LINUX AUTHENTICATION DESIGN

SOLUTION BRIEF

Linux Authentication - Interactive and Service account (M2M) authentication **without Passwords, SSH Keys and Secrets**

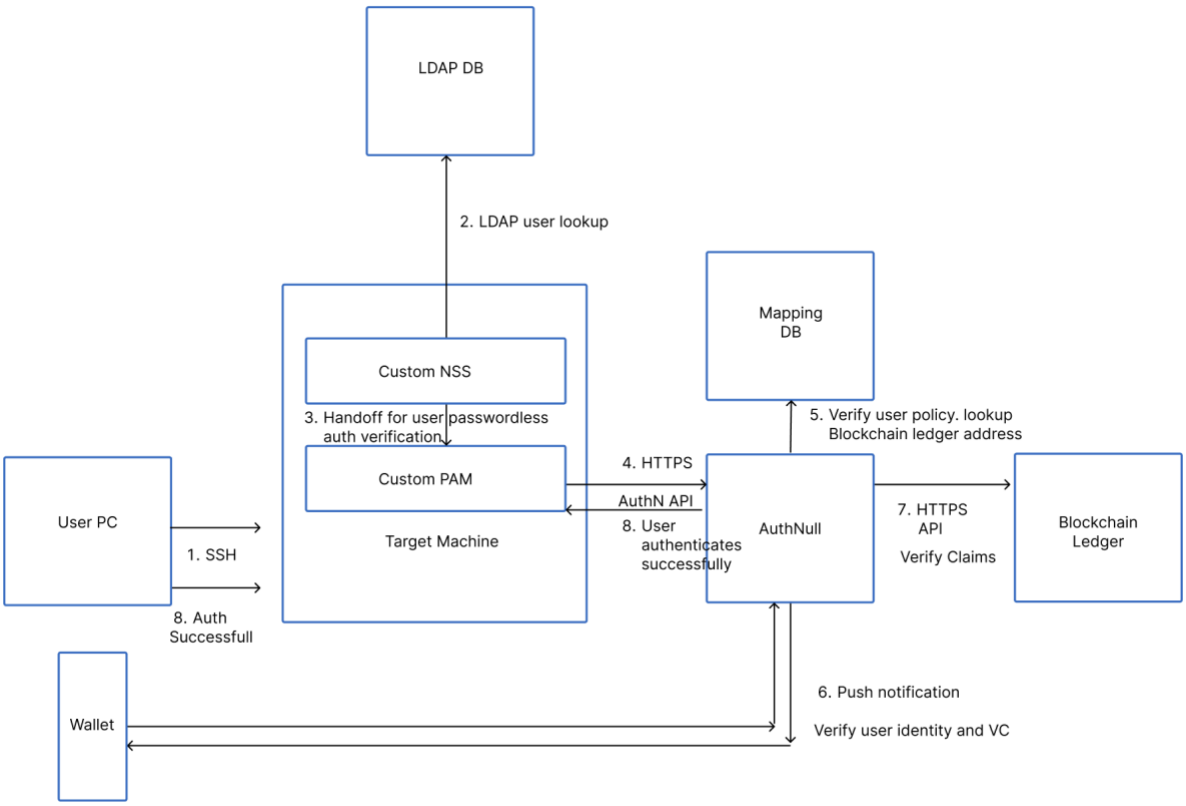
AuthNull

Overview

This document provides an overview of the M2M Linux authentication flows in AuthNull using Active directory users. This method does not require passwords, or SSH keys and entirely uses a Passwordless credential to deliver Machine to machine / service account authentication.

Use case #1: Interactive Passwordless authentication

Logical architecture and flow

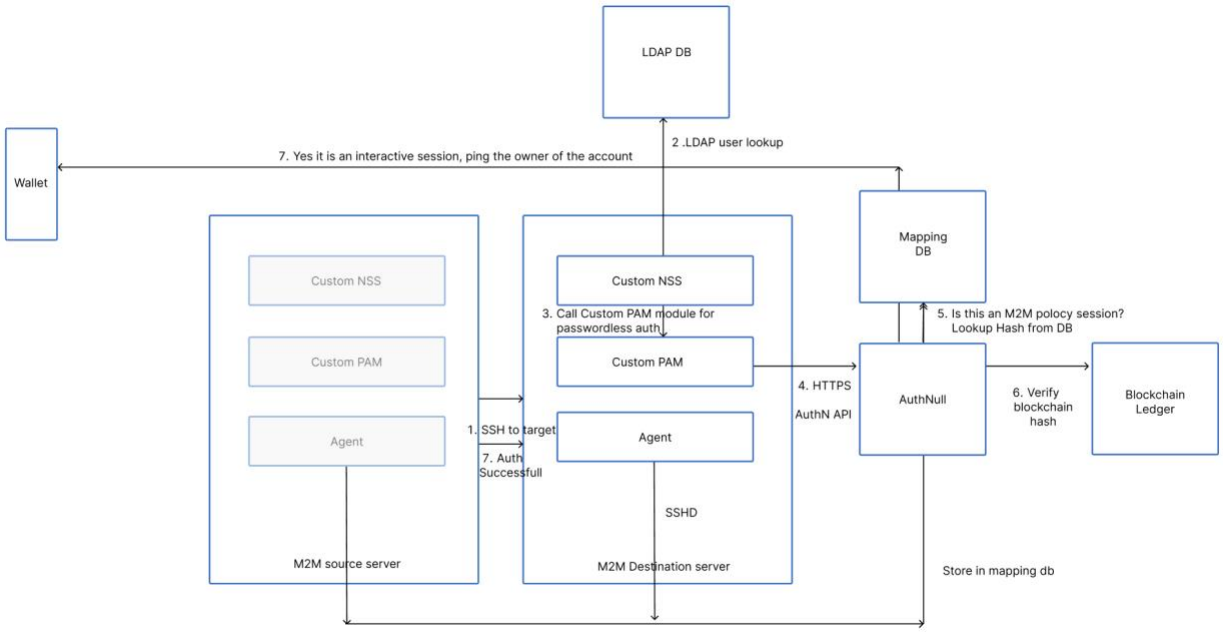


Steps	What happens
Step #1	User attempts to connect to target machine using SSH Ssh asif@1.1.1.1
Step #2	Custom NSS module does a directory lookup to verify the user identity on an LDAP directory (LDAP / TCP). When successful this moves to step #3. If not successful – authentication will be denied.
Step #3	Custom PAM module initiates Passwordless authentication from wallet
Step #4	AuthNull server gets called to verify interactive authentication policy from the mapping db.

Step #5	Mapping DB is used to look up authentication policy and verify - Does this user actually have access rights? Does an interactive policy exist? - Lookup Address of blockchain ledger.
Step #6	User / Owner of account gets a push notification on wallet - User submits presentation submission (credential signed by private key) if he accepts the authentication request, or deny the request if they think someone else is accessing the account. - This PR can be verified using users public key - Additionally, it is converted to a hash with current salt and random string for the day. - This can be considered as user Hash
Step #6	The blockchain ledger hash is looked up from the address from Mapping DB for this user authentication policy. This hash is compared to the computed hash retrieved from the wallet. If both the hashes match, it further verifies that the user has the correct credentials using blockchain
Step #7	User is able to authenticate successfully.

Use case #2: M2M authentication

Logical architecture and flow

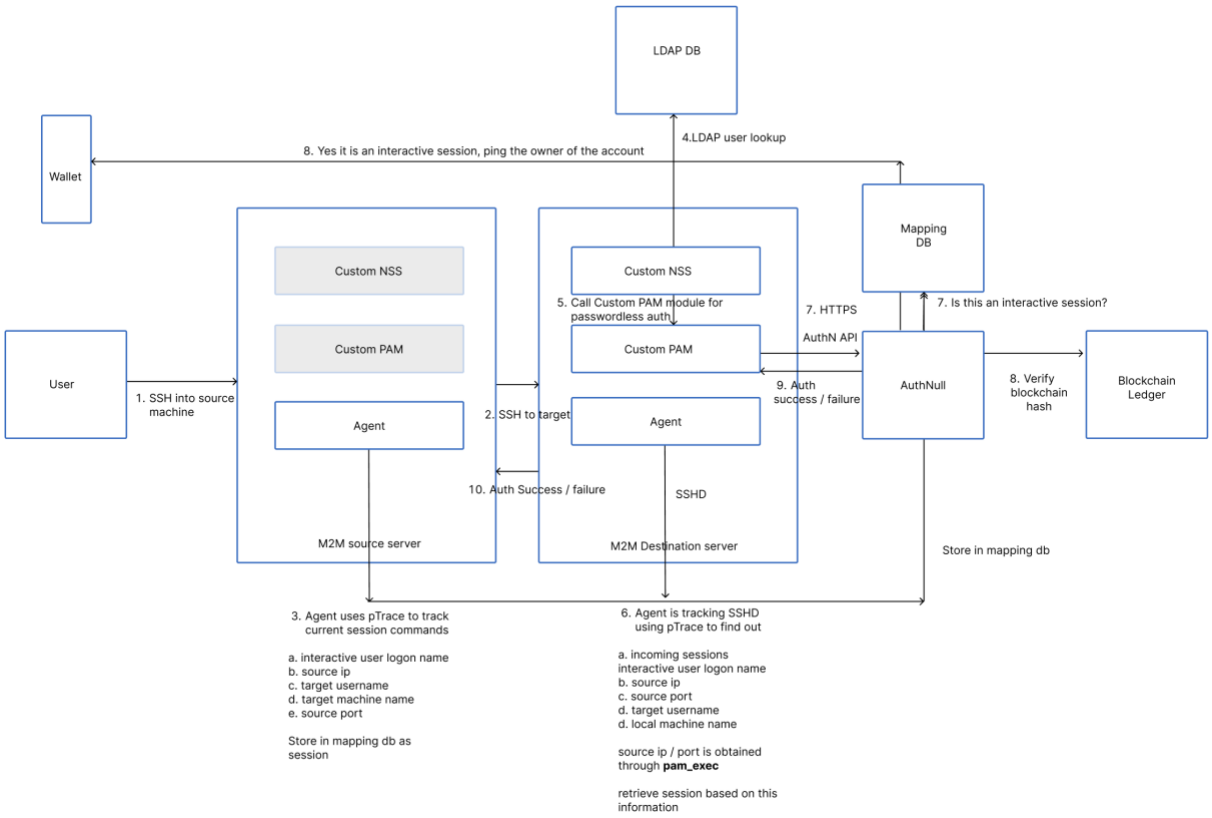


Steps	What happens
Pre-requisite	Admin (who owns m2m credential) first delegates credentials to a blockchain ledger and writes the hash in mapping db, along with address of ledger. Each target authentication machine and its unique IP also have a machine key, and a policy stored in the mapping db. Policy exists defining the authentication as a m2m policy. Daily hash on mapping db (hash1) Hash on Ethereum chain (hash2)
Step #1	Script executes SSH M2M authentication from Source server to Destination server.
Step #2	Custom NSS module does a directory lookup to verify the user identity on an LDAP directory (LDAP / TCP). When successful this moves to step #3.

	If not successful – authentication will be denied
Step #3	Custom PAM module initiates auth for passwordless auth
Step #4	AuthNull server API is called to verify authentication -
Step #5	AuthNull verifies from Mapping db It is verified that this is a M2M policy from the mapping DB. - Does this user have access rights? Does a policy exist. - Address of blockchain ledger. - Hash for verification (original hash)
Step #6	The blockchain ledger hash is looked up from the address from Mapping DB for this user authentication policy. This hash is compared to the computed hash retrieved from the db. If both the hashes match, it further verifies that the m2m authentication is correct has the correct credentials using blockchain
Step #7	User is able to authenticate successfully.

Use case #3: M2M authentication attempted through a privileged interactive user authentication

Logical architecture and flow



Steps	What happens
Prerequisite	Admin (who owns m2m credential) first delegates credentials to a blockchain ledger and writes the hash in mapping db, along with address of ledger. Each target authentication machine and its unique IP also have a machine key, and a policy stored in the mapping db.
Step #1	Privileged User initiates SSH connection to M2M source machine
Step #2	User initiates SSH authentication from M2M Source Server to M2M Destination Server using a m2m policy user example postgres Let's assume current logon user is admin Admin @ 1.1.1.1 $> ssh postgres@2.2.2.2

Step #3	Agent uses Ptrace on source machine to track current session commands and collects the following information: a. interactive user logon name [admin] b. source ip 1.1.1.1 c. target username [postgres] d. target machine ip 2.2.2.2 e. source port that is used for the outbound connection. This is stored in mapping db as a session and a unique hash is generated representing this session.
Step #4	Custom NSS module does a directory lookup to verify the user identity on an LDAP directory (LDAP / TCP). When successful this moves to next step. If not successful – authentication will be denied.
Step #5	Custom PAM module initiates auth for passwordless auth for SSH.
Step #6	Ptrace to track current incoming SSHd sessions and collects the following information: a. incoming session's interactive user logon name b. source ip c. target username d. target machine name e. source port Each source port is unique for each session, and therefore can be used to uniquely identify each session. The source ip and source port is identified through pam_exec which is also used to ensure that a given session is looked up from the mapping db, to find if there's a unique session hash. If a session hash exists, and this has an interactive user
Step #7	AuthNull server is called to verify the user authentication.

	1. AuthNull server checks to find if there's an M2M policy in Mapping db – yes. This is a M2M policy and a hash, with a address of the blockchain. 2. AuthNull checks Mapping db for active **interactive sessions** from source machine The mapping is done using source ip, source remote port, source username, and these values are typically also found on the destination machine using pam_exec, during the course of the session. 3. If this mapping session db has a session hash, along with an interactive user then this session is considered an interactive user. 4. AuthNull finds an active user logged on who initiated this authentication and reverts to evaluating this request as an interactive authentication.
Step #8	Owner of the wallet is notified with a push notification to accept or deny the authentication request. If the owner of the user denies the request, authentication will fail.
Step #7	The blockchain ledger hash is looked up from the address from Mapping DB for this user authentication policy. This hash is compared to the computed hash retrieved from the db. If both the hashes match, it further verifies that the m2m authentication is correct has the correct credentials using blockchain
Step #8	User is able to authenticate successfully.

Key problem statements

How do we ensure we convert non interactive authentication when initiated by an interactive user?

How do we map user sessions from source machine (M1) to destination Machine (M2) and ensure that a privileged interactive user is not leveraging M2M policy and bypassing security controls

Solution approaches to solving the user mapping problem?

1. Map user sessions from source M1 and M2 using session level variables and ensuring that interactive user sessions are identified
2. Block authentication on source M1 when interactive authentication is identified
3. Use BPF to trace inbound SSH connections and block interactive authentications
4. Generate a session key and pass it as arguments using xargs within a SSH session

AuthNull uses solution approach #1 and #2 for identifying and mapping user sessions from source machine (M1) and destination machine (M2)

Approach #1 Map user sessions from source M1 and M2 using session level variables and ensuring that interactive user sessions are identified

Steps:

1. The following information is collected at the time of the session on the source machine using ptrace for every sshd outbound session

Machine M1						
Machine name	Source Privileged User	Service Account	Source Port	Destination Port	Target Machine	Mapping db key (generated)
M1	NA	kloudonedb	5110	22	M2	Key1
M1	NA	kloudonedb	5111	22	M2	Key2
M1	NA	kloudonedb	5112	22	M2	Key3
M1	sharan	kloudonedb	5113	22	M2	Key4
M1	sharan	kloudonedb	5114	22	M2	Key5
M1	asif	kloudonedb	5115	22	M2	Key6
M1	asif	kloudonedb	5117	22	M2	Key7

A unique key is generated for each session. When the session is an interactive user, the source privileged i.e. the interactive user is also stored as a part of the session db.

2. In the destination machine – the following information is collected on a per session basis, especially remote_ip and remote_port using pam_exec.
   Since pam_exec is on a per session basis, the same is used to trigger validation of whether the user is actually an interactive user by mapping the collected information from the db.

   For example: From the above / below collected in the mapping table, a connection with remote port 5117, for service account AuthNulldb from M1 to M2 has a unique privileged user asif who is attempting interactive authentication.

Machine M2				
destination user	remote host	remote port	target user	Target \ip
kloudonedb	M1	5110	kloudonedb	M2
kloudonedb	M1	5111	kloudonedb	M2
kloudonedb	M1	5112	kloudonedb	M2
kloudonedb	M1	5113	kloudonedb	M2
kloudonedb	M1	5114	kloudonedb	M2
kloudonedb	M1	5115	kloudonedb	M2
kloudonedb	M1	5117	kloudonedb	M2

3. Given such a mapping found for interactive user “asif” attempting to connect to “AuthNulldb” – AuthNull finds from the mapping table that this is actually a service account user belonging to wallet muthu@AuthNull.com
4. Therefore Muthu@AuthNull.com will get a notification for this attempted interactive authentication against a service account (m2m) user policy.
5. If Muthu@AuthNull.com allows the interaction on wallet, the interactive user authentication will proceed successfully.
6. If Muthu@AuthNull.com denies the authentication on the wallet, the interactive authentication for service account user AuthNulldb will fail.

Approach #2: Optionally AuthNull also provides a blocking call on source machine M1 to ensure that wallet owner gets a notification

This works as follows:

1. The following information is collected at the time of the session on the source machine using ptrace for every sshd outbound session

	Machine M1					
Machine name	Source Privileged User	Service Account	Source Port	Destination Port	Target Machine	Mapping db key (generated)
M1	NA	kloudonedb	5110	22	M2	Key1
M1	NA	kloudonedb	5111	22	M2	Key2
M1	NA	kloudonedb	5112	22	M2	Key3
M1	sharan	kloudonedb	5113	22	M2	Key4
M1	sharan	kloudonedb	5114	22	M2	Key5
M1	asif	kloudonedb	5115	22	M2	Key6
M1	asif	kloudonedb	5117	22	M2	Key7

2. Since the source machine already knows that the attempted authentication is against AuthNulldb service account user whereas this user is a m2m user – AuthNull looks up the policies for the account “AuthNulldb” for the associated machine “m2”
3. When AuthNull finds “AuthNulldb” belongs to muthu@AuthNull.com – it can initiate a blocking approval request to the owner muthu@AuthNull.com to whether to allow /deny such an authentication request.

4. Subsequently the M2 machine can attempt to re-verify the session and re-initiate authentication against the owner of the wallet as described in approach #1

Advantages of both the approaches:

1. No custom ssh code is required
2. No need of passing session specific variables via SSH / xargs
3. No need of additional AuthNull control plane to Endpoints using TCP
4. No need of re-verification of machine identity (as machine identity is tied to the user

Use case #4: Root bypass

Step #1	User attempts to logon as root
Step #2	AuthNull Bypasses root authentication and enables logon with a root password (no Passwordless authentication required).