An Experimental Approach to Evaluate the Security of Mobile Autofill Frameworks on iOS and Android

Sean Oesch

Anuj Gautam

Scott Ruoti

The University of Tennessee
Knoxville
Overview

• How was the research question born?
• First try
• New direction
• Interactive discussion
• Paper brainstorming
The beginning of the journey

How was the research question born?
Generating Research Ideas

• Zhiyun Qian (https://bit.ly/3x9mzHB)
  • Fill in the blank
  • Expansion
  • Hammer and nails
  • Start small and generalize
  • Reproduction of prior work
  • Needs in industry
Where we began - new context, similar approach

- Replicate our work on desktop managers on mobile
  - USENIX 2020 - That Was Then, This Is Now: A Security Evaluation of Password Generation, Storage, and Autofill in Browser-Based Password Managers
- iOS and Android separate papers
- Replicate & expand work of Aonzo et al. on Android - explore similar vulnerabilities on iOS
USENIX Paper Methods

• Generation
  • Corpus 147 million generated passwords
  • Shannon entropy, $\chi^2$ test, zxcvbn, and a recurrent neural net

• Storage
  • Encryption, metadata, master password requirements

• Autofill
  • iframes, form verification, website verification
USENIX Paper Findings & Recommendations

• Generation
  • Filter weak passwords during generation

• Storage
  • Require strong master passwords

• Autofill
  • Require user interaction before filling credential
    – Prevents automatic credential scraping
    – Increases the probability the user can detect attacks
  • Only autofill passwords into secure field
  • Thoroughly vet the fill page

Desktop would benefit from having first-class support for password management in the browsers and/or OS
Aonzo et al. - Credential Mapping on Android

Table 2: Summary of findings for Keeper (K), Dashlane (D), LastPass (LP), 1Password (1P), and Google Smart Lock (GSL).

<table>
<thead>
<tr>
<th>Feature</th>
<th>K</th>
<th>D</th>
<th>LP</th>
<th>1P</th>
<th>GSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secure mapping</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>One-to-one mapping</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Many-to-one mapping</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Crowdsourced mapping</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Heuristic-based mapping</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>No mapping</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Q1) Vulnerable?</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Q2) Can co-exist on device?</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Q3) Can co-exist on Play Store?</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Q4) Targeted suggestion?</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
First Try

Similar approach, new context
Similar Methodology, New Context

- Looked at ~20 managers on iOS and Android platform based on usage in app store / google play store
- Evaluated generation, storage, and autofill
  - For generation, chose not to repeat check for randomness
  - At this point, autofill was limited to apps and browser (no WebView)
- 2 Papers - 1 for iOS, 1 for Android
Caveat for Android

- Only evaluated generation and autofill on Android
- Expanded Aonzo’s work from 5 managers to ~20
  - These results showed that none of the identified mapping vulnerabilities had been addressed in the last several years
- At the time, I felt this was a very valuable contribution
iOS Paper - PWM Overview

Figure 2: April 2020 Download Estimates
# iOS Paper - Storage

<table>
<thead>
<tr>
<th>System</th>
<th>File type</th>
<th>Algorithm</th>
<th>MP KDF</th>
<th>KDF Rounds</th>
<th>Metadata encryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>1Password</td>
<td>SQLite w/ encrypted cells</td>
<td>AES-256</td>
<td>PBKDF2</td>
<td>100,000</td>
<td>● ● ○ ○ ● ● ○ ● ○ -</td>
</tr>
<tr>
<td>Avast</td>
<td>SQLite w/ encrypted cells</td>
<td>AES-256</td>
<td>PBKDF2</td>
<td>10,240</td>
<td>● ○ ○ ○ ● ○ ○ ● ○ ○</td>
</tr>
<tr>
<td>Bitwarden</td>
<td>Encrypted SQLite file</td>
<td>AES-256</td>
<td>PBKDF2</td>
<td>100,001</td>
<td>● ● ● ● ● ● ● ● ● -</td>
</tr>
<tr>
<td>Dashlane</td>
<td>Encrypted SQLite file</td>
<td>AES-256</td>
<td>Argon2D</td>
<td>3</td>
<td>● ● ● ● ○ ● ○ ● ○</td>
</tr>
<tr>
<td>Enpass</td>
<td>Custom encrypted file</td>
<td>AES-256</td>
<td>PBKDF2</td>
<td>100,000</td>
<td>● ● ● ● ● ● ● ○ ● ○</td>
</tr>
<tr>
<td>iCloud Keychain</td>
<td>Custom encrypted values</td>
<td>AES-256</td>
<td>PBKDF2</td>
<td>100,000</td>
<td>● ● ● ● ● ● ● ● ● ○</td>
</tr>
<tr>
<td>Keeper</td>
<td>SQLite w/ encrypted cells</td>
<td>AES-256</td>
<td>PBKDF2</td>
<td>100,000</td>
<td>● ● ● ● ● ● ● ● ● ○</td>
</tr>
<tr>
<td>Lastpass</td>
<td>Custom encrypted file</td>
<td>AES-256</td>
<td>PBKDF2</td>
<td>100,100</td>
<td>● ● ● ● ● ● ● ● ● ○</td>
</tr>
<tr>
<td>AutoFill</td>
<td>Custom encrypted file</td>
<td>AES-256</td>
<td>PBKDF2</td>
<td>100,100</td>
<td>● ● ● ● ● ● ● ● ● ○</td>
</tr>
<tr>
<td>Lockwise</td>
<td>SQLCipher</td>
<td>AES-256</td>
<td>Uses secure enclave</td>
<td>● ● ● ● ● ● ● ● ● ○</td>
<td></td>
</tr>
<tr>
<td>Norton</td>
<td>Custom encrypted file</td>
<td>AES-256</td>
<td>PBKDF2</td>
<td>1,000</td>
<td>● ● ● ● ● ● ● ● ● ○</td>
</tr>
<tr>
<td>RoboForm</td>
<td>Custom encrypted file</td>
<td>AES-256</td>
<td>PBKDF2</td>
<td>4,000</td>
<td>● ● ● ● ● ● ● ● ● ○</td>
</tr>
<tr>
<td>SafeInCloud</td>
<td>Encrypted SQLite file</td>
<td>AES-256</td>
<td>PBKDF2</td>
<td>10,000</td>
<td>● ● ● ● ● ● ● ● ● ○</td>
</tr>
<tr>
<td>StrongBox</td>
<td>KeePass or PasswordSafe file</td>
<td>AES-256</td>
<td>Argon2D</td>
<td>2</td>
<td>● ● ● ● ● ● ● ● ● ○</td>
</tr>
<tr>
<td>aWallet</td>
<td>Custom encrypted file</td>
<td>AES-256</td>
<td>SHA256</td>
<td>1,000</td>
<td>● ● ● ● ● ● ● ● ● ○</td>
</tr>
<tr>
<td>My Passwords</td>
<td>SQLite w/ encrypted cells</td>
<td>AES-256</td>
<td>PBKDF2</td>
<td>1,000</td>
<td>● ● ● ● ● ● ● ● ● ○</td>
</tr>
<tr>
<td>PasswordSafe</td>
<td>Encrypted SQLite file</td>
<td>AES-256</td>
<td>TwoFish-256</td>
<td>PBKDF2</td>
<td>256 ○ ● ● ● ● ● ● ● ● ● ○</td>
</tr>
<tr>
<td>Chrome</td>
<td>iCloud Keychain</td>
<td>Uses secure enclave</td>
<td>● ● ● ● ● ● ● ● ● ○</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Edge</td>
<td>iCloud Keychain</td>
<td>Uses secure enclave</td>
<td>● ● ● ● ● ● ● ● ● ○</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Firefox</td>
<td>SQLCipher</td>
<td>Uses secure enclave</td>
<td>● ● ● ● ● ● ● ● ● ○</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Secure behavior
- Partially secure behavior
- Insecure behavior
- Not stored

MP = Master Password  KDF = Key Derivation Function

Table 4: Overview of Password Vault Encryption
## iOS Paper - Generation

<table>
<thead>
<tr>
<th>System</th>
<th>Default</th>
<th>Supported Lengths</th>
<th>At Least Difficult Characters</th>
<th>Generate Passwords</th>
<th>Passwords Are Secure</th>
<th>Generate History</th>
</tr>
</thead>
<tbody>
<tr>
<td>1Password</td>
<td>ld 24</td>
<td>4–64</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Avast Passwords</td>
<td>Is 9</td>
<td>4–30</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Bitwarden</td>
<td>ld 14</td>
<td>5–128</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Dashlane</td>
<td>ld 8</td>
<td>4–40</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Enpass</td>
<td>Is 50</td>
<td>4–100</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>iCloud Keychain</td>
<td>lsd 20</td>
<td>20</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Keeper</td>
<td>lsd 20</td>
<td>8–51</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LastPass</td>
<td>lsd 16</td>
<td>8–64</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Norton</td>
<td>lsd 8</td>
<td>4–64</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>RoboForm</td>
<td>lsd 16</td>
<td>4–511</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SafeInCloud</td>
<td>lsd 12</td>
<td>8–31</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>StrongBox</td>
<td>lsd 16</td>
<td>6–88</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>aWallet</td>
<td>lsd 8</td>
<td>4–20</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MyPasswords</td>
<td>ld 10</td>
<td>10</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PasswordSafe</td>
<td>ld 8</td>
<td>1–50</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Chrome</td>
<td>ld 15</td>
<td>15</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Default ✗ Optional

Table 3: Password Generation Features
New Direction

Systematic Analysis of Autofill Frameworks
Secure Autofill Properties

• Managers should only fill credentials when:
  • P1: Users explicitly authorize operation
  • P2: Credential is securely mapped to web domain or app
  • P3: Credential is only accessible to mapped domain
• Protects against credential scraping and phishing

Autofill dialogue tells user it is safe to fill credentials
Autofill on Mobile

• Multiple contexts for autofill
  • Browser
  • Apps
• Multiple approaches to autofill
Contexts for Autofill in Apps

Native UI Elements

WebView

Custom UI Elements
iOS App Extensions

- iOS 8 – 2014
- Popular managers still support – 1Password, Keeper, LastPass
- Older devices – prior iOS 12
iOS AutoFill

- iOS 12 – 2018
- Controls entire autofill process
  - form identification
  - mapping app and domain
  - user interface
  - autofill
Android Autofill Service

- Android 8 (Oreo) 2017 – replaces accessibility service
- Leaves a lot of leeway to individual managers
Methodology Deep Dive

*Testing Autofill in the Browser, Apps, & WebView*
Testing

• Strategy
  • Evaluated 14 managers implemented with the autofill frameworks
  • Considered all three properties in all supported contexts
  • Looking for what the framework enforces, what it fails to enforce, and what it prevents managers from enforcing

• Environment
  • iPhone 7 running iOS 13, using Safari for browser tests
  • Genymotion Android emulator
    – Simulated a Google Pixel 2 running Android 9 (Pie)
    – Chrome for browser
Selecting Managers

- Wanted to determine which managers most utilized
- On Android, used download data from Google Play Store
  - Accessible via API
- iOS does not provide detailed information on downloads from App Store
  - used April 2020 SensorTower estimates as a stand-in
Preparing the Devices

• Android
  • Genymotion emulated devices already rooted
  • “Open GApps” to enable the Google Play Store
  • Appmon / Frida to watch network comms

• iOS
  • Because not open source, no ideal emulation platforms
    • Could not install 3rd party apps - only your own
  • Jailbroke the device
Browser Testing Approach

• Improved browser testing website from USENIX paper
  • Based on vulnerabilities identified in Silver et al., Stock and Johns, Li et al.

• NodeJS website
  • Deployed to Heroku and UTK domain
  • UTK domain allowed broken HTTPS and HTTP
Browser Testing Workflow

- Save a password for a heroku domain and a UTK domain
  - UTK domain allowed me to break the cert (Let’s Encrypt)
- Run test framework at heroku site
- Test HTTP and broken (invalid cert) HTTPS at UTK domain
# Autofill in the Browser

<table>
<thead>
<tr>
<th>Framework</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>iOS Password AutoFill</td>
<td>![Secure behavior]</td>
<td>![Secure behavior]</td>
<td>![Secure behavior]</td>
</tr>
<tr>
<td>iOS App Extensions</td>
<td>![Secure behavior]</td>
<td>![Secure behavior]</td>
<td>![Secure behavior]</td>
</tr>
<tr>
<td>Android Autofill Service</td>
<td>![Secure behavior]</td>
<td>![Secure behavior]</td>
<td>![Secure behavior]</td>
</tr>
<tr>
<td>Most secure desktop manager [48]</td>
<td>![Secure behavior]</td>
<td>![Secure behavior]</td>
<td>![Secure behavior]</td>
</tr>
<tr>
<td>Least secure desktop manager [48]</td>
<td>![Insecure behavior]</td>
<td>![Insecure behavior]</td>
<td>![Insecure behavior]</td>
</tr>
</tbody>
</table>

- ![Secure behavior]: Secure behavior
- ![Partially secure behavior]: Partially secure behavior
- ![Insecure behavior]: Insecure behavior
- ![Delegated to password manager]: Delegated to password manager

Legend:
- User interaction always required
- Maps credentials to domains
- Won’t fill HTTPS→HTTP
- Fills password only on transmission
- Won’t fill different action (static)
- Won’t fill different action (dynamic)
- Won’t fill cross-origin frame
App Testing Approach

- Android
  - Appmon / Frida for network comms
  - dex2jar to reverse apk and inspect code
  - Blackbox testing via custom apps
- iOS
  - Blackbox testing via custom apps
  - Recall that mapping is always handled by OS
## Example Appmon Data

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Method Name</th>
<th>Description</th>
<th>Date</th>
<th>Time</th>
<th>User Agent</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLiteDatabase</td>
<td>getPath</td>
<td>Name: DB Path/data/user/0/com.lastpass.ip.android/databases/autofill.db</td>
<td>Feb 29 2020</td>
<td>09:07 PM</td>
<td></td>
</tr>
<tr>
<td>SQLiteDatabase</td>
<td>rawQueryWithFactory</td>
<td>Name: SQL StatementSELECT * FROM <code>AppHash</code> WHERE <code>packageName</code> = 'com.walmart.gotchya' LIMIT 1 Name: Selection ArgumentsName: Edit Tablenull</td>
<td>Feb 29 2020</td>
<td>09:07 PM</td>
<td></td>
</tr>
<tr>
<td>SQLiteDatabase</td>
<td>execSQL</td>
<td>Name: SQL StatementDELETE FROM <code>AppHash</code> WHERE <code>packageName</code> = 'com.walmart.gotchya' Name: SQL BindArgs</td>
<td>Feb 29 2020</td>
<td>09:07 PM</td>
<td></td>
</tr>
<tr>
<td>SQLiteDatabase</td>
<td>insertWithOnConflict</td>
<td>Name: Table NamepropertiesName: NullColumnHacknullName: Valuesapp_uid=0 hits_count=27 addid=0 params=av=4.11.7.5061&amp;an=LastPass&amp;aid=com.lastpass.ip.android&amp;aid=android.vending tid=UA-44112561-1.cld=0774f1ea-e155-44d5-b941-157d6e756f81Name: conflictAlgorithm5</td>
<td>Feb 29 2020</td>
<td>09:07 PM</td>
<td></td>
</tr>
<tr>
<td>SQLiteDatabase</td>
<td>compileStatement</td>
<td>Name: SQL StatementSELECT CHANGES()</td>
<td>Feb 29 2020</td>
<td>09:07 PM</td>
<td></td>
</tr>
<tr>
<td>SQLiteDatabase</td>
<td>execSQL</td>
<td>Name: SQL StatementDELETE FROM <code>WhitelistedVaultEntry</code> WHERE <code>packageName</code> = 'com.walmart.gotchya' Name: SQL BindArgs</td>
<td>Feb 29 2020</td>
<td>09:07 PM</td>
<td></td>
</tr>
<tr>
<td>SQLiteDatabase</td>
<td>insertOrThrow</td>
<td>Name: Table NamemessagesName: NullColumnHacknullName: Valuesentry=[B@9b8db0e type=0</td>
<td>Feb 29 2020</td>
<td>09:07 PM</td>
<td></td>
</tr>
<tr>
<td>SQLiteDatabase</td>
<td>insertWithOnConflict</td>
<td>Name: Table NamemessagesName: NullColumnHacknullName: Valuesentry=[B@9b8db0e type=0Name: conflictAlgorithmnull</td>
<td>Feb 29 2020</td>
<td>09:07 PM</td>
<td></td>
</tr>
<tr>
<td>SharedPreferences</td>
<td>putString</td>
<td>Name: Keyautofill_selected_ItemName: Value(&quot;accountld&quot;:278259551208598616,&quot;type&quot;:0)</td>
<td>Feb 29 2020</td>
<td>09:08 PM</td>
<td></td>
</tr>
<tr>
<td>SharedPreferences</td>
<td>putLong</td>
<td>Name: Keylast_pause_timeName: Value1583028221795</td>
<td>Feb 29 2020</td>
<td>09:08 PM</td>
<td></td>
</tr>
<tr>
<td>SharedPreferences</td>
<td>putLong</td>
<td>Name: Keytime_activeName: Value3450</td>
<td>Feb 29 2020</td>
<td>09:08 PM</td>
<td></td>
</tr>
<tr>
<td>SharedPreferences</td>
<td>remove</td>
<td>Name: KeyLogged in to Site</td>
<td>Feb 29 2020</td>
<td>09:08 PM</td>
<td></td>
</tr>
<tr>
<td>SharedPreferences</td>
<td>remove</td>
<td>Name: KeyAutofill Item Selected</td>
<td>Feb 29 2020</td>
<td>09:08 PM</td>
<td></td>
</tr>
<tr>
<td>SharedPreferences</td>
<td>putLong</td>
<td>Name: Keytime_activeName: Value3449</td>
<td>Feb 29 2020</td>
<td>09:08 PM</td>
<td></td>
</tr>
</tbody>
</table>
Miscellaneous Things We Checked

• For every PWM:
  • Permissions required on install
    • Autofill service, observe your actions, manage keyboard, observe text you type, etc.
  • If it clears the clipboard after copying a password
  • Form types it would fill
    • Hint type, invisible form, tiny form
  • Warning rooted device
Example Test App for Mapping

Email

Password

Are you sure?

Do you want to use your walmart.com (www.walmart.com) credentials to fill popup (stormy-cove-65327.herokuapp.com)?

These fields will be copied:
Password
Email
Username

DON'T ALLOW   FILL
# Autofill in Native UI Elements

<table>
<thead>
<tr>
<th>Framework</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>iOS Password AutoFill</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>iOS App Extensions</td>
<td>●</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Android Autofill Service</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

- ● Secure behavior
- ○ Insecure behavior
- ◆ Delegated to password manager
WebView Overview

<table>
<thead>
<tr>
<th>Framework</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>iOS Password AutoFill</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>iOS App Extensions</td>
<td>●</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Android Autofill Service</td>
<td>●</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>

- ● Secure behavior
- ○ Insecure behavior
- ○○ Delegated to password manager
Violation P2

• Credential should be mapped to website hosted in WebView
• Some managers/frameworks fill the app credentials into any website hosted in WebView
• Users are conditioned to trust autofill dialogues
# WebView Overview

<table>
<thead>
<tr>
<th>Framework</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>iOS Password AutoFill</td>
<td>●</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>iOS App Extensions</td>
<td>●</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Android Autofill Service</td>
<td>●</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>

- ● Secure behavior
- ○ Insecure behavior
- ⊙ Delegated to password manager
Violation P3

- A host app should not be able to access credentials filled into a WebView
- Both iOS and Android allow JS callbacks
Javascript Callback iOS

```swift
let contentController = WKUserContentController()
contentController.add(self, name: "callbackHandler")

func userContentController(
    _ userContentController: WKUserContentController, didReceive message: WKScriptMessage) {
    if (message.name == "callbackHandler") {
        print("User credentials are \(message.body)")
    }
}
```

Listing 1: Callback to communicate with injected JavaScript

```javascript
var username = document.getElementById("email").value;
var password = document.getElementById("password").value;
var credentials = 'window.location.hostname:username:password';
window.webkit.messageHandlers.callbackHandler.postMessage(credentials);
```

Listing 2: Injected JavaScript that steals credentials and sends to malicious app
Summary & Recommendations

• P1: Users explicitly authorize operation
  • Obeyed by all mobile autofill frameworks in all contexts
• P2: Credential is securely mapped to web domain or app
  • Need a secure bi-directional app-to-domain mapping
  • Should disable autofill in cross-origin iframes
• P3: Credential is only accessible to mapped domain
  • Need secure autofill in WebView and Browser
Questions + Paper Discussion

toesch1@vols.utk.edu
@oeschsec