Pitfalls in Designing Zero-Effort Deauthentication: Opportunistic Human Observation Attacks

Otto Huhta*, Prakash Shrestha†, Swapnil Udar*, Mika Juuti*, Nitesh Saxena† and N. Asokan‡

* Aalto University
† University of Alabama at Birmingham
‡ Aalto University and University of Helsinki

NDSS’16, 24 February, San Diego, CA, USA
The deauthentication problem

• Threat:
  • unauthorized access to a terminal
  • after legitimate user has walked away

• What we actually want is zero-effort deauthentication
• Both innocent and malicious adversaries
Zero-effort deauthentication systems

- Already in use!
  - BlueProximity
  - Keyless Entry in high end cars
- Based on short-range wireless channels: RSS from user devices

http://sourceforge.net/projects/blueproximity/
ZEBRA: a recent proposal for deauthentication

Targeted for hospital wards, factory floors, ...
User may step away from Terminal but lingers nearby

Authenticate:
- Compare both sequences
- Decide “Same User” or “Different User”

- No user profiling!

IEEE Symposium on Security and Privacy (SP) 2014
http://dx.doi.org/10.1109/SP.2014.51
ZEBRA works by averaging out misclassifications [1]

Window size 10,
Threshold 70%

8/10 matches ≥ 70%
User remains logged in

Bracelet data → classes:
1. (any) typing
2. (any) scrolling
3. mouse ↔ keyboard movements (MKKM)
Only interactions seen at Terminal considered [1]

Why? User privacy [1], accuracy of classifier?
ZEBRA vs malicious attackers [1]

- Attacker **required** to mimic all of victim’s interactions
- **20 participants** as attackers; researchers as victims
  - Victims verbally announce their interactions
Does ZEBRA resist malicious attackers? [1]

Fraction of adversaries remaining logged in (window size = 21, threshold=60%)

\[ g = \text{deauthentication at } \# \text{ failed windows} \]

Average window length = 6s
Is this a reasonable adversary model?
More realistic adversary models

1. Naïve all-activity
   – As in Mare et al [1]: mimics all

2. Opportunistic keyboard-only
   – Mimics selected typing

3. Opportunistic all-activity
   – Mimics selected activities

4. Audio-only opportunistic KB-only
   – Mimics selected typing, but no line of sight
Our implementation of ZEBRA

• Implemented end-to-end ZEBRA from scratch

• Using off-the-shelf Android Wear smartwatch
  – Wider applicability: existing affordable models

• Re-use ZEBRA parameters/methodology wherever possible
## Parameter comparison

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Original implementation</th>
<th>Our implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum duration</td>
<td>25 ms</td>
<td>25 ms</td>
</tr>
<tr>
<td>Maximum duration</td>
<td>1 s</td>
<td>1 s</td>
</tr>
<tr>
<td>Idle threshold</td>
<td>1 s</td>
<td>1 s</td>
</tr>
<tr>
<td>Window size</td>
<td>21</td>
<td>20</td>
</tr>
<tr>
<td>Match threshold</td>
<td>60%</td>
<td>60%</td>
</tr>
<tr>
<td>Overlap fraction</td>
<td>Not reported</td>
<td>0</td>
</tr>
<tr>
<td>Grace period</td>
<td>1, 2</td>
<td>1, 2</td>
</tr>
<tr>
<td>Classifier</td>
<td>Random forest</td>
<td>Random forest</td>
</tr>
<tr>
<td>Classifier training data</td>
<td>Form filling</td>
<td>Form filling</td>
</tr>
<tr>
<td>Validation methodology</td>
<td>Not reported</td>
<td>Leave-one-user-out</td>
</tr>
</tbody>
</table>

- Bracelet hardware, datasets used...
Our implementation Architecture

User

Bracelet
- Accelerometer & Gyroscope measurement
- Segmenter
- Feature Extractor
- Communicator
  - Interaction time interval
  - ZEBRA Engine
- Segment data

Terminal
- Input events Listener
- Input Events Listener
- Interaction Extractor
- Authenticator
  - Predicted Interaction sequence
  - Interaction time interval
  - Actual Interaction Sequence
  - Features
- Communicator

ZEBRA Engine
- Synchronize time, transfer interactions and feature set

Android Wear application for smartwatch
Matlab Random Forest classifier for interaction classification
Java application for Terminal

Same user
Or
Different user

Mika Juuti: Pitfalls in Designing Zero-effort Deauthentication
Our implementation of ZEBRA (2)

Synchronize time, transfer interactions and feature set

Zebra/java$ find -name *.java -print | xargs grep -v "\\\\\\" | grep -v "1$" | grep -v "*" | wc -l
Zebra/java$ 7706
Naïve malicious attackers: comparison

Original malicious attacker (naïve) [1]

- 20 participants as victims; researchers as attackers
- All attackers are deauthenticated

Our naïve all-activity attacker

g = deauthentication at # failed windows
ZEBRA does not resist opportunistic malicious attackers

\[ g = \text{deauthentication at \# failed windows} \]

Original malicious attacker (naïve)
- 20 participants as victims; researchers as attackers
- Attackers do not eventually get logged out
Can still protect against innocent “attackers”

- mismatched traces model innocent attackers
- All users eventually deauthenticated
- Avg. window length = 14s

Mismatched user traces

\[ g = \text{deauthentication at } \# \text{ failed windows} \]
What went wrong? [1]

1. Inadequate adversary modeling in [1]!

2. Fundamental design flaw in ZEBRA:
   "Authentication based on input source controlled by adversary"
   
   – Attacker controls Terminal:
     • Can choose type/timing of interactions

   – A case of tainted input:

   ![Comic strip](https://xkcd.com/327/)
Strengthening ZEBRA [1]

- Recognizing more terminal interactions
- Recognizing off-terminal interactions!
- Black/whitelisting, sanitizing input
- Augmenting with trusted input: RSS
Take-home message

1. **Zero-effort security** is appealing
   - Balance between usability and security
   - Care in defining adversary model

2. ZEBRA susceptible to opportunistic attackers, still effective for preventing accidental misuse

*Ask me for a demo!*