Automatic Forgery of Cryptographically Consistent Messages to Identify Security Vulnerabilities in Mobile Services

Chaoshun Zuo†, Wubing Wang†, Rui Wang*, Zhiqiang Lin†

†University of Texas at Dallas
*AppBugs Inc.

Feb 24th, 2016
Mobile Apps Often Need to Talk to a Remote server

- Saving resources (e.g., energy, and storage) on mobile
- Providing customized data (e.g., only retrieving the weather where you live)
Users Have to be Authenticated to Use the Service

- Server needs to know who you are, then push the data of your interest
- Crucial to ensure the authentication process is secure
Various Ways Used for the Authentication Security

App developers have been using:

1. Encryption of crucial data (e.g., user name, password)
2. Hashing (e.g., through MD5, SHA1) the user password
3. Signing (e.g., through HMAC) each message
Are They Enough?

Can a malicious client forge a valid message?
- Completely control a client app execution
- Reverse engineer how a valid message is generated
- Forge new valid authentication messages
Security Implications

- Testing Various Vulnerabilities at Server Side
  - Password brute forcing attack
  - Leaked password probing (password reuse practice)
  - Access token hijacking, SQL injection
Solutions in Web Applications

1. **Limiting the number of login attempts.** One simple solution app developers can adopt is to keep a login attempt state at server side and limit the number of login attempts within a certain time window.

2. **Using CAPTCHA.** Password brute forcing is not a new attack, and there are already solutions to mitigate this. One way that has been widely used on the desktop is the CAPTCHA [VABHL03].

3. **Two-factor authentication.** The most effective way to defeat all these malicious login attacks, we believe, is to adopt two-factor authentication [Wei88].
Introducing **AUTOFORGE**

- Given a mobile app, and few inputs
- A system that can automatically generate legal request messages via *protocol field inference* and *crypto API replay*
- Test various security vulnerabilities at mobile app’s server side
A Running Example: Mini Online Shopping App

“Mini offers a convenient way for customers around the world to shop for a wide variety of cool gadgets, electronic accessories, watches and lifestyle products at affordable prices, all with FREE SHIPPING!”

Installs: 1,000,000 - 5,000,000 (according to Google Play)
Observation of a Traced Network Packet

GET
/api/rest/app_server.php?sign_method=md5&client=android&app_key=A4H0P4JN&format=json&cv=3.9.0&country_code=US&country=USA&currency=USD&timestamp=2015-08-01%2013%3A00%3A59&v=1.2&p=695409430D3127CB969820016CB308F5&email=testappserver%40gmail.com&method=vela.user.login&app_secret=4ce19ca8fcd150a4w4pj91lah24991ut&language=en&sign=424978B759DA07CF8C8C41CCB5B8E718&keys=app_key%2Capp_secret%2Cclient%2Ccoun%2Ccountry_code%2Ccurre%2Ccc%2Cemail%2Cformat%2Clanguage%2Cmethod%2Cpwd%2Csign_method%2Ct%2Cversion%2Csid=1d3a40c25a86417c979fd847d7173e33 HTTP/1.1
x-newrelic-id: XAYCV1ZADgsAUFRTBQ==
User-agent: LightInTheBox 3.9.0(Android; 16; 4.1.1; 480_752; WIFI; generic; M353; en)
Host: api.miniinthebox.com
Connection: Keep-Alive
Accept-Encoding: gzip
Cookie: cookie_test=please_accept_for_session; AKAMAI_FEO_TEST=B; ASRV=A_201505081100

{"result":"fail","code":"1001001","info":[],"error_msg"[:"Invalid email or password (User)"]

- Many fields in a request message (18).
- We are interested in just a few of them, timestamp, pwd, email, sign.
Challenges

- Recognizing the protocol fields
- Identifying the cryptographic functions
- Deciding when to terminate
- Generating the valid messages
Key Insights

- Inferring the message fields with diffed input
- Dynamically hooking well-known cryptographic APIs
- Labeling response message with controlled input
- Replaying the cryptographic function execution
Since we control the client, we installed a root certificate on the emulator to make sure the proxy can get HTTPS messages.
Overview of AUTOFORGE

Since we control the client, we installed a root certificate on the emulator to make sure the proxy can get HTTPS messages.
**API Hooking**

- Run the app and type in the inputs
- Hooks the **well-known cryptographic functions** [Sch99]
Message Field Inference

- **Message field identification** that splits the messages into a set of fields
- **Field semantic inference** that infers the meaning of the identified fields
Message Field Identification: Diffed Message Alignment

GET /api/rest/app_server.php?sign_method=md5&client=android&app_key=A4H0P4JN&format=json&cv=3.9.0&country_code=US&country=USA&currency=USD&timestamp=2015-08-05%2003%3A19%3A26&v=1.2&pwd=695409430D3127CB158002B92FEC1831@email=testappserveralpha%40gmail.com&method=vela.user.login&app_secret=4ce19ca8fc150a4w4pj91lah24991ut&language=en&sign=94056C9BE079510079DBF9A372B4E65&keys=app_key%2Capp_secret%2Cclient%2Ccountry%2Ccountry_code%2Ccurrency%2Cv%2Ctimestamp%2Cemail%2Cformat%2CLanguage%2Cmethod%2Cpwd%2Csign_method%2Cmessage%2Csid

(a) Client Request with a Wrong Password

{"result":"fail","code":"1001001","info":[],"error_msg":"Invalid email or password (User)"}

(b) Server Response for the Wrong Password

GET /api/rest/app_server.php?sign_method=md5&client=android&app_key=A4H0P4JN&format=json&cv=3.9.0&country_code=US&country=USA&currency=USD&timestamp=2015-08-05%2003%3A20%3A01&v=1.2&pwd=A9672D9F5F7414D5B996964A7F07727E@email=testappserverbeta%40gmail.com&method=vela.user.login&app_secret=4ce19ca8fc150a4w4pj91lah24991ut&language=en&sign=D2A173BEB8F169DD1A81CA8D59AD2C69&keys=app_key%2Capp_secret%2Cclient%2Ccountry%2Ccountry_code%2Ccurrency%2Cv%2Ctimestamp%2Cemail%2Cformat%2CLanguage%2Cmethod%2Cpwd%2Csign_method%2Cmessage%2Csid

(c) Client Request with a Correct Password

{"result":"success","code":"1000000","info":{"sessionkey":"6a6ac7ff985eb08524e89392ec1addcb"},"error_msg":[]}

(d) Server Response for the Correct Password
Field Semantic Inference (Optional)

Approaches

- **Pattern Matching.** System data such as `timestamp` always has patterns (e.g., 2015-08-05), we can use pattern

- **Content Matching.** Since we control the user input and some user input would not get changed, then we directly search the diffed field (e.g., a `username` we entered)

- **Degree of Differences.** By measuring the degree of the similarities, we can easily identify the cryptographically computed fields (such as `pwd` and `sign`)
If the Wrong(correct) password responses are identical, we will use the entire message as a Wrong password signature, if the Wrong(correct) password responses are different, we will align them and keep the common string as a signature.
Request Message Generation

1. Modify inputs
2. Re-execute API calls
3. Replace them in message
4. N different wrong passwords and 1 correct password
Experiment Setup: How the 76 Apps Were Chosen

1. Crawled over 20,000 apps from Google Play

2. Filtered out apps that have less than one million installs, and we have 320 apps.

3. Filtered out non-encryption, non-hashing, and non-signing apps, we have 105 apps.

4. Manually ran 105 one-by-one, we found 15 of them do not contain the user login interface, 14 of them do not use HTTP/HTTPS protocols.

5. Therefore, we have 105 - 15 - 14 = 76 apps.
Experiment Setup: How the 76 Apps Were Chosen

1. Crawled over 20,000 apps from Google Play
2. Filtered out apps that have less than one million installs, and we have 320 apps.
Experiment Setup: How the 76 Apps Were Chosen

1. Crawled over 20,000 apps from Google Play
2. Filtered out apps that have less than one million installs, and we have 320 apps.
3. Filtered out non-encryption, non-hashing, and non-signing apps, we have 105 apps.
Experiment Setup: How the 76 Apps Were Chosen

1. Crawled over 20,000 apps from Google Play
2. Filtered out apps that have less than one million installs, and we have 320 apps.
3. Filtered out non-encryption, non-hashing, and non-signing apps, we have 105 apps.
4. Manually run 105 one-by-one, we found
   - 15 of them do not contain the user login interface
   - 14 of them do not use HTTP/HTTPS protocols
Experiment Setup: How the 76 Apps Were Chosen

1. Crawled over 20,000 apps from Google Play
2. Filtered out apps that have less than one million installs, and we have 320 apps.
3. Filtered out non-encryption, non-hashing, and non-signing apps, we have 105 apps.
4. Manually run 105 one-by-one, we found
   - 15 of them do not contain the user login interface
   - 14 of them do not use HTTP/HTTPS protocols
5. Therefore, we have 105 - 15 - 14 = 76 apps
I. Password Brute-force Testing

- Total 76 apps
- 86% of apps’ server side are vulnerable to password brute-forcing attack
- Including CNN, Expedia, iHeartRadio, and Walmart.
Other Testing

1. II. Leaked Username and Password Probing Testing.
2. III. Facebook Access Token Hijacking Testing.
**A Serious Security Problem at Server Side**

- **AUTOFORGE** has demonstrated that lack of security checks at server side can lead to several severe attacks:
  1. Password brute forcing
  2. Leaked username and password probing
  3. Access token hijacking.

- This is a very serious problem considering that a large volume of popular apps, including CNN, Expedia, iHeartRadio, and Walmart as demonstrated in our testing, are vulnerable to these attacks.

- **HTTPS alone cannot defeat password brute-forcing, nor can hashing and signing of client request messages**
Related Work

1. **Protocol Reverse Engineering.** A large body of research focusing on protocol reverse engineering [Bed, MLK+06, CKW07, CS07, WMKK08, LJXZ08, MWKK09, CPKS09]

2. **Application Dialogue Replay.** AUTOFORGE employs cryptographic function replay to generate the authenticated messages, which is similar to the existing application dialogue replay systems: RolePlayer [CPWK06] and Replayer [NBFS06].

3. **Mobile App Vulnerability Discovery.** A considerate amount of efforts have focused on discovering various vulnerabilities in mobile apps. TaintDroid [EGC+10], PiOS [EKKV11], CHEX [LLW+12], SMV-Hunter [SSG+14]. However, few efforts have been focusing on identifying the vulnerabilities in app’s server side.
**AutoForge**

- Given a mobile app, and few inputs
- A system that can automatically generate legal request messages via protocol field inference and crypto API replay
- Test various security vulnerabilities at mobile app’s server side

**Experimental Result w/ 76 apps**

- 86% of servers (including CNN, and Walmart) are vulnerable to password brute-forcing
- 100% are vulnerable to leaked password probing
- 12% are vulnerable to Facebook access token hijacking


References III
