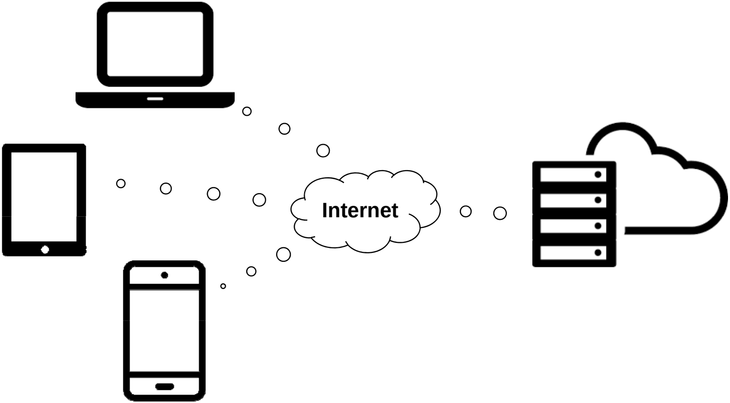
Automatic Forgery of Cryptographically Consistent Messages to Identify Security Vulnerabilities in Mobile Services

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

[†]University of Texas at Dallas
[*]AppBugs Inc.

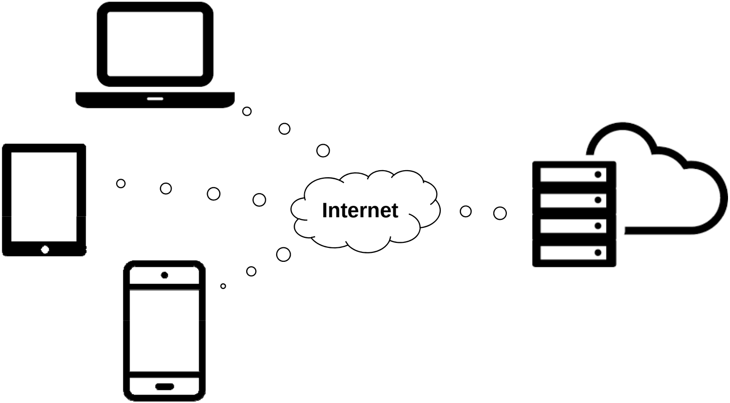
Feb 24th, 2016

Mobile Apps Often Need to Talk to a Remote server



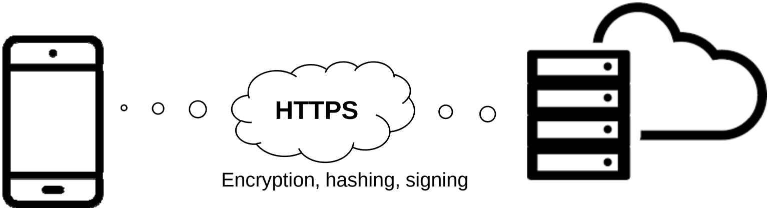
- Saving resources (e.g., energy, and storage) on mobile
- Providing customized data (e.g., only retrieving the weather where you live)

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

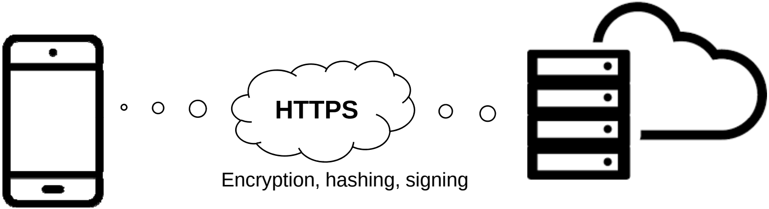
Various Ways Used for the Authentication Security



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

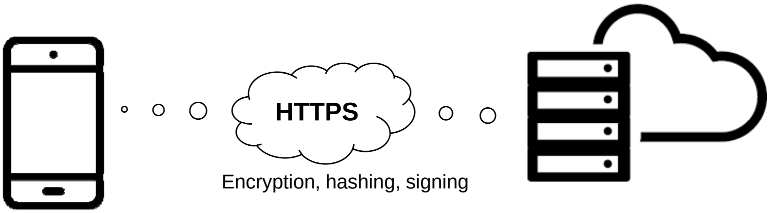
Are They Enough?



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



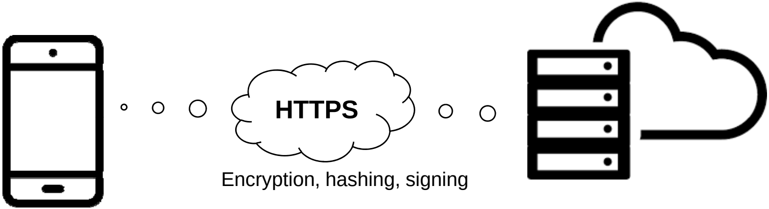
Testing Various Vulnerabilities at Server Side

- 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].

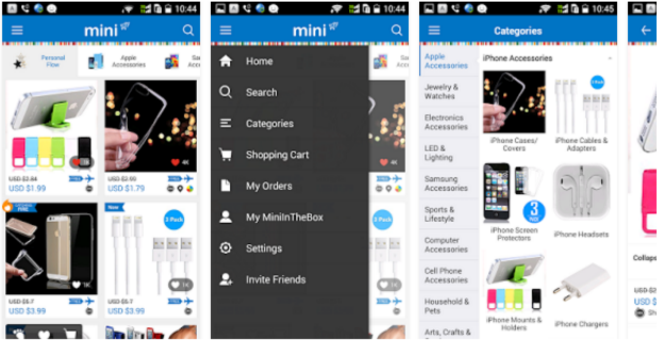
Introducing AUTOFORGE



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&pwd=695409430D3127CB969820016CB308F5&email=testappserver%40gmail.com
&method=vela.user.login&app_secret=4ce19ca8fcd150a4w4pj9llah24991ut&language=en&sign=424978B
759DA07CF8C8C41CCB5B8E718&keys=app_key%2Capp_secret%2Cclient%2Ccountry%2Ccountry_code%2Ccurr
ency%2Ccv%2Cemail%2Cformat%2Clanguage%2Cmethod%2Cpwd%2Csign_method%2Ctimestamp%2Cv&sid=1d3a4
0c25a86417c979fd847d7173e33 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

```
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&pwd=695409430D3127CB969820016CB308F5&email=testappserver%40gmail.com
&method=vela.user.login&app_secret=4ce19ca8fcd150a4w4pj9llah24991ut&language=en&sign=424978B
759DA07CF8C8C41CCB5B8E718&keys=app_key%2Capp_secret%2Cclient%2Ccountry%2Ccountry_code%2Ccurr
ency%2Ccv%2Cemail%2Cformat%2Clanguage%2Cmethod%2Cpwd%2Csign_method%2Ctimestamp%2Cv&sid=1d3a4
0c25a86417c979fd847d7173e33 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
```

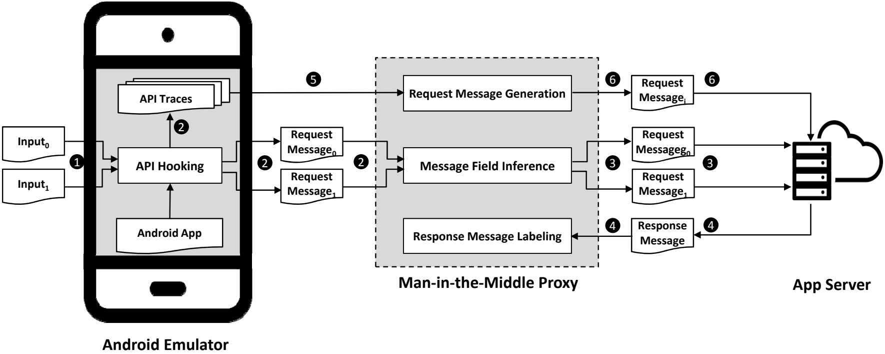
```
{"result":"fail","code":"1001001","info":[],"error_msg":["Invalid email or password (User)"]}
```

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

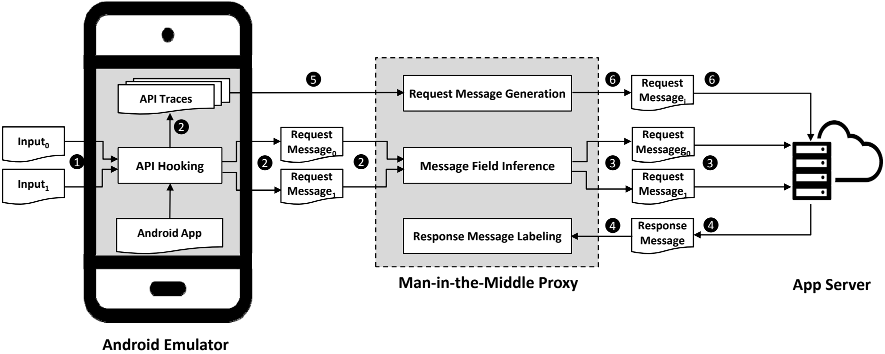
Key Insights

- **Inferring the message fields with diffed input**
- **Dynamically hooking well-known cryptographic APIs**
- **Labeling response message with controlled input**
- **Replaying the cryptographic function execution**

Overview of AUTOFORGE



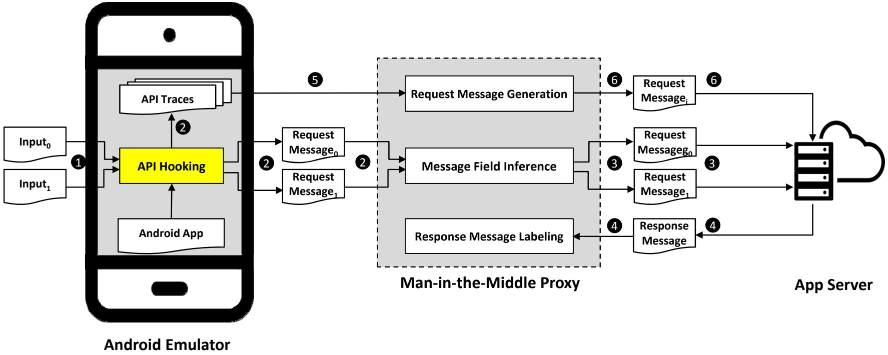
Overview of AutoForge



HTTPS

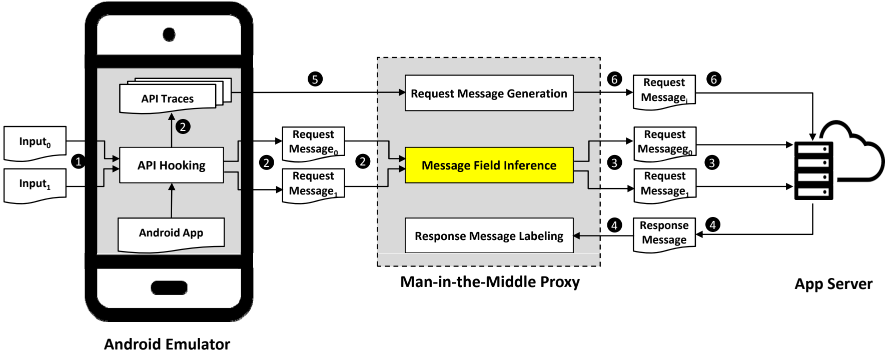
Since we control the client, we installed a root certificate on the emulator to make sure the proxy can get HTTPS messages.

API Hooking



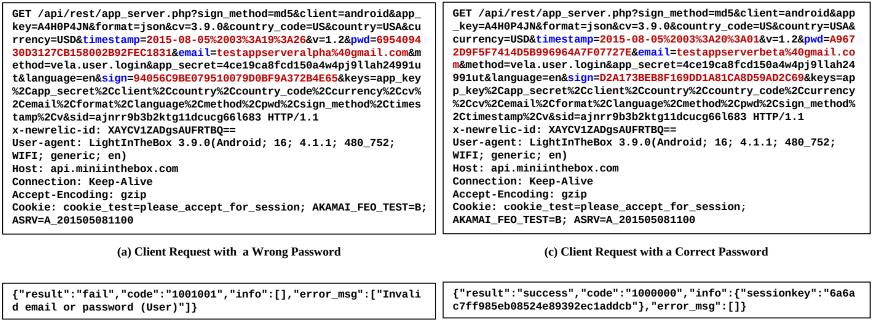
- Run the app and type in the inputs
- Hooks the well-known cryptographic functions [Sch99]

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&cu
rrency=USD&timestamp=2015-08-05%2003%3A19%3A26&v=1.2&pwd=6954094
30D3127CB158002B92FEC1831&email=testappserveralpha%40gmail.com&m
ethod=vela.user.login&app_secret=4ce19ca8fcd150a4w4pj9llah24991u
t&language=en&sign=94056C9BE079510079D0BF9A372B4E65&keys=app_key
%2Capp_secret%2Cclient%2Ccountry%2Ccountry_code%2Ccurrency%2Ccv%
2Cemail%2Cformat%2Clanguage%2Cmethod%2Cpwd%2Csign_method%2Ctimes
tamp%2Cv&sid=ajnrr9b3b2ktg11dcucg66l683 HTTP/1.1
x-newrelic-id: XAYCV1ZADgsAUFRTBQ==
User-agent: LightInTheBox 3.9.0(Android; 16; 4.1.1; 480_752;
WIFI; generic; 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
```

(a) Client Request with a 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=A967
2D9F5F7414D5B996964A7F07727E&email=testappserverbeta%40gmail.co
m&method=vela.user.login&app_secret=4ce19ca8fcd150a4w4pj9llah24
991ut&language=en&sign=D2A173BEB8F169DD1A81CA8D59AD2C69&keys=ap
p_key%2Capp_secret%2Cclient%2Ccountry%2Ccountry_code%2Ccurrency
%2Ccv%2Cemail%2Cformat%2Clanguage%2Cmethod%2Cpwd%2Csign_method%
2Ctimestamp%2Cv&sid=ajnrr9b3b2ktg11dcucg66l683 HTTP/1.1
x-newrelic-id: XAYCV1ZADgsAUFRTBQ==
User-agent: LightInTheBox 3.9.0(Android; 16; 4.1.1; 480_752;
WIFI; generic; 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
```

(c) Client Request with a Correct Password

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

(b) Server Response for the Wrong Password

```
{"result":"success","code":"1000000","info":{"sessionkey":"6a6a
c7ff985eb08524e89392ec1addcb"},"error_msg":[]}
```

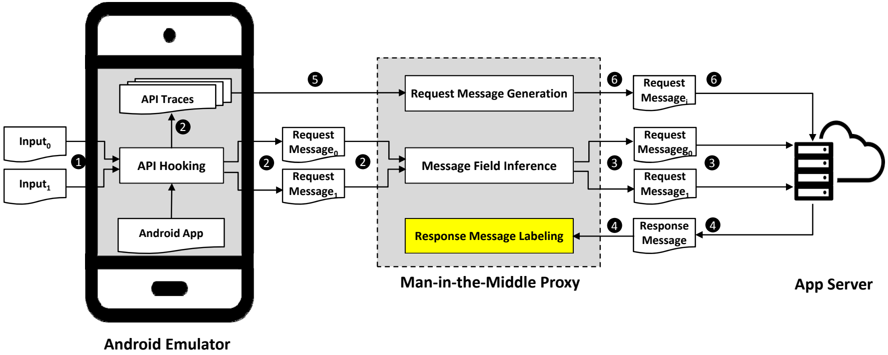
(d) Server Response for the Correct Password

Field Semantic Inference (Optional)

Approaches

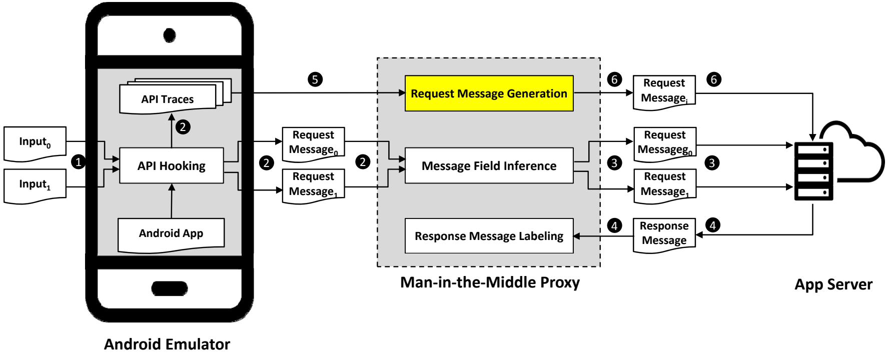
- **Pattern Matching.** System data such as `timestamp` always has patterns (e.g., 2015-08-05), we can use pattern
- **Content Matching.** Since we control the user input and some user input would not get changed, then we directly search the diffed field (e.g., a `username` we entered)
- **Degree of Differences.** By measuring the degree of the similarities, we can easily identify the cryptographically computed fields (such as `pwd` and `sign`)

Response Message Labeling



- 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



- Modify inputs
- Re-execute API calls
- Replace them in message

- 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

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.**

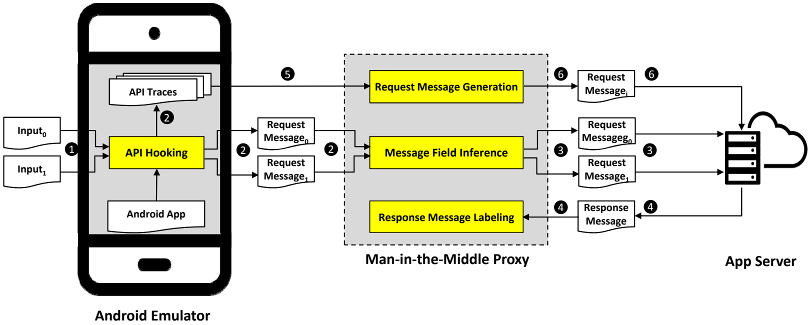
A Serious Security Problem at Server Side

- 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



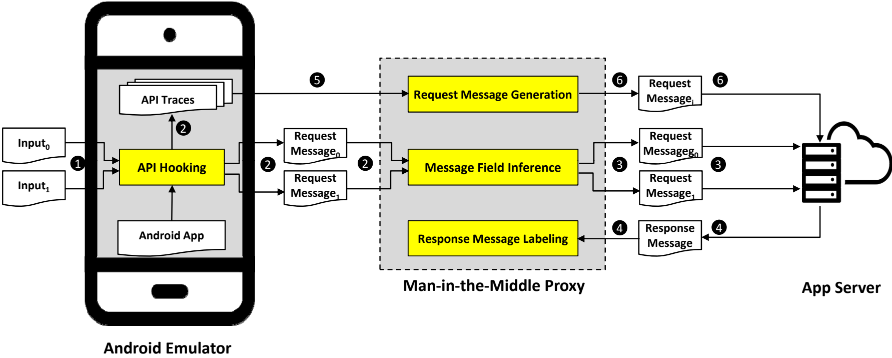
AutoForge

- 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

Q&A


API Traces
2
Input$_0$
Input$_1$
1
API Hooking
Android App
Android Emulator
5
2
Request Message$_0$
Request Message$_1$
2
Request Message Generation
Message Field Inference
Response Message Labeling
Man-in-the-Middle Proxy
6
Request Message$_i$
6
3
Request Message$_0$
Request Message$_1$
3
4
Response Message
4
App Server

References I

- Marshall Beddoe, The protocol informatics project, `http://www.4tphi.net/~awalters/PI/PI.html`.
- Weidong Cui, Jayanthkumar Kannan, and Helen J. Wang, Discoverer: Automatic protocol reverse engineering from network traces, Proceedings of the 16th USENIX Security Symposium (Security'07) (Boston, MA), August 2007.
- Juan Caballero, Pongsin Poosankam, Christian Kreibich, and Dawn Song, Dispatcher: Enabling active botnet infiltration using automatic protocol reverse-engineering, Proceedings of the 16th ACM Conference on Computer and and Communications Security (CCS'09) (Chicago, Illinois, USA), 2009, pp. 621–634.
- Weidong Cui, Vern Paxson, Nicholas Weaver, and Randy H. Katz, Protocol-independent adaptive replay of application dialog, Proceedings of the 13th Annual Network and Distributed System Security Symposium (NDSS'06) (San Diego, CA), February 2006.
- Juan Caballero and Dawn Song, Polyglot: Automatic extraction of protocol format using dynamic binary analysis, Proceedings of the 14th ACM Conference on Computer and and Communications Security (CCS'07) (Alexandria, Virginia, USA), 2007, pp. 317–329.
- W. Enck, P. Gilbert, B.G. Chun, L.P. Cox, J. Jung, P. McDaniel, and A.N. Sheth, TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones, OSDI, 2010.
- M. Egele, C. Kruegel, E. Kirda, and G. Vigna, Pios: Detecting privacy leaks in ios applications, NDSS, 2011.
- Zhiqiang Lin, Xuxian Jiang, Dongyan Xu, and Xiangyu Zhang, Automatic protocol format reverse engineering through context-aware monitored execution, Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08) (San Diego, CA), February 2008.

References II

- Long Lu, Zhichun Li, Zhenyu Wu, Wenke Lee, and Guofei Jiang, Chex: statically vetting android apps for component hijacking vulnerabilities, Proceedings of the 2012 ACM conference on Computer and communications security, ACM, 2012, pp. 229–240.
- Justin Ma, Kirill Levchenko, Christian Kreibich, Stefan Savage, and Geoffrey M. Voelker, Unexpected means of protocol inference, Proceedings of the 6th ACM SIGCOMM on Internet measurement (IMC'06) (Rio de Janeriro, Brazil), ACM Press, 2006, pp. 313–326.
- Paolo Milani Comparetti, Gilbert Wondracek, Christopher Kruegel, and Engin Kirda, Prospex: Protocol Specification Extraction, IEEE Symposium on Security & Privacy (Oakland, CA), 2009, pp. 110–125.
- James Newsome, David Brumley, Jason Franklin, and Dawn Song, Replayer: Automatic protocol replay by binary analysis, Proceedings of the 13th ACM Conference on Computer and and Communications Security (CCS'06), 2006.
- Bruce Schneier, Cryptography: The importance of not being different, Computer **32** (1999), no. 3, 108–109,112.
- David Sounthiraraj, Justin Sahs, Garrett Greenwood, Zhiqiang Lin, and Latifur Khan, Smv-hunter: Large scale, automated detection of ssl/tls man-in-the-middle vulnerabilities in android apps, Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS'14) (San Diego, CA), February 2014.
- Luis Von Ahn, Manuel Blum, Nicholas J Hopper, and John Langford, Captcha: Using hard ai problems for security, Advances in Cryptology — EUROCRYPT 2003, Springer, 2003, pp. 294–311.

References III

Kenneth P Weiss, Method and apparatus for positively identifying an individual, January 19 1988, US Patent 4,720,860.

Gilbert Wondracek, Paolo Milani, Christopher Kruegel, and Engin Kirda, Automatic network protocol analysis, Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08) (San Diego, CA), February 2008.