Continuous Authentication via Bluetooth and Mobile Devices

Asish Gummidi
*asishgummidi96@gmail.com

* Corresponding author
doi: https://doi.org/10.21467/proceedings.7.6.12

***Abstract*—Passwords are often insecure and easily forgotten, and current passwordless authentication methods, such as bio- metrics and hardware authenticators, come with additional costs and only offer one-time authentication. This creates vulnerabilities when users leave their computers unlocked. ProxyAuth, an open-source continuous authentication system, addresses these issues by using an Android phone as a hardware authenticator. The system continuously authenticates and de-authenticates the user based on their proximity to the computer, detected through Bluetooth bandwidth analysis. When the user moves away from the computer, the system locks, ensuring security without requiring additional hardware. ProxyAuth aims to provide a secure, cost-effective, and convenient alternative to traditional password- based authentication methods.**

I. Introduction

A. Problem Statement

There is right now no broadly accessible passwordless ver- ification technique for signing all through a Linux Work area Climate utilizing a cell phone without client communication. Existing arrangements require extra equipment, for example, YubiKeys or unique mark scanners, which may not be reason- able for all clients. In addition, these strategies come up short on auto-lockout highlight, depending on clients to physically log out, which is in many cases dismissed, prompting potential security dangers like unapproved access, information robbery, or malware establishment.

ProxyAuth proposes a savvy arrangement involving a cell phone as an equipment authenticator by means of Bluetooth. This approach guarantees passwordless confirmation, persis- tent verification/de-validation, and programmed framework locking in view of the telephone's nearness while safeguarding against Bluetooth enhancer assaults.

B. Rationale

The requirement for ProxyAuth emerges from individual and group encounters, where leaving frameworks opened during breaks prompted potential security concerns. Signing in over and again with long passwords was awkward. ProxyAuth means to address these difficulties with a functional and creative arrangement.

The group utilized different abilities for this task, including Android improvement, Linux programming, and an eagerness to learn new innovations like PAM and Bluetooth conventions. The venture gave a potential chance to upgrade Linux information and work on framework level programming, lining up with individual interests and abilities.

C. ProxyAuth Implementation

By the last run, the accompanying parts were created:

1) A Linux PAM module empowering validation through trusted, matched cell phones.
2) An Android application that ceaselessly screens vicinity and keeps the PC when the telephone is out of reach.

The PAM module checks matched gadgets against a whitelist of Macintosh addresses and validates the client in the event that a match is found. A foundation de-validation server consistently speaks with the telephone to keep up with confirmation. On the off chance that the believed gadget detaches or an unapproved gadget endeavors correspondence, the framework locks right away.

D. Security Considerations

ProxyAuth tends to the accompanying expected assaults:

- **Bluetooth Spoofing:** Messages between the telephone and PC are scrambled utilizing AES-GCM to guarantee trustworthiness and genuineness.
- **Bluetooth Intensification Attacks:** Proposed counter- measures incorporate breaking down Bluetooth parcel in- activity and data transmission to appraise the telephone's separation from the PC.

Proceedings DOI: 10.21467/proceedings.7.6; Series: AIJR Proceedings; ISSN: 2582-3922; ISBN: 978-81-989164-9-5

- **Device Theft:** While relieving taken telephone chances is past the task's extension, future elements might address this worry.

Testing included endeavors to take advantage of Bluetooth weaknesses utilizing Ubertooth One, an open-source Bluetooth testing gadget. While effective execution of these assaults was deficient inside the venture course of events, plans for hearty safeguards were laid out.

II. Background and Related Work

A. Background

Secret key-based validation is generally viewed as unreliable because of issues like feeble, taken, or reused passwords, and weakness to phishing and perception attacks [1], [2]. Notwithstanding progressions, for example, secret phrase supervisors, these arrangements actually depend on an expert secret phrase, making them helpless against compromise [3]. Multifaceted validation (MFA) and biometric frameworks have shown guarantee however frequently require extra equipment or effect client convenience [4]. Adjusting convenience, deployability, and security is fundamental for compelling confirmation methods [5].

ProxyAuth expects to give passwordless validation by utilizing a client's cell phone and Bluetooth closeness, limiting extra equipment costs. The framework verifies clients in light of ownership (cell phone) and area (vicinity) while keeping up with comfort. ProxyAuth can incorporate with other confirmation strategies, empowering multi-modular security without critical client trouble.

B. Related Work

Different closeness based validation arrangements have been created. XYLOC by Guarantee Advancements uses custom radio frequencies however requires extra hardware [6]. Essentially, wearable-based frameworks, similar to Apple Watch's Macintosh open element, stay bound to explicit ecosystems [7]. Outsider applications, like Close to Lock, give practically identical usefulness yet need cross-stage support [8].

Projects like ZEBRA and Glass Wearable Gadgets propose imaginative arrangements utilizing movement following or wearable gadgets however present equipment conditions or complexity [9]. ProxyAuth separates itself by utilizing existing cell phones and Bluetooth innovation, guaranteeing more extensive openness and negligible client exertion. Not at all like earlier works, it targets Android and Linux clients, giving a minimal expense, easy to understand arrangement.

III. Methodology and Plan Goals

A. Methodology

ProxyAuth comprises of an Android application, PAM, and a de-confirmation server. Advancement followed a spry technique with iterative runs, empowering adaptability and gradual advancement. The Android application handles correspondence with the server utilizing Bluetooth, integrating highlights, for example, posting matched Macintosh addresses, laying out RFCOMM attachments, and supporting foundation administrations, with encryption as a future objective. PAM validates the PC by checking believed gadgets through a matched Macintosh address list and conjures the de-confirmation server post-login. The de-verification server keeps up with persistent correspondence with the cell phone, locking the PC assuming the gadget neglects to answer difficulties in 10 seconds or less. Vicinity based lockout and extra encryption were arranged however not finished.

B. Design Goals

ProxyAuth's key elements incorporate portable based PC confirmation and programmed lockout when the client is out of reach. The Android application records matched gadgets, works with secure correspondence, and incorporates an off but- ton, while PAM guarantees login just with confided in gadgets and alternatively upholds multifaceted confirmation. The de- verification server oversees challenge-reaction correspondence and vicinity-based lockouts, with encryption improvements as a forthcoming component.

IV. Towards AI-Enhanced Continuous Authentication

A. Integrating AI for Behavioral and Proximity Intelligence

While ProxyAuth already leverages proximity sensing via Bluetooth, expanding this into a more intelligent, adaptive system unlocks new layers of security. Future iterations aim to introduce machine learning and deep learning components that bring human-like intuition to device behavior analysis.

Proceedings DOI: 10.21467/proceedings.7.6
ISBN: 978-81-989164-9-5

Series: AIJR Proceedings
ISSN: 2582-3922

a) 1. Anomaly Detection via Device Behavior Modeling: We envision training models to detect abnormal interaction patterns between the phone and computer—such as unexpected disconnections, timing irregularities, or Bluetooth signal spoofing. These can be flagged as potential threats in real- time using recurrent neural networks (RNNs) or lightweight anomaly detectors like Isolation Forests, ensuring active defense against sophisticated intrusions.

b) 2. Proximity Prediction with Machine Learning: Beyond fixed bandwidth thresholds, we plan to implement models (e.g., regression-based or neural proximity classifiers) trained on Bluetooth RSSI (Received Signal Strength Indicator) and device orientation data to estimate proximity more accurately. These models adapt over time to the user's movement patterns and environment, enabling context-aware locking and unlocking mechanisms.

c) 3. Gait and Behavioral Biometrics: AI can be harnessed to understand user-specific mobility through gait analysis using accelerometer and gyroscope data from the phone. Each user's walking pattern forms a unique biometric signature. Combined with time-of-day usage patterns and device interaction behavior, this allows ProxyAuth to verify identity continuously without the user lifting a finger.

V. Tools and Libraries Used

A. Hardware

ProxyAuth utilizes a few equipment parts: an Android cell phone running SDK rendition 20 or higher, which was chosen for its wide reception and open-source nature; a Linux PC with Bluetooth support, normally running Ubuntu 18.04 LTS or 19.04, for confirmation purposes; Bluetooth connectors for testing, particularly on virtual machines lacking local Bluetooth; and the Ubertooth One, an open-source 2.4 GHz remote improvement stage, for trying different things with Bluetooth signal assaults, in spite of the fact that its utilization was restricted (see Segment 6.3).

B. Software

Programming instruments incorporate Android Studio for fostering the Android application, Elf Work area Climate for similarity with the de-verification server and PAM, and Ubuntu for the advancement climate. We utilized the gcc compiler (rendition 7.4) and the Linux PAM Module for framework validation. Virtual Box, at first involved by some colleagues for testing, was subsequently dropped. D-Feet, an instrument for troubleshooting D-Transport messages, was likewise utilized.

C. Programming Language and Scripts

C was picked for advancement because of its similarity with Linux and PAM, as well as its documentation and local area support. Kotlin was chosen for Android improvement because of its compact sentence structure and present-day highlights, while Perl was utilized for prearranging undertakings like reliance establishment.

D. Collaboration/Task The board Tools

Slack filled in as the primary correspondence stage, coordinating with Github to advise colleagues of updates. Github was utilized for adaptation control and task the board, including issue following and pull demands, improving cooperation and association.

E. Additional Programming Packages

We involved Bluez for Bluetooth convention support on Linux, Talkative for D-Transport connections, and PAM libraries for framework verification. Instruments like Apache 2 and a switch were considered for future elements, for example, boycotting lost gadgets and confining confirmation to a similar organization.

VI. Project Overview

ProxyAuth comprises of three fundamental parts: the Android application, situated in proxyAuth/android; the Linux PAM module, situated in proxyAuth/pam/src/pam_proxy.c; and the de-validation server, situated in proxyAuth/pam/src/deauth.c. The validation interaction can be separated into two primary stages. To begin with, during the **one-time authentication**, the client should turn on Bluetooth on both the Android gadget and the Linux machine, match the gadgets, and lock the PC. After awakening, ProxyAuth will confirm the client assuming that the gadgets are as yet matched. Second, during **continuous authentication**, the de-validation server persistently checks in the event that the matched Android gadget is in scope of the PC. Assuming the gadget moves out of reach or quits answering, the framework locks the PC.

Proceedings DOI: 10.21467/proceedings.7.6
ISBN: 978-81-989164-9-5

Series: AIJR Proceedings
ISSN: 2582-3922

VII. Prerequisites for Clients and Developers

The framework requires a Linux machine with Bluetooth capacities and a little person work area climate, along- side an Android cell phone. The product conditions incorporate libpam0g-dev, libglib2.0-dev, bluez, and libbluetooth dev. To set up the framework, clients should introduce these conditions and make important registries, for example, /lib/security for the PAM mod- ules. The /and so on/pam.d/gdm-password docu- ment should be altered to incorporate pam_proxy.so, and believed Bluetooth Macintosh addresses should be added to the /and so on/proxy_auth/ index for every client. Moreover, the Bluetooth server design should be adjusted to address similarity issues with Bluez 5, and consents for /var/run/sdpshould be refreshed on reboot.

VIII. PAM Implementation

PAM (Pluggable Validation Modules) considers the execution of custom confirmation strategies in Linux. In this venture, the ProxyAuth PAM module (pam_proxy.so) is coordinated into the gdm-password setup to verify clients in light of the vicinity of their Bluetooth-empowered Android gad- gets. The confirmation cycle includes checking the matched Android gadget against a rundown of believed gadgets put away on the framework. This confirmation is performed by the Bluetooth login capability, which checks in the event that the Android gadget is on the confided in list.

A. De-confirmation Program

The de-confirmation server speaks with the authenticator and locks the framework in the event that there is an issue, for example, not getting information or recognizing the au- thenticator is out of reach. The server is sent off by the PAM module with a solitary boundary: the Macintosh address of the gadget that verified PAM. The source code is situated in proxyAuth/pam/src/deauth.c.

1) Checking Deauth's Argument: Prior to beginning, deauth checks in the event that the given contention is a substantial Bluetooth address by calling check_arg. It additionally checks assuming that the gadget is matched and confided in utilizing is_trusted_client. If both of these checks' fizzles, the program locks the framework.

2) Bluetooth Overview: Bluetooth works at 2.4 GHz with a scope of 10-100 meters, contingent upon the class of the gad- get. The de-verification program utilizes the Serial Port Profile (SPP) to empower information move between the PC and the authenticator. The capability init_server sets up the Bluetooth administration, including restricting the attachment and enrolling the help.

```
set_service
set_bluetooth_servic
e_info set_browsable
set_l2cap_info
register_rfcomm_soc
k register_service
```

The interaction incorporates setting a UUID for the help, characterizing the help class, setting Bluetooth profiles, making the assistance browsable, and enrolling the RFCOMM channel.

3) Communication among Deauth and Authenticator: When a Bluetooth server is laid out, deauth checks assuming that the authenticator is the very gadget that validated PAM. In the event that it isn't matched or verified, the frame- work is locked utilizing a D-Transport call to Little person's screensaver. In the event that no message is gotten from the authenticator in somewhere around 10 seconds, the association is ended, and the machine is locked.

4) Conflicting UUID because of Different Instances: A contention can happen when different occasions of deauth are brought forth after the client locks their framework. This happens on the grounds that a similar UUID is publicized by various examples. The issue is settled by having deauth end itself when it recognizes the framework is locked, utilizing a D-Transport intermediary call to Little person's Presence object.

5) Checking Vicinity of the Authenticator: To forestall as- saults utilizing repeaters, deauth computes the transmission capacity between the authenticator and PC. In the event that the data transmission falls under a specific edge, showing the authenticator is excessively far from the framework, the PC is locked. This technique depends on the way that repeaters can't send information at a similar speed as the authenticator in closeness.

B. Android Application

The Android application sends messages to the deauth program once the client is confirmed. In any case, the ongo- ing execution requires manual association with the deauth program for persistent validation. The application comprises of three exercises:

- MainActivity.kt: Records recently matched gadgets
- ControlActivity.kt: Makes an occurrence of

BluetoothInteraction administration for corre- spondence

- BluetoothInteraction.kt: Handles foundation correspondence with deauth

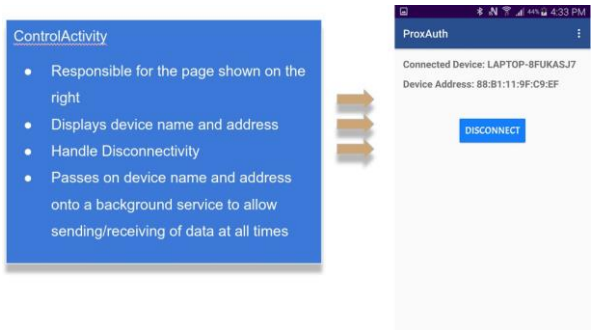

Fig. 1. An overview of what ControlActivitydoes

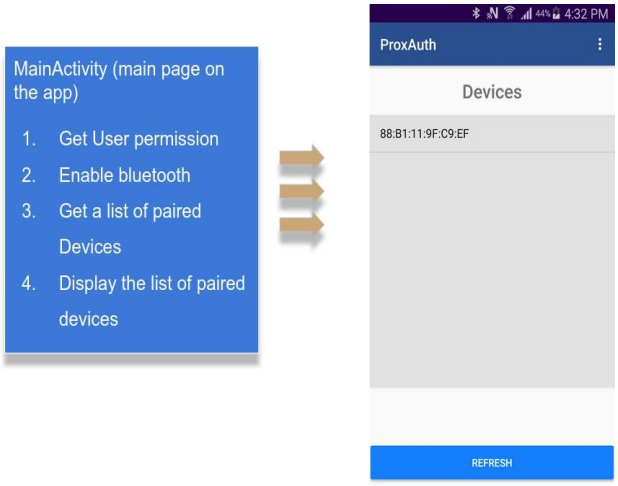

Fig. 2. An overview of what MainActivitydoes

Service Record

attribute ID	attribute value
ServiceRecordHandle	<int>
ServiceClassIDList	<UUID(s)>
ServiceID	<UUID>
ProtocolDescriptorList	<sequence>
BrowseGroupList	<UUID(s)>
DocumentationURL	<URL>
...	

optional

Fig. 3. Data representation of the service record. Taken from a paper about the development of Service Discovery Architecture [10]

Test 2: Deauth spawn by PAM Manual Lock

Check the pid of `deauth`

```
$ date
Wed Apr  1 01:30:53 EDT 2020
$ ps -u zaku | grep deauth
16484 ?        00:00:00 deauth
```

Unfortunately to test PAM with the version of the android application I have, I have to manually send messages. But if I continously send message and manually lock the system, I should see the pid of `deauth` to change.

Result: The pid has changed

```
$ date
Wed Apr  1 01:31:13 EDT 2020
zaku@zaku-laptop:~/Documents/csc490/proxyAuth/pam$ ps -u zaku | grep deauth
16657 ?        00:00:00 deauth
```

[SUCCESS]

Simple Regression Testing

Lock the system if the device is not either valid, trusted, or paired

Test Case: The device is not paired

```
$ bluetoothctl info F0:81:73:92:2E:C2 | grep Connected
```

Result:

```
$ /etc/proxy_auth/deauth F0:81:73:92:2E:C2
pika: 20:DA:22:DE:0F:68
Registering UUID 00001101-0000-1000-8000-00805f9b34fb
accepted connection from 00:00:00:00:00:00
pika: 20:DA:22:DE:0F:68
00:00:00:00:00:00 is not trusted or not authorized to deauthenticate the system
```

Fig. 4. A sample showing how Kim documents and tests every issue and pull request assigned to him

IX. Evaluation and Illustrations Learned

A. Technical and Security Evaluation

1) Technical Evaluation: Testing was necessary to our improvement interaction. We tried the PAM and de-validation server by verifying clients through Bluetooth and guaranteeing the highlights met the objectives in Segment. Genuine gadgets, not virtual machines, were utilized for testing to stay away from issues, and relapse tests were performed after each run. Testing devices and documentation, particularly for little high- lights like Macintosh address enlistment, were carried out and itemized by Kim. Testing for PAM involved matching gadgets and checking in the event that confirmation works, while the de-validation server was tried with Android applications and data transfer capacity checks for closeness identification.

2) Security Evaluation: Because of time imperatives and issues with Ubertooth One, security assessments, for example, repeater

Proceedings DOI: 10.21467/proceedings.7.6
ISBN: 978-81-989164-9-5

Series: AIJR Proceedings
ISSN: 2582-3922

assaults and Bluetooth mocking, were not directed. Be that as it may, AES-GCM encryption was endeavored to get correspondence between gadgets. Since Bluetooth security re- lies upon the fundamental matching techniques, we depended on Bluetooth's P-256 matching calculation and AES-CCM encryption for correspondence security. A couple of worries were raised about potential security weaknesses in the C code, for example, support spills over and memory access issues, however severe gathering and sanitizer banners were utilized to limit gambles. In spite of the fact that there are chances, for example, the de-validation server being killed by aggressors, we expect these weaknesses have low probability in the ongoing extension. Future work will plan to address these worries, particularly with respect to Bluetooth satirizing and nearness checks.

B. From Partial to Full Implementation

The initial ProxyAuth implementation laid the groundwork for robust authentication, but key security layers remain under development. Upcoming versions will:

- Integrate end-to-end AES-GCM encryption for all communication streams.
- Implement spoof-resistance mechanisms using crypto- graphic challenge-response authentication.
- Enable fully hands-free operation with background Blue- tooth handshakes and seamless re-authentication when the user returns.

These enhancements ensure the system moves from a working proof-of-concept to a production-ready tool.

C. Expanding the Evaluation Scope

Security and technical evaluations so far have focused on core stability and functional correctness. We plan to widen the evaluation criteria to include:

- Simulated spoofing and relay attack scenarios using adversarial tools (e.g., Bluetooth repeaters).
- Longitudinal studies on battery usage and performance impact across devices.
- Usability tests with diverse users to refine interaction models and minimize friction.

Coupling empirical testing with statistical models will guide iterative design choices and ensure real-world robustness.

X. Concluding Remarks

ProxyAuth is a zero-exertion validation framework involving a cell phone as an authenticator. It locks the PC when the client moves away, however, nonstop verification actually requires manual activity. While fragmented, it offers a practical option in contrast to customary techniques, with potential for higher security and comfort.

Learning Outcomes

This venture featured the difficulties of collaboration and coordination.

Teamwork: Unfortunate correspondence obstructed progress.

Workflow: Devices like GitHub, Kanban, and Slack aided, and GitHub Activities might have robotized testing.

Linux, Android, and C: I learned Linux verification, D- Transport, Bluetooth, memory the board, and contrasted Kotlin with Java.

XI. Future Work

The venture has significant opportunity to get better, especially in gathering its underlying plan targets. In the Android application, we mean to show just matched and validated Macintosh addresses in the MainActivity, with the capacity for clients to enlist new gadgets physically. Auto-matching and persistent correspondence with the PC will be executed for consistent verification without client mediation. Furthermore, we intend to add an additional layer of encryption on top of Bluetooth security, offer discretionary security highlights like PIN or biometric validation, and give a site to boycott lost or taken telephones from verifying laptops.

For the PAM module, we will uphold multifaceted validation by means of an adjustable setup generator or particular PAM module. We will likewise upgrade security by executing challenge-reaction verification and nearness checks to forestall ridiculing and assaults. On the de-confirmation server side, we intend to guarantee that the PC locks on the off chance that the authenticator doesn't interface inside a brief period of time, support other work area conditions, and supplant GDBus with libdbus for more extensive similarity. Furthermore, we will empower clients to tweak the vicinity distance for lockouts and acquaint a more mind-boggling challenge with increment protection from parodying and man-in-the-center assaults.

Proceedings DOI: 10.21467/proceedings.7.6
ISBN: 978-81-989164-9-5

Series: AIJR Proceedings
ISSN: 2582-3922

A. Scalability and Platform Generalization

At present, ProxyAuth is optimized for GNOME-based Linux distributions. To reach broader adoption, the architecture will be modularized and adapted for:

- KDE Plasma and other popular Linux desktop environments.
- macOS integration via CoreBluetooth and PAM plug-ins.
- Windows compatibility through native Bluetooth APIs and Credential Provider extensions.

This platform independence ensures ProxyAuth's applicability across enterprise and academic environments alike.

REFERENCES

[1] Verizon, "2017 data breach investigations report," Verizon, Report, 2017.

[2] HYPR, "New password study by hypr finds 78% of people had to reset a password they forgot in past 90 days," *HYPR*, Dec 2019. [Online] Available: https://www.hypr.com/hypr-password-study-findings/.

[3] S. Palfy, "How much do passwords cost your business?"InfoSecurity Magazine, June 2018. [Online]. Available:https://www.infosecurity-magazine.com/opinions/how-much-passwords-cost/

[4] Microsoft, "Password-less protection: Reduce your risk exposure with password alternatives," Microsoft, Report, 2018.

[5] P. C. V. Oorschot, *Computer Security and the Internet: Tools and Jewels*. Springer, 2020.

[6] E. Technologies, "Xyloc solutions overview," Product Brochure, Ensure Technologies, 2009. [Online]. Available: https://www.ensuretech.com/wp-content/uploads/2011/08/xyloc-overview-brochure.pdf

[7] Apple, "How to unlock your mac with your apple watch," https://support.apple.com/en-ca/HT206995, Apple, Oct 2019.

[8] N. Lock, "A new way to lock your mac. just walk away," https://nearlock.me/press, Near Lock, 2015. [Online]. Available: https://nearlock.me/press

[9] S. Mare, A. M. Markham, C. Cornelius, R. Peterson, and D. Kotz, "Zebra: Zero-effort bilateral recurring authentication," in *2014 IEEE Symposium on Security and Privacy*, 2014, pp. 705–720.

[10] C. Schwingenschlogl and A. Heigl, "Development of a service discovery architecture for the bluetooth radio system," 09 2000.