

# گزارش تحقیق درباره‌ی SSL Pinning و نحوه‌ی دور زدن آن

تهیه و تنظیم: مبین خیبری

شماره دانشجویی: 994421017

استاد راهنما: دکتر میرسامان تاجبخش

## چکیده:

در گزارش پیش‌رو قصد داریم با نحوه‌ی عملکرد SSL Pinning آشنا شده و راهکارهایی را در خصوص دور زدن آن ارائه کنیم. این گزارش به کمک منابع پرشمار موجود در سطح اینترنت تهیه و تدوین شده که لیستی از مهم‌ترین این مراجع، در انتهای گزارش گنجانده شده است.

## SSL Pinning چیست؟

هنگامی که برنامه‌ها و اپلیکیشن‌های موبایل با یک سرویس دهنده ارتباط برقرار می‌کنند، معمولاً از SSL برای محافظت از اطلاعات در حال انتقال در شبکه در برابر استراق سمع، کپی برداری و دستکاری غیر مجاز استفاده می‌کنند. به طور پیش فرض، اجرای SSL مورد استفاده در برنامه‌های موبایل به این صورت خواهد بود که این برنامه‌ها برای حصول اطمینان از جعلی نبودن (trusted بودن) بودن یا نبودن یک سرور، گواهی‌نامه آن را در همان ابتدا با گواهی‌نامه‌های پیش فرض موجود در سیستم عامل گوشی مقایسه می‌کنند که اصطلاحاً محل ذخیره‌ی این گواهی‌نامه‌ها Trust Store نامیده می‌شود. این Trust Store فهرستی از گواهی‌نامه‌هایی است که به همراه سیستم عامل به صورت پیش فرض عرضه می‌گردد.



با این حال، با استفاده از SSL Pinning، تنظیمات اپلیکیشن به صورتی خواهد بود که به غیر از تعداد محدودی از گواهینامه های از پیش تعریف شده، باقی آن ها را رد کند. هر زمان که اپلیکیشن به یک سرور متصل شود، گواهی سرور را با گواهی های پین شده مقایسه می کند. فقط تنها زمانی که آنها مطابقت داشته باشند، سرور مورد اعتماد است و اتصال SSL برقرار می گردد.

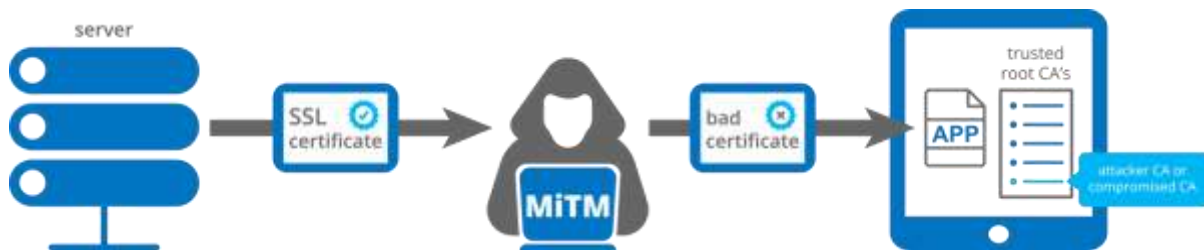


### چرا اپلیکیشن به SSL Pinning نیاز دارد؟

تنظیم و نگهداری جلسات (Session) های SSL (، معمولاً به یک library یا کتابخانه‌ی سیستم محول می گردد. این بدان معنا است که اپلیکیشنی که تلاش می کند یک ارتباط را با سرور برقرار نماید، قادر به تعیین قابل اعتماد بودن یا نبودن یک گواهینامه نمی باشد و به طور کامل بر روی گواهینامه هایی که در Trust Store سیستم عامل موجود است، تکیه می کند.

هنگامی که یک گواهینامه SSL تولید و در این Trust Store قرار میگیرد، می تواند یک حمله مرد میانی (Man-in-the-Middle) علیه هر برنامه ای که از SSL استفاده می کند را ایجاد کند، به طوری که هر جلسه (Session) از SSL قابل خواندن و دستکاری شود. در این حالت مهاجم می تواند از این فرصت برای معکوس کردن پروتکل استفاده شده در برنامه یا استخراج کلیدهای API از درخواست ها، استفاده نماید.

مهاجمان همچنین می توانند جلسات SSL را با فریب دادن کاربر و ترغیب وی به نصب CA از طریق یک صفحه وب مخرب و جعلی مورد حمله قرار دهند. همچنین ممکن است CA های اصلی و مورد اعتماد دستگاه، مورد حمله و دسترسی قرار گرفته و برای تولید گواهی جعلی استفاده شوند.



محدود کردن مجموعه ای از گواهی های قابل اعتماد از طریق پیاده سازی SSL Pinning ، به طور موثری سبب محافظت از برنامه های کاربردی موبایل در برابر حملات از راه دور شده می گردد. این امر همچنین از مهندسی معکوس و افزوده شدن CA سفارشی به Trust Store دستگاه و تحلیل عملکرد برنامه و نحوه ارتباط آن با سرور، جلوگیری می کند.

### پیاده سازی SSL Pinning در اپلیکیشن های تحت iOS

به صورت ساده پیاده سازی SSL Pinning از طریق ذخیره ی اطلاعات اضافی در داخل اپلیکیشن جهت شناسایی سرور و حصول اطمینان از عدم وجود حمله مرد میانی (Man-in-the-Middle)، صورت می گیرد.

### چه چیزی را پین کنیم؟

در واقع در این مرحله، کار به دو صورت می تواند انجام گیرد: اختصاص گواهینامه به سرور توسط خود سرور یا پین شدن کلید عمومی سرور. از طرفی شما خود انتخاب می کنید که داده های واقعی ذخیره گردند یا هش شده ی داده ها بکار گرفته شود. در واقع در این حالت می توان یک فایل هش شده از گواهی یا یک هش از رشته ی کلید عمومی را در اپلیکیشن ذخیره نمود.

انتخاب میان دو گزینه پین کردن گواهی یا کلید عمومی می تواند از جنبه های مختلف امنیت و نگهداری برنامه مورد بررسی قرار گیرد که این بحث در خارج از موضوع بحث این مطلب است. برای کسب اطلاعات بیشتر در این خصوص می توانید از [راهنماهای OWASP در این لینک](#) استفاده نمایید.

### نحوه جاسازی اطلاعات پین شده در برنامه

داده های مورد نیاز برای پیکربندی SSL می توانند در برنامه به دو روش کلی جاسازی شوند:

- جاسازی به صورت یک فایل دارایی (Asset File)
- جاسازی به عنوان یک رشته در کد برنامه

اگر از روش اول استفاده نمایید یعنی فایل گواهی را پین کنید، گواهی معمولاً به عنوان یک فایل دارای پین می‌گردد و هر بار که یک ارتباط SSL ساخته می‌شود، گواهی دریافت شده سرور با فایل (های) گواهی (های) شناخته شده مقایسه می‌شود و اگر فایل‌ها دقیقاً با یکدیگر مطابقت داشته باشند، اتصال مورد اعتماد خواهد بود و ارتباط برقرار می‌شود.

از طرفی با استفاده از روش دوم، هنگام پین کردن کلید عمومی سرور، کلید را می‌توان به عنوان یک رشته در کد برنامه تعبیه کرد یا آن را می‌توان در یک فایل دارای ذخیره نمود. در این حالت هر زمان که یک اتصال SSL ایجاد می‌شود، کلید عمومی از گواهی سرور دریافت شده استخراج می‌شود و با رشته ذخیره شده مقایسه می‌گردد. اگر با رشته‌ها دقیقاً مطابقت داشته باشند، اتصال مورد اعتماد است و ارتباط برقرار می‌شود.

### کتابخانه‌ها و روش‌های متداول برای SSL Pinning

کتابخانه‌های زیر گزینه‌های محبوب و متداولی برای پیاده‌سازی SSL در برنامه‌های Swift و زبان C می‌باشند.

لینک	نوع	زبان	روش و نحوه پین	نام
<a href="#">لینک</a>	Apple networking library	C شی گرا	Certificate file, public key	NSURLSession
<a href="#">لینک</a>	Networking library	Swift	Certificate file, public key	Alamofire
<a href="#">لینک</a>	Networking library	C شی گرا	Certificate file, public key	AFNetworking
<a href="#">لینک</a>	SSL Pinning	C شی گرا	public key	TrustKit

NSURLSession یک API ارائه شده از سوی اپل برای تسهیل ارتباطات شبکه است که یک فریم ورک سطح پایین می‌باشد، بنابراین پیاده‌سازی SSL با استفاده از آن سخت است و نیازمند بسیاری از تنظیمات به صورت دستی می‌باشد.

TrustKit، Alamofire و AFNetworking کتابخانه هایی هستند که به طور گسترده ای در بالای NSURLSession مورد استفاده قرار گرفته اند. هر دو AFNetworking و Alamofire کتابخانه های کامل شبکه هستند که از چگونگی پیکربندی SSL به عنوان بخشی از API خود پشتیبانی می کنند. TrustKit یک فریم ورک کوچک است که فقط بررسی پیاده سازی SSL Pinning را انجام می دهد. AFNetworking برای برنامه هایی با زبان شی گرای C و یا Alamofire برای برنامه های Swift انتخاب خوبی هستند. زمانی که شما به دنبال یک کتابخانه شبکه کامل هستید و فقط نیاز به SSL دارید، TrustKit می تواند گزینه مناسبی باشد.

با توجه به اینکه یکی از اولین اقداماتی که مهاجم در هنگام معکوس کردن کد و ساختار یک برنامه کاربردی موبایل انجام می دهد، دور زدن و عبور از SSL/TLS برای به دست آوردن اطلاعات و شناخت بهتر در عملکرد نرم افزار و نحوه ارتباط با سرور آن می باشد. در بخش بعد، به بررسی تکنیک های دور زدن و عبور از SSL Pinning در iOS و اقدامات متقابل هنگام وقوع آن، پرداخته می شود.

### روش های دور زدن مکانیزم امنیتی SSL Pinning در برنامه های موبایل

یکی از متداول ترین روش های امن سازی ارتباطات میان کلاینت و سرور بهره گیری از مکانیزم SSL می باشد. در خصوص اپلیکیشن های موبایل نیز برای استفاده از این راهکار، از مکانیزم SSL pinning استفاده می شود. در بخش قبلی ضمن معرفی این مکانیزم و شیوه عملکرد آن در امن سازی ارتباطات اپلیکیشن های موبایل، ضرورت استفاده و چگونگی پیاده سازی آن را معرفی کردیم. در این قسمت نیز روش های دور زدن این مکانیزم امنیتی با استفاده از یک اپلیکیشن نمونه و همچنین معرفی ابزارهای مربوطه توضیح داده شده و برخی پیشنهادات در خصوص چگونگی امن سازی اپلیکیشن های موبایل در برابر این روش ها ارائه گردیده است.

به طور کلی دور زدن مکانیزم امنیتی SSL Pinning توسط مهاجمین به یکی از دو روش زیر قابل انجام است:

1. از طریق جلوگیری از بررسی SSL پین شده و یا دستکاری نتیجه حاصل از این بررسی.
  2. از طریق جایگزینی داده های پین شده در اپلیکیشن، به عنوان مثال جایگزینی گواهی موجود در asset ها و یا کلید هش شده.
- در قسمت های بعدی، هر دو روش با استفاده از یک اپلیکیشن نمونه و همچنین معرفی ابزارهای مربوطه توضیح داده خواهد شد.

## آزمون و هدف

در ادامه به توضیح چگونگی دور زدن TrustKit SSL Pinning در نرم افزار نسخه دمو TrustKit که بر روی نسخه ی جیلبریک شده آیفون اجرا می گردد، پرداخته می شود. برای این کار، از ابزارهای زیر استفاده خواهیم کرد:

- از [mitmproxy](#) برای تجزیه و تحلیل داده های ارسالی در شبکه استفاده می شود که ابزارهای جایگزین آن [Burp Suite](#) یا [Charles](#) هستند.
  - ابزار [Frida](#) برای متدها و حملات hooking و patching استفاده می شود. از دیگر فریمورک های محبوب برای hooking میتوان به [Cydia Substrate](#)، [Cycrypt](#) یا [Substitute](#) اشاره نمود.
  - برای جایگزینی رشته ها در باینری، از ابزار [Disassembler Hopper](#) استفاده خواهیم کرد.
- طبیعتاً نرم افزار نسخه دمو TrustKit قابلیت کمتری نسبت به نسخه تجاری آن را دارد و تنها قابلیتی که ما از آن استفاده می کنیم، تلاش برای اتصال به <https://www.yahoo.com> با استفاده از یک هش پین نامعتبر برای آن دامنه می باشد.

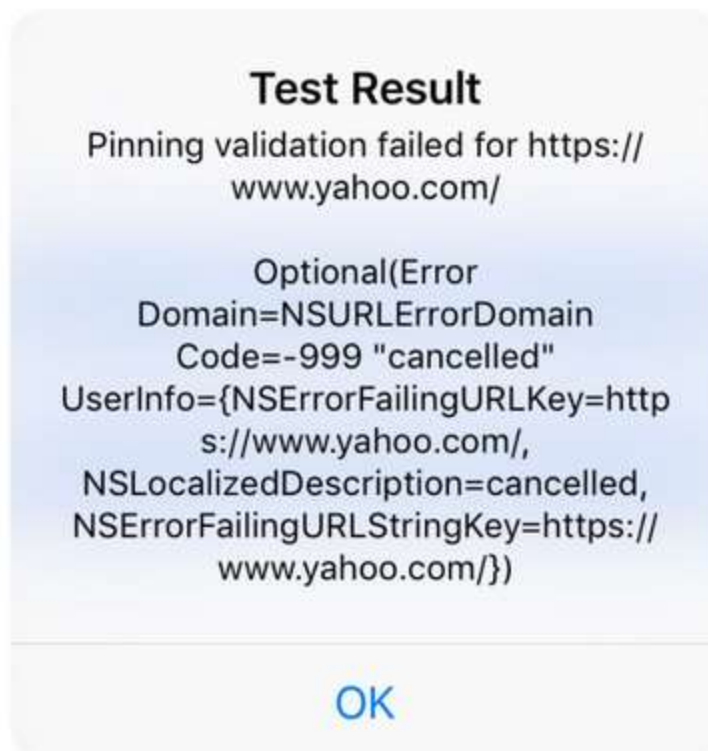
```
let trustKitConfig: [String: Any] = [
    kTSKSwizzleNetworkDelegates: false,
    kTSKPinnedDomains: [
        "yahoo.com": [
            kTSKEnforcePinning: true,
            kTSKIncludeSubdomains: true,
            kTSKPublicKeyAlgorithms: [kTSKAlgorithmRsa2048],

            // Invalid pins to demonstrate a pinning failure
            kTSKPublicKeyHashes: [
                "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA",
                "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB"
            ],
        ],
    ],
    kTSKReportUri: ["https://overmind.datatheorem.com/trustkit/report"],
],
```

توجه داشته باشید حتی اگر هشهای ارائه شده برای دامنه yahoo.com معتبر باشند، اعتبار سنجی SSL Pinning بایستی رد شود زیرا ما از پراکسی مردمیانی یا Man-in-the-Middle استفاده میکنیم.

هنگام اتصال به دامنه yahoo.com، ابزار mitmproxy نشان می دهد که دامنه در واقع بازدید نمی شود و فقط گزارش اعتبارسنجی SSL Pinning به سرورهای مورد نظر ارسال می گردد. از سوی دیگر دستگاه خود پیامی با محتوای رد شدن اعتبارسنجی Pinning نمایش می دهد. تمامی این رفتارها با توجه به فعال بودن SSL Pinning کاملاً طبیعی و قابل پیش بینی هستند.

```
POST https://overmind.datatheorem.com/trustkit/report
- 200 text/html [no content] 314ms
POST https://overmind.datatheorem.com/trustkit/report
- 200 text/html [no content] 225ms
```



### متد اول: جلوگیری از بررسی SSL پین شده

در ادامه به بررسی چگونگی دور زدن SSL Pinning با استفاده از ابزار Frida می پردازیم. اما پیش از آن بایستی بدانیم که در کدام قسمت از کد، در واقع چک کردن و بررسی SSL Pinning انجام می شود.

### پیدا کردن محل بررسی

از آنجا که TrustKit منبع باز است، بنابراین به راحتی می توان دریافت که منطق اعتبار سنجی گواهی واقعی در کجا می افتد. [TSKPinningValidator evaluateTrust:forHostname:]- در مواردی که سورس کد در دسترس نباشد، با یک بررسی دقیق تر از API کتابخانه SSL Pinning می توان دریافت فعالیت اصلی اعتبار سنجی در کجا اتفاق می افتد.

امضای evaluateTrust:forHostname: حاوی اطلاعات زیادی در خصوص متد مربوطه می باشد.

```
- (TSKTrustDecision) evaluateTrust : (SecTrustRef _Nonnull) serverTrust
forHostname : (NSString * _Nonnull)serverHostname
```

همان طور که مشاهده می شود این متد 2 ورودی شامل نام سروری (Hostname) که قصد اتصال به آن وجود دارد را شامل شده و در نهایت TSKTrustDecision را به عنوان خروجی باز می گرداند. این متغیر از نوع enum می باشد.

```
/**
 Possible return values when verifying a server's identity against a set of pins.
 */
typedef NS_ENUM(NSUInteger, TSKTrustEvaluationResult)
{
    TSKTrustEvaluationSuccess,
    TSKTrustEvaluationFailedNoMatchingPin,
    TSKTrustEvaluationFailedInvalidCertificateChain,
    TSKTrustEvaluationErrorInvalidParameters,
    TSKTrustEvaluationFailedUserDefinedTrustAnchor,
    TSKTrustEvaluationErrorCouldNotGenerateSpkiHash,
};
```

همان طور که در سورس کد مشاهده می گردد، هر یک از فیلدهای مربوطه مورد اشاره قرار گرفته است، اما روشن است که مهمترین مقدار در میان آنها، مقدار فیلد TSKTrustEvaluationSuccess می باشد.

## دور زدن بررسی SSL

برای دور زدن بررسی SSL pinning TrustKit، ما متد [TSKPinningValidator estimateTrust:forHostname:] را با استفاده از ابزار Frida، دستکاری (یا به اصطلاح hook) خواهیم کرد و اطمینان حاصل می کنیم که همیشه مقدار مورد نظر ما را برمی گرداند. در ابتدا اسکریپت مورد نیاز ابزار Frida را ایجاد می کنیم و آن را با عنوان disable\_trustkit.js ذخیره می کنیم.

```
var evalTrust = ObjC.classes.TSKPinningValidator["- evaluateTrust:forHostname:"];
Interceptor.attach(evalTrust.implementation, {
    onLeave: function(retval) {
        console.log("Current return value: " + retval);
        retval.replace(0);
        console.log("Return value replaced with (TSKTrustDecision) \
            TSKTrustDecisionShouldAllowConnection");
    }
});
```

این اسکریپت در واقع Frida را به متد evaluateTrust:forHostname: instance در محیط TSKPinningValidator متصل می کند و کد مربوطه را هر بار که این متد باز گردانده می شود، اجرا می



کند. این کد، بدون در نظر گرفتن مقدار قبلی و واقعی (TSKTrustEvaluationSuccess) ، آن را با مقدار صفر (0) جایگزین کرده و بر می گرداند.

حال Frida را اجرا نموده و به فرآیند TrustKitDemo موجود در دستگاهمان متصل نموده و اسکریپ را اجرا می نماییم:

`frida -U -l disable_trustkit.js -n TrustKitDemo-Swift.`

اکنون اگر `https://www.yahoo.com` را لود کنیم، می بینیم که در mitmproxy suite ، این URL با موفقیت بارگذاری شده است.

```
POST https://overmind.datatheorem.com/trustkit/report
  → 200 text/html [no content] 314ms
POST https://overmind.datatheorem.com/trustkit/report
  → 200 text/html [no content] 225ms
GET https://www.yahoo.com/
  → 302 text/html 17b 89ms
-> GET https://be.yahoo.com/?p=us
  → 200 text/html 85.18k 431ms
POST https://overmind.datatheorem.com/trustkit/report
  → 200 text/html [no content] 355ms
POST https://overmind.datatheorem.com/trustkit/report
  → 200 text/html [no content] 242ms

[4/6] :help [*:8080]
```

```
2018-03-05 10:22:56 GET https://be.yahoo.com/?p=us
  → 200 OK text/html 85.18k 431ms

Request: Response: Detail:
[decoded gzip] HTML [auto]
<!DOCTYPE html>
<html id="atomic" lang="nl-BE" class="atomic my3columns 1-out Pos-r https fp fp-v2 rc1 fp-default
mini-uh-on viewer-right two-col ntk-wide ltr desktop Desktop bkt919">
<head>
  <title>Yahoo</title>
  <meta http-equiv="x-dns-prefetch-control" content="on">
  <link rel="dns-prefetch" href="//s.yimg.com">
  <link rel="preconnect" href="//s.yimg.com">
  <link rel="dns-prefetch" href="//y.analytics.yahoo.com">
  <link rel="preconnect" href="//y.analytics.yahoo.com">
  <link rel="dns-prefetch" href="//geo.query.yahoo.com">
  <link rel="preconnect" href="//geo.query.yahoo.com">
  <link rel="dns-prefetch" href="//csc.beap.bc.yahoo.com">
  <link rel="preconnect" href="//csc.beap.bc.yahoo.com">
  <link rel="dns-prefetch" href="//geo.yahoo.com">
  <link rel="preconnect" href="//geo.yahoo.com">
  <link rel="dns-prefetch" href="//comet.yahoo.com">
  <link rel="preconnect" href="//comet.yahoo.com">

[4/6] :help :back [*:8080]
```

همچنین مطابق با شکل زیر در موبایل نیز این پیام را مشاهده می کنیم: تأیید پین با موفقیت انجام شده است.

## Test Result

Pinning validation succeeded for  
<https://www.yahoo.com/>

OK

همچنین، Frida خروجی زیر را برای حصول اطمینان از اینکه فرآیند دستکاری (hook) مطابق با انتظار ما عمل کرده و مقدار مطلوب را برگردانده است، ارائه می دهد.

```
[iPhone::TrustKitDemo-Swift]->
```

```
Current return value: 0x1
```

```
Return value replaced with (TSKTrustDecision)
```

```
TSKTrustDecisionShouldAllowConnection
```

```
Current return value: 0x1
```

```
Return value replaced with (TSKTrustDecision)
```

```
TSKTrustDecisionShouldAllowConnection
```

اکنون فرآیند دور زدن SSL Pinning TrustKit با موفقیت انجام شده و تمامی درخواست های وب قابل مشاهده و تغییر می باشند. البته مثال ارائه شده تنها یک نمونه ساده و ابتدایی از دور زدن SSL Pinning تنها با تغییر مقدار بازگشتی متد می باشد .

### استفاده از سایر ابزارها

دور زدن SSL Pinning را می توان حتی با استفاده از ترفندهای موجود برای موبایل های جیلبریک شده، از طریق روش های سادهتری انجام داد. برای مثال، SSL Kill Switch 2 پشته TLS در سیستم عامل iOS را پچ کرده و بدینوسیله کلید SSL پیاده سازی شده که از آن استفاده می نمایند را غیرفعال می کند. یکی از این ترفندها می باشد. از سوی دیگر ابزار [Objection SSL Pinning disabler](#) در Frida، بررسی های سطح پایین SSL Kill Switch 2 را اجرا می کند و چند نمونه هوک در اسن فریمورک را ایجاد می نماید.

جدول زیر متدهایی را که می توانند برای بعضی از فریمورک های SSL Pinning ، هوک شوند را تشریح می کند.

libcoretls_cfhelpers.dylib	tls_helper_create_peer_trust
NSURLSession	-[* URLSession:didReceiveChallenge:completionHandler:]
NSURLConnection	+[* connection:willSendRequestForAuthenticationChallenge:]
AFNetworking	-[AFSecurityPolicy setSSLPinningMode:] -[AFSecurityPolicy setAllowInvalidCertificates:] +[AFSecurityPolicy policyWithPinningMode:] +[AFSecurityPolicy policyWithPinningMode:withPinnedCertificates:]

## روش مقابله: تشخیص و شناسایی hooking

قبل از تأیید SSL Pin ، می توان به منظور شناسایی حملات هوکینگ، یکپارچگی و عدم دستکاری شدن یا تغییر غیرمجاز توابع فوق را مورد بررسی قرار داد. به عنوان مثال، از SSL Kill Switch 2 که در بالای فریمورک معروف Cydia Substrate جهت انجام حملات هوکینگ در زمان اجرا، ساخته شده است، استفاده خواهیم کرد. هوکینگ در این فریمورک از طریق MSHookFunction API انجام می شود.

روش توضیح داده شده در اینجا تنها یک اثبات مفهومی است و پیشنهاد می شود از کد شناسایی هوک که در این روش توضیح خواهیم داد، در نرم افزارهای تولیدی خود استفاده نکنید. در واقع این یک روش ساده است و تنها نوع خاصی از هوک را در ARM64 تشخیص می دهد. استفاده از این روش بررسی، بدون بهره گیری از مکانیزم های مبهم سازی کد (Obfuscation) ، حذف آن را بسیار آسان خواهد کرد.

یک روش معمول برای هوک کردن توابع اساسی (native) ، جایگزین نمودن چند دستور اولیه آنها با یک ترامپالین (Trampoline) است. ترامپالین به مجموعه ای از دستورها گفته می شود که مسئول انتقال جریان کنترل به یک قطعه کد جدید برای جایگزینی یا تقویت رفتار اولیه است. با استفاده از lldb ، می توانیم دقیقاً متوجه شویم "ترامپالین" چیست و چگونه به نظر می رسد.

10 دستور اول تابع اولیه (unhook) به شرح ذیل است:

```
(lldb) dis -n tls_helper_create_peer_trust
libcoretls_cfhelpers.dylib`tls_helper_create_peer_trust:
0x1a8c13514 <+0>: stp    x26, x25, [sp, #-0x50]!
0x1a8c13518 <+4>: stp    x24, x23, [sp, #0x10]
0x1a8c1351c <+8>: stp    x22, x21, [sp, #0x20]
0x1a8c13520 <+12>: stp    x20, x19, [sp, #0x30]
0x1a8c13524 <+16>: stp    x29, x30, [sp, #0x40]
0x1a8c13528 <+20>: add    x29, sp, #0x40
0x1a8c1352c <+24>: sub    sp, sp, #0x20
0x1a8c13530 <+28>: mov    x19, x2
0x1a8c13534 <+32>: mov    x24, x1
0x1a8c13538 <+36>: mov    x21, x8
```

10 دستور اول تابع هوک شده به شرح ذیل است:

```
(lldb) dis -n tls_helper_create_peer_trust
libcoretls_cfhelpers.dylib`tls_helper_create_peer_trust:
0x1a8c13514 <+0>: ldr    x16, #0xc8                ; <+8>
0x1a8c13518 <+4>: br     x16
0x1a8c1351c <+8>: .long 0x00267c2c                ; unknown opcode
0x1a8c13520 <+12>: .long 0x00000081                ; unknown opcode
0x1a8c13524 <+16>: stp    x29, x30, [sp, #0x40]
0x1a8c13528 <+20>: add    x29, sp, #0x40           ; <+0x40
0x1a8c1352c <+24>: sub    sp, sp, #0x20           ; <+0x20
0x1a8c13530 <+28>: mov    x19, x2
0x1a8c13534 <+32>: mov    x24, x1
0x1a8c13538 <+36>: mov    x21, x0
```

در تابع هوک شده، 16 بایت اول، ترامپلین را تشکیل می دهند. آدرس 0x00000001002ebc2c در رجیستر x16 بارگذاری می شود و سپس به آن آدرس می رود. (BR X16) این آدرس به المان زیر اشاره می کند:

SSLKillSwitch2.dylib`replaced\_tls\_helper\_create\_peer\_trust

همان گونه که مشاهده می شود، در آن SSL Kill Switch 2 جایگزین شده است.

```
(lldb) dis -a 0x00000001002ebc2c
SSLKillSwitch2.dylib`replaced_tls_helper_create_peer_trust:
0x1002ebc2c <+0>: sub    sp, sp, #0x20                ; <+0x20
0x1002ebc30 <+4>: mov    w8, #0x0
0x1002ebc34 <+8>: strb   x0, [sp, #0x10]
0x1002ebc38 <+12>: strb   w1, [sp, #0x17]
0x1002ebc3c <+16>: str    x2, [sp, #0x8]
0x1002ebc40 <+20>: mov    x0, x8
0x1002ebc44 <+24>: add    sp, sp, #0x20                ; <+0x20
```

اگر پیاده سازی تابع از پیش مشخص شده باشد، چند بایت اول از تابع یافت شده را می توان با بایت های مشخص شده مقایسه کرد. بدین ترتیب می توان بدون نقض Pinning را اجرا نمود. در خصوص Cydia Substrate، مشاهده می شود که تابع با استفاده از یک برنچ غیرشرطی به یک رجیستر (BR Xn) پچ شده است، در این حالت می توانیم وجود این دستور را در چند بایت اول بررسی نماییم. در صورتیکه دستور برنچ یافت شد، فرض بر این است که تابع هوک شده و در غیر اینصورت تابع معتبر است.

**روش مقابله: مبهم سازی اسمی (Name Obfuscation)**

همان طور که در بالا دیدیم، برای دور زدن مکانیزم SSL Pinning، نفودگر ابتدا باید بفهمد که کدام مکانیزم را باید هوک کند. با استفاده از یک ابزار مبهم سازی (Obfuscation) متادیتاهای اپلیکیشن های

iOS نوشته شده به زبان Swift و Objective-C، برنامه نویسان می توانند این تشخیص را برای نفوذ گر بسیار دشوار سازند.

مبهم سازی اسامی همچنین قادر است، مانع از عملکرد صحیح تمامی ابزارهای خودکار برای جستجوی نام متدهای شناخته شده شود. یک ابزار مبهم سازی می تواند متدها را به شیوه ای متفاوت در ساختار هر یک از نسخه های اپلیکیشن تغییر دهد، به گونه ای که نفوذگر را مجبور به جستجوی نام واقعی متدها در هر نسخه جدید گرداند.

توجه داشته باشید که مبهم سازی اسامی فقط قادر است از برنامه در مقابل ابزارهای دور زدن SSL که در آنها، گواهی SSL در کد برنامه یا کتابخانه های برنامه جاگذاری شده است، محافظت نماید. ابزارهایی که با فریم ورک های هوکینگ سیستم کار می کنند، با این روش قابل جلوگیری نمی باشند.

### متد دوم: جایگزینی داده های SSL Pinning

راه دیگر برای دور زدن SSL Pinning این است که داده های پین شده را در داخل برنامه جایگزین نماییم. اگر ما قادر به جایگزینی فایل گواهی پین شده اصلی یا رشته کلید عمومی با گواهی یا کلید موجود در سرور مردمیانی (Man-in-the-Middle) خودمان باشیم، قاعدتا میتوانیم سرور خودمان را به جای سرور اصلی پین کنیم.

جایگزینی یک فایل گواهی جاسازی شده می تواند به آسانی جایگزینی یک فایل در پکیج IPA باشد.

در اپلیکیشن هایی که هش کلید عمومی سرور را پین می کنند، می توانیم رشته را با هش کلید عمومی سرور خودمان جایگزین کنیم. تصویر زیر نشان می دهد که چگونه برنامه Demo TrustKit در ابزار Hopper بارگذاری شده است. Hopper به مهاجم اجازه می دهد که رشته ها را در فایل MachO جایگزین نموده و آن را مجدداً به شکل یک فایل قابل اجرا و معتبر تبدیل نماید.



هنگامی که فایل یا رشته جایگزین می شود، دایرکتوری تحت عملیات باید مجدداً به صورت یک IPA امضا و زیپ شود. این موضوع خارج از محدوده بحث این وبلاگ قرار دارد، اما اطلاعات بیشتر را می توانید در [اینجا](#) پیدا نمایید.

## روش مقابله: مبهم سازی رشته ها (String Obfuscation)

هنگام پین کردن گواهینامه ها با یک لیست از هش های کلید عمومی hard-code شده، بهتر است که مقادیر و ارزش ها را رمزگذاری کنید. این عمل در واقع اپلیکیشن شما را در برابر حملات hooking محافظت نخواهد کرد، اما نفوذگر را برای جایگزینی هش های اصلی با گواهی ساختگی از سوی او، با مشکل جدی رو به رو می سازد. مبهم سازی و رمزگذاری رشته ها (مقادیر) در این بخش قابل استفاده می باشد. ابزارهای [DexGuard](#) برای اپلیکیشن های اندرویدی و همچنین [iXGuard](#) برای برنامه های iOS می توانند مبهم سازی رشته های (حساس) مدنظر برنامه نویس را انجام دهند.

## روش مقابله: مبهم سازی جریان کنترل (Control Flow Obfuscation)

یک مهاجم با استفاده از تکنیک های مهندس معکوس قادر است جریان و روند کنترل برنامه را تجزیه و تحلیل نموده تا بتواند از این طریق محل دقیقی که در آن، برنامه هش واقعی را تایید می کند، پیدا نماید. اگر او موفق به پیدا کردن این محل گردد، می تواند ببیند که کدام رشته مورد استفاده قرار گرفته و همچنین می تواند محل رشته هش در باینری را پیدا کند. مبهم سازی جریان کنترل برنامه توسط برنامه نویس، باعث می شود تا تجزیه و تحلیل دستی از کد برای نفوذگر بسیار مشکل گردد. ابزارهای [DexGuard](#) و همچنین [iXGuard](#) می توانند مبهم سازی جریان کنترلی نرم افزار را انجام دهند.

## متصل کردن گواهی (Certificate Pinning) به برنامه اندروید یا ios خود

وقتی ما توسعه دهندگان در حال توسعه هر نوع نرم افزاری هستیم، نمی توان امنیت را فراموش کرد. حداقل معیار امنیتی که باید از آن استفاده کنیم HTTPS به عنوان پروتکل برای به اشتراک گذاری اطلاعات بین مشتری (در این حالت یک برنامه اندروید/ios) و یک سرور است؛ به دنبال آن یک پروتکل رمزنگاری بروز شده مانند TLS 1.2 (SSL 3.0) آسیب پذیر است!

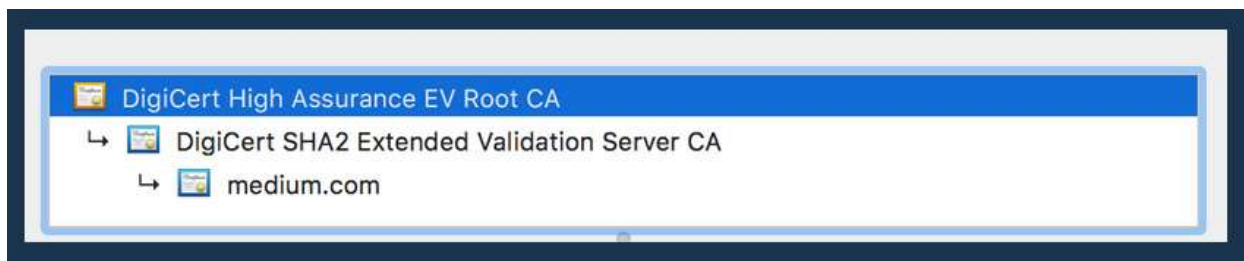
ممکن است فکر کنید که استفاده از HTTPS کافی است اما در بعضی موارد مانند برنامه های بانکی، که ممکن است داده های حساس بین مشتری و سرور ارسال شود، می تواند خطرناک باشد.

به طور پیش فرض هنگام برقراری اتصال TLS، کلاینت دو مورد را بررسی می کند:

- گواهی سرور با نام میزبان درخواستی مطابقت دارد
- گواهی سرور دارای زنجیره ای از حقایق براساس گواهی ریشه است

آنچه انجام نمی دهد بررسی این است که آیا این گواهی، گواهی خاصی است که می دانید سرور از آن استفاده می کند، و این یک آسیب پذیری امنیتی احتمالی است: اگر کلاینت به خطری افتد و یک گواهی ناامن نصب شود، کسی می تواند یک حمله بین آن انجام دهد.





راه حل این مشکل، متصل کردن گواهی است: ذخیره گواهی در کلاینت برای اطمینان از اینکه هر درخواست SSL ساخته شده مطابق با سرور ما است. بگذارید چگونگی این کار را در هر دو برنامه اندروید و ios برای شما توضیح دهم.

## اندروید

کتابخانه [Okhttp](#) یک کلاس CertificatePinner ارائه می دهد تا به نمونه OkHttpClient اضافه شود. آسان ترین راه برای متصل کردن به میزبان این است که pinning را بر روی پیکربندی نقض شده فعال کنیم و در صورت عدم موفقیت اتصال پیکربندی مورد انتظار خوانده شود.

```
1 CertificatePinner certificatePinner = new CertificatePinner.Builder()
2     .add("mydomain.com", "sha256/AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=")
3     .build();
4 OkHttpClient client = OkHttpClient.Builder()
5     .certificatePinner(certificatePinner)
6     .build();
```

پس از اجرای یک درخواست، این پیام را روی کنسول مشاهده خواهید کرد:

```
javax.net.ssl.SSLPeerUnverifiedException: Certificate pinning failure!
```

Peer certificate chain:

```
sha256/afwiKY3RxoMmLkuRW1I7QsPZTJPwDS2pdDROQjXw8ig=: CN=mydomain.com, OU=PositiveSSL
```

```
sha256/kIO23nT2ehFDXCfx3eHTDRESMz3asj1muO+4aldjiuY=: CN=COMODO RSA Secure Server CA
```

```
sha256/grX4Ta9HpZx6tSHkmCrvpApTQGo67CYDnvprLg5yRME=: CN=COMODO RSA Certification Authority
```

```
sha256/ICppFqbkrIJ3EcVFAkeip0+44VaoJUymbnOaEUK7tEU=: CN=AddTrust External CA Root
```

Pinned certificates for mydomain.com:

```
sha256/AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=
```

```
at okhttp3.CertificatePinner.check(CertificatePinner.java)
```

```
at okhttp3.Connection.upgradeToTls(Connection.java)
```

```
at okhttp3.Connection.connect(Connection.java)
```

```
at okhttp3.Connection.connectAndSetOwner(Connection.java)
```

این exception باعث می شود شناسه کلید عمومی گواهی سرور برای شما فراهم شود. آن ها را روی CertificatePinner بچسبانید و تمام!

```
1 CertificatePinner certificatePinner = new CertificatePinner.Builder()
2     .add("mydomain.com", "sha256/afwiKY3RxoMmLkuRW1l7QsPZTJPwDS2pdDR0QjXw8ig=")
3     .add("mydomain.com", "sha256/k1023nT2ehFDXCfx3eHTDRESMz3asj1mu0+4aIdjiuY=")
4     .add("mydomain.com", "sha256/grX4Ta9HpZx6tSHkmCrvpApTQGo67CYDnvprLg5yRME=")
5     .add("mydomain.com", "sha256/lCppFqbkr1J3EcVFAkeip0+44VaoJUymbn0aEUk7tEU=")
6     .build();
```

## ios

راه حل ios چندان ساده نیست، زیرا باید خود گواهی را درون برنامه خود ذخیره کنید. من از Alamofire به عنوان کتابخانه HTTP در سوئیفت استفاده می کنم.

ابتدا باید گواهی سرور را با فرمت der دریافت کنید و آن را به پروژه ios خود اضافه کنید.

```
1 openssl s_client -showcerts -servername mydomain.com -connect mydomain.com:
2 443 </dev/null | openssl x509 -outform DER > mydomainCert.der
```

و اکنون بیاید اتصال گواهی را فعال کنیم: برای انجام این کار به هر دو شیء ServerTrustPolicy و SessionManager نیاز داریم. مورد اول نام میزبان و گواهی هایی که در این فرایند استفاده می شوند را تعریف می کند.

```
1 var serverTrustPolicies = [
2     "mydomain.com": .pinCertificates(
3         certificates: ServerTrustPolicy.certificates(),
4         validateCertificateChain: true,
5         validateHost: true
6     ),
7 ]
```



ServerTrustPolicy.certificates() کليه گواهی‌های ذخيره شده را برمی‌گرداند و booleanها زنجيره گواهی و نام ميزبان را تاييد می‌کنند.

در آخر با استفاده trust policy یک شیء SessionManger ایجاد کنید:

```
1 var sessionManager = SessionManager(serverTrustPolicyManager:  
2   ServerTrustPolicyManager(policies: serverTrustPolicies!))
```

تمام شد! برای اجرای درخواست فقط از این شیء sessionManger استفاده کنید.

```
sessionManager.request("https://mydomain.com/api", method: .get, headers: headers)...
```

## روش جدید برای پیاده‌سازی آن در ios

TrustKit یک کتابخانه منبع‌باز برای پیاده‌سازی اتصال SSL است. این یک راه‌حل آسان‌تر و انعطاف‌پذیرتر نسبت به استفاده از ServerTrustPolicy است.

هنگامی که یکبار TrustKit در برنامه ادغام شد، ما فقط باید آن را فعال کنیم، به عنوان مثال در فایل AppDelegate.

```
let trustKitConfig = [  
    kTSKEnforcePinning: true,  
    kTSKIncludeSubdomains: true,  
    kTSKSwizzleNetworkDelegates: false,  
    kTSKPinnedDomains: [  
        "mydomain.com": [  
            kTSKPublicKeyAlgorithms: [kTSKAlgorithmRsa2048, kTSKAlgorithmRsa4096],  
            kTSKPublicKeyHashes: [  
                "afwiKY3RxoMmLkuRW1l7QsPZTJPwDS2pdDROQjXw8ig=",  
                "kIO23nT2ehFDXCfx3eHTDRESMz3asj1muO+4aldjiuY=",
```

```
"grX4Ta9HpZx6tSHkmCrvpApTQGo67CYDnvprLg5yRME="
```

```
],
```

```
]
```

```
]
```

```
] as [String : Any]
```

```
TrustKit.initSharedInstance(withConfiguration:trustKitConfig)
```

بدین ترتیب، اجرای آن مانند اندروید است: ما گواهی der و ServerTrustPolicy را فراموش می‌کنیم و اکنون از کلیدهای عمومی استفاده می‌کنیم، که راهی انعطاف‌پذیرتر برای پیاده‌سازی اتصال گواهی است زیرا نیازی به فایل به روز شده گواهی نداریم.

منابع:

- i. <https://www.ashnasecure.com/blog/post/158/%D8%B1%D9%88%D8%B4%D9%87%D8%A7%DB%8C%D8%AF%D9%88%D8%B1%D8%B2%D8%AF%D9%86%D9%85%DA%A9%D8%A7%D9%86%DB%8C%D8%B2%D9%85%D8%A7%D9%85%D9%86%DB%8C%D8%AA%DB%8C%20SSL%20Pinning%D8%AF%D8%B1%D8%A8%D8%B1%D9%86%D8%A7%D9%85%D9%87%D9%87%D8%A7%DB%8C%D9%85%D9%88%D8%A8%D8%A7%DB%8C%D9%84>
- ii. <https://roocket.ir/articles/certificate-pinning-your-android-and-ios-apps>
- iii. <https://medium.com/@anuj.rai2489/ssl-pinning-254fa8ca2109>
- iv. <https://www.indusface.com/learning/what-is-ssl-pinning-a-quick-walk-through/>
- v. [https://www.ashnasecure.com/blog/post/154/SSL Pinning: %D8%B1%D8%A7%D9%87%DA%A9%D8%A7%D8%B1%D8%A7%D9%85%D9%86%DB%8C%D8%AA%D8%A7%D8%B1%D8%AA%D8%A8%D8%A7%D8%B7%D8%A7%D8%AA%D8%A7%D9%BE%D9%84%DB%8C%DA%A9%DB%8C%D8%B4%D9%86%D9%85%D9%88%D8%A8%D8%A7%DB%8C%D9%84](https://www.ashnasecure.com/blog/post/154/SSL%20Pinning:%D8%B1%D8%A7%D9%87%DA%A9%D8%A7%D8%B1%D8%A7%D9%85%D9%86%DB%8C%D8%AA%D8%A7%D8%B1%D8%AA%D8%A8%D8%A7%D8%B7%D8%A7%D8%AA%D8%A7%D9%BE%D9%84%DB%8C%DA%A9%DB%8C%D8%B4%D9%86%D9%85%D9%88%D8%A8%D8%A7%DB%8C%D9%84)

پایان.