENX.

The API was released in iOS 13.5 and the source-code is largely published here.

Apple's implementation also allows for an app-less approach (Exposure Notifications Express). This feature was released in iOS 13.7. However, the source-code for this feature has not been published.

The main difference between the Exposure Notification API and ENX is that ENX implements support for a second server, called the test verification server. Health practitioners report positive tests to this server, and the user then receives a code which then needs to be verified. From here on the system appears to work the same way as the EN API. Due to its closed-source nature, we have been unable to obtain a full picture of the security model of this feature, however we have reported on the parts we were able to identify and analyze.

2 Methodology

2.1 Planning

Our approach for this audit has been as follows:

1. **Source code analysis**
   We have evaluated the published source code to gain an understanding of the the operation of the exposure notification framework as well as its security model.
2. **Cross reference source code against binary implementation**
   We have cross referenced the source code against the binary implementation. In this way we are able to find differences and nuances in between the published source code and the actual implementation.
3. **Deep dive into the binary**
   Since we discovered there is more functionality implemented than the published source code suggests, we took a deep dive into the relevant binaries to gain an understanding how they work.
4. **Dynamic analysis**
   We used several dynamic analysis techniques to confirm our hypotheses on how the implementations work, as well how the exposure notification framework functions in the bigger picture. Ie, which servers is it connected to, how are these connections secured?

3 Reconnaissance and Fingerprinting

We were able to gain information about the software and infrastructure through the following automated scans. Any relevant scan output will be referred to in the findings.

- Binary ninja – https://binary.ninja
- Objective ninja – https://github.com/jonpalmisc/ObjectiveNinja
- sslscan – https://github.com/rbsec/sslscan
- FRIDA – https://frida.re/
- SandBlaster – https://github.com/malus-security/sandblaster
- jtool2 – http://www.newosxbook.com/tools/jtool.html

4 Security Architecture

The main service implementing exposure notifications runs as an XPC service with the endpoint `com.apple.ExposureNotificationService`, launched on activation by the system init process, `launchd`, as configured in the `System/Library/xpc/launchd.plist`:

```
"/System/Library/Frameworks/ExposureNotification.framework/ExposureNotification" => {
  "_serviceBundles" => [
    0 => {
      "_executablePath" => "/System/Library/Frameworks/ExposureNotification.framework/XPCServices/
ExposureNotificationService.xpc/ExposureNotificationService"
      "_infoPlist" => {
        "CFBundleExecutable" => "ExposureNotificationService"
        "CFBundleIdentifier" => "com.apple.ExposureNotificationService"
        "CFBundleName" => "ExposureNotificationService"
        "CFBundlePackageType" => "XPC!"
        "CFBundleVersion" => "1"
        "XPCService" => {
          "ServiceType" => "Application"
        }
      }
    }
  ]
}
```

The main code for it is implemented in `ExposureNotificationDaemon.framework`, which is part of the dyld_shared_cache and is part of both `bluetoothd` and `ExposureNotificationService`.

Access to the XPC service provided by the ExposureNotificationService by an App happens through the public `ExposureNotification.framework`, which, as part of the `dyld_shared_cache`, is loaded into the app. Apps use its exposed interface through an `ENManager` ObjectiveC-Object. Its public interfaces are covered by Apple's Developer Documentation. It acts as an XPC client to the `com.apple.ExposureNotification` XPC service. As part of the app itself, it bridges the undocumented XPC interface to a public API, but does not represent a security boundary.

A second version of the exposure notification daemon runs as part of the system bluetooth daemon, `bluetoothd`. It is configured to accept requests on the `com.apple.ExposureNotification` mach service:

```
"/System/Library/LaunchDaemons/com.apple.bluetoothd.plist" => {
  "EnableTransactions" => 1
  "KeepAlive" => {
    "SuccessfulExit" => 0
  }
  "Label" => "com.apple.bluetoothd"
  "LaunchEvents" => {
    "com.apple.notifyd.matching" => {
      "com.apple.mobile.lockdown.activation_state" => {
        "Notification" => "com.apple.mobile.lockdown.activation_state"
      }
    }
  }
  "MachServices" => {
    "com.apple.bluetooth.BTPacketLogger" => 1
    "com.apple.bluetooth.cloudkit.xpc" => 1
    "com.apple.bluetooth.xpc" => 1
    "com.apple.BTAudioHALPlugin.xpc" => 1
    "com.apple.ExposureNotification" => 1
    "com.apple.server.bluetooth" => 1
    "com.apple.server.bluetooth.bbprovider.xpc" => 1
    "com.apple.server.bluetooth.classic.xpc" => 1
    "com.apple.server.bluetooth.general.xpc" => 1
    "com.apple.server.bluetooth.le.att.xpc" => 1
    "com.apple.server.bluetooth.le.pipe.xpc" => 1
    "com.apple.usernotifications.delegate.com.apple.ExposureNotification.UserNotification" => 1
    "com.apple.wirelessproxd" => 1
  }
  "POSIXSpawnType" => "Interactive"
  "ProgramArguments" => [
    0 => "/usr/sbin/bluetoothd"
  ]
  "PublishesEvents" => "com.apple.bluetooth.discovery"
  "RunAtLoad" => 1
  "SuccessfulExit" => 0
  "UserName" => "mobile"
}
```

When started as part of `ExposureNotificationService`, the `xpcMain` function of the `ExposureNotificationDaemon.framework` is invoked, providing an XPC service, `com.apple.ExposureNotificationService`, which clients can connect to.

Access control

Access control to the NE feature is implemented in 3 separate mechanisms

1. Checking the client app's entitlements.
2. A check to determine if the app sending the XPC message is the same as the one that activated the NE feature.
3. A check of the XPC token.

These checks are performed in the XPC message handlers, after the connection is made, and before any functional work is done. All message handlers perform check 1, but not all handlers perform checks 2 and 3.

On a different organizational level (XPC), the connection between the client and the service is validated by using the client's audit token. This prevents other programs from hijacking the connection and masquerading as a legitimate app.

See the implementation of

```
- (void) _xpcConnectionAccept:(xpc_connection_t) inCnx
```

for more details.

Entitlements

Entitlements are Apple's core instrument for process authorization and authentication. They are embedded in a process' code signature, and can be queried by the kernel or other processes. On non-jailbroken iPhones, where the Apple App Store is the only way to run third-party software, access to entitlements is controlled by Apple – Apps with entitlements that they are not allowed to possess are not allowed in the App Store.

Access to this XPC service provided by `ExposureNotificationService` is controlled on two levels: * The iOS kernel sandbox, `Sandbox.kext`, contains a sandbox profile `DataActivation.sb`, which requires apps to have the `com.apple.developer.exposure-notification` entitlement to look up the mach port associated with the XPC service. It does this by leveraging kernel hooks and the TrustedBSD framework. The following excerpt from `DataActivation.sb`, recovered from an iOS 15 kernelcache using a modified version of SandBlaster shows part of the relevant rules: `scheme (require-all (global-name "com.apple.ExposureNotification") (require-entitlement "com.apple.developer.exposure-notification"))` This coarse-grained access control is limited to processes constrained by this sandbox profile. This seems to be the case for iOS apps, but not for iOS system processes. * Fine-grained access control is performed by the `ExposureNotificationDaemon` in its `[ENXPCConnection _xpcConnectionRequest:]` implementation. After retrieving the message type `mTyp` from the message dictionary, the corresponding handler is invoked and is responsible for authorizing the client using its entitlements.

The implementation defines four levels of access, in increasing order:

Access Level	Name	Entitlements Required
1	`ENAccessLevelNone`	None
2	`ENAccessLevelPublic`	`com.apple.developer.exposure-notification`
3	`ENAccessLevelPublicTest`	`com.apple.developer.exposure-notification-test`
4	`ENAccessLevelPrivate`	`com.apple.private.exposure-notification`

Higher access levels include the permissions of lower access levels, i.e. an XPC client that has the `ENAccessLevelPrivate` access level may perform any actions that require `ENAccessLevelPublicTest`.

The official national/regional app holds the `com.apple.exposure-notification` entitlement. For testing purposes during development, app developers are also provided with a profile for the `com.apple.developer.exposure-notification-test` entitlement; however according to Apple, it is “not allowed in the App Store”.

A number of additional entitlements are used to secure special-purpose interfaces and permissions: * `com.apple.developer.exposure-notification-test-skip-file-verification` In order to determine exposure, the device will download diagnosis keys from the key server. These are then processed and used to match against received RPI’s from other devices. These diagnosis keys are digitally signed on the key server. After downloading, the `ENDaemon` verifies the signature. When a client app has this entitlement set, `ENDaemon` will skip verification of the digital signature.

- `com.apple.developer.exposure-notification-logging` If this on the client app, the `ENDaemon` logging system is notified that debug information may be logged. It depends on the settings of ENDaemon if this information is actually logged.

- `com.apple.private.exposure-notification-bypass-key-release-prompt` When a user tests positive for COVID-19 the regional PHA can collect the key to share with a central server of positive keys. Under normal operating conditions, the user is prompted with a request for approval of transmission of the key to the server. When this entitlement is set on the client app, this request for approval is bypassed and the user is not presented with a request for approval.
- `com.apple.private.exposure-notification-test-verification` This entitlement is not used in the public part of the framework, but is used in undocumented parts of the framework. When this entitlement is set on the client app, it will be allowed to call `[ENXPCConnection _xpcStartTestVerificationSession:]` on ENDaemon. This appears be part of Exposure Notification Express.
- `com.apple.private.exposure-notification-show-buddy` This entitlement is not used in the public part of the framework, but is used in undocumented parts of the framework. When this entitlement is set on the client app, it will be allowed to call `[ENXPCConnection _xpcShowBuddy:]` This appears be part of Exposure Notification Express.
- `com.apple.private.security.storage.ExposureNotification`

Enumerating binaries on iOS posessing the above entitlements paints a picture of the different components involved in Exposure Notification. Jonathan Levin's OS X / iOS Entitlement Database is helpful for this, however, at the time of writing it only covered iOS up to 15.2.

The following Table contains a List of binaries containing the appropriate entitlements:

Name	1	2	3	4	5	6	7	8	9	10	11
`ExposureNotificationService`	X	X	X	X	X	X	X	X	X		
`DPSubmissionService`	X				X		X	X			
`HealthENLauncher`	X				X					X	
`ExposureNotificationRemoteViewService`	X	X			X						
`HealthENBuddy`	X				X						
`powerlogHelperd`					X						
`PerfPowerServicesExtended`					X						
`dasd`					X						
`aggregated`					X						
`imagent`					X						
`indexSettingsManifest`					X						
`Preferences`					X						
`bluetoothd`											X

Legend

1. `com.apple.developer.exposure-notification`
2. `com.apple.developer.exposure-notification-test`

3. `com.apple.developer.exposure-notification-logging`
4. `com.apple.developer.exposure-notification-test-skip-file-verification`
5. `com.apple.private.exposure-notification`
6. `com.apple.private.exposure-notification-bypass-key-release-prompt`
7. `com.apple.private.exposure-notification-device-identity`
8. `com.apple.private.exposure-notification-differential-privacy`
9. `com.apple.private.exposure-notification-show-buddy`
10. `com.apple.private.exposure-notification-test-verification`
11. `com.apple.private.security.storage.ExposureNotification`

Active app check

This check is implemented in

```
- (BOOL) _appActiveStatusWithError:(NSErrorOutType) outError
```

Its main purpose is to compare the signing identity of the connecting client app with the signing identity that is associated with the daemon when it is first activated.

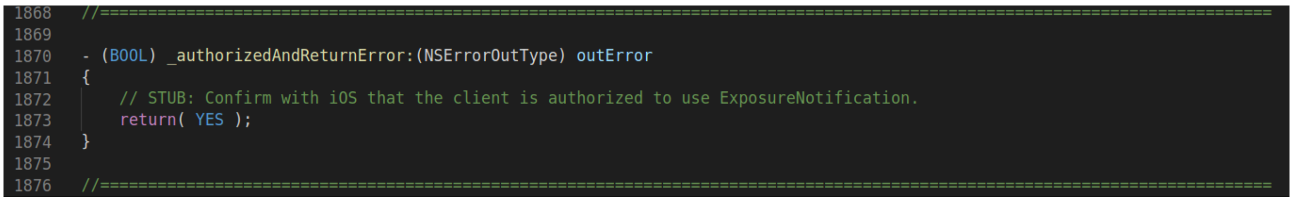
It's purpose seems to be to prevent other apps from interacting with the daemon after it has been activated with a client app.

XPC token check

This check is implemented in

```
- (BOOL) _authorizedAndReturnError:(NSErrorOutType) outError
```

However, this function is not implemented in the code.



```
//==========================================================================================================

- (BOOL) _authorizedAndReturnError:(NSErrorOutType) outError
{
    // STUB: Confirm with iOS that the client is authorized to use ExposureNotification.
    return( YES );
}

//==========================================================================================================
```

Analysis of this function in the binary revealed that the system checks whether the XPC token of the client is authorized to use the NE service.

Overview

The table below lists the messages that can be handled by the NE service. The client communicates with the service by sending these messages and processing the result. Each message is received by the service and then is passed on to the handler for processing. Each handler performs security checks before it continues to do its work.

To prevent any confusion, the `public` access level does not mean that any app can use the function; It means that an app that has an entitlement that gives it `public` access can use the function.

Message	handling function	access level	active app check	XPC token check
ENXPCMessageTypeManagerActivate	_xpcManagerActivate: inRequest	public	no	no
ENXPCMessageTypeSetEnabled	_xpcSetEnabled: inRequest	public	yes	yes
ENXPCMessageTypeEntitlementCheck	_xpcEntitlementCheck: inRequest	public	no	no
ENXPCMessageTypeGetUserTraveled	_xpcGetUserTraveled: inRequest	public	yes	yes
ENXPCMessageTypePreAuthorizeDiagnosisKeys	_xpcPreAuthorizeDiagnosisKeys: inRequest	public	no	no
ENXPCMessageTypeRequestDiagnosisKeys	_xpcRequestPreAuthorizedDiagnosisKeys: inRequest	public	yes	no
ENXPCMessageTypeGetTravelStatusEnabled	_xpcGetTravelStatusEnabled: inRequest	private	no	yes
ENXPCMessageTypeSetTravelStatusEnabled	_xpcSetTravelStatusEnabled: inRequest	private	no	yes
ENXPCMessageTypeGetDiagnosisKeys	_xpcGetDiagnosisKeys:inRequest testMode:NO	public	yes	yes
ENXPCMessageTypeGetTestDiagnosisKeys	_xpcGetDiagnosisKeys:inRequest testMode:YES	public test	yes	yes
ENXPCMessageTypeResetData	_xpcResetData:inRequest	private	no	yes
ENXPCMessageTypeExposureDetection FileActivate	_xpcExposureDetectionFileActivate: inRequest	public	yes	yes
ENXPCMessageTypeExposureDetection FileAdd	_xpcExposureDetectionFileAdd: inRequest	public	no	no
ENXPCMessageTypeExposureDetection FileFinish	_xpcExposureDetectionFileFinish:inRequest	public	no	no
ENXPCMessageTypeExposureDetection FileGetExposures	_xpcExposureDetectionFileGetExposures: inRequest	public	no	no
ENXPCMessageTypeExposureDetectionFile GetExposureWindows	_xpcExposureDetectionFileGetExposureWin inRequest	public	no	no
ENXPCMessageTypeGetRemotePresentation Request	_xpcGetRemotePresentation RequestIfNeeded:inRequest	public	no	yes
ENXPCMessageTypeRemotePresentation Decision	_xpcRemotePresentationReceived Decision:inRequest	private	yes	no

Legend

`Message`: The message type sent to the EN XPC service

`Handling function`: The function within the service that handles the message

`access level`: The access level the handler requires for it to perform its function. The client app's access level is determined by its entitlements.

`active app check`: If the handler performs the active app check

`xpc token check`: If the handler performs the xpc token check.

Analysis

Of the three security controls, the `access level` control, determined by the app's entitlements, is the basis of the security architecture surrounding access to service endpoints. In order to get access to the handler's functionality, an adversary would need to have an app with the necessary entitlement, or to exploit a program that has the entitlement.

Since not all handlers perform the `active app check`, technically, an adversary having access to an app with `access level` public, could gain access to the handler's functionality.

The last access control layer is on the XPC level, and authorizes whether a connection is allowed to use the service.

It is unclear why these controls are not implemented for functionality where it would not hinder correct operation (for instance initialization). Nonetheless, we believe these controls provide a sufficient degree of protection.

Extra endpoints

Apart from the in the source code documented endpoints to the NE service, we found a set of endpoints that are not documented in code. We believe that these endpoints belong to the ENX addition that has been shipped with iOS since version 13.7

handling function	access level
xpcStartTestVerificationSession	private
xpcFetchTestVerificationMetadata	private
xpcFinishTestVerificationSession	private
xpcStartSelfReportWebSession	none
xpcGetStatusForBundleIdentifier	private
xpcGetInfo	private
xpcSetActiveApp	private
xpcGetLastExposureNotification	private
xpcDownload	private
xpcSetActiveRegion	private
xpcSetAutomaticRegionSwitch	private
xpcOnboardingDidStart	private
xpcGetPreAuthorizeDiagnosisKeysEnabled	private
xpcSetPreAuthorizeDiagnosisKeysEnabled	private
xpcGetRemotePresentationRequestIfNeeded	public
xpcRemotePresentationReceivedDecision	private
xpcGetDataVaultSize	private
xpcSetAvailabilityAlertEnabled	private
xpcSetMonthlySummaryAlertEnabled	private
xpcShowBuddy	none
xpcVerifyTextMessage	private

xpcLegalConsentPageCount	none

Legend

`handling function` The function handling the XPC message
`access level` The access level required to use the function. This is determined by the app's entitlement

Analysis

The functions have been analysed for their security properties. What stands out that unlike the API, which requires mostly public access levels, is that the presumed ENX functions require predominantly private access levels. We believe that since ENX is a feature that does not require a third party client app to function, the public access level is largely unnecessary.

Another thing that stands out is that unlike the API function handlers, the extra security controls `active app check` and `xpc token check` have been omitted. The ENX part of the EN service relies entirely on the private entitlement instead. The omission of these controls indicates that apps with the private access level are inherently trusted, underlining the notion that the security model is primarily geared towards protecting against rogue apps.

Lastly, there are three API calls that require no access level at all.

The `xpcStartSelfReportWebSession` allows for starting a self-report session, if the region the user is currently in allows for this feature. It's not clear why there are no security controls in place. A possible reason is to remove a barrier for other apps/browsers to initiate this process. Based on static analysis, there appear to be no security problems.

Attack surfaces

bluetoothd

The bluetooth daemon `bluetoothd` handles everything bluetooth-related on iOS. Apart from providing an XPC service for apps using bluetooth connections, it provides a number of bluetooth protocol implementations, such as A2DP and AACP for audio, Magic Pairing, support for AirPods, HID Gamepads, and other accessories. It therefore possesses a significant external attack surface, both over-the-air and through malicious third-party apps.

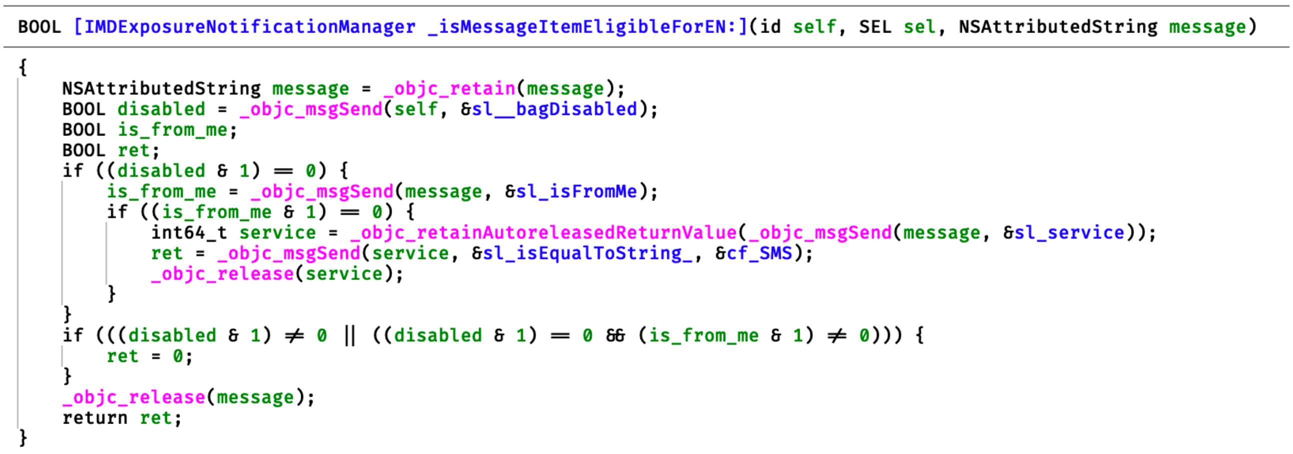
In the context of ExposureNotification, `bluetoothd` handles a number of core tasks:

- It implements the cryptographic protocols defined in the Google/Apple Exposure Notification Specification, and holds the sensitive key material needed to do so. This entails deriving the RPIs and AEMK from the TEKs, and encrypts the AEM.
- It periodically broadcasts the Bluetooth Low-Energy beacons containing the RPI and the AEM, rotating the bluetooth MAC address together with the RPIs.
- It scans for Exposure Notification Bluetooth Low-Energy beacons broadcast by other devices, and stores them in an SQLite database.

imagent

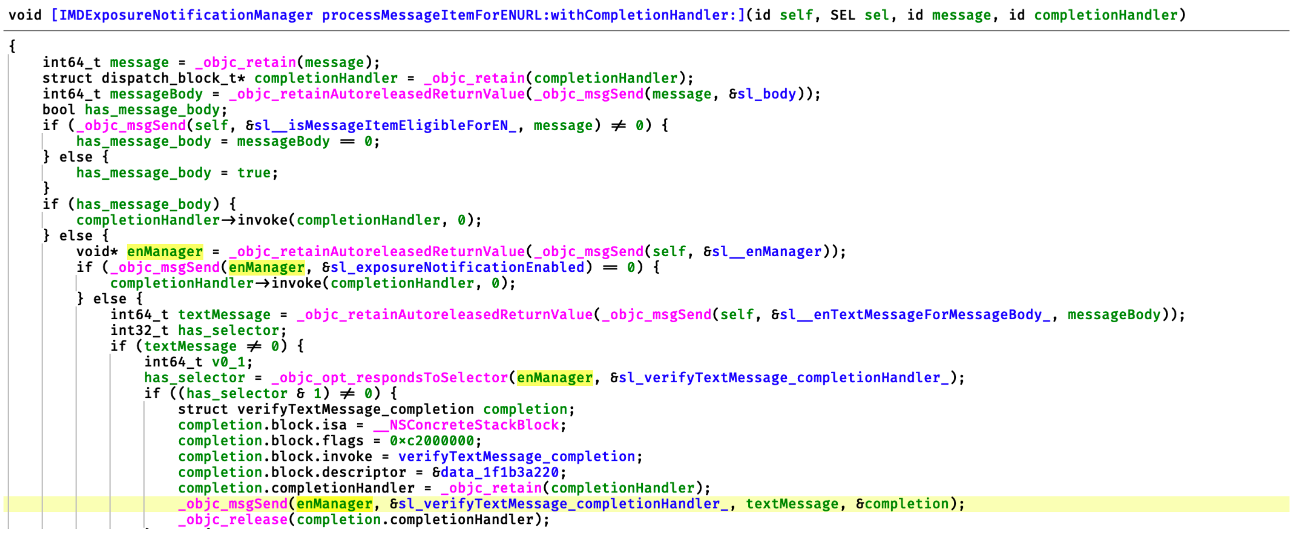
The imagent daemon is responsible for handling instant messaging. Large parts of the functionality are implemented in the `IMDaemonCore` framework (part of the `dyld_shared_cache`).

`IMagentCore` contains functionality to scan SMS messages for URLs:



```
BOOL [IMDExposureNotificationManager _isMessageItemEligibleForEN:](id self, SEL sel, NSAttributedString message)
{
    NSAttributedString message = _objc_retain(message);
    BOOL disabled = _objc_msgSend(self, &sl__bagDisabled);
    BOOL is_from_me;
    BOOL ret;
    if ((disabled & 1) == 0) {
        is_from_me = _objc_msgSend(message, &sl_isFromMe);
        if ((is_from_me & 1) == 0) {
            int64_t service = _objc_retainAutoreleasedReturnValue(_objc_msgSend(message, &sl_service));
            ret = _objc_msgSend(service, &sl_isEqualToString_, &cf_SMS);
            _objc_release(service);
        }
    }
    if (((disabled & 1) ≠ 0 || ((disabled & 1) == 0 && (is_from_me & 1) ≠ 0))) {
        ret = 0;
    }
    _objc_release(message);
    return ret;
}
```

Within these, it scans for URLs specified by the `ExposureNotificationDaemon`, and passes them back to the `ExposureNotificationDaemon` via XPC through the `enManager`.



```
void [IMDExposureNotificationManager processMessageItemForENURL:withCompletionHandler:](id self, SEL sel, id message, id completionHandler)
{
    int64_t message = _objc_retain(message);
    struct dispatch_block_t* completionHandler = _objc_retain(completionHandler);
    int64_t messageBody = _objc_retainAutoreleasedReturnValue(_objc_msgSend(message, &sl_body));
    bool has_message_body;
    if (_objc_msgSend(self, &sl__isMessageItemEligibleForEN_, message) ≠ 0) {
        has_message_body = messageBody == 0;
    } else {
        has_message_body = true;
    }
    if (has_message_body) {
        completionHandler→invoke(completionHandler, 0);
    } else {
        void* enManager = _objc_retainAutoreleasedReturnValue(_objc_msgSend(self, &sl__enManager));
        if (_objc_msgSend(enManager, &sl_exposureNotificationEnabled) == 0) {
            completionHandler→invoke(completionHandler, 0);
        } else {
            int64_t textMessage = _objc_retainAutoreleasedReturnValue(_objc_msgSend(self, &sl__enTextMessageForMessageBody_, messageBody));
            int32_t has_selector;
            if (textMessage ≠ 0) {
                int64_t v0_1;
                has_selector = _objc_opt_respondsToSelector(enManager, &sl_verifyTextMessage_completionHandler_);
                if ((has_selector & 1) ≠ 0) {
                    struct verifyTextMessage_completion completion;
                    completion.block.isa = __NSConcreteStackBlock;
                    completion.block.flags = 0×c2000000;
                    completion.block.invoke = verifyTextMessage_completion;
                    completion.block.descriptor = &data_1f1b3a220;
                    completion.completionHandler = _objc_retain(completionHandler);
                    _objc_msgSend(enManager, &sl_verifyTextMessage_completionHandler_, textMessage, &completion);
                    _objc_release(completion.completionHandler);
```

The list of allowed URLs is retrieved from Apple's servers at `https://init.ess.apple.com/WebObjects/VCInit.woa/wa/getBag?ix=3`. This contains a base64-encoded and signed FaceTime configuration, which includes the exposure notification configuration:

```
(page 17)  <key>en-push-allow-domains</key>
(page 17)  <array>
(page 17)    <string>en.express</string>
(page 17)    <string>enexpress.app</string>
(page 17)  </array>
(page 17)  <key>en-push-disabled</key><false/>
```

On the `ExposureNotificationDaemon` side, the corresponding text XPC handler requires the `ENAccessLevelPrivate` access level, and hence the `com.apple.private.exposure-notification` entitlement, granting the highest level of privileges to a number of unrelated XPC calls. This does not conform to the principle of least privilege; `imagent` holds a number of privileges it does not need through this entitlement.

Further parsing of the SMS message happens within the `ExposureNotificationDaemon` framework – and hence within `bluetoothd` – in `[ENTextMessageManager verifyTextMessage:phoneNumber:verificationDate:publicKey:publicKeyVersion:userReport:outError:]`. This opens up external attack surface to the `ExposureNotificationDaemon` via SMS.

Using SEEMOO-Lab's ARIstoteles, we were able to inject messages containing URLs to allowed domains and observe them arriving in `bluetoothd` in the `-[ENXPCConnection _xpcVerifyTextMessage:]` method.

These code paths are not part of Apple's published Exposure Notification source code, and therefore are unlikely to have faced much public scrutiny. Further security research could focus on the parsing logic of `IMDaemonCore` and the parsing and verification logic in `ExposureNotificationDaemon`, e.g. by fuzzing these code paths.

Other components

Dasd

Dasd is the Duet Activity Scheduler Daemon. It will repeatedly launch the currently active Exposure Notification App to allow it to perform background processing. Other apps are not allowed to do this.

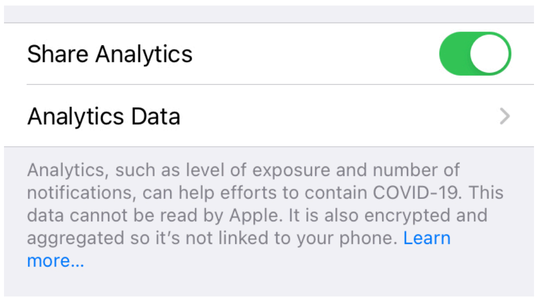
This is only performed when exposure notifications are enabled. To determine whether exposure notifications are enabled, and to query the currently active app, `dasd` connects to the XPC service offered by `bluetoothd`, `com.apple.ExposureNotification`. It holds the `com.apple.private.exposure-notification` entitlement, and therefore has a very high privilege level when connecting to the service.

HealthENBuddy and HealthENLauncher

These components seem to be related to ENX. Since this feature does not require an external app to function, it does need to provide some means of interaction to the user. HealthENBuddy and launcher are presumed to be the means of interaction. We have not been able to fully reverse engineer these components since they are written in swift and our tooling was not suited to decompile swift binaries.

Differential Privacy (DP)

Apple uses differential privacy to collect telemetry from devices. The User must opt in to sharing analytics in the settings menu for this to be active:


Share Analytics

Analytics Data

Analytics, such as level of exposure and number of notifications, can help efforts to contain COVID-19. This data cannot be read by Apple. It is also encrypted and aggregated so it's not linked to your phone. Learn more...

Google and Apple's claims about their differential privacy construction can be found in their white paper at https://covid19-static.cdn-apple.com/applications/covid19/current/static/contact-tracing/pdf/ENPA_White_Paper.pdf. The cryptographic properties of the analytics collection were not the subject of this review. Previous reviews of Apple's differential privacy implementation have found shortcomings, however it is unclear whether these shortcomings are still present and whether and how they affect data collected from Exposure Notifications.

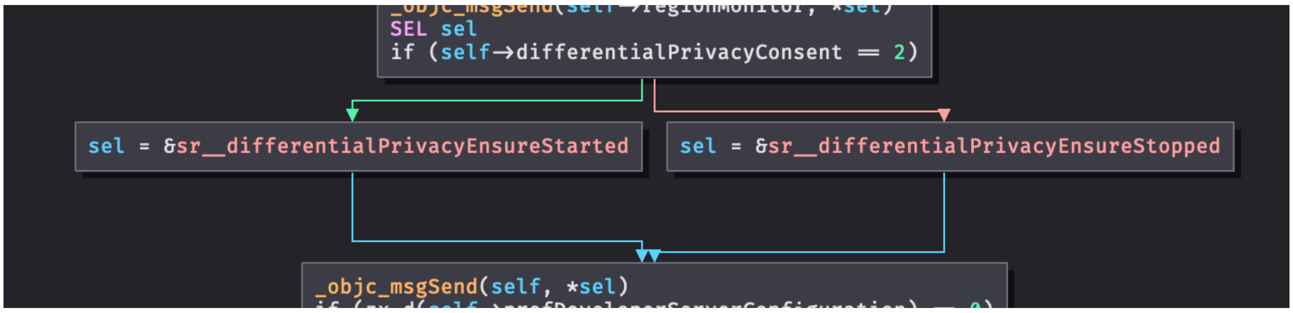
It’s important to note that Google and Apple’s implementations of differential privacy rely on multiparty computation where the parties may not collaborate to compromise privacy. In the US, Google and Apple provide anonymization of data from each of their respective devices, ISRG and NIH aggregate the anonymized data, and MITRE reconstructs the aggregate data and displays the statistics. Privacy of the collected data is to a large extent contingent on these parties not collaborating to unmask individual users.

The differential privacy implementation and telemetry collection code paths are not part of Apple’s source code release for Exposure Notifications. End users can therefore not – without significant reverse engineering effort – verify Apple’s claims about the privacy properties of the system.

Analytics are aggregated in `bluetoothd` in an `ENDifferentialPrivacyManager` instance. For analytics to be collected, the region’s server configuration needs to opt in to analytics, and the user needs to grant explicit consent.

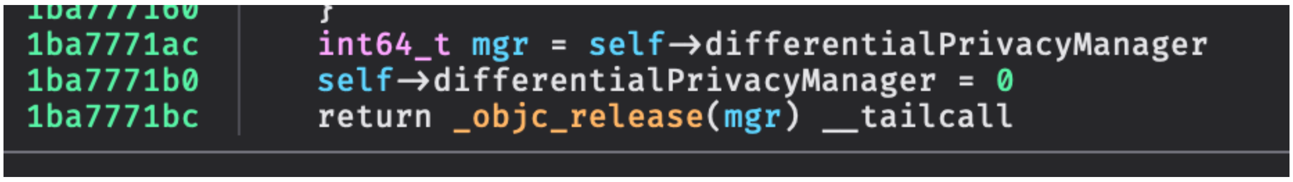
Through static reverse engineering, we verified that analytics are not collected or submitted to Google and Apple’s ingestion servers when either of these conditions is not met.

Within the `[ENDaemon _update]` function, there is a code path that starts or stops the differential privacy collection when consent is not given (indicated by value 2).



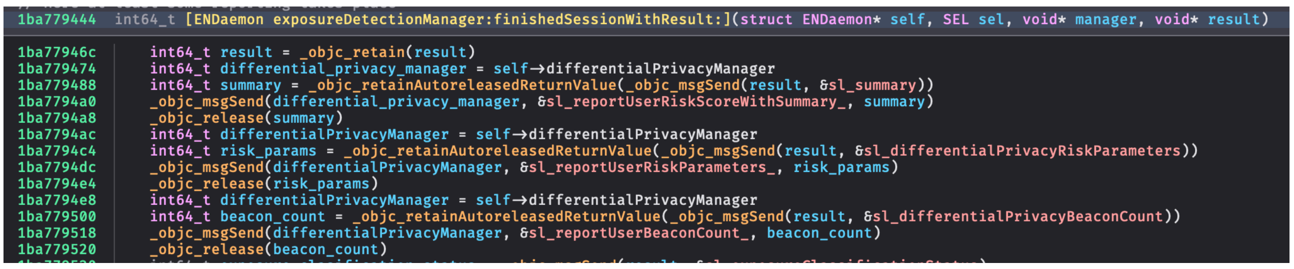
```
SEL sel
if (self->differentialPrivacyConsent == 2)

sel = &sr__differentialPrivacyEnsureStarted
sel = &sr__differentialPrivacyEnsureStopped

_objc_msgSend(self, *sel)
```

When `[ENDaemon differentialPrivacyEnsureStopped]` is called, this sets the `ENDaemon`’s differential privacy manager to 0, and decreases its reference count. This ensures that future attempts to store telemetry on this `ENDifferentialPrivacyManager` will not succeed. We did not, however, find any indication that withdrawal of consent deletes already collected, but not-yet uploaded data.



```
1ba7771ac    int64_t mgr = self->differentialPrivacyManager
1ba7771b0    self->differentialPrivacyManager = 0
1ba7771bc    return _objc_release(mgr) __tailcall
```

Metrics are stored on the `ENDifferentialPrivacyManager` object. For example, on finishing an exposure detection session, the number of beacons, the user’s risk score, the parameters that make up that risk score, and the number of beacons seen are reported to the `ENDifferentialPrivacyManager` object.



```
1ba779444  int64_t [ENDaemon exposureDetectionManager:finishedSessionWithResult:](struct ENDaemon* self, SEL sel, void* manager, void* result)

1ba77946c      int64_t result = _objc_retain(result)
1ba779474      int64_t differential_privacy_manager = self->differentialPrivacyManager
1ba779488      int64_t summary = _objc_retainAutoreleasedReturnValue(_objc_msgSend(result, &sl_summary))
1ba7794a0      _objc_msgSend(differential_privacy_manager, &sl_reportUserRiskScoreWithSummary_, summary)
1ba7794a8      _objc_release(summary)
1ba7794ac      int64_t differentialPrivacyManager = self->differentialPrivacyManager
1ba7794c4      int64_t risk_params = _objc_retainAutoreleasedReturnValue(_objc_msgSend(result, &sl_differentialPrivacyRiskParameters))
1ba7794dc      _objc_msgSend(differentialPrivacyManager, &sl_reportUserRiskParameters_, risk_params)
1ba7794e4      _objc_release(risk_params)
1ba7794e8      int64_t differentialPrivacyManager = self->differentialPrivacyManager
1ba779500      int64_t beacon_count = _objc_retainAutoreleasedReturnValue(_objc_msgSend(result, &sl_differentialPrivacyBeaconCount))
1ba779518      _objc_msgSend(differentialPrivacyManager, &sl_reportUserBeaconCount_, beacon_count)
1ba779520      _objc_release(beacon_count)
```

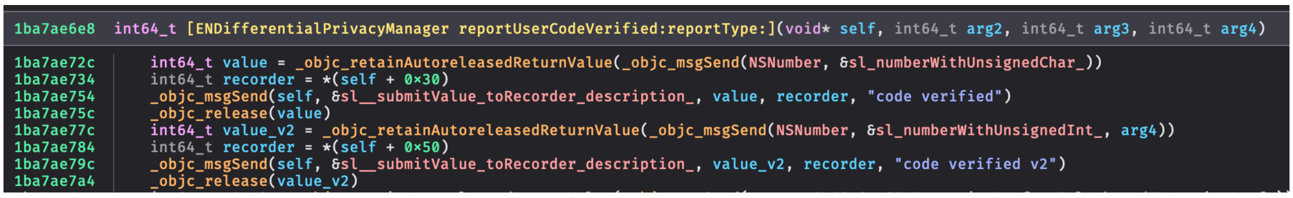
In Objective-C, it is a no-op to send messages to the `nil` receiver when the differential privacy manager is `nil`, meaning that the metrics are not collected.

The `DifferentialPrivacyManager` passes the metrics to the `DPSubmissionService`, which uploads them to the analytics endpoints configured in the `ExposureNotification` configuration.

The following metrics are collected:

- User risk score and user risk parameters (`com.apple.EN.UserRisk`, `com.apple.EN.UserRiskParameters`). The risk score contains the daily maximum score, sum over scores, and a weighted sum over durations for multiple days. The user risk parameters are calculated from the attenuation and duration map over exposure windows.
- User beacon count (`com.apple.EN.BeaconCount`) - the number of beacons collected.
- User exposure notification and user exposure notification v2 (`com.apple.EN.UserNotification`, `com.apple.EN.UserNotificationV2D14`) The exposure classification is recorded when an exposure notification is posted.
- User exposure notification interaction (`com.apple.EN.UserNotificationInteraction`): The exposure classification is recorded when a user notification is tapped.
- Code verified and code verified v2 (`com.apple.EN.CodeVerified`, `com.apple.EN.CodeVerifiedWithReportTypeV2D14`) The latest exposure information and report type is recorded when a test is verified. This may be the case only when Exposure Notification Express is enabled.
- Secondary attack (`com.apple.EN.SecondaryAttackV2D14`) – The value of this metric is yet unclear. Whether it is reported depends on data from the latest exposure.
- User diagnosed vaccine status and user diagnosed vaccine status v2 (`com.apple.EN.DiagnosedVaccineStatus`, `com.apple.EN.DiagnosedVaccineStatusV2D14`) is collected when diagnosis keys are released. It encodes whether a user is vaccinated, and whether they have had a non-expired exposure registered.
- Keys uploaded and keys uploaded v2 (`com.apple.EN.KeysUploaded`, `com.apple.EN.KeysUploadedWithReportTypeV2D14`)
- Delay between exposure and notification (`com.apple.EN.DateExposure`, `com.apple.EN.DateExposureV2D14`) This is recorded in bins of under 4 days, under 7 days, and under 11 days when a user notification is posted.

The v2 versions of many metrics may be the same value, but encoded in a number type that can handle larger values, or a value containing additional information such as the report type. It's unclear how submitting the same value twice affects the differential privacy properties. Depending on the implementation, this may need to be adjusted for in the appropriate privacy budgets.



```
1ba7ae6e8  int64_t [ENDifferentialPrivacyManager reportUserCodeVerified:reportType:](void* self, int64_t arg2, int64_t arg3, int64_t arg4)

1ba7ae72c      int64_t value = _objc_retainAutoreleasedReturnValue(_objc_msgSend(NSNumber, &sl_numberWithUnsignedChar_))
1ba7ae734      int64_t recorder = *(self + 0x30)
1ba7ae754      _objc_msgSend(self, &sl__submitValue_toRecorder_description_, value, recorder, "code verified")
1ba7ae75c      _objc_release(value)
1ba7ae77c      int64_t value_v2 = _objc_retainAutoreleasedReturnValue(_objc_msgSend(NSNumber, &sl_numberWithUnsignedInt_, arg4))
1ba7ae784      int64_t recorder = *(self + 0x50)
1ba7ae79c      _objc_msgSend(self, &sl__submitValue_toRecorder_description_, value_v2, recorder, "code verified v2")
1ba7ae7a4      _objc_release(value_v2)
```

Configuration

Apart from the configuration of the app itself, EN uses a configuration per region. This configuration is fetched every time the Daemon's preferences change, and on a regular time interval, and is ultimately retrieved from Apple's server at `https://gateway.icloud.com/enservice`. The configuration items include, among other properties:

- healthAuthorityId
- region (country code / state code)
- tekLocalDownloadBaseURL (The URL diagnosis keys are downloaded from)
- tekUploadURL (the URL exposure keys should be uploaded to)
- If text message verification is enabled (default `no`). If this parameter is `no`, no text verification will take place. This parameter is tied to the text message processing by `imagent`.
- If test verification is enabled (default `no`)
- Public key information

A sample configuration pulled from the mentioned URL is provided in the appendix.

Apart from the above items, there are a set of items that control the behaviour of the Exposure Notification Framework when a user moves from one region to another, other parameters to fine-tune the behaviour of the ENF, and parameters to control the Differential Privacy feature of the ENF.

These configurations are set per region, and regions may correspond to nations or states.

It is presumed these configuration parameters, in combination with the DP feature, are used as a control and feedback loop that adjusts the operation of the system in real-time.

This places a great deal of control in the hands of whomever controls the configuration endpoint. A successful attack on Apple's DNS server together with a compromise of their certificate authority could place control of the ENF in the hands of an adversary.

The configuration file is stored in `/var/mobile/Library/ExposureNotification/Regions/Configurations/Server/{regioncode}.json`.

Since configuration takes place in `EnConfigurationStore`, which is a part of `ENDaemon`, it runs in the system’s sandbox. An adversary, having broken the sandbox, may have access to the file and the ability to change its contents, causing the app to malfunction.

Updating of the configuration

Apple provides a service to update the EN configuration through a service this is documented here: `https://developer.apple.com/documentation/exposurenotification/changing_configuration_values_using_the_server_to_server_api?language=objc`.

This allows a PHA to update the configuration for the region it has authority over by issuing a POST request on the documented endpoint. This request has to be signed with a private key of which its public counterpart is known to Apple.

This system allows for server-to-server communication to automate changes under certain conditions.

Assuming this security control works as intended, only the entity in possession of the private key is able to make changes to region’s configuration.

Great care must be taken to protect this private key from breach of confidentiality. If an adversary obtains this private key, and no other security controls are in place, they may be able to alter the configuration of the EN feature. The extent to which configuration items may be changed is not clear, and is beyond the scope of this assignment. The point is that if the private key falls into unauthorized hands, they may be able to control the behaviour of the EN-enabled devices present in the region.

Transport security

The EN feature communicates with the configuration server using TLS. The server supports TLSv1.3 and TLSv1.2 which are regarded as secure versions of TLS. Though the preferred cipher suites for TLSv1.2 and TLSv1.3 are regarded as safe, TLSv1.2 offers support for weak CBC based cipher suites.

The security risk for EN regarding transport security is considered low. The region configuration could be considered public as there is no authentication control in place. Regarding the updating of the information, this is protected by both TLS and asymmetric cryptography, minimizing the risk of tampering by an adversary.

Furthermore, Apple’s certificates are pinned on the device. Attempts at intercepting network traffic by adding a trusted wildcard certificate to the system trust store were not successful.

Databases

EN uses two databases to store exposure related information.:

1. An advertisements database (ENAdvertisementSQLiteStore), where information of other devices is stored. To circumvent a technical issue, there is a central store, and temporary stores that are merged into the central store periodically.
2. An exposure database (ENExposureDatabase). In this database matching advertisements, are copied into the "advertisements" table. This database also holds Temporary Exposure Keys (TEKs) and other metadata.

The technology used to store the databases holding above tables is SQLite. This is a database format that is stored in a single file on the filesystem and has no authentication / authorization system. Instead, it relies on the permissions of the filesystem for authorization and higher-level systems for authentication. The contents of these files are not encrypted.

The central advertisements store is in `/var/mobile/Library/ExposureNotification/Advertisements/en_advertisements.db`

Temporary advertisements stores are stored using the following naming convention: `/var/mobile/Library/ExposureNotification/Advertisements/tmp_en_advertisements_{uuid}.db`

The exposure advertisements database is in `/var/mobile/Library/ExposureNotification/Exposure/en_exposure.sqlite`

The logic controlling the database runs as part of `ENDaemon` and as such runs in the system sandbox, protecting it from unauthorized access.

Backup and Restore

Exposure notification data is included in standard iPhone backups. However, the cryptographic key material, i.e. the TEK and the observed beacons, are not saved. Only previously downloaded diagnosis key caches are retained.

On an iPhone which had exposure notifications enabled in Germany, and the AL, CO and HI states in the US, the following (abridged) list of files was found in the backup:

```
Library/ExposureNotification
Library/ExposureNotification/Downloads
Library/ExposureNotification/Downloads/B0566516-9BE0-4675-8BB9-D44D399DF278
Library/ExposureNotification/Downloads/B0566516-9BE0-4675-8BB9-D44D399DF278/Downloads
Library/ExposureNotification/Downloads/B0566516-9BE0-4675-8BB9-D44D399DF278/index.txt
Library/ExposureNotification/Downloads/DD076728-7F45-4DBA-9CAF-0DA004B0D881
Library/ExposureNotification/Downloads/DD076728-7F45-4DBA-9CAF-0DA004B0D881/Downloads
Library/ExposureNotification/Downloads/DD076728-7F45-4DBA-9CAF-0DA004B0D881/index.txt
Library/ExposureNotification/Downloads/state.dat
Library/ExposureNotification/Regions
Library/ExposureNotification/Regions/Configurations
Library/ExposureNotification/Regions/Configurations/Server
Library/ExposureNotification/Regions/Configurations/Server/de.json
Library/ExposureNotification/Regions/Configurations/Server/us-ak.json
...
Library/ExposureNotification/Regions/Configurations/Server/us-wy.json
Library/ExposureNotification/Regions/Configurations/System
Library/ExposureNotification/Regions/Configurations/System/de.data
Library/ExposureNotification/Regions/Configurations/System/us-al.data
Library/ExposureNotification/Regions/Configurations/System/us-co.data
Library/ExposureNotification/Regions/Configurations/System/us-hi.data
Library/ExposureNotification/Regions/Subdivisions
Library/ExposureNotification/Regions/Subdivisions/Server
Library/ExposureNotification/Regions/Subdivisions/Server/de.plist
Library/ExposureNotification/Regions/Subdivisions/Server/us.plist
Library/ExposureNotification/VerificationHash
Library/ExposureNotification/VerificationHash/verificationHashes.dat
Library/Preferences/com.apple.ExposureNotification.plist
```

These databases were not part of the backup.

Manipulating this backup to change preferences or manipulate the server configurations is possible, but exposure notifications are disabled on restore. The server configurations are re-downloaded when enabling Exposure Notifications. We were therefore not able to abuse this to e.g. change telemetry or TEK upload URLs.

5 Future Work

- **Commission a further audit of the Exposure Notification Express feature**
  This feature is undocumented in the source code; we stumbled upon it by chance. Though we examined the parts we were able to see, we recommend commissioning a full evaluation of this feature.

6 Conclusion

The iOS sandboxing model is focused on rogue apps, however, it is not sufficient to protect against vulnerabilities in Apple's own code. Large parts of the exposure notification framework's supporting services are implemented within iOS daemons which handle potential attacker-controlled input, such as `bluetoothd`, present large external attack surfaces, and have had a number of vulnerabilities identified in them in the past.

Compared with previously published source code, the source code published by Apple at the time of testing matches very well with parts of the implementations in iOS 15. However, large parts of the implementation have been omitted from the source code, including many APIs that are used internally between Apple services. The published source code suggests a much more limited functionality compared to the actual implementation. This is especially true for the Exposure Notifications Express mode.

Regarding user privacy, we have not found any clear issues where sensitive, personally identifying information could be obtained by an unauthorized third party. Though exposure telemetry is collected, this uses differential privacy. Some shortcomings of this system have been found by third parties. It is unclear whether these issues have been fixed for iOS, or what their impact on the EN framework might be.

Appendix 1 Sample Configuration

```
{
    "subdivisions": [],
    "appConfigs": [
        {
            "countryCode": "NL",
            "state": "",
            "appBundleId": "nl.rijksoverheid.en",
            "publicKey":
 "MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAExqcE4vibzfNTYyLDaMK4JVYI4PFsrsMiowu8vQpC9eB7uClAU4nJVKNeJ1AB9+M5dFTZib3ARZ
\u003d\u003d",
            "publicKeyVersion": "v15",
            "signAuthorityName": "signAuthorityName",
            "config": {
                "enVersion": 1,
                "agencyColor": [
                    0.72,
                    0.15,
                    0.36
                ],
                "reportTypeConfirmedClinicalDiagnosisWeight": 100,
                "isMatchingRestrictedRegion": false,
                "exposureDetailsBodyText_1_EN_US": "",
                "notificationSubject_3_EN_US": "",
                "hasStateRegions": false,
                "tekUploadURL": "",
                "agencyRegionName_DE_DE": "Die Niederlande",
                "attenuationNearMedThreshold": 50,
                "tekLocalDownloadBasePath": "",
                "testVerificationURL": "",
                "agencyRegionName_FR_FR": "Pays-Bas",
                "reportTypeConfirmedTestWeight": 100,
                "reportTypeNoneMap": 3,
                "agencyRegionName_AR_AE": "######",
                "agencyRegionName_TR_TR": "Hollanda",
                "agencyDisplayName_NL_NL": "Ministerie van Volksgezondheid",
                "attenuationImmediateWeight": 150,
                "exposureDetailsBodyText_4_EN_US": "",
                "regionDisabledTransitionGracePeriodMinutes": 1440,
                "perDayMaxERVThreshold_2": 0,
                "perDayMaxERVThreshold_3": 0,
                "perDayMaxERVThreshold_1": 0,
                "partnerTelemetryAppleCertificateChain": "",
                "infectiousnessHighWeight": 100,
                "perDayMaxERVThreshold_4": 0,
                "isTestRegion": false,
                "resetAvailabilityAlertForDeclinedUsers": false,
                "agencyVerificationAPIKey": "",
                "weightedDurationAtAttenuationThreshold_4": 0,
                "agencyHeaderStyle": 0,
                "agencyMessage_PL_PL": "CoronaMelder to oficjalna, holenderska aplikacja wysyłająca
 powiadomienia na temat koronawirusa, opracowana pod nadzorem holenderskiego Ministerstwa
 Zdrowia, Opieki Społecznej i Sportu. Więcej informacji można znaleźć na stronie internetowej
 coronamelder.nl",
                "weightedDurationAtAttenuationThreshold_1": 0,
                "weightedDurationAtAttenuationThreshold_3": 0,
                "publicHealthAuthorityTelemetryAppleCertificateChain": "",
                "weightedDurationAtAttenuationThreshold_2": 0,
                "applicationBackgroundRuntimeIntervalinHours": 2,
                "attenuationImmediateNearThreshold": 30,
                "attenuationNearWeight": 100,
                "flags": 0,
                "serverConfigVersion": 1,
                "agencyVerificationHeaderTitle": "",
                "V1Enable": true,
                "agencyVerificationCertificateURL": "",
                "dynamicAlgorithmEnabled": true,
                "agencyVerificationURL": "",
                "enEnabled": true,
                "notificationSubject_2_EN_US": "",
                "agencyMessage_BG_BG": "CoronaMelder е официалното приложение на Нидерландия
за известяване във връзка с коронавируса, разработено под надзора на Министерството на
здравеопазването, благосъстоянието и спорта. Повече информация ще откриеш на coronamelder.nl.",
                "dynamicThrottleUpAdvDensity": 20,
                "enablePreArmVerification": false,
                "legalConsentVersion": "",
                "agencyHeaderSubtitle_EN_US": "",
                "agencyRegionName_RO_RO": "Țările de Jos",
                "reportTypeSelfReportWeight": 100,
                "agencyMessage_NL_NL": "CoronaMelder is de officiële corona notificatie-app van
Nederland, ontwikkeld onder het toezicht van het Ministerie van Volksgezondheid, Welzijn en Sport.
Meer informatie vind je op coronamelder.nl.",
                "chaffPercent": 0.006,
                "agencyRegionName_BG_BG": "Нидерландия",
                "privacyParameterInputCandence": 7,
                "appleTelemetryEndpoint": "https://exposure-notification.apple.com",
                "chaffTotal": 1,
                "verificationCodeLearnMoreURL": "",
                "agencyDisplayName_AR_AE": "##### #####",
                "agencyDisplayName_PL_PL": "Holenderskie Ministerstwo Zdrowia",
                "attenuationMedWeight": 50,
                "clinicalDiagnosisPerDaySumERVThreshold_1": 0,
                "clinicalDiagnosisPerDaySumERVThreshold_2": 0,
                "notificationSubject_1_EN_US": "",
                "agencyDisplayName_TR_TR": "Sağlık Bakanlığı",
                "agencyRegionName_NL_NL": "Nederland",
                "clinicalDiagnosisPerDaySumERVThreshold_3": 0,
                "rpiAdvertisementToleranceInMinutes": 120,
                "clinicalDiagnosisPerDaySumERVThreshold_4": 0,
                "reportTypeRecursiveWeight": 100,
                "forceAPWakeIntervalInMinutesThreshold": 5,
                "symptomOnsetToInfectiousnessMap": 288605699003383808,
                "daysSinceExposureThreshold": 14,
                "classificationName_3": "",
                "agencyRegionName_PL_PL": "Holandia",
                "classificationName_2": "",
                "classificationName_4": "",
                "classificationName_1": "",
                "enableRecursiveType": false,
                "detectExposureNKDLimit": 6,
                "regionIdentifier": "",
                "rpiDuplicateAdvertisementToleranceInMinutes": 20,
                "agencyDisplayName_DE_DE": "Gesundheitsministerium",
                "agencyVerificationMessage": "",
                "agencyDisplayName_ES_ES": "Ministerio de Salud",
                "recursivePerDaySumERVThreshold_4": 0,
                "recursivePerDaySumERVThreshold_3": 0,
                "recursivePerDaySumERVThreshold_2": 0,
                "exposureDetailsBodyText_3_EN_US": "",
                "recursivePerDaySumERVThreshold_1": 0,
                "testVerificationAPIKey": "",
                "agencyMessage_FR_FR": "CoronaMelder est l\u0027application officielle de
 notification du coronavirus aux Pays-Bas développée sous la supervision du ministère de la Santé,
 du Bien-être et des Sports. Vous trouverez plus d\u0027informations sur coronamelder.nl.",
                "numberOfAdvSamplesForRPIThreshold": 1,
                "agencyMessage_AR_AE": "###### ## ####### ##### (CoronaMelder) ## ###### ####### ##
###### .######## ######### ##### ##### ##### ### ###### ## ##### ####### ###### ## ####### ######
### ######### ## ###### #####  coronamelder.nl.",
                "agencyMessage_TR_TR": "CoronaMelder, Sağlık, Refah ve Spor Bakanlığı gözetiminde
 geliştirilen, Hollanda\u0027daki resmi Korona bildirim uygulamasıdır. Daha fazla bilgiyi
 coronamelder.nl adresinde bulabilirsiniz.",
                "agencyDisplayName_EN_US": "Ministry of Health",
                "agencyHeaderTextColor": [
                    1,
                    1,
                    1
                ],
                "agencyMessage_RO_RO": "CoronaMelder este aplicația oficială de notificare pentru
 coronavirus în Țările de Jos, dezvoltată sub supravegherea Ministerului Sănătății Publice,
 Bunăstării și Sportului. Mai multe informații găsiți pe coronamelder.nl",
                "agencyWebsiteURL": "",
                "infectiousnessStandardWeight": 100,
                "agencyFAQWebsiteURL": "",
                "confirmedTestPerDaySumERVThreshold_1": 0,
                "testVerificationCertificateURL": "",
                "confirmedTestPerDaySumERVThreshold_4": 0,
                "confirmedTestPerDaySumERVThreshold_2": 0,
                "confirmedTestPerDaySumERVThreshold_3": 0,
                "osTriggeredAppRunTimeInSeconds": 300,
                "dynamicThrottleDownAdvDensity": 6,
                "agencyDisplayName_BG_BG": "Министерство на здравеопазването",
                "enableChaff": true,
                "tekLocalDownloadIndexFile": "",
                "regionTransitionGracePeriodInMinutes": 180,
                "agencyRegionName_EN_US": "The Netherlands",
                "healthAuthorityID": "",
                "selfReportPerDaySumERVThreshold_3": 0,
                "agencyMessage_ES_ES": "CoronaMelder es la app de notificación de coronavirus
 oficial de los Países Bajos desarrollada bajo la supervisión del Ministerio de Salud, Bienestar y
 Deportes de los Países Bajos. Para más información, visita coronamelder.nl.",
                "selfReportPerDaySumERVThreshold_4": 0,
                "agencyImage": "https://coronamelder.nl/img/coronamelder-rijksoverheid-logo.png?v
\u003d2",
                "selfReportPerDaySumERVThreshold_1": 0,
                "selfReportPerDaySumERVThreshold_2": 0,
                "dynamicThrottleDownRSSI": -55,
                "enableAssociatedDomains": true,
                "tekPublishInterval": 24,
                "notificationSubject_4_EN_US": "",
                "perDaySumERVThreshold_2": 0,
                "callbackIntervalInMin": 1440,
                "exposureDetailsBodyText_2_EN_US": "",
                "perDaySumERVThreshold_1": 0,
                "perDaySumERVThreshold_4": 0,
                "phaPackageName": "",
                "perDaySumERVThreshold_3": 0,
                "dynamicThrottleUpDurationInMinutes": 900,
                "agencyMessage_DE_DE": "CoronaMelder ist die offizielle Corona-Benachrichtigungs-
App der Niederlande, die unter der Aufsicht des Ministeriums für Gesundheit, Gemeinwohl und Sport
 entwickelt wurde. Weitere Informationen finden Sie unter coronamelder.nl.",
                "classificationURL_2": "",
                "classificationURL_3": "",
                "classificationURL_4": "",
                "agencyMessage_EN_US": "CoronaMelder is the official Dutch coronavirus notification
 app, developed under supervision by the Ministry of Health, Welfare and Sport. Find more
 information on coronamelder.nl.",
                "dynamicThrottleDownDurationInSeconds": 216,
                "attenuationMedFarThreshold": 60,
                "exposureMatching": false,
                "stateRegionConfigs": {},
                "classificationURL_1": "",
                "agencyDisplayName_FR_FR": "Ministère de la Santé",
                "agencyRegionName_ES_ES": "Países Bajos",
                "attenuationOtherWeight": 0,
                "detectExposureDailyLimit": 20,
                "agencyDisplayName_RO_RO": "Ministerul Sănătății Publice",
                "chaffPercentAlt": 0.5
            }
        }
    ]
}
```

Appendix 2 Data model

The datamodel used by EN is as follows:

ENAdvertisementSQLiteStore:

```
rpi BLOB,
encrypted_aem BLOB
timestamp INTEGER
scan_interval INTEGER
rssi INTEGER
max_rssi INTEGER
saturated BOOLEAN
counter INTEGER
PRIMARY KEY(rpi, timestamp)
```

ENExposureDatabase:

```
@"CREATE TABLE teks ("
            @"ROWID INTEGER PRIMARY KEY AUTOINCREMENT, "
            @"region_id TEXT, "
            @"app_bundle_id TEXT, "
            @"key BLOB NOT NULL UNIQUE, "
            @"start INTEGER NOT NULL, "
            @"period INTEGER NOT NULL, "
            @"end INTEGER NOT NULL, "
            @"onset_days INTEGER NOT NULL, "
            @"report_type INTEGER NOT NULL, "
            @"original_report_type INTEGER, "
            @"transmission_risk INTEGER NOT NULL)",
        @"CREATE TABLE advertisements ("
            @"rpi BLOB NOT NULL, "
            @"encrypted_aem BLOB NOT NULL, "
            @"timestamp INTEGER NOT NULL, "
            @"scan_interval INTEGER NOT NULL, "
            @"rssi INTEGER NOT NULL, "
            @"max_rssi INTEGER NOT NULL, "
            @"saturated INTEGER NOT NULL, "
            @"counter INTEGER NOT NULL, "
            @"tek_id INTEGER NOT NULL REFERENCES teks(ROWID) ON DELETE CASCADE, "
            @"PRIMARY KEY(rpi, timestamp)) WITHOUT ROWID",
        @"CREATE INDEX teks_end ON teks(end)",
        @"CREATE INDEX advertisement_tek_id ON advertisements(tek_id)",
        @"CREATE TABLE kvs ("
            @"ROWID INTEGER PRIMARY KEY AUTOINCREMENT, "
            @"key TEXT NOT NULL UNIQUE, "
            @"value, " // Note: intentionally not typed
            @"type INTEGER NOT NULL, "
            @"mod_date REAL NOT NULL, "
            @"expiration_date REAL)",
        @"CREATE TABLE session_history("
            @"ROWID INTEGER PRIMARY KEY AUTOINCREMENT, "
            @"uuid BLOB NOT NULL UNIQUE, "
            @"date REAL NOT NULL, "
            @"app_bundle_id TEXT, "
            @"region_cc TEXT, "
            @"region_sc TEXT, "
            @"file_count INTEGER NOT NULL, "
            @"match_count INTEGER NOT NULL, "
            @"build TEXT, "
            @"exp_class TEXT)",
        @"CREATE INDEX session_history_date ON session_history(date)",
        @"CREATE TABLE file_history("
            @"ROWID INTEGER PRIMARY KEY AUTOINCREMENT, "
            @"hash BLOB NOT NULL, "
            @"date REAL NOT NULL, "
            @"session_id INTEGER NOT NULL REFERENCES session_history(ROWID) ON DELETE CASCADE, "
            @"key_count INTEGER NOT NULL, "
            @"match_count INTEGER, "
            @"app_bundle_id TEXT, "
            @"region_cc TEXT, "
            @"region_sc TEXT, "
            @"metadata BLOB)",
        @"CREATE INDEX file_history_session_date ON file_history(session_id, date)",
        @"CREATE INDEX file_history_hash ON file_history(hash)",
```

Appendix 3 Configuration server sslscan output

```
sslscan gateway.icloud.com
         Version: 2.0.11-static
         OpenSSL 1.1.1n-dev  xx XXX xxxx

         Connected to 17.248.176.230

         Testing SSL server gateway.icloud.com on port 443 using SNI name gateway.icloud.com

           SSL/TLS Protocols:
         SSLv2     disabled
         SSLv3     disabled
         TLSv1.0   disabled
         TLSv1.1   disabled
         TLSv1.2   enabled
         TLSv1.3   enabled

           TLS Fallback SCSV:
         Server supports TLS Fallback SCSV

           TLS renegotiation:
         Session renegotiation not supported

           TLS Compression:
         Compression disabled

           Heartbleed:
         TLSv1.3 not vulnerable to heartbleed
         TLSv1.2 not vulnerable to heartbleed

           Supported Server Cipher(s):
         Preferred TLSv1.3  128 bits  TLS_AES_128_GCM_SHA256        Curve 25519 DHE 253
         Accepted  TLSv1.3  256 bits  TLS_AES_256_GCM_SHA384        Curve 25519 DHE 253
         Accepted  TLSv1.3  256 bits  TLS_CHACHA20_POLY1305_SHA256  Curve 25519 DHE 253
         Preferred TLSv1.2  256 bits  ECDHE-ECDSA-AES256-GCM-SHA384 Curve 25519 DHE 253
         Accepted  TLSv1.2  256 bits  ECDHE-RSA-AES256-GCM-SHA384   Curve 25519 DHE 253
         Accepted  TLSv1.2  256 bits  ECDHE-ECDSA-CHACHA20-POLY1305 Curve 25519 DHE 253
         Accepted  TLSv1.2  256 bits  ECDHE-RSA-CHACHA20-POLY1305   Curve 25519 DHE 253
         Accepted  TLSv1.2  128 bits  ECDHE-ECDSA-AES128-GCM-SHA256 Curve 25519 DHE 253
         Accepted  TLSv1.2  128 bits  ECDHE-RSA-AES128-GCM-SHA256   Curve 25519 DHE 253
         Accepted  TLSv1.2  256 bits  ECDHE-ECDSA-AES256-SHA384     Curve 25519 DHE 253
         Accepted  TLSv1.2  256 bits  ECDHE-RSA-AES256-SHA384       Curve 25519 DHE 253
         Accepted  TLSv1.2  256 bits  ECDHE-ECDSA-CAMELLIA256-SHA384 Curve 25519 DHE 253
         Accepted  TLSv1.2  256 bits  ECDHE-RSA-CAMELLIA256-SHA384  Curve 25519 DHE 253
         Accepted  TLSv1.2  128 bits  ECDHE-ECDSA-AES128-SHA256     Curve 25519 DHE 253
         Accepted  TLSv1.2  128 bits  ECDHE-RSA-AES128-SHA256       Curve 25519 DHE 253
         Accepted  TLSv1.2  128 bits  ECDHE-ECDSA-CAMELLIA128-SHA256 Curve 25519 DHE 253
         Accepted  TLSv1.2  128 bits  ECDHE-RSA-CAMELLIA128-SHA256  Curve 25519 DHE 253

           Server Key Exchange Group(s):
         TLSv1.3  128 bits  secp256r1 (NIST P-256)
         TLSv1.3  192 bits  secp384r1 (NIST P-384)
         TLSv1.3  260 bits  secp521r1 (NIST P-521)
         TLSv1.3  128 bits  x25519
         TLSv1.3  224 bits  x448
         TLSv1.2  128 bits  secp256r1 (NIST P-256)
         TLSv1.2  192 bits  secp384r1 (NIST P-384)
         TLSv1.2  260 bits  secp521r1 (NIST P-521)
         TLSv1.2  128 bits  x25519
         TLSv1.2  224 bits  x448

           SSL Certificate:
         Signature Algorithm: sha256WithRSAEncryption
         ECC Curve Name:      prime256v1
         ECC Key Strength:    128

         Subject:  gateway.icloud.com
         Altnames: DNS:gateway-india.icloud.com, DNS:gateway-sandbox.icloud.com, DNS:gateway-sr-
china.icloud.com, DNS:gateway-australia.icloud.com, DNS:gateway.icloud.com
         Issuer:   Apple IST CA 2 - G1

         Not valid before: Jun 22 11:54:06 2021 GMT
         Not valid after:  Jul 22 11:54:05 2022 GMT
```

Appendix 4 Glossary

Client app

The app usually built by national governments that interacts with the Exposure Notification framework

ENDaemon

Exposure Notification Daemon. The part of the exposure notification framework that runs as a daemon. This carries out the bulk of the operations involved with exposure notification.

Exposure Notification API (EN API)

The Application Programming Interface (API) developed by Apple and Google that allows for building apps that use Exposure Notification framework.

Exposure Notification Express (ENX)

The second evolution of the contact tracing technology developed by Google and Apple. ENX allows for contact tracing without a client app.

Rolling Proximity Identifier(RPI)

The ID derived from temporary exposure keys. This ID is broadcast as part of a bluetooth advertisement. This ID is used instead of the key itself for anonymity purposes.

Appendix 5 Testing team

Fabian Freyer	After winning multiple high-profile international CTF tournaments and qualifying for events such as DEF CON CTF and 0CTF with his CTF team Tasteless, Fabian is now focusing on code auditing and pentesting. Among his public work, he has identified critical vulnerabilities in iTerm2 (e.g. CVE-2019-9535 together with Stefan Grönke), Homebrew and RocketChat. While he prefers reviewing low-level and native code in Rust, C and Assembly and currently focuses on Apple software ecosystems such as iOS and macOS, Fabian has a well-founded understanding of the security pitfalls of web-frontend desktop applications and has identified a number of security flaws in Electron-based applications (e.g. https://hackerone.com/reports/899964). Recently, Fabian has been part of a collaborative effort to investigate the security of the Apple AirTags and has held a talk on this topic at a hardware security conference.
Daniel Attevelt	Daniel started programming at a young age, writing demos and games in assembly. He then began developing hardware interfaces and control software for home-brew hardware in C++. Daniel studied Cognitive Neuroscience at Utrecht University but chose to follow a more practical path into software development. After completing a career in software development, he switched to the security field and is now using his skills to help protect society's information systems.
Melanie Rieback	Melanie Rieback is a former Asst. Prof. of Computer Science from the VU, who is also the co-founder/CEO of Radically Open Security.