RSAConference2019
Asia Pacific & Japan
Singapore | 16–18 July | Marina Bay Sands

SESSION ID: **CMI-W02**

Insights from the trenches Must-have secure coding lessons in mobile

Yair Amit

VP, Symantec
CTO & Co-Founder, Skycure

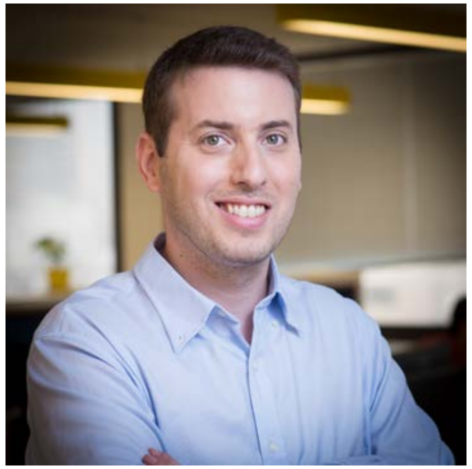
Igal Kreichman

Dev Manager, Symantec

Meet the Speakers

Symantec
Skycure
Google
IBM
IDF 8200

Igal Kreichman
Dev Manager, Symantec



Symantec
Skycure
IBM
watchfire
IDF 8200

Yair Amit
VP, Symantec
CTO & Co-founder, Skycure

RSA Conference2019
Asia Pacific & Japan

Agenda

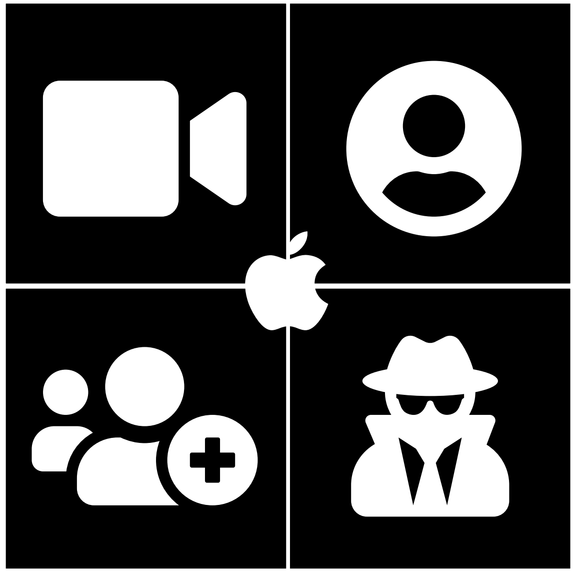
- OS-level pitfalls
- IDE-level pitfalls
- **App-level pitfalls**
- Live demo: exploiting a few consecutive pitfalls
- Summary

RSAConference2019 Asia Pacific & Japan

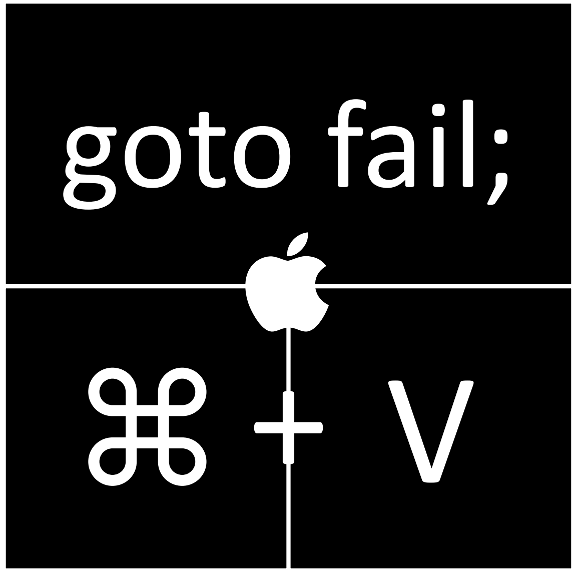
OS Level Risks

OS level bugs (Apple)

FaceTime



gotofail



OS level bugs: GoToFail

```
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer signedParams,
                                 uint8_t *signature, UInt16 signatureLen) {
    …
    if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
        goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;
    err = sslRawVerify(ctx,
                       ctx->peerPubKey,
                       dataToSign,                /* plaintext */
                       dataToSignLen,             /* plaintext length */
                       signature,
                       signatureLen);
    …
fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;
}
```

Always goto "fail", even if err==0

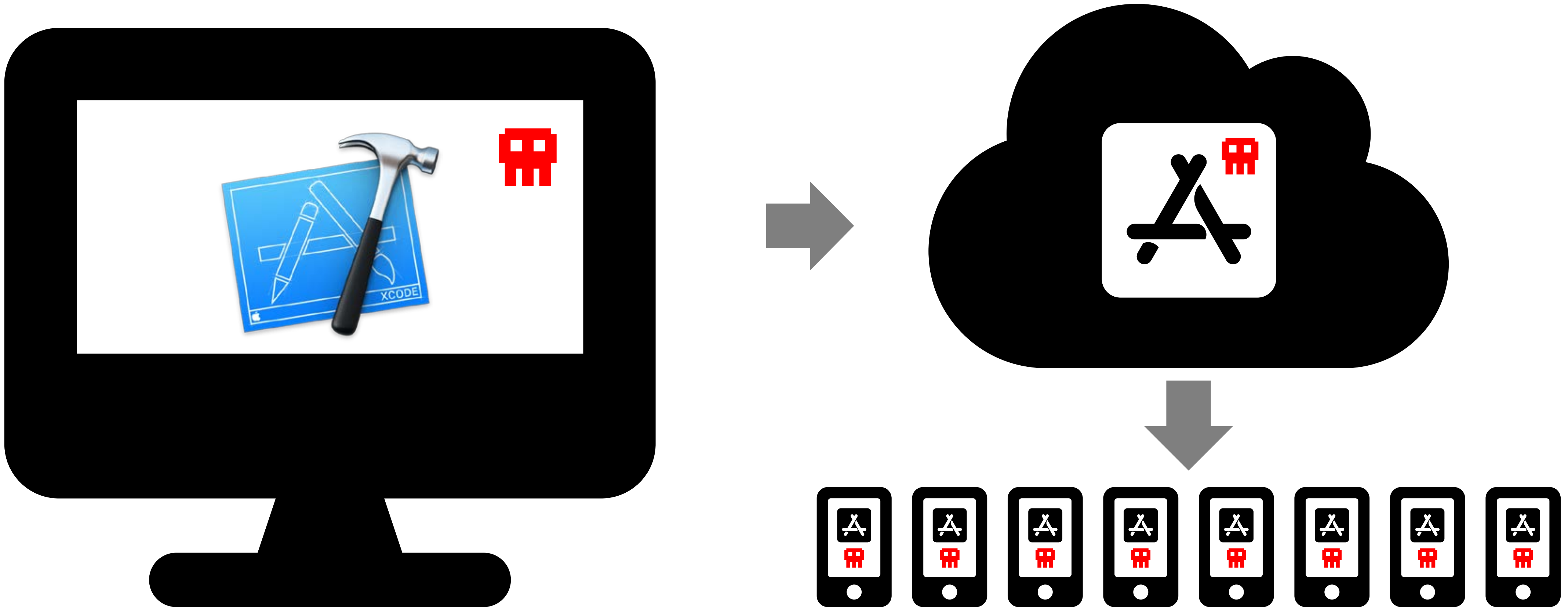
Code is skipped (even though err == 0)

Function returns 0 (i.e. verified), even though sslRawVerify was not called

RSAConference2019
Asia Pacific & Japan

IDE Level Risks

Malicious/Fake IDE: XcodeGhost

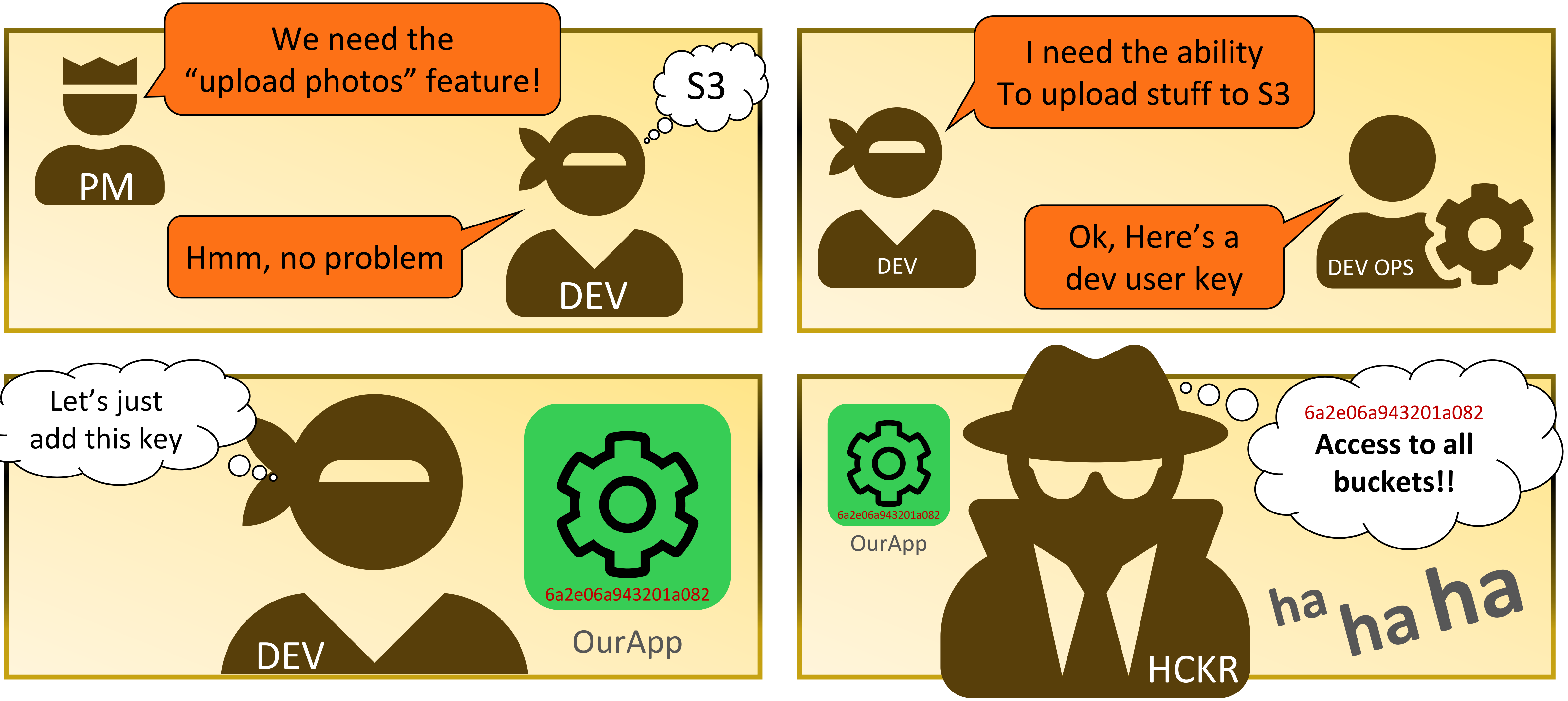


RSAConference2019 Asia Pacific & Japan

Coding Pitfalls

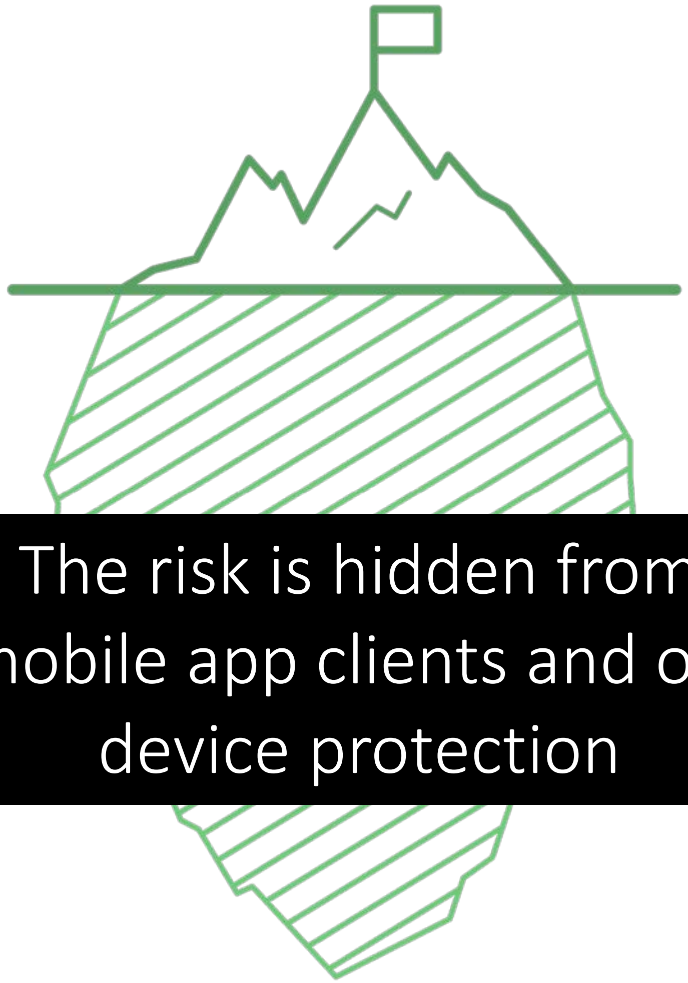
Exposed Backend

How often do you see this happen?



The hidden iceberg

Data security and modern servers lack of security



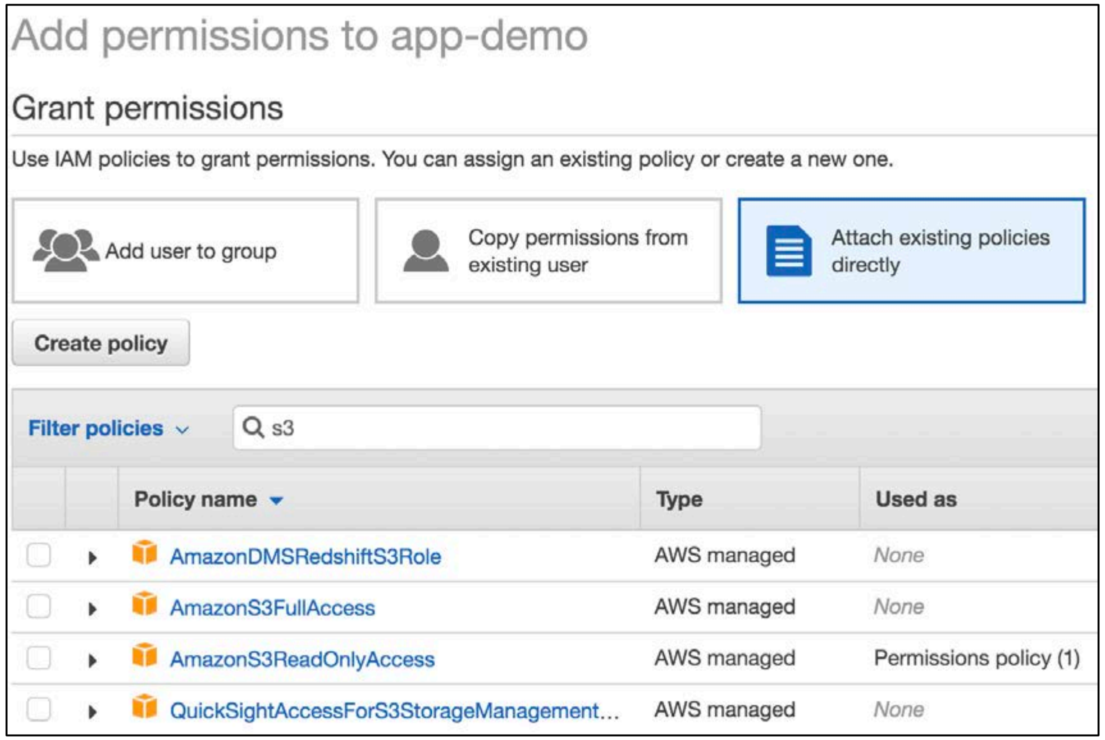
The risk is hidden from mobile app clients and on-device protection

- Mobile apps routinely offload data to cloud based data servers
- Most popular datastores used by mobile apps (in order) from sample size of:
  - Amazon AWS S3
  - Google Firebase
  - Microsoft Azure Storage
  - Elasticsearch, Twilio, MySQL, CouchDB

The largest mobile data leak to date: ~100TB of data and 500+ million records

Power of the default

Default policies for S3 access



Add permissions to app-demo

Grant permissions

Use IAM policies to grant permissions. You can assign an existing policy or create a new one.

Add user to group
Copy permissions from existing user
Attach existing policies directly
Create policy
Filter policies
s3

		Policy name	Type	Used as
☐	▸	AmazonDMSRedshiftS3Role	AWS managed	*None*
☐	▸	AmazonS3FullAccess	AWS managed	*None*
☐	▸	AmazonS3ReadOnlyAccess	AWS managed	Permissions policy (1)
☐	▸	QuickSightAccessForS3StorageManagement...	AWS managed	*None*

How the default write access policy looks

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:*",
            "Resource": "*"
        }
    ]
}
```

Best practices

Pre-signed URLs (client-server)

1. On the server side, generate a pre-signed URL for a file to be uploaded.
2. Share with the app.

Pros

- High isolation
- Temporary access (due to expiration)
- Very low risk exposure

Cons

- Overhead: requires an interaction mechanism with the server before each upload

Best practices

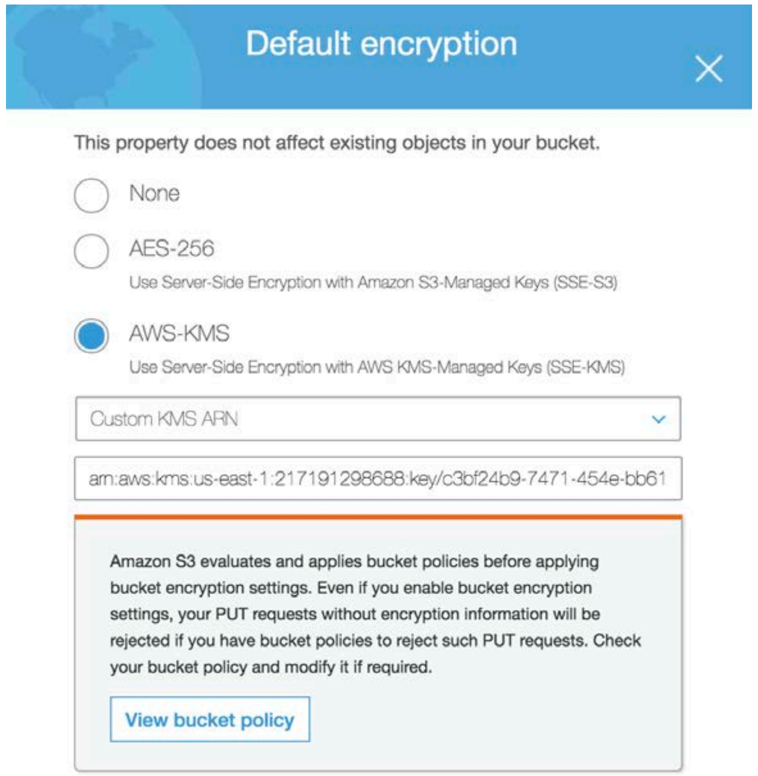
Custom policy (client-side)

1. Define a custom policy for the mobile-app user with access to a specific bucket & write-file only.
2. Use the key of the dedicated mobile-app user via the app to upload files.
3. Use a different user (==higher permissions) for the server side.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::demo-app-uploads/*"
        }
    ]
}
```

In order to keep this path safe:

- **Overwrite risk:** generate randomized file names
- **Mitigate others' errors:** use default encryption to prevent other from getting accidental access



Default encryption

This property does not affect existing objects in your bucket.

None

AES-256
Use Server-Side Encryption with Amazon S3-Managed Keys (SSE-S3)

[✓ AWS-KMS]
Use Server-Side Encryption with AWS KMS-Managed Keys (SSE-KMS)

Custom KMS ARN

arn:aws:kms:us-east-1:217191298688:key/c3bf24b9-7471-454e-bb61

Amazon S3 evaluates and applies bucket policies before applying bucket encryption settings. Even if you enable bucket encryption settings, your PUT requests without encryption information will be rejected if you have bucket policies to reject such PUT requests. Check your bucket policy and modify it if required.

View bucket policy

Symantec

RSAConference2019 Asia Pacific & Japan

RSAConference2019 Asia Pacific & Japan

Coding Pitfalls

Media File Jacking / Insecure use of device shared storage

Device storage options in Android

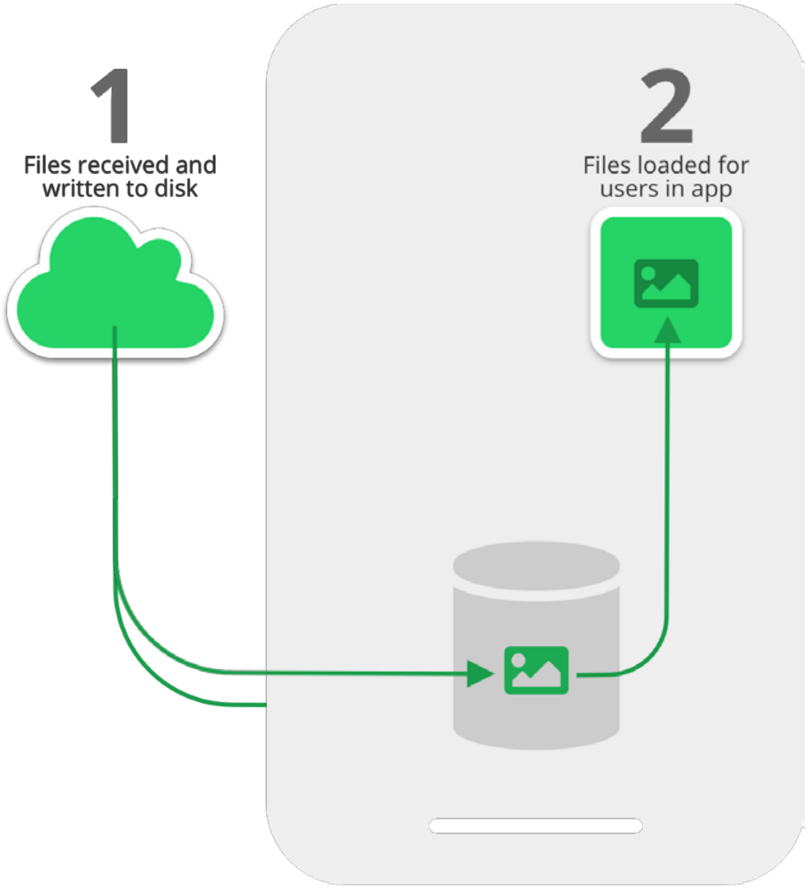
- How to use different storage types in Android
  - “Internal storage is best when you want to be sure that neither the user nor other apps can access your files.”
  - By contrast, “external storage is the best place for files that don't require access restrictions and for files that you want to share with other apps or allow the user to access with a computer.”
- The big problem starts when external storage is used for tasks better suited for internal storage…

Modern Instant Messaging

- Classic Instant Messaging (IM), such as SMS, are known to be susceptible to sender and content spoofing.
- Modern IM apps utilize end-to-end encryption - an important mechanism to ensure the integrity of communications.
  - However, users are not safe if app-level vulnerabilities exist in the code.

WhatsApp & Telegram – Media File Jacking

- WhatsApp for Android stores media unencrypted in external storage by default
  - Telegram does so if “Save to Gallery” is enabled
- The mechanism:
  - Store the media files to disk
  - Load the media files from disk when the user engages with the app
- The problem:
  - Attacker’s malware can manipulate the media during the time-lapse!


1
Files received and written to disk
2
Files loaded for users in app

RSAConference2019
Asia Pacific & Japan

Demo

Video manipulation

RSAConference2019 Asia Pacific & Japan

Demo

Invoice manipulation

How can this problem be avoided?

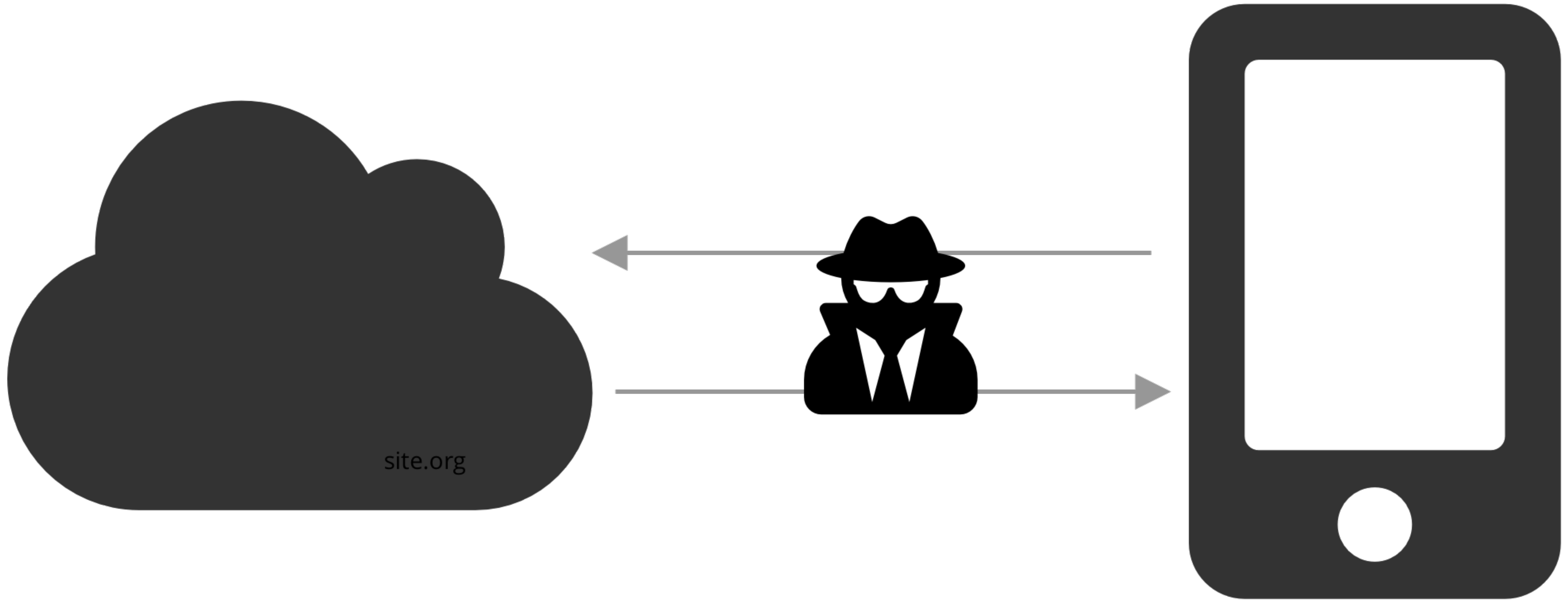
- Internal storage
  - Limit risk from malware by limiting access to the resources
- Encryption
  - Store in the external storage but encrypt to avoid modifications
- Media Hash
  - Generate an hash per file, store in metadata, don't load it if it was modified

RSAConference2019
Asia Pacific & Japan

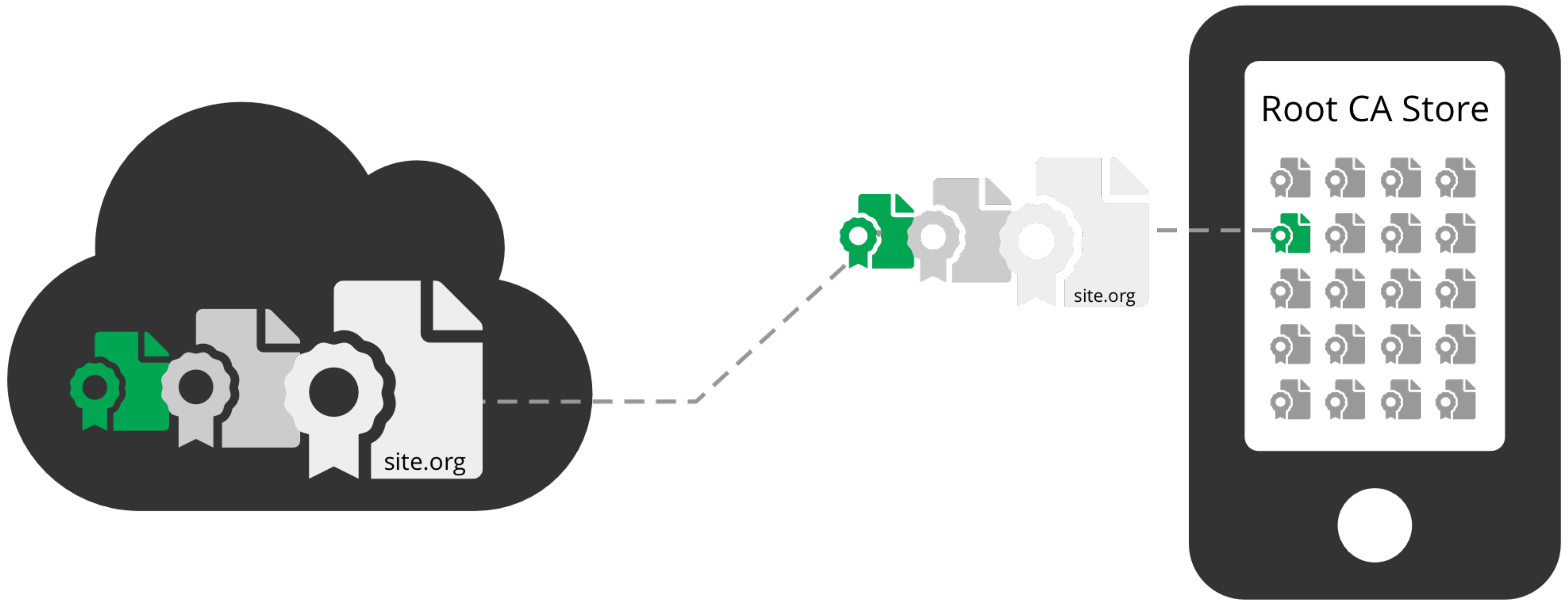
Coding Pitfalls

SSL Pinning

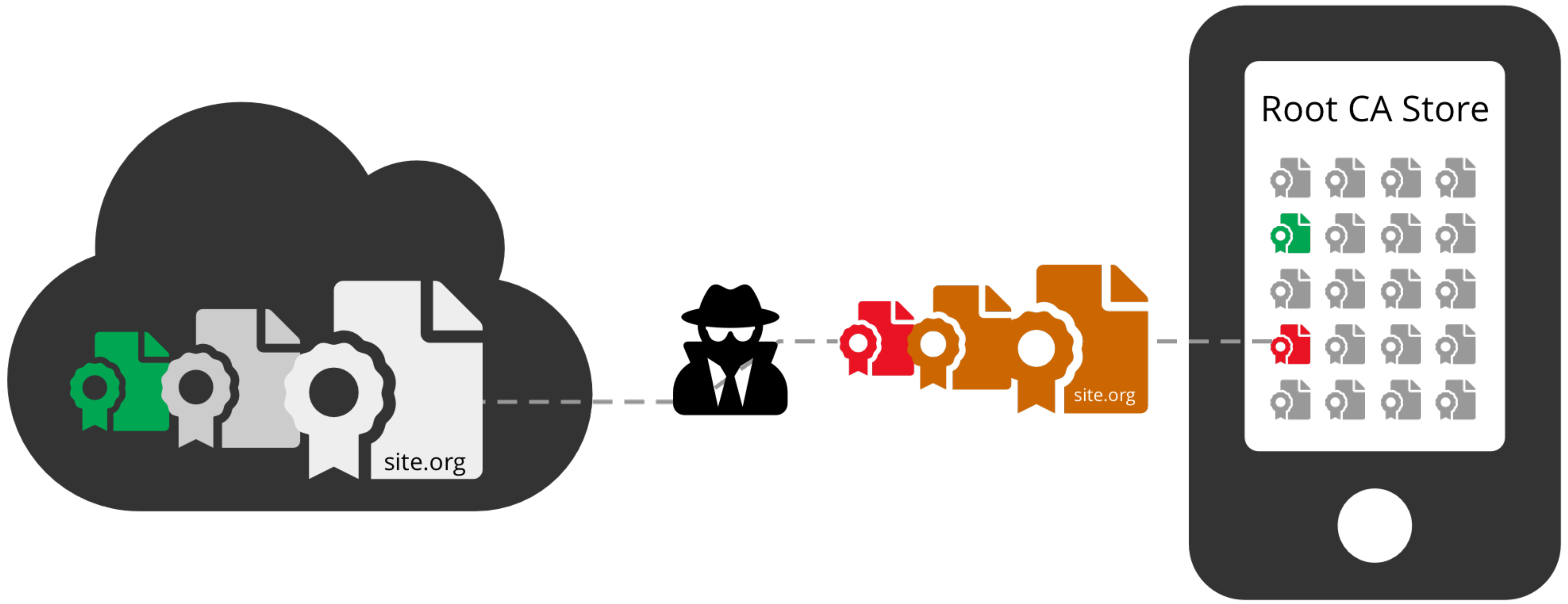
MITM



Certificates chain of trust



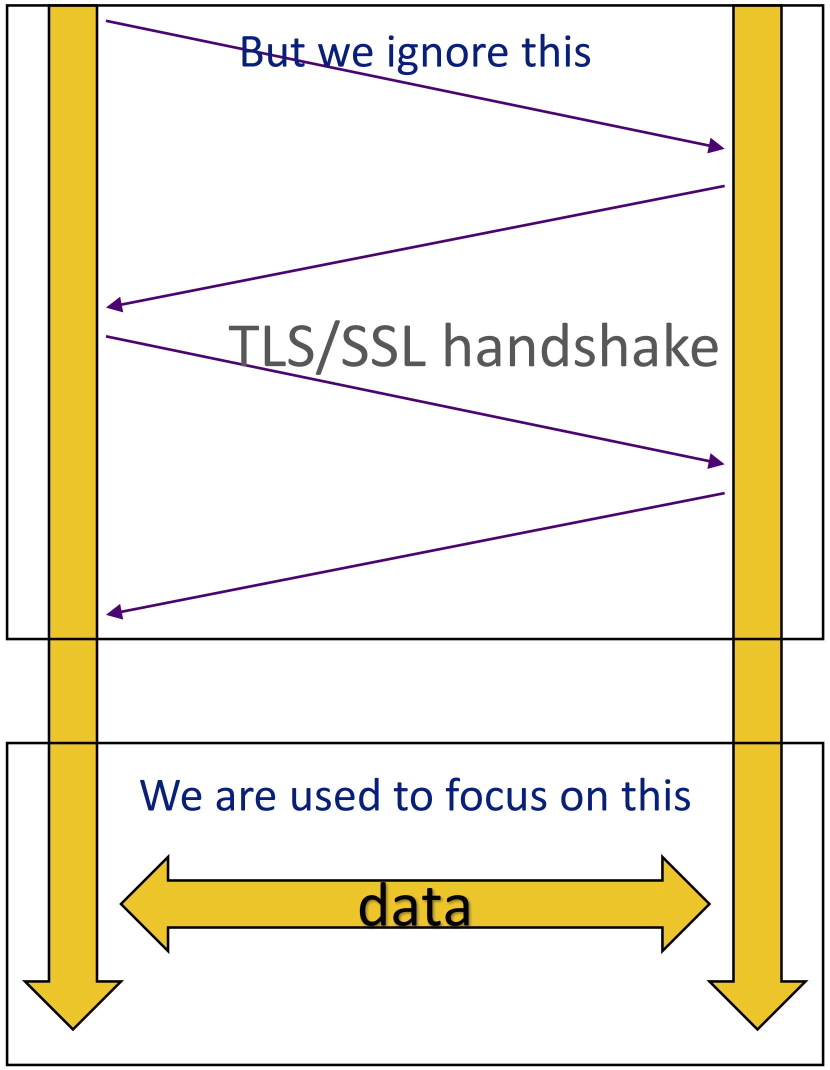
Circumventing the chain of trust…



TLS/SSL Certificate Pinning

"How do I do it?"

1. Hook into the handshake phase
2. Utilize OS validations (host, expiry, revocation, etc.)
3. Validate the certificate is the one we expect



TLS/SSL Certificate Pinning

Look under the hood

```
(1) func urlSession(_ session: URLSession, didReceive challenge: URLAuthenticationChallenge, completionHandler: @escaping
        (URLSession.AuthChallengeDisposition, URLCredential?) -> Void) {
        let serverTrust = challenge.protectionSpace.serverTrust

(2)     // Set SSL policies for domain name check
        let policies = NSMutableArray()
        policies.add(SecPolicyCreateSSL(true, (challenge.protectionSpace.host as CFString)))
        SecTrustSetPolicies(serverTrust!, policies);

        // Evaluate server certificate
        var error: CFError?
        if SecTrustEvaluateWithError(serverTrust!, &error) {
(3)         checkCertPinning(challenge.protectionSpace.serverTrust!, completionHandler: completionHandler)
        } else {
            completionHandler(.cancelAuthenticationChallenge, nil)
        }
    }
```

TLS/SSL Certificate Pinning

Option #1: verify the entire-certificate

```
func checkCertPinning(_ serverTrust: SecTrust, completionHandler: @escaping
    (URLSession.AuthChallengeDisposition, URLCredential?) -> Void) {

    let certRef = SecTrustGetCertificateAtIndex(serverTrust, 0)!
    let certificateChain = SecCertificateCopyData(certRef) as Data
    let certificateHash = CertUtil.sha256(certificateChain)

    if (AppConfig().allowedCertChainHashes().contains(certificateHash)) {
        let credential:URLCredential = URLCredential(trust: serverTrust)
        completionHandler(.useCredential, credential)
    } else {
        completionHandler(.cancelAuthenticationChallenge, nil)
    }
}
```

TLS/SSL Certificate Pinning

Option #2: verify only the public key

```
func checkCertPinning(_ serverTrust: SecTrust, completionHandler: @escaping
    (URLSession.AuthChallengeDisposition, URLCredential?) -> Void) {

    let secKey = SecTrustCopyPublicKey(serverTrust)!
    var error: Unmanaged<CFError>?
    let certificatePublicKey = SecKeyCopyExternalRepresentation(secKey, &error)! as Data
    let publicKeyHash = CertUtil.sha256(certificatePublicKey)

    if (AppConfig().allowedCertPublicKeyHashes().contains(publicKeyHash)) {
        let credential:URLCredential = URLCredential(trust: serverTrust)
        completionHandler(.useCredential, credential)
    } else {
        completionHandler(.cancelAuthenticationChallenge, nil)
    }
}
```

Practical considerations & solutions

"What do I do if I get locked out?"

- Use another secure channel to be able to add a temporary key

For example:

Use APNs / FCM to send a temporary key so the app is able to pull a new configuration

RSAConference2019
Asia Pacific & Japan

Coding Pitfalls
Path traversal

Unarchivin

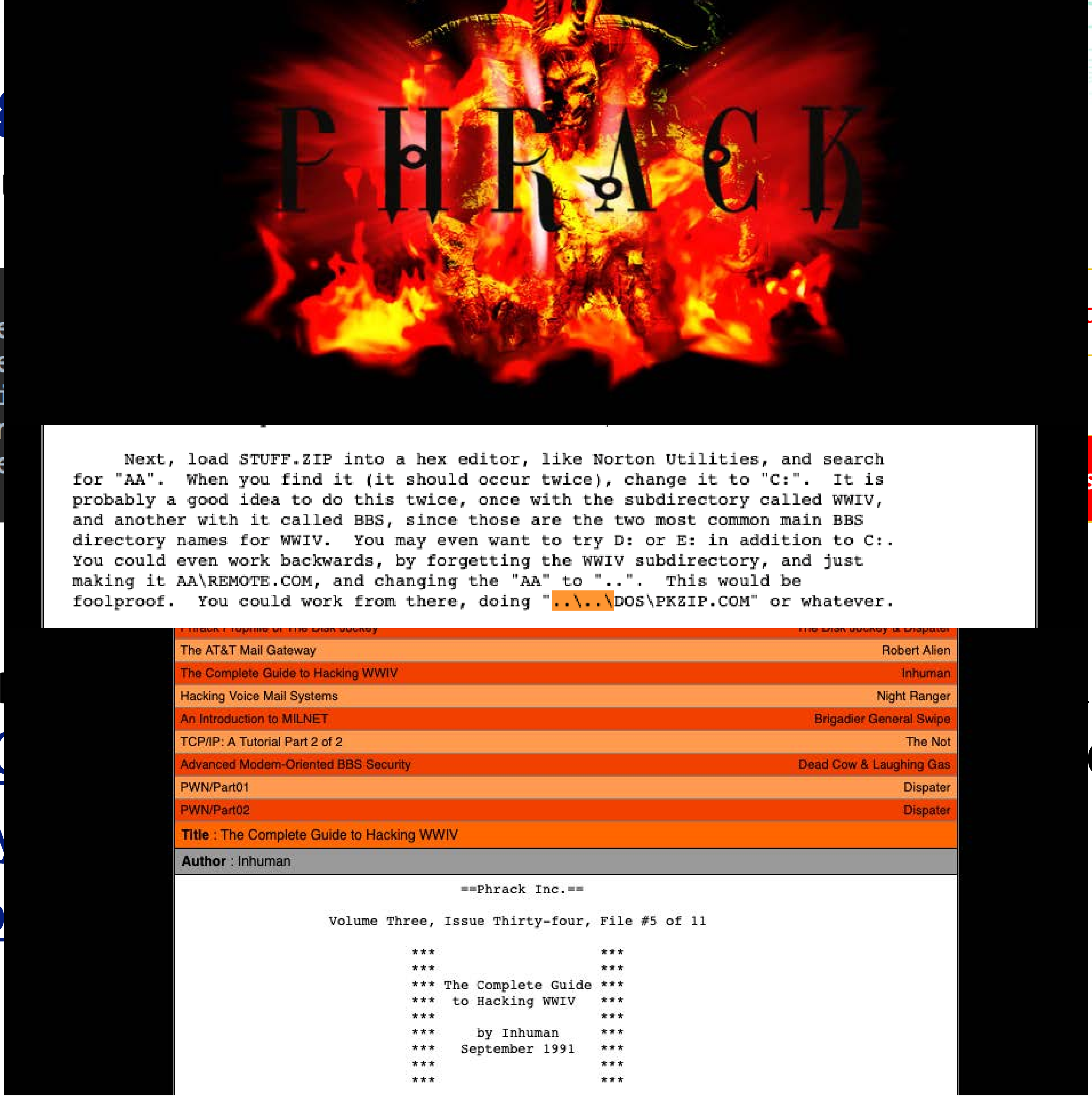
How would yo

```
Enumeration<? extends
while (entries.hasMore
    ZipEntry entry = e
    File file = new Fi
    InputStream conter
    IOUtils.copy(conte
}
```

u/.ssh/authorized_keys

s

- **Path traver** 35 [1991], CVE-2001-0 days…
  - https://sn
  - https://zip



```
     Next, load STUFF.ZIP into a hex editor, like Norton Utilities, and search
for "AA".  When you find it (it should occur twice), change it to "C:".  It is
probably a good idea to do this twice, once with the subdirectory called WWIV,
and another with it called BBS, since those are the two most common main BBS
directory names for WWIV.  You may even want to try D: or E: in addition to C:.
You could even work backwards, by forgetting the WWIV subdirectory, and just
making it AA\REMOTE.COM, and changing the "AA" to "..".  This would be
foolproof.  You could work from there, doing "..\..\DOS\PKZIP.COM" or whatever.
```

The AT&T Mail Gateway	Robert Alien
The Complete Guide to Hacking WWIV	Inhuman
Hacking Voice Mail Systems	Night Ranger
An Introduction to MILNET	Brigadier General Swipe
TCP/IP: A Tutorial Part 2 of 2	The Not
Advanced Modem-Oriented BBS Security	Dead Cow & Laughing Gas
PWN/Part01	Dispater
PWN/Part02	Dispater

Title : The Complete Guide to Hacking WWIV

Author : Inhuman

```
                    ==Phrack Inc.==

      Volume Three, Issue Thirty-four, File #5 of 11

               ***                     ***
               ***                     ***
               *** The Complete Guide  ***
               ***   to Hacking WWIV   ***
               ***                     ***
               ***      by Inhuman     ***
               ***    September 1991   ***
               ***                     ***
               ***                     ***
```

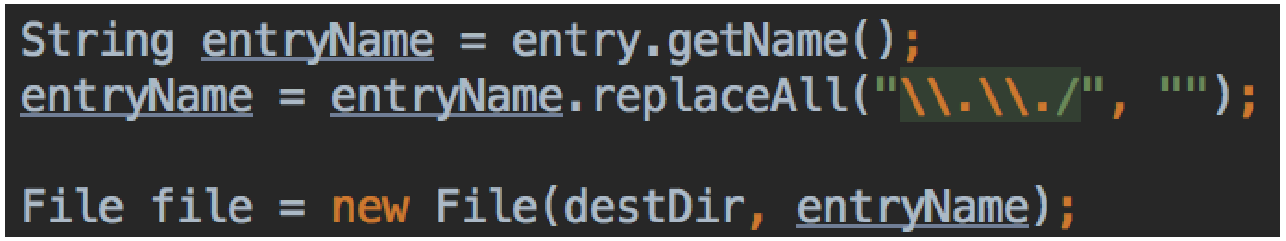
Symantec.

RSAConference2019 Asia Pacific & Japan

Best practices to avoid Path Traversal vulns

How would you fix it?

“Let’s sanitize!!!”



```
String entryName = entry.getName();
entryName = entryName.replaceAll("\\.\\./", "");

File file = new File(destDir, entryName);
```

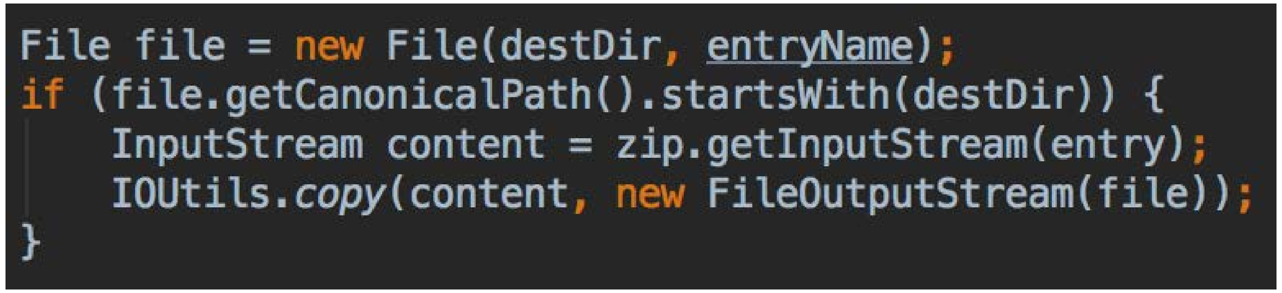
Sanitization is tricky and is susceptible to bypasses...

```
.../.\/file -> ../file
```

Best practices to avoid Path Traversal vulns

What you should really do…

- Generate the target path programmatically.
- Make sure that the canonicalized version of the target path starts with the canonicalized version of the target path you expect that files to be written to / read from.

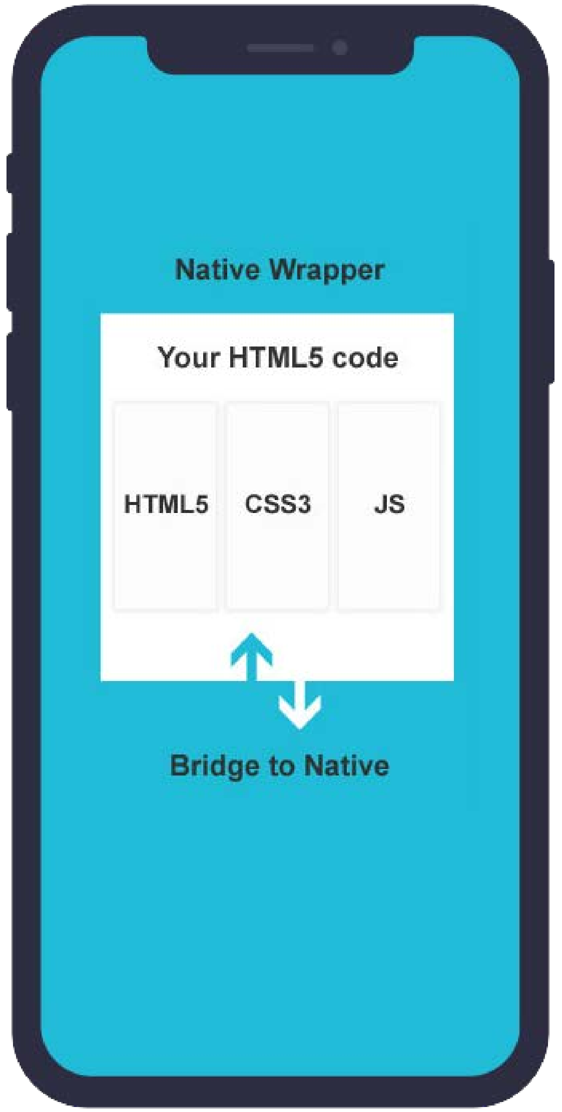


```
File file = new File(destDir, entryName);
if (file.getCanonicalPath().startsWith(destDir)) {
    InputStream content = zip.getInputStream(entry);
    IOUtils.copy(content, new FileOutputStream(file));
}
```

RSAConference2019 Asia Pacific & Japan

Let’s connect the dots…

Hybrid apps

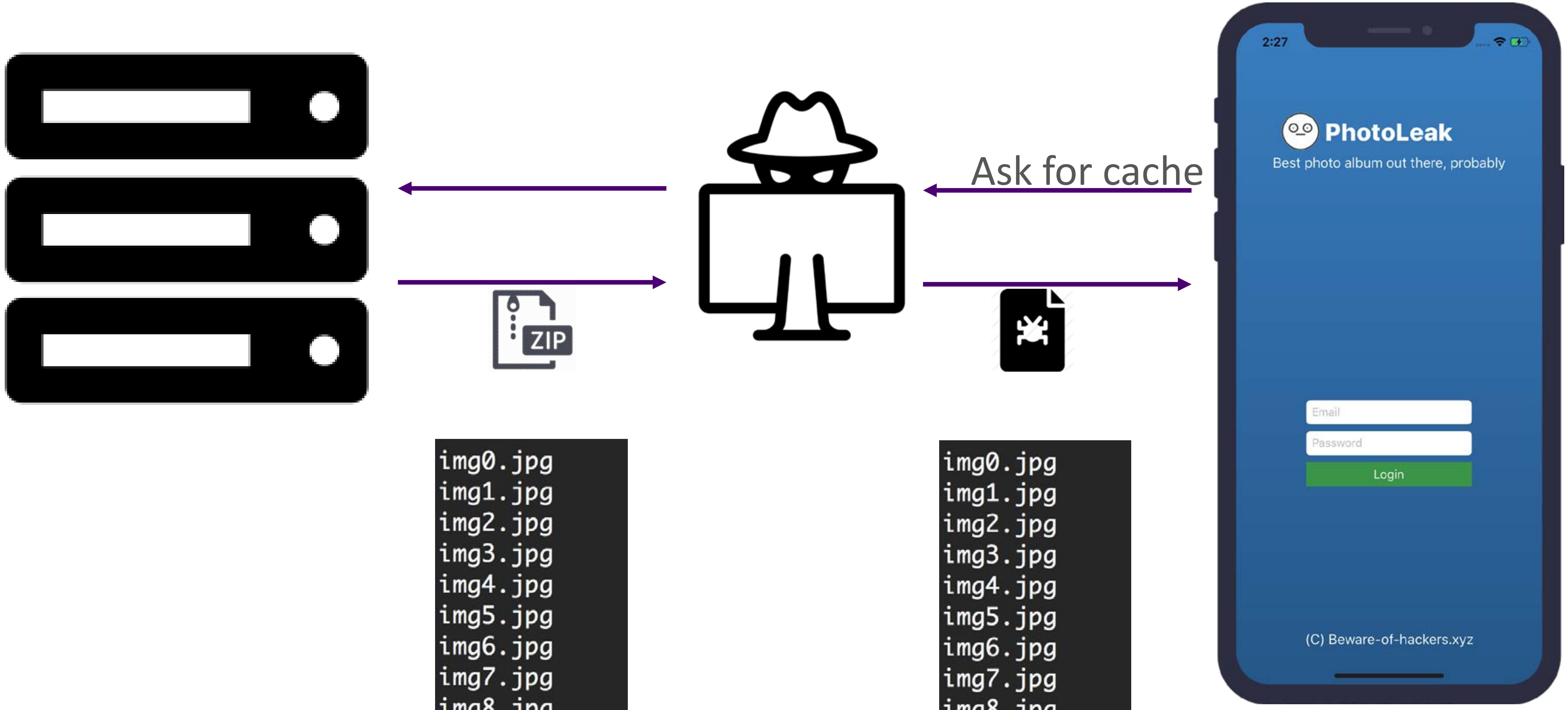

Native Wrapper
Your HTML5 code
HTML5
CSS3
JS
Bridge to Native

RSAConference2019 Asia Pacific & Japan

Live Demo

Exploiting a few consecutive pitfalls

What just happened?


Ask for cache
ZIP
2:27
PhotoLeak
Best photo album out there, probably
Email
Password
Login
(C) Beware-of-hackers.xyz

```
img0.jpg
img1.jpg
img2.jpg
img3.jpg
img4.jpg
img5.jpg
img6.jpg
img7.jpg
img8.jpg
img9.jpg
```

```
img0.jpg
img1.jpg
img2.jpg
img3.jpg
img4.jpg
img5.jpg
img6.jpg
img7.jpg
img8.jpg
img9.jpg
../app.js
```

Summary

Key Takeaways

- Neverending story
  - Mobile Apps are like a black-box to users.
  - As utilization of mobile goes up, so will be the ramifications of app-vulnerability exploits.
- Education & awareness are key
  - On an ongoing basis!
  - Thinking before copying (the StackOverflow syndrome)
- The importance of secure by design APIs
  - OS & cloud-infrastructure vendors responsibility

Thank You!