نگاهی بر جیلبرک از گذشته تا به امروز

مقدمه

سلام خدمت شما دوست عزیزی که این مقاله را میخوانید, ایده ی این مقاله تقریبا یکی دو سالی ذهن بنده را مشغول کرده بود. هر موقع نیاز به تحقیقی در این زمینه میدیدم به سایت های شناخته شده مانند (theiphonewiki.com) مراجعه میکردم, البته نقش سایت هایی نظیر (exploit-db.com) را نیز نمیتوان نادیده گرفت. اما مشکل اصلی از آنجایی شروع میشود که بسیاری از افراد علاقه مند به زمینه هک و امنیت و به خصوص افرادی که میخواهند در زمینه ی سیستم عامل iOS فعالیت کنند, نمیدادند که باید از کجا و چگونه شروع کنند.

در این مقاله سعی شده تا توضیحاتی با توجه به مقاله های شناحته شده و همچنین تجربیات شخصی از مباحث ابتدایی تا بحث های پیشرفته تر گرد آوری گردد تا مشتاقان این زمینه بتوانند سرنخی برای شروع (در واقع ادامه) این سفر هیجان انگیز داشته باشند.

در ضمن نداشتن فهرست و سایر مشکلات مقاله را نیز به بزرگی خودتان ببخشید, در هر حال این اولین مقاله من است و امیدوارم تجربه لازم برای نوشتن مقالات بعدی همراه با تکامل روزانه ی iOS را کسب کنم.

در آخر هم لازم میبینم از عمو گوگل تشکر ویژه ای داشته باشم که کمک های بسیاری در زمینه ی پیدا کردن توضیحات متعارف تر به من کردند.

مهیار رزقی

نگاهی بر جیلبرک از گذشته تا به امروز

همیشه با نوشتن تیتر های کلیشه ای مشکل داشتم اما در حقیقت موضوع این قسمت مقاله دقیقا همین تیتر کلیشه ای هست که میبینید!

برای اینکه تقسیم بندی گذشته و امروز iOS از نظر خودم را برای شما واضح تر بیان کنم نیاز هست که در ابتدا چند اصطلاح را که در این مقاله زیاد به آنها برخورد میکنیم را برای شما توضیح دهم.

Userland describes the software running on an iOS device after the kernel has started. A userland-based exploit or jailbreak, being entirely software-based, can be patched by Apple. Userland jailbreaks differ from jailbreaks that affect the boot chain of trust, in that they do not allow custom firmwares/IMG3 files to be flashed.

اصولا شما به هر دستگاه کامپیوتری که نگاه کنید دو بخش اصلی در قسمت سیستم عامل میبینید:

Kernel mode, User mode

هسته سیستم عامل (Kernel) یک نرم افزار سطح پایین است که وظیفه کنترل درخواست‌های خروجی یا ورودی از طرف نرم افزارهای دیگر (و همچنین سخت افزارها) را دارد.

کرنل واسط بین اپلیکیشن‌ها و سخت افزارها است؛ یعنی هر درخواستی که هر یک از اپلیکیشن‌ها (حتی منابع خود سیستم عامل) برای استفاده از منابع سخت افزاری داشته باشد، ابتدا به کرنل فرستاده می‌شود تا مورد بررسی و تحلیل قرار بگیرد. کرنل همچنین وظایف دیگری نیز بر عهده دارد؛ از جمله مدیریت منابع سیستم، آماده سازی سیستم عامل و برنامه‌ها، مدیریت آدرس‌ها و حافظه RAM و...

بعد از کرنل, User mode یا همان Userland که در بالا به آن اشاره کردم وجود دارد, که در سطح بالاتری قرار میگیرد و در واقع هرچیزی که کاربر با آن سر و کار دارد در Userland وجود دارد, بگذارید کلمه ی وجود دارد را به زندانی شده تغییر دهیم!

خوب این زندان همان Jail و هدف ما از این مقاله فرار از زندان (Jailbreak) است!

اینجاست که تازه بحث گذشته و امروز به میان می آید. در گدشته یعنی تا آخر سری نسخه های iOS 4 جدال اپل با جیلبرکر ها مثل الآن زیاد نبود و اکثرا جیلبرک ها بر پایه باگ هایی بودند که در سطح پایینتر یعنی در سطح کرنل وجود داشتند و البته این نوع جیلبرک در زمان فعلی بسیار با ارزش شده اند. حتما میپرسید چرا؟! دلیلش بسیار ساده و روشن است, معمولا وقتی شرکتی محصولی تولید میکند و سپس سیستم عامل آن را پس از مدتی به روز رسانی میکند در بیشتر موارد این تغییرات در قسمت Usermode است چرا که کاربر با این قسمت سر و کار دارد. پس اگر فکر کنیم میبینیم که اگر باگی در سطح کرنل داشته باشیم بدون توجه به تغییراتی که در Userland ایجاد میشود میتوانیم دسترسی خود را حفظ کنیم.

زمانی بود که چند روز پس از عرضه ی عمومی نسخه ای از iOS جیلبرک آن نسخه به طور عمومی عرضه میشد. به لطف Geohot باگی از bootrom آیفون ۴ به دست آمد که نفوذ را در هر نسخه ای ممکن میساخت چرا که تنها راه از بین بردن باگ در سطح Bootrom تغییر سخت افزاری از طرف شرکت سازنده است که البته کاری پر هزینه است اپل هم زیاد به خود زحمت نداده و این مشکل را نه در آیفون ۴ بلکه در 4S رفع کرده, امیدوارم چرایی در ذهنتان به وجود آمده باشد! اولا که جمع آوری و تغییر این همه محصول که به بازار عرضه شده کاری

بسیار سخت و پر هزینه است و هیچ شرکتی حاضر نیست چنین ضرر عظیمی ببیند. دوما کسی که قصد استفاده از چنین باگی را دارد مسلما در زمان فراخوان عمومی برای تعویض دستگاه اقدام به چنین عمل احمقانه ای نمیکند!

این مثالی ساده از باگ های در سطح Bootrom بود, امروزه نیز اینگونه باگها وجود دارند ولی جیلبرکی بر پایه این باگ ها عرضه نمیشود چرا که بسیار با ارزش هستند. به عنوان مثال ویدیو ای از iH8sn0w منتشر شده و نشان میدهد که iOS 9 را جیلبرک کرده, این کار با استفاده از یکی از باگ های iBoot در آیفون ۵ انجام شده, البته که این کار کمک شایانی به پیدا کردن باگ هایی در زمینه Userland میکند اما چیزی که واضح است این است که قطعا شما جیلبرکی بر پایه iBoot برای iOS 9 نخواهید دید.

پس تا به اینجای مقاله باید متوجه شده باشید که ما تنها جیلبرک هایی که در قسمت Userland هستند را بررسی میکنیم.

بررسی یک آسیب پذیری ساده

برای اینکه قبل از شروع کار اصلی آشنایی مختصری با نوع فکر کردن و دید نسبت به آسیب پذیری های جزیی در زمینه ی جیلبرک پیدا کنید, تصمیم به شرح آسیب پذیری که چندی پیش در یکی از نرم افزار های معروف در زمینه ویرایش و ساخت فایل های PDF روی آیفون و آیپد پیدا شده بود گرفتم.

PDF Converter & Text Editor

By Dixant Vijayvargiya

Text Editor & PDF Creator is your all-in-one document management solution for iPhone, iPod touch and iPad.

It can catch documents from PC or Mac via USB cable or WIFI, email attachments, Dropbox and box and save it on your iPhone, iPod Touch or iPad locally.

https://itunes.apple.com/it/app/text-editor-pdf-creator/id639156936

اول آنکه این آسیب پذیری در نسخه آخر این برنامه (که در زمان نوشتن این مقاله ۲٫۱ بوده است) موجود میباشد.

این آسیب پذیری از نوع local file include بوده و به هکر اجازه میدهد که بتواند از راه دور فایل یا پوشه ای ایجاد نماید و یا به فایل یا پوشه خاصی دسترسی پیدا کند, البته قبل از هر چیز این را در نظر بگیرید که در وضعیت فعلی چنین آسیب پذیری به آسانی منتهی به جیلبرک نمیشود, چرا که هر برنامه در سیستم عامل iOS به صورت Sandbox اجرا شده و به سایر قسمت های سیستم عامل دسترسی ندارد!

ممکن است زیاد در مورد Sandbox یا " جعبه شنی " چیزی نشنیده باشید, ما در اینجا به صورت مختصر به این مورد اشاره خواهیم کرد.

در واقع سند باکس یک سیستم حفاظتی قدرتمند است که این روزها در اکثر برنامه ها به کار گرفته می شود. به عنوان مثال مرورگرها. اما این سیستم حفاظتی به صورت دقیق چکار میکند؟ سند باکس در واقع یک محیط بسته ایجاد می کند تا اگر یک برنامه مخرب بود و یا محتوایی که از طریق آن به کامپیوتر شما راه پیدا کرده است مخرب باشد، مشکلی متوجه سیستم شما نشود.

چنین آسیب پذیری تنها در صورتی منتهی به جیلبرک میشود که شما راهی برای دور زدن این محیط سندباکس پیدا کنید, که البته چندان دور از فکر هم نیست. در مباحث بعدی پس از بررسی جیلبرک های اخیر گروه TaiG میتوانید به این موضوع پی ببرید که ایده ها و آسیب پذیری های ساده میتوانند چه نقش بزرگی داشته باشند.

در این مثال آسیب پذیری در مقدار "filename" در ماژول "submit upload" وجود دارد. در اینجا هکر میتواند با قرار دادن کد مورد نظر خود در به عنوان اسم فایل و سپس آپلود آن از طریق پنل تحت وب برنامه کد های خود را اجرا کند. لازم به ذکر است که اینکار به آسانی و از طریق متد POST انجام میگیرد و نیاز به هیچگونه دسترسی سطح مدیریت و... ندارد.

Request Method(s):

[+] [POST]

Vulnerable Module(s):

[+] Submit (Upload)

Vulnerable Parameter(s):

[+] filename

Affected Module(s):

[+] Index File Dir Listing (http://localhost:52437/)

PoC: Upload File (http://localhost:52437/Box/)

<bq>Files</bq> ..

<a href="<iframe>2.png"><../[LOCAL FILE INCLUDE VULNERABILITY IN FILENAME!]>2.png (0.5 Kb, 2015-04-30 10:58:46 +0000)

<form action="" method="post" enctype="multipart/form-data" name="form1" id="form1"><label>upload file<input type="file" name="file" id="file" /></label><label><input type="submit" name="button" id="button" value="Submit" /></label></form> </body></html></iframe>

--- PoC Session Logs [POST] (LFI - Filename) ---

Status: 200[OK]

POST http://localhost:52437/Box/

Load Flags[LOAD_DOCUMENT_URI LOAD_INITIAL_DOCUMENT_URI] Größe des Inhalts[3262] Mime Type[application/x-unknown-content-type]

Request Header:

Host[localhost:52437]

User-Agent[Mozilla/5.0 (Windows NT 6.3; WOW64; rv:37.0) Gecko/20100101 Firefox/37.0]

Accept[text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8]

Accept-Language[de,en-US;q=0.7,en;q=0.3]

Accept-Encoding[gzip, deflate]

Referer[http://localhost:52437/Box/]

Connection[keep-alive]

POST-Daten:

POST_DATA[-----------------------------321711425710317

Content-Disposition: form-data; name="file"; filename="../[LOCAL FILE INCLUDE VULNERABILITY IN FILENAME!]>2.png"

Content-Type: image/png

برای انجام حمله موفق در این روش, نیاز است مراحل زیر را انجام دهید:

۱. ابتدا برنامه مورد نظر را روی دستگاه خود نصب کنید
۲. برنامه را اجرا کنید و وب سرور را فعال کنید
۳. صفحه تحت وب برنامه را در مرورگر اینترنت کامپیوتر خود باز کنید (localhost:52437) توجه داشته باشید که به جای عبارت localhost باید IP دستگاه حود را قرار دهید
۴. یک سِشِن (Session) برای دیدن و ویرایش درخواست ها در صفحه مورد نظر با استفاده از برنامه Tamper Data و یا Tamper Chrome ایجاد کنید
۵. حالا با ویرایش درخواست اسم فایل مورد نظر را با دستوری که میخواهید اجرا شود جا به جا کنید
۶. درخواست را ذخیره کنید و دستور انجام ادامه کار را بدهید
۷. کد مورد نظر ما در شاخه اصلی پنل تحت وب اجرا میشود (localhost:52437)
۸. حال یکبار دیگر همان فایل را بدون ایجاد تغییرات در درخواست آپلود کنید
۹. خوب تبریک میگوییم شما از یک آسیب پذیری به نفع خودتان استفاده کردید

خوب در اینجا ما موفق شدیم که از یک آسیب پذیری ساده به نفع خودمان استفاده کنیم, اما هدف از ساختن یک پیلود (Payload) که یک فایل خاص را به تمام فایل های آپلود شده پیوند میزند, چیست؟

اگر کمی در جیلبرک های iOS برگردیم عقب نام برنامه نام آشنای JailbreakMe به چشمان میخورد.

این برنامه یکی از اولین جیلبرک های ساخته شده در Userland میباشد که از ورژن ۳٫۱٫۲ سر و کله اش پیدا شد.

اساس کار برنامه بسیار ساده تر از چیزی بود که در بالا توضیح دادیم. (البته اگر صرفا نوع اجرای عملیات و نه کد نویسی را در نظر بگیریم)

در آن زمان باگی در فایل های PDF پیدا شده بود که به هکر اجازه میداد کد مورد نظر خود را بدون داشتن دسترسی مدیریت اجرا کند. در این زمان هم Comex از این موضوع سوء استفاده کرد و با طراحی کردن سایتی که کاربر را به سمت PDF مورد نظر هدایت میکرد ابزار جیلبرک جدیدی را به وجود آورد.

البته در نهایت از یکی از باگهای تحت کرنل (IOSurface.framework) برای گرفتن دسترسی روت (root) استفاده شده بود.

https://www.theiphonewiki.com/wiki/Malformed_CFF_Vulnerability

https://www.theiphonewiki.com/wiki/IOSurface_Kernel_Exploit

اگر علاقه مند به اطلاعات بیشتر هستید میتوانید از لینک زیر استفاده کنید:

https://www.theiphonewiki.com/wiki/Star

خوب با این تفاسیر چندان دور از انتظار نیست که روزی شما هم چنین باگی پیدا کنید و در طی یک فرایند این چنینی جیلبرک جدیدی عرضه کنید.

من همیشه اعتقاد داشتم هکر کسی است که بتواند به ساده ترین شکل مشکلات را حل کند, نه فقط در زمینه کامپیوتر, بلکه هر مشکلی! پس از خلاقیت خود استفاده کنید.

امیدوارم خود شما در هر بخش تحقیقات لازم را انجام دهید زیرا در اینجا تنها توضیح مختصری از روش انجام کار گفته میشود.

بررسی جیلبرک TaiG برای ورژن های ۸/۰ تا ۸/۱/۲

خوب بالاخره iOS 9 هم منتشر شد و چیزی که واضح است این است که قطعا توضیح روشن درباره ی نحوه عملکرد برنامه های معروف Pangu8 و TaiG آسیبی به کمپانی اپل و همچنین افراد نخواهد زد. (این موضوع را در نظر بگیرید که جیلبرک بعد روشن ماجرا است و میتوان از این آسیب پذیری ها برای ساخت بدافزار هم استفاده کرد!)

قصد دارم در دو مقاله مجزا ابتدا نحوه عملکرد برنامه جیلبرک TaiG برای ورژن های ۸/۰ تا ۸/۱/۲ و سپس برای ورژن های ۸/۱/۳ تا ۸/۴ را شرح دهم, البته برای درک بهتر این قسمت از مقاله توصیه میکنم کمی در مورد نحوه دیباگ کردن (Debugging) و دی اسمبلر ها (Disassembler) و مانیتور کردن برنامه ها تحقیق کنید.

این آسیب پذیری توسط اپل رفع شده و در وبسایت اپل از هر دو تیم برای پیدا کردن باگ ها تشکر شده است.

http://lists.apple.com/archives/security-announce/2015/Jan/msg00001.html

اما جیلبرک TaiG دارای کد نویسی بسیار تمیزتر و با ارزش تری نسبت به Pangu8 است و میتوان مطالب بسیاری را از آن یاد گرفت.

نرم افزار های مورد نیاز

چون برنامه TaiG در اصل تحت ویندوز است ما نیز این برنامه را در ویندوز بررسی میکنیم, برای اینکار شما به برنامه هایی نیاز دارید که در زیر لیست آنها را برایتان آماده کردم:

- Process Explorer: برنامه ای که توسط Mark Russinovich نوشته شده و اطلاعات بسیار خوبی را در مورد پروسه های در حال اجرا در اختیار ما میگذارد
- Process Monitor: این برنامه نیز مانند برنامه بالا اطلاعاتی در مورد پروسه های در حال اجرا در اختیار ما میگذارد اما وجه تمایز این دو برنامه این است که این برنامه میتواند لاگ های مربوط به API را نیز به ما نشان بدهد
- IDA: از بهترین برنامه های دی اسمبلر (Disassembler) برای ویندوز
- WinDBG: از دیباگر های خوب برای ویندوز است ولی در این مقاله استفاده کمی دارد چون TaiG همکاری خوبی با ما در این زمینه ندارد!

در هنگام نوشتن مقاله دستگاه ما با استفاده از نرم افزار Pangu8 جیلبرک شده بود (در واقع خراب شده بود!)

در زیر لیست برنامه های استفاده شده روی دستگاه iOS را برای شما قرار میدهم:

- FileMon: برنامه ای برای مشاهده ی FSEvents
- fs_usage: برنامه ای بر پایه kdebug مکینتاش که کارایی مشابهی دارد که خروجی چندان جالبی ارائه نمیدهد ولی برا چک کردن دوباره حروجی ها مناسب است.
- Jtool: یک دی اسمبلر خوب برای iOS

در واقع اگر از دستگاه جیلبرک نشده استفاده میکردیم بسیار بهتر بود زیرا میتوانستیم تمام پروسه ها را به صورت زنده مانیتور کنیم, اما یک مزیت که باید به آن توجه کنیم این است که کد نویسی نرم افزار Pangu8 بسیار ضعیف تر TaiG بود و هرگونه درخواستی به سمت OSKextRequest میتوانست باعث پنیک (Panic) کردن کرنل و در نتیجه ری استارت شدن دستگاه شود, در نتیجه ریسک ایجاد تغییرات دائمی توسط TaiG کاملا از بین میرود و میتوانیم با خیال راحت و بدون ترس از بریک شدن دستگاه به کار خود ادامه دهیم. هر دو شرکت

سامسونگ و اپل از مهندسی های خاصی در ساخت دستگاه های خود استفاده کردند که حتی اگر عمدا قصد بریک کردن دستگاه خود را داشته باشید باید بسیار تلاش کنید!

اگر دقت کرده باشید حجم برنامه TaiG به نسبت زیاد بود (۵۱٫۹۳۸٫۸۱۶ بایت) زیرا برنامه تماما در فرمت PE نوشته شده بود که کار دی اسمبل کردن بسیار سخت میکرد, زیرا شما در هنگام کار با برنامه معمولا با کد های بسیار پیچیده و زیاد مواجه میشدید که در آخر معلوم میشد که عکس هایی با فرمت PNG هستند که به صورت Base64 اینکد شده اند!

- PE (پی‌ئی) یا اجرایی انتقال‌پذیر (Portable Executable) بیانگر نوعی ساختار از پرونده‌های اجرایی، آبجکت‌فایل‌ها یا دی‌ال‌الها در سیستم‌عامل ۳۲ یا ۶۴بیتی ویندوز است. عبارت "انتقال‌پذیر" به انطباق‌پذیری آن با محیط‌های بی‌شمار و معماری‌های گوناگون نرم‌افزارهای سیستم اشاره دارد.

در نگاه اول هیچ وابستگی به فایل های کتابخانه ای اپل مانند CoreFoundation, AppleMobileDevice, iTuneSMobileDevice دیده نمیشود, که جای کمی فکر دارد!

راز این شعبده بازی با نگاه کردن به فایل های موقتی که ابزار جیلبرک TaiG ایجاد میکند روشن میشود. در این زمان یک فایل دی ال ال کوچک با نام Tai***.tmp (که به جای *** یک اسم تصادفی وجود دارد) به وجود می آید. همچنین با مانیتور کردن پروسه های در حال اجرا به سادگی میبینید که یک دی ال ال اجرا میشود که یک پروسه به نام CreateDevHelp دارد. در واقع اسم دقیق این دی ال ال TGHelp.dll است.

بخش ویندوزی کار چندان چنگی به دل نمیزند, به همین دلیل این بخش را به صورت خلاصه تر توضیح میدم و فایل دی ال ال را در صورتی که خواستید خودتان روی آن کار کنید را برایتان قرار میدهم.

http://t3hosting.ir/dl/taig/Tai3V6K.tmp

در زیر لیست آدرس های مهم را با هم مرور میکنیم:

```
.text:10012288                mov     dword_1008677C, eax
.text:1001228D                call    edi ; GetProcAddress
.text:1001228F                mov     dword_10086778, eax
.text:10012294
.text:10012294 _do_mobile_device:                     ; CODE XREF: __gets_all_symbols_from_Apple_libs+4F↑j
.text:10012294                mov     eax, _MobileDevice_dll_handle
.text:10012299                test    eax, eax
.text:1001229B                jnz     short _got_mobileDevice_dll_handle
.text:1001229D                mov     esi, offset aMobiledevice_d ; "MobileDevice.dll"
.text:100122A2                call    _loads_library
.text:100122A7                test    eax, eax
.text:100122A9                mov     _MobileDevice_dll_handle, eax
.text:100122AE                jz      _got_iTunesMobileDevice_dll_handle
.text:100122B4                push    offset aAmdevicecreate ; "AMDeviceCreateFromProperties"
.text:100122B9                push    eax             ; hModule
.text:100122BA                call    edi ; GetProcAddress
.text:100122BC                mov     dword_1008676C, eax
.text:100122C1                mov     eax, _MobileDevice_dll_handle
.text:100122C6                push    offset aAmdcopysystemb ; "AMDCopySystemBonjourUniqueID"
.text:100122CB                push    eax             ; hModule
.text:100122CC                call    edi ; GetProcAddress
.text:100122CE                mov     ecx, _MobileDevice_dll_handle
.text:100122D4                push    offset a_createpairing ; "_CreatePairingMaterial"
.text:100122D9                push    ecx             ; hModule
.text:100122DA                mov     _AMDCopySystemBonjourUniqueID, eax
.text:100122DF                call    edi ; GetProcAddress
.text:100122E1                mov     _CreatePairingMaterial, eax
.text:100122E6
.text:100122E6 _got_mobileDevice_dll_handle:          ; CODE XREF: __gets_all_symbols_from_Apple_libs+43B↑j
.text:100122E6                mov     eax, _iTunesMobileDevice_dll_handle
.text:100122EB                test    eax, eax
.text:100122ED                jnz     loc_100129A7
.text:100122F3                mov     esi, offset aItunesmobile_0 ; "iTunesMobileDevice.dll"
.text:100122F8                call    _loads_library
.text:100122FD                test    eax, eax
.text:100122FF                mov     _iTunesMobileDevice_dll_handle, eax
.text:1001304                 jz      _got_iTunesMobileDevice_dll_handle
.text:1001230A                push    offset aUsbmuxconnectb ; "USBMuxConnectByPort"
```

0x 10011E60: تابعی است که تمام آبجکت های مربوط به کتابخانه ی اپل (مانند: CoreFoundation.dll, MobileDevice.dll, iTunesMobileDevice.dll, AirTrafficHost.dll) را فراخوانی میکند, که اینکار را با فراخوانی آدرس 0x 10011DE0 که از تابع LdrLoadDll در فایل ntdll.dll به جای LoadLibrary در kernel32 استفاده میکند که پروسه ی کار چندان هم واضح نباشد.

0x1001465C: چک کردن وضعیت اکتیوشن دستگاه

0x10013AC9: چک کردن وجود یا عدم وجود پسورد روی دستگاه

0x10017DE0: برقراری ارتباط با com.apple.mobilebackup2

0x10011360: برقراری ارتباط با com.apple.afc2

لیست بالا چندان کامل نیست اما اطلاعات تقریبا کافی برای انجام مهندسی معکوس به شما میدهد. اما اگر خواستید لیست کامل تری را درست کنید میتوانید از طریق آدرس زیر اقدام کنید و لیست تمام API ها را مشاهده کنید.

http://www.libimobiledevice.org/docs/html/files.html

خوب به قسمت سرگرم کننده ی مقاله رسیدیم! میخواهیم فرایند جیلبرک را روی خود دستگاه اپلیمان بررسی کنیم.

اپل تعدادی از این آسیب پذیری ها را در APPLE-SA-2015-01-27-2 لیست کرده است ولی همانطور که قبلا اشاره کردم به هیچ عنوان توضیحات کامل و یا حتی مشخصی وجود ندارد که کدام آسیب پذیری مسئولیت کدام قسمت از فرایند جیلبرک را بر عهده دارد.

به علت ساختار بسیار سخت و محکم (ولی نه ضد گلوله!) iOS معمولا باید از ترکیب چند باگ استفاده کرد تا بتوانن به چنین آزادی عمل و جیلبرکی دست پیدا کرد. در زیر لیست فرایند هایی که در همه ی زمان ها از گذشته تا به امروز برای انجام جیلبرک با آن ها سر و کار داریم را بیان میکنم:

- از بین بردن سندباکس: اپل به صورت دیفالت تمامی برنامه ها را محدود کرده, البته به لطف Sandbox.kext و AppleMobileFileIntegrity.kext میتوان این محدودیت را از بین برد, که در ادامه توضیح میدهم.
- به دست آوردن قابلیت اجرای کد دلخواه: چیزی شبیه مثالی که در مبحث قبل زدیم, اما با این تفاوت که این کار معمولا با استفاده از حمله کردن به یکی از فایل های امضا شده (Signed) توسط اپل و اکسپلویت (Exploit) کردن آن مانند libmis انجام میگیرد.
- گرفتن دسترسی روت (root): از هک ساده وبسایت ها تا هک کردن دستگاه های اپل همیشه هدف رسیدن به بالاترین سطح دسترسی در سیستم عامل بوده و هست, که در دستگاه های اپلی از طریق حمله کردن به یکی از دیمون هایی (daemon) که به صورت روت در حال اجرا است, بسیار محترمانه خواهش میکنیم که دیمون launchd را به صورت روت برای ما اجرا کند!
- پچ کردن کرنل (Kernel Patching): بیشتر روش های این عمل در iOS به لطف Comex ممکن شده و ما در اینجا در واقع کرنل را بعد از روی کار آمدن Userland با استفاده از دستوراتی در هنگام بوت شدن (boot-args) و نه در nvram (همانطور که گفتیم امنیت iBoot بسیار پیشرفت کرده و تمام دستورات ما را رد میکند) انجام میدهیم.

البته نکته ای باید در نظر بگیرید این است که همیشه شما نیاز به ۴ پیلود مختلف برای انجام اینکار ندارید و در بیشتر موارد میتوان با حمله به یکی از سرویس های در حال اجرا روی دستگاه به هر سه مورد اول رسید. (که این دقیقا همان کاری است که TaiG با استفاده کردن از کندی سرویس AFC و کد نویسی بد آن انجام داده است!) اما پچ کردن کرنل همیشه یک پیلود جداگانه نیاز خواهد داشت, زیرا امنیت XNU در طی سال های اخیر بسیار بالاتر رفته تا جلوی جیلبرک های متعدد با یک روش یکسان گرفته شود.

گفتیم که در اینجا جیلبرک از طریق حمله به سرویس AFC انجام شده است, اما چگونه؟

AFC در واقع مسئولیت برقراری ارتباطات میان iTunes (در این مورد TaiG) و iDevice (همان دستگاه اپلی) را بر عهده دارد و تنها سرویس قابل دسترسی است که در نهایت اجازه ی ساخت فایل و فولدر و همینطور پاک کردن آن ها در iDevice را دارد.

اگر قبلا به این موضوع توجه کرده باشید با برنامه هایی مانند iTools, iFunbox, iMazing و... حتی اگر دستگاه شما جیلبرک هم نباشد شما میتوانید عکس هایتان را به Camera roll یا همان فولدر /private/var/mobile/Media/DCIM انتقال دهید. دلیل این امر هم آن است که اپل اجازه ی دسترسی کامل به فولدر /private/var/mobile/Media را به AFC داده است و هر چیزی (عکس, فیلم, موزیک و کتاب ها) که شما به کمک iTunes و سایر برنامه ها در iDevice خود انتقال میدهید در این فولدر ذخیره میشود.

TaiG دقیقا از همین موضوع استفاده کرد و فایل ها و فولدر هایی در /private/var/mobile/Media ایجاد میکند.

به خروجی های زیر توجه کنید, ابتدا فولدر _exhelp و سپس _mvhelp ایجاد میشود.

```
126 afcd  Created dir  /private/var/mobile/Media/_exhelp
 126 afcdCreated dir  /private/var/mobile/Media/_exhelp/a
 126 afcdCreated dir  /private/var/mobile/Media/_exhelp/a/a
 126 afcdCreated dir  /private/var/mobile/Media/_exhelp/a/a/a
 126 afcdCreated dir  /private/var/mobile/Media/_exhelp/var
 126 afcdCreated dir  /private/var/mobile/Media/_exhelp/var/mobile
 126 afcdCreated dir  /private/var/mobile/Media/_exhelp/var/mobile/Media
 126 afcdCreated dir  /private/var/mobile/Media/_exhelp/var/mobile/Media/Books
 126 afcdCreated dir  /private/var/mobile/Media/_exhelp/var/mobile/Media/Books/Purchases
 126 afcdCreated  /private/var/mobile/Media/_exhelp/var/mobile/Media/Books/Purchases/mload
 126 afcdCreated  /private/var/mobile/Media/_exhelp/a/a/a/c        (symlink)
 126 afcdCreated dir  /private/var/mobile/Media/_mvhelp
 126 afcdCreated dir  /private/var/mobile/Media/_mvhelp/a
 126 afcdCreated dir  /private/var/mobile/Media/_mvhelp/a/a
 126 afcdCreated dir  /private/var/mobile/Media/_mvhelp/a/a/a
 126 afcdCreated dir  /private/var/mobile/Media/_mvhelp/a/a/a/a
 126 afcdCreated dir  /private/var/mobile/Media/_mvhelp/a/a/a/a/a
 126 afcdCreated dir  /private/var/mobile/Media/_mvhelp/a/a/a/a/a/a
 126 afcdCreated dir  /private/var/mobile/Media/_mvhelp/a/a/a/a/a/a/a
 126 afcdCreated dir  /private/var/mobile/Media/_mvhelp/private
 126 afcdCreated dir  /private/var/mobile/Media/_mvhelp/private/var
```

126 afcd Created /private/var/mobile/Media/_mvhelp/private/var/run

126 afcd Created /private/var/mobile/Media/_mvhelp/a/a/a/a/a/a/a/c (symlink)

-- BackupAgent wakes up

279 BackupAgent Chowned /private/var/MobileDevice/ProvisioningProfiles

Some events dropped

*** Warning: Some events may be lost

279 BackupAgent Created dir /private/var/.backup.i/var/MobileDevice

279 BackupAgent Created dir /private/var/.backup.i/var/MobileDevice/ProvisioningProfiles

279 BackupAgent Chowned /private/var/.backup.i/var/MobileDevice/ProvisioningProfiles

279 BackupAgent Created dir /private/var/.backup.i/var/mobile/Media

279 BackupAgent Created dir /private/var/.backup.i/var/mobile/Media/PhotoData

279 BackupAgent Renamed /private/var/mobile/Media/_mvhelp/a/a/a/a/a/a/a/c
/private/var/.backup.i/var/mobile/Media/PhotoData/c

279 BackupAgent Chowned /private/var/run

279 BackupAgent Chowned /private/var/run

279 BackupAgent Renamed /private/var/mobile/Media/_exhelp/a/a/a/c /private/var/run/mobile_image_mounter

279 BackupAgent Chowned /private/var/mobile/Media/Books/Purchases/mload

پس از پایان کار و گرفتن دسترسی مورد نظر فایل ها را پاک میکند

126 afcd Deleted /private/var/mobile/Media/_mvhelp/a/a/a/a/a/a/a/a

126 afcd Deleted /private/var/mobile/Media/_mvhelp/a/a/a/a/a/a/a

126 afcd Deleted /private/var/mobile/Media/_mvhelp/a/a/a/a/a/a

126 afcd Deleted /private/var/mobile/Media/_mvhelp/a/a/a/a/a

126 afcd Deleted /private/var/mobile/Media/_mvhelp/a/a/a/a

126 afcd Deleted /private/var/mobile/Media/_mvhelp/a/a/a

126 afcd Deleted /private/var/mobile/Media/_mvhelp/a/a

126 afcd Deleted /private/var/mobile/Media/_mvhelp/a

126 afcd Deleted /private/var/mobile/Media/_mvhelp/private/var/run

126 afcd Deleted /private/var/mobile/Media/_mvhelp/private/var

126 afcd Deleted /private/var/mobile/Media/_mvhelp/private

126 afcd Deleted /private/var/mobile/Media/_mvhelp

126 afcd Deleted /private/var/mobile/Media/_exhelp/a/a/a

126 afcd Deleted /private/var/mobile/Media/_exhelp/a/a

126 afcd Deleted /private/var/mobile/Media/_exhelp/a

126 afcd Deleted /private/var/mobile/Media/_exhelp/var/mobile/Media/Books/Purchases/mload

126 afcd Deleted /private/var/mobile/Media/_exhelp/var/mobile/Media/Books/Purchases

126 afcd Deleted /private/var/mobile/Media/_exhelp/var/mobile/Media/Books

126 afcd Deleted /private/var/mobile/Media/_exhelp/var/mobile/Media

126 afcd Deleted /private/var/mobile/Media/_exhelp/var/mobile

126 afcd Deleted /private/var/mobile/Media/_exhelp/var

126 afcd Deleted /private/var/mobile/Media/_exhelp

شاید کمی گیج کننده باشد ولی چیزی که در نهایت به آن دست یافتیم بدین شکل است:

/private/var/run/mobile_image_mounter -> ../../../var/mobile/Media/Books/Purchases/mload

چرا این همه دردسر؟ اصلا چرا فایل mobile_image_mounter ؟

خوب AFCD راه ورود را برای ما از میکند ولی فقط یکی دو قدم میتوانیم برداریم! بقیه دستورات باید توسط DeveloperDiskImage پشتیبانی بشوند. DeveloperDiskImage (DDI) زمانی روی کار می آید که شما دستگاه خود را به Xcode متصل میکنید (DDI بخشی از SDK مورد استفاده در Xcode است) که تعدادی ابزار و فریم ورک (Framework) را در اختیار ما میگذارد.

وقتی که دستگاه را به Xcode متصل میکنیم این ابزارها و فریم ورک ها در Developer/ مانت (Mount) میشود, و شما میتوانید در Developer/Library/Frameworks/ و System/Library/Frameworks/ و جند فولدر دیگر جست و جو کنید! البته فقط این فولدر نیست اپل این کار را برای فولدر های دیگر مانند AppleInternal/ نیز تکرار کرده. خوب پس اگر کسی بتواند یک DDI را شبیه سازی کند میتواند به فال های فولدر Developer/ دسترسی داشته باشد و آن ها را تغییر دهد.

قضیه به این سادگی ها هم نیست! باید به این موضوع توجه داشته باشید که DDI ساین شده است و شما نمیتوانید هر فایلی که خواستید را به عنوان DDI به iDevice بخورانید و افکار پلید خود را پیاده کنید! اینجا پای یک روش قدیمی به نام The race condition یا وضعیت مسابقه ای میان می آید.

در بین زمان چک کردن امضاء و خواندن فایل یک فاصله زمانی بسیار بسیار کوتاه وجود دارد, تصور کنید بتوانید DDI اصلی را در زمان چک کردن امضاء بدهید و سپس در هنگاه خواندن فایل خود را ارائه بدهید, فکر جالب و البته شدنی است!

TaiG دقیقا همین کار را کرده, در هنگام انجام عملیات جیلبرک, در حدود موقعی که کار به ۳۰% میرسد یک فایل شبیه به DDI در فرمت DMG که بسیار با دقت ساخته شده است با نام input (http://t3hosting.ir/dl/taig/input) در کنار DDI اصلی به نام input2 در دستگاه آپلود میشود که به صورت پشت سر هم تلاش میکند تا MobileStorageMounter.app را مجبور به خواندن DMG اشتباه کند, این کار آنقدر تکرار میشود تا سرانجام فایل مورد نظر ما خوانده شود که به این عمل وضعیت مسابقه ای گفته میشود.

در اینجا فایل DMG که قرار است جایگزین شود ساخته میشود

126 afcd Created /private/var/mobile/Media/Books/Purchases/mload/input

126 afcd Modified /private/var/mobile/Media/Books/Purchases/mload/input

فایل ساخته شده را در MobileStorageMounter.app ثبت میکند. به خاطر داشته باشید که قبلا از طریق متد سیم لینک (Symlink) این پوشه را به یکی از پوشه هایی که AFC به آن دسترسی داشت متصل کردیم (/private/var/mobile/Media/Books/Purchases/mload/)

6362 mobile_storage_p Created
/private/var/mobile/Media/Books/Purchases/mload/6d55c2edf0583c63adc540dbe8bf8547b49d54957ce9dc8032d1a9f9ad759e2b1fe99fcb2baeb3db5348ab322cb65c7fc38b59cb75697cbc29221dce1ecd120d/909b75240921fc3f2d96ff08d317e199e033a7f8a8ff430b0c97bf3c6210fc39f35e1c239d1bf7d568be613aafef53104f3bc1801eda87ef963a7abeb57b8369/37kpPQ.dmg

6362 mobile_storage_p Modified
/private/var/mobile/Media/Books/Purchases/mload/6d55c2edf0583c63adc540dbe8bf8547b49d54957ce9dc8032d1a9f9ad759e2b1fe99fcb2baeb3db5348ab322cb65c7fc38b59cb75697cbc29221dce1ecd120d/909b75240921fc3f2d96ff08d317e199e033a7f8a8ff430b0c97bf3c6210fc39f35e1c239d1bf7d568be613aafef53104f3bc1801eda87ef963a7abeb57b8369/37kpPQ.dmg

DMG اصلی به input2 انتقال داده میشود

126 afcd Renamed
/private/var/mobile/Media/Books/Purchases/mload/6d55c2edf0583c63adc540dbe8bf8547b49d54957ce9dc8032d1a9f9ad759e2b1fe99fcb2baeb3db5348ab322cb65c7fc38b59cb75697cbc29221dce1ecd120d/909b75240921fc3f2d96ff08d317e199e033a7f8a8ff430b0c97bf3c6210fc39f35e1c239d1bf7d568be613aafef53104f3bc1801eda87ef963a7abeb57b8369/37kpPQ.dmg to

/private/var/mobile/Media/Books/Purchases/mload/input2

نام DMG ساخته شده از input به نام اصلی تغییر میکند

126 afcd Renamed /private/var/mobile/Media/Books/Purchases/mload/input /private/var/mobile/Media/Books/Purchases/

mload/6d55c2edf0583c63adc540dbe8bf8547b49d54957ce9dc8032d1a9f9ad759e2b1fe99fcb2baeb3db5348ab322cb65c7fc38b59cb75697cbc29221dce1ecd120d/909b75240921fc3f2d96ff08d317e199e033a7f8a8ff430b0c97bf3c6210fc39f35e1c239d1bf7d568be613aafef53104f3bc1801eda87ef963a7abeb57b8369/37kpPQ.dmg

خوب بار اول کار نکرد! بالاخره MobileStorageMounter.app آنقدرها هم احمق نیست و فایل را پاک میکند!

6347 MobileStorageMou Deleted
/private/var/mobile/Media/Books/Purchases/mload/6d55c2edf0583c63adc540dbe8bf8547b49d54957ce9dc8032d1a9f9ad759e2b1fe99fcb2baeb3db5348ab322cb65c7fc38b59cb75697cbc29221dce1ecd120d/909b75240921fc3f2d96ff08d317e199e033a7f8a8ff430b0c97bf3c6210fc39f35e1c239d1bf7d568be613aafef53104f3bc1801eda87ef963a7abeb57b8369/37kpPQ.dmg

MobileStorageMounter.app لاگ فایلی به منظور توضیح فرایند انجام شده ایجاد میکند.

6362 Modified /private/var/mobile/Library/Logs/Device-O-Matic/com.apple.mobile.storage_proxy.log.0

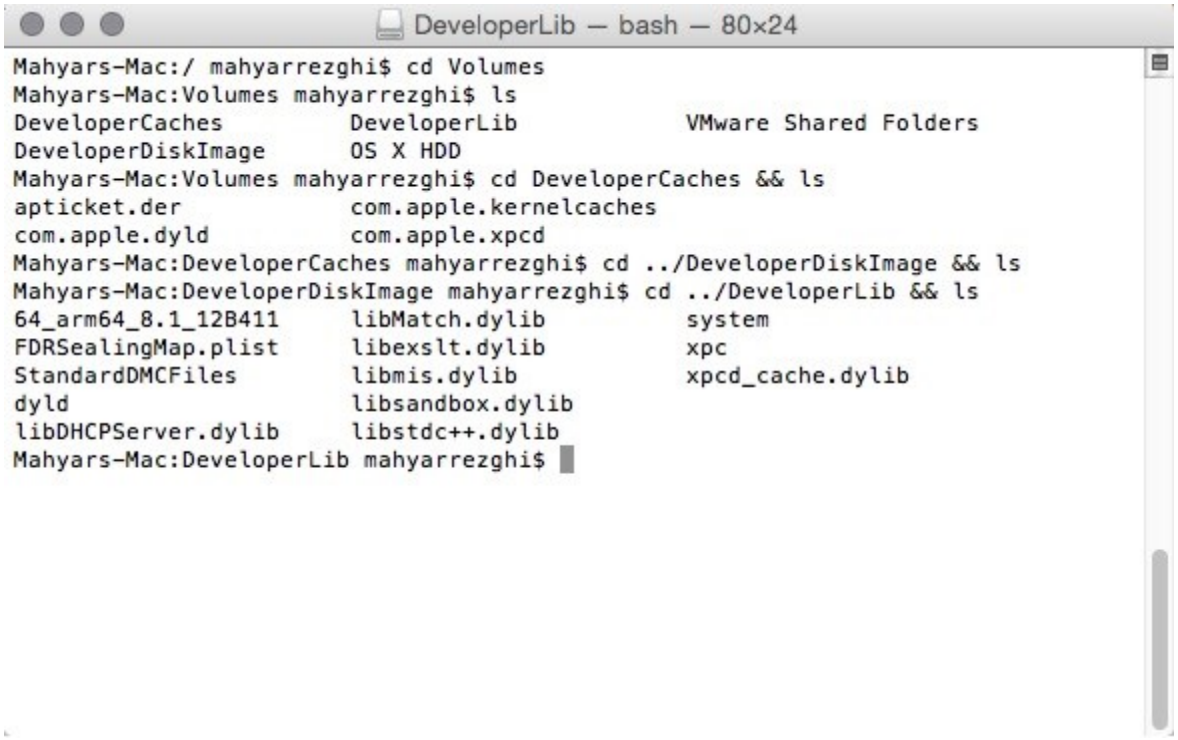
خوب این فرایند باید دوباره تکرار شود پس فایل input2 را پاک میکنیم

126 afcd Deleted /private/var/mobile/Media/Books/Purchases/mload/input2

این فرایند آنقدر تکرار میشود تا بالاخره موفق شویم.

معمولا پروسه بالا باید چندین بار تکرار شود تا فایل مورد نظر ما خوانده شود.

فایل input به راحتی در مک قابل خواندن است به همین دلیل من در اینجا از یک مکینتاش نصب شده در نرم افزار VMware استفاده کردم که در زیر لیست فایل های موجود در input را برای شما قرار میدهم:



اگر دقت کنید دستور مربوط به پارتیشن DeveloperDiskImage هیچ خروجی ندارد که نشان میدهد این فولدر خالی است, این موضوع شاید در نگاه اول کمی جلب توجه کند و عجیب به نظر برسد ولی در واقع چندان مهم هم نیست! در واقع در اینجا مهم ترین قسمت Volumes/DeveloperLib/ است که داخل usr/lib/ مانت میشود! و به همین شکل Volumes/DeveloperCaches/ در System/Library/Caches/ مانت میشود که شامل فایل های apticket.der (که حداقل من نیازی به وجودش نمیبینم!) و com.apple.dyld/enable-dylibs-to-override-cache که کار اصلی را بر عهده دارد و بار ها و بار ها اجرا میشود تا عملیات موفقیت آمیز بوده و فایل های مورد نظر ما در کش جایگزین شوند. در این حال عملیات ساین با مشکل انجام میشود و دستگاه نمیتواند فایل ها را با DDI ساین کند در نتیجه از /usr/libexec/amfid استفاده میکند و در نهایت پس از چندین بار تلاش بی نتیجه کد ساین نشده ی شما, ساین میشود! تبریک!

اما (همیشه یک اما وجود دارد!) به این توجه کنید که ما هنوز جلوی چک کردن کد ساین در amfid را نگرفتیم بلکه فقط فایلی که amfid باید چک میکرد را جایگزین کردیم پس با این وجود نمیتوانیم از DYLD_INSERT_LIBRARIES استفاده کنیم که این موضوع عملا جا به جایی فایل ها را بلا استفاده میکند چرا که در نهایت amfid هر پروسه ای که ساین نشده باشد را میبندد!

خوب تا به حال DMG مورد نظر ما مانت شده که دارای dylib های مورد نظر ماست, اگر فایل libmis.dylib را با jtool باز کنیم با خروجی های زیر رو به رو میشویم:

```
mahyarrezghi@Mahyars-Mac (~/Documents/iOS/JB/TaiG) % jtool -l /Volumes/DeveloperLib/libmis.dylib
Fat binary, big-endian,  3 architectures: armv7, armv7s, armv8
Specify one of these architectures with -arch switch, or export the ARCH environment variable
```

دقیقا همان فایل هایی که در جیلبرک evasi0n استفاده شده بود!

```
mahyarrezghi@Mahyars-Mac (~/Documents/iOS/JB/Pangu) % ARCH=armv7 jtool -S /Volumes/DeveloperLib/libmis.dylib
    I _MISValidateSignature (indirect for _CFEqual)
    I _kMISValidationInfoEntitlements (indirect for _kCFUserNotificationTokenKey)
    I _kMISValidationInfoSignerCertificate (indirect for _kCFUserNotificationTokenKey)
    I _kMISValidationInfoSigningID (indirect for _kCFUserNotificationTokenKey)
    I _kMISValidationInfoValidatedByProfile (indirect for _kCFUserNotificationTokenKey)
    I _kMISValidationOptionAllowAdHocSigning (indirect for _kCFUserNotificationTokenKey)
    I _kMISValidationOptionExpectedHash (indirect for _kCFUserNotificationTimeoutKey)
    I _kMISValidationOptionLogResourceErrors (indirect for _kCFUserNotificationTokenKey)
    I _kMISValidationOptionUniversalFileOffset (indirect for _kCFUserNotificationTokenKey)
    I _kMISValidationOptionValidateSignatureOnly (indirect for _kCFUserNotificationTokenKey)
    U _CFEqual
    U _kCFUserNotificationTimeoutKey
    U _kCFUserNotificationTokenKey
    U dyld_stub_binder
mahyarrezghi@Mahyars-Mac (~/Documents/iOS/JB/TaiG) % ARCH=armv7 jtool -l /Volumes/DeveloperLib/libmis.dylib
LC 00: LC_SEGMENT             Mem: 0x00000000-0x00001000      __TEXT
    Mem: 0x00001000-0x00001000           __TEXT.__text   (Normal)
LC 01: LC_SEGMENT             Mem: 0x00001000-0x00002000      __LINKEDIT
LC 02: LC_ID_DYLIB            /usr/lib/libmis.dylib
..
LC 15: LC_CODE_SIGNATURE      Offset: 38576, Size: 752 (0x96b0-0x99a0)
LC 16: LC_SEGMENT             Mem: 0xfffff000-0x1ffff000      __DATA

mahyarrezghi@Mahyars-Mac (~/Documents/iOS/JB/TaiG) % ARCH=armv8 jtool -l /Volumes/DeveloperLib/libmis.dylib
LC 00: LC_SEGMENT_64          Mem: 0x000000000-0x4000   __TEXT
    Mem: 0x000004000-0x000004000           __TEXT.__text   (Normal)
LC 01: LC_SEGMENT_64          Mem: 0x000004000-0x8000   __LINKEDIT
LC 02: LC_ID_DYLIB            /usr/lib/libmis.dylib
..
LC 16: LC_SEGMENT_64          Mem: 0xfffffffffffc000-0x1fffc000       __DATA
```

فایل جایگزین libmis که توسط TaiG نوشته شده چندان هم کامل نیست به همین دلیل ممکن است گاهی در زمانی که نتواند کد ها را تایید کند, کرش کند.

{"app_name":"amfid","app_version":"","os_version":"iPhone OS 8.1.2 (12B440)",
"slice_uuid":"b99c5334-10f8-3014-865b-b0977d6f1a45","share_with_app_devs":false,
"build_version":"","is_first_party":true,"bug_type":"109","name":"amfid"}

Incident Identifier: 59E17AA8-469A-4395-AC6B-53C1F2B9657B (yep, Apple, that was me :-)

CrashReporter Key: a72d111b6b52f7c1ae1360e923e7c0da675b9261

Hardware Model: iPhone7,2

Process: amfid [22]

Path: /usr/libexec/amfid

Identifier: amfid

Version: ???

Code Type: ARM-64 (Native)

Parent Process: launchd [1]

Date/Time: 2015-05-21 08:13:39.385 -0400

Launch Time: 2015-05-21 08:13:21.101 -0400

OS Version: iOS 8.1.2 (12B440)

Report Version: 105

Exception Type: EXC_BREAKPOINT (SIGTRAP)

Exception Codes: 0x0000000000000001, 0x00000001200fd088

Triggered by Thread: 2

Dyld Error Message:

Symbol not found: _MISCopyInstalledProvisioningProfiles

Referenced from: /usr/libexec/amfid

Expected in: /usr/lib/libmis.dylib

Dyld Version: 353.6

خوب بخش سخت ماجرا تمام شده و تنها یک بخش دیگر باقی مانده است, آن هم پروسه ای است که به untether معروف شده و در واقع در هر بار ری استارت کردن سیستم تمامی این کار ها را به صورت اتوماتیک انجام میدهد! /taig/taig نام این فایل دوست داشتنی است که در انتهای پروسه اجرا شده و دستگاه را ری استارت میکند. (قبل از ری استارت شدن دستگاه کار هایی مانند مانت کردن روت, و انتقال فایل های

کش شده به یک محل دائمی و اضافه کردن /DeveloperPatch که شامل AFC2 است و به ما اجازه میدهد تا به پوشه / که بالانرین سطح است دسترسی پیدا کنیم.)

تا اینجا برنامه TaiG عدد ۴۰% را نشان میدهد و با پیغام زیبای Success از ما پذیرایی میکند و سپس پیغام injecting jailbreak program را به ما نشان میدهد. حالا فایل /taig/taig میتواند اجرا شود. میتوانید این فایل را از لینک زیر دانلود کنید:

http://t3hosting.ir/dl/taig/taig

جالب ترین بخش کار بررسی فایل taig است:

mahyarrezghi@Mahyars-Mac (~/Documents/iOS/JB/TaiG) % file taig

/Volumes/VMware Shared Folders/iOS/JB/taig/taig: Mach-O universal binary with 2 architectures

/Volumes/VMware Shared Folders/iOS/JB/taig/taig (for architecture armv7): Mach-O executable arm

/Volumes/VMware Shared Folders/iOS/JB/taig/taig (for architecture arm64): Mach-O 64-bit executable

چیزی که با نگاه کردن به خروجی های بالا معلوم میشود این است که فایل taig به صورت universal نوشته شده است و هم روی دستگاه های ۳۲بیتی و هم ۶۴بیتی (آیفون 5S و آیپد ایر به بعد) کار میکند.

من در این مقاله فایل taig را تحت معماری ۶۴بیتی بررسی میکنم, اما معماری ۳۲ بیتی هم تا حد زیادی به همین شکل است.

اگر به یاد داشته باشید موقعی که این جیلبرک عرضه شد بسیار از هکر های قدیمی تر به مردم اعلام کردند که فعلا از این جیلبرک استفاده نکنند تا صحت کار آن مشخص شود و مبادا به بد افزاری آلوده باشد. پس قطعا آن ها هم تمامی این مراحل را طی کردن تا به این فایل برسند زیرا این فایل مهم ترین فایل این جیلبرک است. اما چگونه فهمیدند که این فایل آلوده نیست؟

عکس مربوط به بررسی فایل با نرم افزار jtool در صفحه بعد آمده است و شما با نگاه کردن به لیست فایل های مورد نیاز و آبجکت های فراخوانی شده به راحتی میتوانید تایید کنید که هیچگونه فایل یا کد مخربی در این جیلبرک استفاده نشده است.

- خروجی که با کادر قرمز مشخص کردم به ما نشان میدهد که این فایل به هیچ عنوان کد گذاری نشده و ما میتوانیم تمام محتویات آن را ببینیم که همین امر نیز میتواند دلیلی بر آلوده نبودن این فایل باشد.
- قسمتی که که با کادر سبز مشخص کردم به صورت پیشفرض در Xcode وجود دارد, پس قرار دادن بدون تغییر قطعا یک اشتباه در برنامه نویسی بوده است.

برای اجرای این فایل یک سری پیش نیاز ها فراخوانی میشود که در بین دو کادر قرار دارند, که میتوانید در زیر کار هر یک از آن ها را مشاهده کنید:

libsystem.B: بدون این فایل هیچ برنامه یا فایلی نمیتواند پیش نیاز های خود را فراخوانی کند, میتوان کار این فایل را مانند Kernel32+NtDll+MSVCRT در ویندوز و همچنین libc.so در لینوکس در نظر گرفت.

libc++: نشان میدهد که در این برنامه از c++ هم استفاده شده است.

libobjc و Foudation/CoreFoundation: نشان دهنده ی استفاده شدن از Objective C است.

IOKit: مهر تاییدی بر این موضوع است که عملیات پچ کردن کرنل توسط این فایل انجام میشود.

libafc.dylib: در بررسی این فایل من نتوانستم اطلاعات دقیقی از استفاده این پیش نیاز در اینجا به دست بیاورم!

```
Mahyars-Mac:Desktop mahyarrezghi$ ARCH=armv8 ./jtool -l taig
LC 00: LC_SEGMENT_64          Mem: 0x000000000-0x100000000      __PAGEZERO
LC 01: LC_SEGMENT_64          Mem: 0x100000000-0x100014000      __TEXT
        Mem: 0x100005e2c-0x1000115f8            __TEXT.__text   (Normal)
        Mem: 0x1000115f8-0x100011d90            __TEXT.__stubs  (Symbol Stubs)
        Mem: 0x100011d90-0x100012528            __TEXT.__stub_helper    (Normal)
        Mem: 0x100012528-0x100012de3            __TEXT.__cstring        (C-String Literals)
        Mem: 0x100012de3-0x10001309e            __TEXT.__objc_methname  (C-String Literals)
        Mem: 0x1000130a0-0x100013c27            __TEXT.__const
        Mem: 0x100013c27-0x100013c40            __TEXT.__objc_classname (C-String Literals)
        Mem: 0x100013c40-0x100013caf            __TEXT.__objc_methtype  (C-String Literals)
        Mem: 0x100013cb0-0x100013ccc            __TEXT.__ustring
        Mem: 0x100013ccc-0x100013db4            __TEXT.__gcc_except_tab
        Mem: 0x100013db4-0x100014000            __TEXT.__unwind_info
LC 02: LC_SEGMENT_64          Mem: 0x100014000-0x100018000      __DATA
        Mem: 0x100014000-0x100014098            __DATA.__got    (Non-Lazy Symbol Ptrs)
        Mem: 0x100014098-0x1000145a8            __DATA.__la_symbol_ptr  (Lazy Symbol Ptrs)
        Mem: 0x1000145a8-0x1000145b0            __DATA.__mod_init_func  (Module Init Function Ptrs)
        Mem: 0x1000145b0-0x100014890            __DATA.__const
        Mem: 0x100014890-0x100014898            __DATA.__objc_classlist (Normal)
        Mem: 0x100014898-0x1000148a0            __DATA.__objc_catlist   (Normal)
        Mem: 0x1000148a0-0x1000148a8            __DATA.__objc_imageinfo
        Mem: 0x1000148a8-0x100014b28            __DATA.__objc_const
        Mem: 0x100014b28-0x100014bc0            __DATA.__objc_selrefs   (Literal Pointers)
        Mem: 0x100014bc0-0x100014be0            __DATA.__objc_classrefs (Normal)
        Mem: 0x100014be0-0x100014c30            __DATA.__objc_data
        Mem: 0x100014c30-0x100014f10            __DATA.__cfstring
        Mem: 0x100014f10-0x100015138            __DATA.__data
        Mem: 0x100015140-0x100016b70            __DATA.__bss    (Zero Fill)
LC 03: LC_SEGMENT_64          Mem: 0x100018000-0x100020000      __LINKEDIT
LC 04: LC_DYLD_INFO
LC 05: LC_SYMTAB
        Symbol table is at offset 0x19a70 (105072), 186 entries
        String table is at offset 0x1ab6c (109420), 2984 bytes
LC 06: LC_DYSYMTAB
            2 local symbols at index     0
            1 external symbols at index  2
          183 undefined symbols at index 3
           No TOC
           No modtab
          343 Indirect symbols at offset 0x1a610

LC 07: LC_LOAD_DYLINKER         /usr/lib/dyld
LC 08: LC_UUID                  UUID: AC536657-237F-39BE-A7B2-ACBC3A671692
LC 09: LC_VERSION_MIN_IPHONEOS  Minimum iOS  version:    7.0.0
LC 10: LC_SOURCE_VERSION        Source Version:          0.0.0.0.0
LC 11: LC_MAIN                  Entry Point:             0x9674 (Mem: 100009674)
LC 12: LC_ENCRYPTION_INFO_64    Encryption: 0 from offset 16384 spanning 65536 bytes
LC 13: LC_LOAD_DYLIB            /usr/lib/libafc.dylib
LC 14: LC_LOAD_DYLIB            /usr/lib/libc++.1.dylib
LC 15: LC_LOAD_DYLIB            /System/Library/Frameworks/IOKit.framework/Versions/A/IOKit
LC 16: LC_LOAD_DYLIB            /System/Library/PrivateFrameworks/ServiceManagement.framework/ServiceManagement
LC 17: LC_LOAD_DYLIB            /System/Library/Frameworks/Foundation.framework/Foundation
LC 18: LC_LOAD_DYLIB            /usr/lib/libobjc.A.dylib
LC 19: LC_LOAD_DYLIB            /usr/lib/libSystem.B.dylib
LC 20: LC_LOAD_DYLIB            /System/Library/Frameworks/CoreFoundation.framework/CoreFoundation
LC 21: LC_RPATH                 @executable_path/Frameworks
LC 22: LC_FUNCTION_STARTS       Offset: 104640, Size: 344 (0x198c0-0x19a18) with 217 functions
LC 23: LC_DATA_IN_CODE          Offset: 104984, Size: 0 (0x19a18-0x19a18)
LC 24: LC_DYLIB_CODE_SIGN_DRS   Offset: 104984, Size: 88 (0x19a18-0x19a70)
LC 25: LC_CODE_SIGNATURE        Offset: 112416, Size: 1104 (0x1b720-0x1bb70)
Mahyars-Mac:Desktop mahyarrezghi$
```

موقعی که در مکینتاش این فایل را بررسی میکردم مشاهده کردم که طبق اعلام مکینتاش این فایل ساین نشده است و عملا اجرا شدنش در iDeviceیک معجزه است! اما معجزا ای درکار نیست و با با یک دستور ساده توسط jtool میتوانیم به راز این عمل پی ببریم:

```
mahyarrezghi@Mahyars-Mac (~/Documents/iOS/JB/TaiG) % JCOLOR=1 jtool  -arch armv8 --sig -v --ent taig
Blob at offset: 112416 (1104 bytes) is an embedded signature of 1089 bytes, and 3 blobs
        Blob 0: Type: 0 @36: Code Directory (709 bytes)
                Version:    20001
                Flags:      none (0x0)
                Identifier:  taig
                CDHash:      09059af87d64372658826d73e419d42383e2cd75
                # of Hashes: 28 code + 5 special
                Hashes @149 size: 20 Type: SHA-1
                        Entitlements blob:  3b3feeeb6676cb7bcd61e03436a8a39742feb44c (OK)
                        Application Specific:           Not Bound
                        Resource Directory:             Not Bound
                        Requirements blob:              3a75f6db058529148e14dd7ea1b4729cc09ec973 (OK)
                        Bound Info.plist:     Not Bound
                        Slot   0 (File page @0x0000):  b00b96b29f6a86c3bf2538d5b94c5d5a9ee0429e (OK)
                        Slot   1 (File page @0x1000):  1ceaf73df40e531df3bfb26b4fb7cd95fb7bff1d (OK)
                                # تمامی اسلات ها دارای هش سالم بودند به همین دلیل برای خلاصه شدن کد آنها را حذف کردم
                        Slot  27 (File page @0x1b000):          bf05426e9a67734564f1e8bcd752885fd6182b05 (OK)
        Blob 1: Type: 2 @745: Empty requirement set
        Blob 2: Type: 5 @757: Entitlements (332 bytes)


<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
   <key>platform-application</key>
   <true/>
   <key>get-task-allow</key>
   <true/>
   <key>task_for_pid-allow</key>
```

```
    <true/>
</dict>
</plist>
#
# Normally, there'd be a (potentially empty) CMS (RFC3852) signature blob here. This confuses codesign

        Superblob ends @36
```

- تمامی هش ها سالم میباشند
- فایل با شناسه taig ساین شده است
- امضاء فایل به شکل ad hoc نیست (این نوع امضا ها برای فایل ها نوشته شده توسط اپل استفاده میشوند و همچنین در Xcode هم روشی برای ساین کردن برنامه ها به صورت ad hoc وجود دارد.)
- این فایل شامل entitlement است که حتی قابلیت دیباگ کردن هم در آن فعال است

```
    <key>get-task-allow</key>
    <true/>
```

مهمترین دلیل اینکه این فایل حتما باید ساین شده باشد این است که ابتدا جلوی اجرا شدن پیش از موعد فایل توسط AMFI.kext گرفته شود و دوم اینکه entitlement هایی که در بالا ذکر شد در حافظه کرنل ذخیره میشود و سیستم عامل دیگر راهی برای تغییر دادن آنها ندارد در نتیجه میتوانیم همیشه از نعمت استفاده از کد های ساین نشده استفاده کنیم.

platform-application: من به این مشخصه لقب قاتل سندباکس را میدهم, چراکه در حالت عادی جلوی هر کدی که خارج از سندباکس اجرا شود, توسط amfid گرفته میشود, اما با این مشخصه ما به amfid اعلام میکنیم که این فایل یکی از برنامه های سیستم عامل هست در نتیجه مشکلی برای اجرا شدن ندارد.

get-task-allow و task_for_pid_allow: این دو دستور بخشی از دستور های اصلی اپل برای دیباگ کردن برنامه ها هستند که به ما اجازه بارگذاری هر پروسه ای توسط یوزر روت را میدهند.

در واقع در زمانی که این فایل میخواهد اجرا شود چیز هایی که اتفاق می افتد به این شکل است:

۱. برنامه اجرا میشود
۲. AMFI.kext پروسه را نمی بندد زیرا فایل مورد نظر ساین شده است!
۳. وقتی که AMFI.kext میبیند امضاء برنامه از نوع Ad Hoc نیست و با نام خاصی ثبت شده است نتیجه میگیرد که این یکی از برنامه های ثبت شده در App Store است پس از amfid با استفاده از دستور معروف HOST_AMFID_PORT میخواهد که یکبار دیگر فایل را چک کند
۴. اگر amfid هنوز اجرا نشده باشد, launchd آنرا اجرا میکند و دستور را انتقال میدهد.

۵. amfid-imwit فایل libmis.dylib را اجرا میکند, اگر به خاطر داشته باشید این فایل در مرحله قبل توسط فایل مورد نظر ما جایگزین شده بود! پس دستور سالم بود امضای فایل را میدهد
۶. AMFI.kext هم با کمال میل اجازه میدهد که فایل ما با سطح دسترسی روت اجرا شود.

بار اولی که این فایل میخواهد اجرا شود به صورت taig –s اجرا میشود و این وقتی است که پیغام injecting the jailbreak program را روی برنامه مشاهده میکنید.

هشدار خیلی خیلی مهم: به هیچ عنوان این دستور را به صورت دستی و پس از جیلبرک اجرا نکنید! طبق تجربه شخصی taig آنقدر هوشمند نیست که بفهمد یکبار روی دستگاه نصب شده و نباید دوباره اجرا شود, اینکار باعث میشود تا این فایل دوباره ساخته شود ولی به صورت خالی! در تنیجه دستگاه دچار بوت لوپ میشود و حتی سرویس ssh هم اجرا نمیشود تا به داد شما برسد. و مجبور به ریستور به آخرین نسخه میشوید که جیلبرکی هم هنوز برای آن موجود نیست.

پروسه نصب شدن شامل کپی شدن فولدر taig/ که شامل دو فایل taig و lockdown_patch.dmg میشود.

```
Mahyars-iPhone:~ root# ls -lR /DeveloperPatch

/DeveloperPatch:

total 0

drwxrwxrwx 4 mobile staff 136 Oct 12 20:55 Library

drwxrwxrwx 4 mobile staff 136 Oct 21 00:19 bin


/DeveloperPatch/Library:

total 0

drwxrwxrwx 4 mobile staff 136 Nov 27  2013 Lockdown


/DeveloperPatch/Library/Lockdown:

total 0

drwxrwxrwx 4 mobile staff 136 Oct 21 19:54 ServiceAgents


/DeveloperPatch/Library/Lockdown/ServiceAgents:

total 4

-rwxrwxrwx 1 mobile staff 179 Oct 21 19:54 com.apple.afc2.plist


/DeveloperPatch/bin:

total 28

-rwxrwxrwx 1 mobile staff 24928 Oct 12 23:36 afcd2
```

afcd2 نسخه غیر محدود سرویس afc است که در مرحله ی اول تمام این اعمال را برای ما ممکن کرد.

محتویات فایل com.apple.afc2.plist:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
	<key>AllowUnauthenticatedServices</key>
	<true/>
	<key>Label</key>
	<string>com.apple.afc2</string>
	<key>ProgramArguments</key>
	<array>
		<string>/DeveloperPatch/bin/afcd2</string>
		<string>--lockdown</string>
		<string>-d</string>
		<string>/</string>
	</array>
</dict>
```

برای اینکه مطمئن شویم که فایل ما در هر بار بوت شدن دستگاه اجرا میشود فایل com.taig.untether.plist در launchd ایجاد میکنیم:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
	<key>Label</key>
	<string>com.taig.untether</string>
	<key>LaunchOnlyOnce</key>
	<true/>
	<key>POSIXSpawnType</key>
	<string>Interactive</string>
	<key>ProgramArguments</key>
	<array>
		<string>/taig/taig</string>
```

```
        </array>
        <key>RunAtLoad</key>
        <true/>
        <key>UserName</key>
        <string>root</string>
</dict>
</plist>
```

خوب کار جیلبرک با موفقیت به اتمام رسید! اینجاست که دستور ری استارت شدن به دستگاه داده میشود و پس از روشن شدن مجدد شما یک iDevice با جیلبرک untether خواهید داشت.

سخن پایانی

از همه ی شما خوانندگان عزیز ممنونم که تا اینجای مقاله را دنبال کردید, سعی میکنم در آینده نیز بتوانم با مطالب بهتر و کاملتر در خدمت شما باشم.

نیاز میبینم که نام افرادی که هم بنده در این مقاله از صحبت ها و مقالاتشان استفاده کردم و هم کمک شایانی به جامعه جیلبرک کردند را ذکر کنم:

@comex

@Technologeeks

@iH8sn0w

اگر خواستید بیشتر در جریان کار های آینده من قرار بگیرید میتوانید از اکانت توییتر من دیدن فرمایید:

@mahyar_rezghi

مهیار رزقی