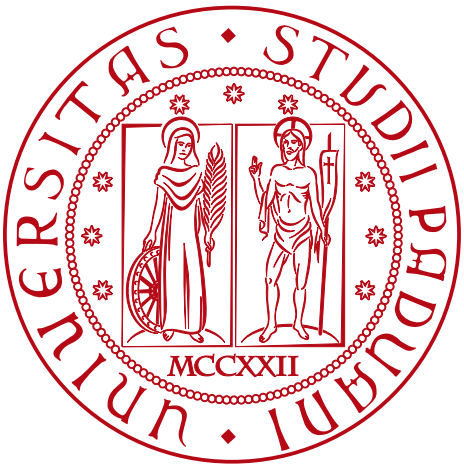
Università degli studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Self-sovereign Identity: Sviluppo di un sistema di login SSH tramite credenziali verificabili in Linux

Tesi di laurea

Relatore

Prof. Claudio Enrico Palazzi

Laureando

Marco Marchiante

Matricola 1222397

ANNO ACCADEMICO 2022-2023

Dedicato a tutti coloro che mi hanno sopportato e mi sono stati vicini durante questi anni di studio.

Una dedica speciale per la nonna Adriana che è venuta a mancare improvvisamente e due settimane prima della mia Laurea.

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di circa 300 ore, dal laureando Marco Marchiante presso l'azienda Athesys s.r.l..

L'obiettivo dello stage era l'esplorazione del tema della SSI[G] nel processo di autenticazione SSH[G]. In particolare era richiesta l'implementazione di un PoC[G] che dimostrasse un sistema di login SSH tramite VC[G] su GNU Linux.

L'attività di stage prevedeva la comprensione del funzionamento interno del processo di autenticazione del protocollo SSH[G] e dell'interazione dell'ultiimo con i moduli di autenticazioe del sistema Linux NSS[G] e PAM[G].

Ulteriore obiettivo dello stage consisteva nell'implementare una feature chiave del futuro prodotto, cioè il provisioning[G] di utenti non registrati tramite dati forniti nelle credenziali utilizzate per effettuare l'accesso.

Ringraziamenti

nnanzitutto, vorrei esprimere la mia gratitudine al Prof. Claudio Enrico Palazzi relatore della mia tesi, per l'aiuto e il sostegno fornitomi durante la stesura del lavoro."

Inoltre, desidero ringraziare l'azienda Athesys s.r.l. per avermi ospitato e offerto questa opportunità, il tutor aziendale Dr. Mattia Zago per avermi seguito durante questa esperienza e tutti i colleghi per avermi aiutato lungo il percorso.

Desidero ringraziare con affetto la mia famiglia per il sostegno, il grande aiuto e per essermi stati vicini in ogni momento durante gli anni di studio.
In particolare ringrazio i miei genitori per aver creduto in me durante questo periodo travagliato.

Desidero poi ringraziare i miei amici, in particolare Marco, per tutti i bellissimi anni passati insieme e le mille avventure vissute e i miei colleghi per essere stati sempre di supporto.

Infine vorrei ringraziare e mille persone conosciute durante il percorso di studi dalle quali ho imparato molto e spero di essere riuscito a ricambiare almeno in parte.

Grazie a tutti di cuore per avermi sopportato e avermi accompagnato in questo percorso.

Padova, Dicembre 2023 *Marco Marchiante*

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

In questo capitolo viene introdotto il contesto operativo, quindi l'azienda, processi e strumenti utilizzati, e l'organizzazione del presente testo

1.1 L'azienda

Figura 1: Logo di Athesys s.r.l.

Athesys s.r.l è un system integrator che si occupa principalmente di soluzioni di IAM[G].

L'azienda nasce nel 2010 a Padova dalla sinergia di affermati professionisti del settore IT, con lo scopo di offrire consulenza altamente specializzata in ambito System Integration, Database Management, Sicurezza applicativa, Governance Cloud Platform, Hyperconvergenza e Sviluppo Software in modalità Agile[G]. [5]

Figura 2: Logo di Monokee s.r.l.

Monokee s.r.l. [6] una *startup* fondata nel 2017 e si occupa di soluzioni per l'IAM.

Il loro prodotto principale è Monokee SSO[G], una piattaforma SSO[G] sviluppata in collaborazione con Athesys. Supporta multiple tecnologie di autenticazione, tra cui la SSI[G]. [7]

1.2 Modello di sviluppo

Il modello di sviluppo adottato dall'azienda segue il modello Agile[G] consentendo forti benefici di efficienza.

Il modello descritto in seguito è simile al popolare modello Scrum[G] ma non lo ricalca completamente. Infatti i due termini sono spesso confusi

- Il progetto viene diviso in periodi con lunghezza di una settimana per lasciare libertà di manovra ai membri del team ma non abbastanza per andare fuori strada.
- Tra la fine di ogni periodo e l'inizio del successivo il team effettua una riunione dove effettua diverse attività riguardo il periodo precedente e quello successivo.
- Vengono condivisi i progressi e/o frustrazioni in modo che tutto il team sia al corrente dello stato di sviluppo del progetto.
- Vengono condivisi i dubbi e/o punti di blocco in modo da potersi aiutare a vicenda.
- Viene fatta della retrospettiva in modo che di possa imparare dagli errori e puntare sui punti di forza.
- In base alla retrospettiva e lo stato di avanzamento e del progetto vengono decise e assegnate le attività da effettuare nel prossimo periodo.

1.2.1 Strumenti utilizzati

Vengono utilizzati diversi strumenti tecnologici per supportare tale modello di sviluppo, utile a migliorare l'efficienza della comunicazione. Spesso questi strumenti offrono funzionalità aggiuntive che vanno da semplici comfort fino a essere degli strumenti importanti nella vita di tutti i giorni offrendo alti livelli di flessibilità.

Ad esempio nel caso di Athesys, gli strumenti scelti per supportare il modello di sviluppo consentono a ogni membro del team di lavorare in *smart working* senza perdere acceso a strumenti importanti, comunicazione con gli altri membri del team o efficienza.

La successiva lista comprende strumenti che ho avuto l'occasione di usare in prima persona durante l'esperienza di stage:

Miscrosoft Teams Software pensato per la comunicazione collaborazione aziendale sviluppato da Microsoft. Permette il facile scambio di contenuti e messaggi in forma di chat testuale o video chiamata e mostra lo stato in tempo reale di ogni utente (i membri del team in questo caso) facilitando le comunicazioni interne, specialmente asincrone, e rendendo possibile una esperienza di lavoro remoto con i minimi compromessi.

Google Calendar Calendario digitale sviluppato da Google facente parte della suite di strumenti G-Suite. Consente di creare e riempire diversi calendari che possono essere condivisi in maniera selettiva consentendo a tutti o selezionati membri del team di controllare le disponibilità dei propri pari per chiedere un consulto o programmare meeting, riunioni o eventi.

Google Mail Software di posta elettronica sviluppato da Google e facente parte della suite di strumenti G-Suite. Consente di memorizzare, inviare e ricevere e-mail in un contesto aziendale. È utilizzabile come servizio di posta interna per comunicazioni importanti e/o ufficiali all'interno del team o dell'organizzazione e per intrattenere conversazioni ufficiali con membri non appartenenti all'organizzazione ma in modo da far riconoscere l'appartenenza a quest'ultima.

VPN^G L'azienda utilizza un servizio di VPN per permettere a ogni membro del team di connettersi alla rete interna dell'ufficio da qualunque posizione, a patto di avere connessione Internet. Questo strumento è utile per accedere alle e risorse aziendali in diversi contesti come ad esempio lo *smart working* o la dimostrazione di una demo complessa a un cliente fuori sede.

VPS^G Per agevolare lo sviluppo vengono mesi a disposizione dei server virtuali utili per simulare facilmente interazioni di rete o delegare operazioni di computazione intense allo scopo di migliorare il tempo di esecuzione o avere un ambiente standard di riferimento.

1.3 Introduzione al problema

In un mondo dove ogni informazione è a un passo dall'essere di dominio pubblico, il concetto di privacy ha iniziato ad assumere tutto un altro significato. Oggi giorno è un aspetto fondamentale della vita di ognuno e tutti ne hanno diritto in ogni aspetto della quotidianità.

Con l'ascesa dei social e la digitalizzazione di pratiche, pubbliche amministrazioni e in generale i servizi, sta tuttavia diventando sempre più difficile da un punto di vista tecnico garantire questo fondamentale diritto. Le informazioni sono sempre a un click dall'essere di dominio pubblico e garantire che ciò non succeda è un problema sempre più stringente a cui sia l'opinione pubblica che le grandi compagnie stanno iniziando a sensibilizzarsi.

Molte tecnologie concorrono a rendere ciò possibile e ciò che è più evidente è anche il pezzo più importante di tutti: l'IAM^G, cioè un software che gestisce l'accesso a una risorsa in base all'identità di chi vi vuole accedere. Un classico esempio è ogni *form* di login, dove la richiesta di nome utente e password è un metodo di conferma dell'identità che una volta confermata garantisce l'accesso a una o più risorse.

Il modo in cui questi sistemi vengono gestiti oggi ha molti problemi che stanno cominciando a diventare evidenti, spaziando dalla sconvenienza di avere più set di credenziali al problema di sicurezza legato alla molteplicità o alle possibilità di monopolio di chi controlla le identità digitali.

Il paradigma della SSI^G si propone di risolvere questo problema restituendo agli individui il controllo delle loro identità digitali che è finora è stato gestito da terzi senza che fosse ovvio.

1.4 Organizzazione del testo

Di seguito viene dettagliata la divisione del testo in capitoli e vengono dichiarate le convenzioni tipografiche che sono state adottate nella stesura del presente documento.

1.4.1 Struttura del testo

Il testo è stato articolato nei seguenti capitoli:

Il primo capitolo Questo capitolo introduttivo descrive l'azienda, il suo modo di lavorare e gli strumenti utilizzati a tale scopo nel corso del progetto.

Secondo capitolo Descrive lo stage facendo riferimento al piano di lavoro^G.

Terzo capitolo Fornisce una descrizione di massima delle tecnologie impiegate.

Quarto capitolo Descrive il i processi di ricerca, progettazione e implementazione del lavoro effettuato durante lo stage indicando le sfide affrontate.

Quinto capitolo Descrive la fase di test.

Sesto capitolo Effettua delle considerazioni sui possibili sviluppi derivanti dallo studio presentato e possibili miglioramenti individuati durante lo stage.

Settimo capitolo Conclude il documento riflettendo sull'esperienza di stage lungo l'aspetto dell'aderenza alla pianificazione e dell'arricchimento personale derivato.

1.4.2 Convenzioni tipografiche

Riguardo la stesura del testo, relativamente al documento sono state adottate le seguenti convenzioni tipografiche:

- Gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
- Per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: parola^{G} oppure parola (*breve descrizione*)^{G}
- I termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.
- Le citazioni sono evidenziate dalla sintassi [numero] in prossimità della stessa citazione, dove il numero indica la corrispondente fonte riportata nella bibliografia che si trova a fine documento;

Capitolo 2

Descrizione dello stage

Questo capitolo presenta ed espande sulla proposta di stage concordata con l'azienda, come da piano di lavoro.

2.1 L'idea

Lo scopo dello stage è implementare un meccanismo di autenticazione per SSH[G] che utilizzi le VC[G] al posto delle credenziali utente, consentendo inoltre il provisioning[G] JIT (*just-in-time*)[G] dell'account in caso di assenza. Nello specifico, lo studente andrà a lavorare sul sistema Linux e a realizzare due moduli personalizzati per NSS[G], e PAM[G].

Il candidato lavorerà in principio esplorando il protocollo SSH[G] (principalmente la fase di autenticazione) e i moduli di autenticazione di *Linux* alla ricerca di possibili limitazioni tecniche per suggerire una possibile architettura e successivamente si cimenterà nella realizzazione di un sistema dimostrativo con tecnologie adatte all'architettura scelta e da concordare con il team.

L'obiettivo finale è la realizzazione di un PoC[G] dimostrativo abbinato a una documentazione tecnica che dettagli l'architettura realizzata e il processo decisionale.

2.2 Requisiti

I requisiti concordati con l'azienda, e quindi definiti nel piano di lavoro[G], sono molteplici e divisi in 3 categorie ordinate per importanza/priorità:

[O]bbligatori costituiscono l'obiettivo primario e sono perciò requisiti vincolanti. Devono essere quindi soddisfatti perché il progetto sia considerato come completato.

[D]esiderati non sono requisiti vincolanti o necessari ma costituiscono un considerabile valore aggiunto e contribuiscono alla completezza del prodotto finale.

[F]acoltativi rappresentano un valore aggiunto con bassa priorità e sono i primi a essere tagliati qualora le risorse siano insufficienti.

I requisiti sono contraddistinti da un codice univoco nel formato

R[TIPO][NUMERO]

Codice	Descrizione
RO01	Progettazione e sviluppo di un modulo NSS^G personalizzato per l'autenticazione basata su VC^G.
RO02	Progettazione e sviluppo di un modulo PAM^G personalizzato per l'integrazione dell'autenticazione SSH con le credenziali verificabili e il provisioning^G JIT (*just-in-time*)^G degli account.
RO03	Realizzazione di una PoC^G funzionante che dimostri l'autenticazione SSH^G basata su credenziali verificabili e il provisioning^G JIT (*just-in-time*)^G degli account.
RD01	Gestione delle politiche di accesso e delle autorizzazioni tramite PAM^G.
OD02	Integrazione con un sistema di gestione delle identità e degli accessi esistente.
RF01	Documentazione di integrazione per consentire ad altri sviluppatori di utilizzare il meccanismo di autenticazione SSH^G basato su VC^G.
RF02	Implementazione di meccanismi di revoca delle VC^G.

Tabella 1: Requisiti di progetto concordati con l'azienda

2.3 Pianificazione

Il periodo di svolgimento dello stage è stato dal 10/07/2023 al 22/09/2023 per un totale di 10 settimane. Oltre alle 8 settimane richieste per il completamento delle 300 ore, sono state incluse infatti due settimane aggiuntive utili a coprire eventuali imprevisti, come gli esami residui, oltre alla chiusura per ferie dell'azienda.

Lo stage è stato pianificato su 304 ore e diviso in 3 fasi consecutive: Formazione, Implementazione e Documentazione.

Sono stati previsti inoltre dei *deliverable* per le prime due fasi da presentare il martedì successivo al completamento delle stesse.

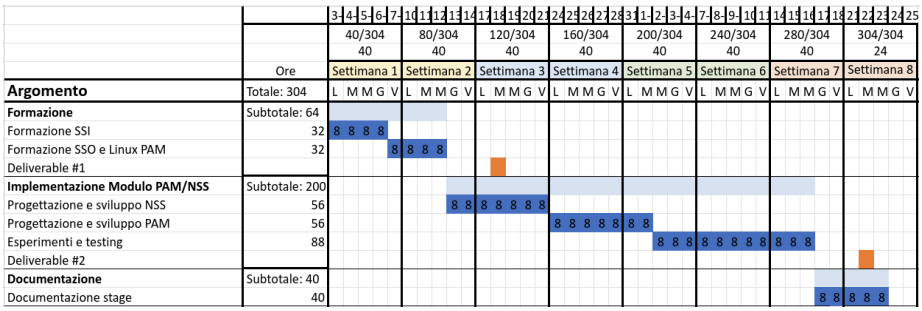


Figura 3: Pianificazione dello stage su 304 ore

2.3.1 Formazione

Questa fase costituisce il fondamento dello stage ed ha lo scopo di fornire solidi fondamenti teorici per familiarizzare con il dominio applicativo e i concetti teorici trattati nel corso dello stage. Inoltre ha il fondamentale compito di dare al candidato il vocabolario adeguato per interagire con il team.

Ha una durata prevista di due settimane e consiste nei seguenti punti:

- formazione su SSO (*Single Sign-on*)^G, IAM (*Identity and Access Management*)^G e SSI (*Self-Sovereign Identity*)^G. Comprende una panoramica teorica e l'approfondimento delle applicazioni pratiche del paradigma SSI^G.
- studio approfondito di unix PAM (*Pluggable Authentication Modules*)^G durante la formazione su SSI^G, SSO^G e IAM^G.

Questa fase è generalmente guidata dal tutor aziendale ma prevede sia formazione da parte del team su argomenti specifici che auto formazione da parte del candidato.

Deliverable

Documentazione interna aziendale che riassume i concetti e le *best practice* relative a SSO, IAM, SSI e UNIX PAM.

2.3.2 Implementazione ed Esperimenti

Questa fase è il fulcro dell'intero stage avendo durata stimata di 5 settimane e prevede i seguenti punti:

- Progettazione e sviluppo del modulo NSS personalizzato per l'autenticazione basata su credenziali verificabili.
- Progettazione e sviluppo del modulo PAM personalizzato per l'integrazione dell'autenticazione SSH con le credenziali verificabili e il provisioning^G JIT (*just-in-time*)^G degli account.
- Esperimenti e testing per valutare l'efficacia del meccanismo di autenticazione SSH basato su credenziali verificabili e il provisioning^G JIT (*just-in-time*)^G degli account.

Una volta comprese le tecnologie in gioco, lo scopo di questa fase è quello affrontare la nuance del progetto

Deliverable

POC funzionante che dimostra l'autenticazione SSH basata su credenziali verificabili e il provisioning^G JIT (*just-in-time*)^G degli account. Report completo delle attività di sviluppo, inclusi dettagli tecnici, decisioni di progettazione e risultati ottenuti.

2.4 Documentazione

Questa fase costituisce la conclusione del progetto con una durata prevista di una settimana ed è interamente focalizzata su di un solo punto:

- Stesura della documentazione tecnica dettagliata che descrive l'implementazione del meccanismo di autenticazione SSH basato su credenziali verificabili e il provisioning[G] JIT (*just-in-time*)[G] degli account.

Deliverable

Documentazione tecnica completa.

2.5 Divisione settimanale delle attività

Segue la divisione settimanale delle attività indicando per ciascuna la quantità di ore allocate e il periodo indicativo di svolgimento

Settimana	Attività	Ore
1 - 2	Formazione	56
2 - 3	Progettazione e sviluppo NSS	32
4 - 5	Progettazione e sviluppo PAM	56
5 - 8	Esperimenti e testing	88
8	Documentazione stage	16
	Totale	304

Tabella 2: Scansione settimanale delle attività

Capitolo 3

Descrizione delle tecnologie

In questo capitolo sono elencate e dettagliate le tecnologie e paradigmi tecnologici utilizzati nell'arco del progetto.

3.1 Rust

Rust è un linguaggio di programmazione a basso livello moderno *open-source* supportato dalla *Rust Foundation* dal 2020 (precedentemente da *Mozilla Foundation* dal 2015). Pone enfasi particolare sulla correttezza del codice e la performance. Come altri linguaggi a basso livello, è fortemente tipizzato, compilato e garantisce controllo a basso livello della memoria.

Offre una varietà di feature innovative ma alcune fra le principali lo distinguono nella sua categoria:

- **Derivazione funzionale**: *Rust* è un linguaggio procedurale ma incorpora alcune caratteristiche dai linguaggi funzionali come lo stato immutabile, aiutando la correctness.
- **Sitnassi potente e moderna**: Essendo un linguaggio moderno offre una sinstassi molto potente grazie anche all'integrazione del paradigma funzionale
- ***Borrow checker***: Un nuovo modo di pensare alla concorrenza basandosi sul concetto di prestito e proprietà dei dati. In questo modo diminuiscono significativamente le *race condition* e i *memory leak*.
- **Compilazione incrementale**: Compila il codice incrementalmente *out of the box* senza bisogno di *build tools* complessi come CMake[G] o Ninja[G].

È utilizzato tipicamente per system level programming[G] ma grazie alla sua prestazione e sintassi moderna è molto apprezzato per scrivere WASM[G]

È costantemente il linguaggio più amato dagli sviluppatori [8] e ciò porta molti vantaggi. La documentazione è abbondante e ben fatta spaziando da documentazine tecnica a varie guide *domain-specific*, molte delle quali contribuite dalla community estremamente attiva e amichevole.

Uno dei suoi punti forti è che essendo un linguaggio moderno comprende molte feature pensaate per migliorare la *developer experience*. Ad esempio comprende i tool rustup per getire completamente l'intera *toolchain*, rustfmt per una formattazione uniforme e standardizzata del codice, rust-analyzer per integrare un LSP[G]. Inoltre include un supporto integrato per test e documentazione e possiede un suo *package manager* cargo ed una *repository* online chiamata crates.io.

3.2 Apache Guacamole

Apache Guacamole[G] è un *gateway desktop* remoto (o sulizione per desktop remoto) sviluppato inizialmente nel 2010 e gestito tutt'ora da *Apache Software Foudation*. Il progetto è completamente gratuito ed open-source e la sua principale caratteristica è il funzionamento *clientless*[G]. Infatti una volta installato su di un server, utilizza HTML5 per creare un web-client *on-the-fly* sulla finestra del broswer senza la necessità di un più classico client discreto installato sulla macchina.

Il prodotto è molto flessibile grazie alla varietà di API[G] offerte e la estensiva documentazine che comprende manuali e tutorial oltre la documentazione tecnica. È offerto il supporto per multipli protocolli tra cui SSH[G], VNC[G]e RDP[G].

Oltretutto supporta l'utilizzo commerciale sia tramite la *Apache License* che tramite compagnie di terze parti che supportano e promuovono l'utilizzo del software. [9]

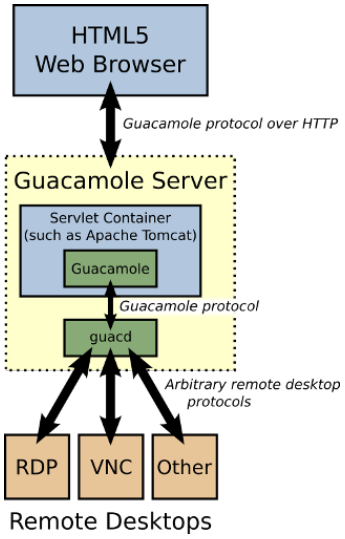


Figura 4: [1] Architettura di Apache Guacamole[G]

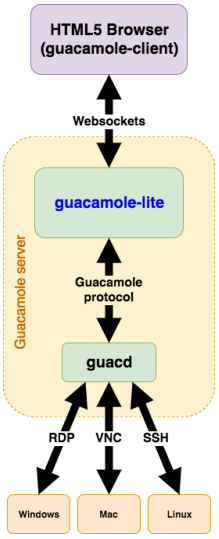


Figura 5: [2] Architettura di Apache Guacamole[G] modificata con *guacamole-lite*

L'architettura di Apache Guacamole[G] è composta da 3 componenti: un **client web**, un **daemon**[G] e un *"frontend"* che agisce da collegamento tra i due.

Web browser è considerato il *client* dell'applicazione e ha il compito di raccogliere l'input, speditro al *frontend* tramite *tunneling* ed elaborare la risposta che ne consegue.

Frontend è un api backend che accetta le connessioni dal *web browser* e le inoltra al daemon guacd.

Daemon è il vero e proprio Apache Guacamole[G].

Comunica con il *frontend* per ricevere l'input dal *client* e lo elabora per interfacciarsi con uno tra i protocolli supportati al client deskto premoto-

3.2.1 guacamole-lite

Guacamole-lite è un "*frontend*" per guacd che va a sostituire quello di default di Apache Guacamole[G]. iI suoi vantaggi sono limitati all'utilizzo di *websocket* per la comunicazioné e l'utilizzo di *Node.js* anzichè *Java* come linguaggio di programmazione

3.3 Self-sovereign Identity (SSI)

La SSI (*Self-Sovereign Identity*)[G] è un nuovo paradigma nella gestione dell'identità digitale e si propne come la soluzione definitiva al problema delel moderne souzioni di SSO e autenticazoine i ngenerale con prevenzione delle frodi.

3.3.1 Background

Al giorno d'oggi per la gestione dell'identità digitale vengono urilizzati due diversi modelli, spesso offerti parallelamente: **centralizzato** o "a silos" e **federato** "third party".

Il modello **centralizzato** detto anche "a silos" è il classico modello di "sign-up/sign-in" ritrovabile tutt'ora nella maggioranza dei siti Web. Prevede che un individuo/entità si iscriva ad ogni servizio che adotta questo modello e gli venga assegnata per ciascuno una nuova identità.

Il principale vantaggio di qeusto modello è la semplicità di implementazione, che l'esatto motivo della sua diffusione. Infatti l'identità esiste solo localmente al contesto applicativo rendendo necessario l'immagazzinamento dei soli dati richiesti senza ulteriore overhead.

I vantaggi finiscono però qui poichè questa pratica genera due problematiche molto serie causate dall'esistenza delle identità in multiple forme allo stesso tempo:

multipli *Point of Failure* esistendo le identità in più luoghi contemporanei, i point of failure crescono con la moltiplicita dei luoghi. Questa molteplicità espone i dati ad attacchi hacker. Consideriamo in particolare che secondo la statistica molti servizi non utiilzzano livelli di sicurezza adeguati e ogni individuo nel corso della sua vita si iscrive a centinaia di servizi che memorizzeranno permanentemente i dati.

fenomeno delle *multiple identities* esistendo in più forme è impossibile assicurare consistenza tra le diverse identità portando al fenomeno delle *multiple identities*. Ciò può generare diversi problemi spaziando da disagi minori a problemi seri come nella fusione di archivi con dati inconsistenti.

Il modello **federato** detto anche "third party" prevede che un provider si faccia carico della gestione dell'dentità e altri servizi si appoggino al provider per fruire dei servizi di identità.

Questo modello è sempre più comune ed offerto in molti siti come alternativ al modello centralizzato. Ad esempio basta pensare agli SSO di Google, Facebook o Apple. In italia anche lo SPID ricade sotto questa categoria.

I suoi vantaggi sono chiari rispetto al modello centralizzato poichè i dati esistono in un solo luogo risolvendo entrambi i problemi sopra citati.

Ora però viene introdotto il problema del *monopolio*. Infatti nulla impedirebbe agli Identity Provider di alterare i dati a piacimento. In questo modo gli si conferisce l'autorità pratica di controllare l'identità degli utenti e di spiarne l'attività del momento che il flusso d'autenticazine passa forzatamente per la loro infrastruttura.

I due modelli in uso si basano dulla centralizzazione delle informazioni ma la SSI usa un approccio diverso concentrandosi sulla decentralizzazione dei dati tramite la blockchain e un sistema di chiavi pubbliche e private.

Lo standard, pur se in stato di draft, è molto ricco ma il funzionamento può essere riassunto brevemente.

3.3.2 Terminologia e workflow

Essenzialmente la blockchain viene utilizzata cme un ***disctibuted ledger*** per immagazzinare le identità degli utenti in maniera anonima. Nello specifico il ledger memorizza un identità astratta contenente la chiave pubblica del soggetto a cui si riferisce e altri dati necessari per provarne la correlazione. Le vere informazioni sono in possesso dell'utente che tramite la sua chiave privata può correlarle alla cua "identità anonima" nella blockchain che è collegata lui stesso poichè in possesso della chiave privata corretta.

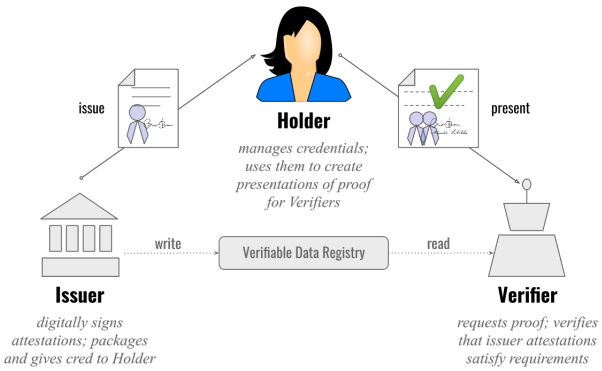


Figura 6: [3] Overview del funzionamento di SSI (*Self-Sovereign Identity*)^G

Nella SSI ci sono 3 entità:

issuer^G Entità che asserisce delle affermazioni riguardo ad un soggetto e genera una credenziale che le contiene.

holder^G Entità che possiede delle credenziali che spesso (ma non necessariamente) lo riguardano

verifier^G Entità che riceve una credenziale o presentazione verificabile e la verifica.

Il pezzo principale dell'intero sistema è il DID^G che si potrebbe descrivere come una specie di "identità astratta" ma in realtà è un identificatore univoco. Come accenato precedentemente è salvata in modo indelebile sulla *blockchain*

sotto forma di DID Document^G e si riferisce a un particolare soggetto chiamato DID Subject^G.

Questo DID Document^G contiene quindi informazioni che permettano di identificare unovicamente il correlato DID Subject^G senza rivelarne l'identità. In particolare è contenuto il DID^G del DID Subject^G, la sua chiave pubblica e le informazioni riguardo il metodo utilizzabile per il processo di verifica.

In realtà un DID Document^G può contenere molti altri campi come ad esempio il campo DID Controller^G che contiene il DID^G di un soggetto a cui sono permesse modifiche del DID Document^G.

È chiaro quindi che a un DID^G è collegato un DID Document^G che lo descrive. Per ricavare un DID Document^G è necessario risolverlo a partire da un DID^G in un processo chiamato DID Resolution^G.

Come accennato precedentemente pur rappresentaando un soggetto, un DID^G non vi è collegato esplicitamente in alcun modo. L'unico modo di efettuare quindi questo collegamento è di utilizzare la propria chiave privata in un processo chiamato DID Auth^G. Questo processo ha lo scopo di provare l'appartenenza di un did ad un soggetto mostrando ci poterlo controllare ed è tipicamente protratto da un verifier^G. Si parte da un DID^G che viene usato tramite DID Resolution^G per risolvere il relativo DID Document^G da cui viene estratta la chiave pubblica. Il verifier^G a questo punto genera una *challenge* con la chiave pubblica appena recuperata, ch epotrè essere risolta unicamente con la chiave privata associata.. Se il soggetto è in grado di risolverla correttamente utilizzando la propria chiave privata, significa che è in grado di apportare modifiche e quindi controllare il DID Document^G ed è quindi correlato al DID^G di partenza.

Facendo quindi riferimento a Figura 6, un tipico workflow è come segue:

Se un issuer^G deve effettuare delle asserzioni chiamate claim^G riguardo ad un soggetto, emetterà un certificato contenente le stesse claim^G ed il DID^G che ne descrive l'oggetto, oltre che al proprio DID^G. Il certificato verrà poi firmato con la propria chiave privata dall'issuer^G generando una proof^G in modo che successivamente sia possibile verificare l'identità dell'issuer risolvendone il DID^G. Una volta firmato tale certificato è chiamatp VC^G.

A questo punto la VC^G è in possesso di un holder^G che può non coincidere con il subject. Più avanti sarà evidente perchè ciò non costituisce un problema. Un holder^G, usando la propria chiave privata, può utilizzare le credenziali in suo possesso presso un verifier^G. Può inoltre decidere di farlo singolarmente presentando una sola VC^G o combinandone una quantità superiore in una VP^G.

Una VP^G è una collezione di una o più VC^G ed à sempre creata da un holder^G. In questo caso l'holder^G si comporta effettivamente da issuer^G e iserire il proprio DID^G e una proof^G genrata con la sua chiave privata colelgata al DID^G. È importante notare vhe una VP^G dovrebbe avere una vita il più breve possibile

Quando un verifier^G necessita di verificare una VC^G o VP^G deve effettuare due operazioni in ordine non rilevante:

1. Verificare che il DID Subject^G indicato nella VC^G o VP^G sia controllato dall'holder^G tramite il processo di DID Auth^G.

2. Verificare che le claim^G siano effettifamente state rilasciate dall'issuer^G. In caso di VP^G qeusta verifica viene ripstuta più volte, una per ogni VC^G contenuta.

Solo quando entrambe le verifiche sono completate con successo l'verifier^G può fidarsi dell'autenticità delle claim^G e dell'identità dell'holder^G abbastanza da poter prendere decisioni basandosi su tali dati.

Quest'ultima verifica consiste nel controllo delle/a proof^G. Infatti nel caso di una VP^G potrebbero essere contenute più VC^G oltre alla proof^G che firma la presentazine stessa. Per l'effettiva verifica il verifier^G recupera il DID^G dell'issuer^G e tramite il processo di DID Resolution^G il DID Document^G associato e di coneguenza la chiave pubblica con la quale è possibile verificare la proof^G utilizzata per la firma.

Vale la pena notare che il sistema è molto resistente grazie alla separazione fisica a logica tra DID^G e claim^G. Infatti anche perdendo le VC^G queste sono inutili senza la chiave privata del soggetto poichè anche verificando le stesse, non ci sarebbe modo per correlarle ad un sogggetto.

Anche nel malaugurato evento della perdita della chiave privata, esite la possibilità di emettere credenciali revocabili.

Un esempio significativo del modo in cui questa tecnologia potrebbe essere rilevante come futuro dell'identità digitale, è la sostituzione degli stessi documenti di identità (probabilmente il documento più importante tra tutti) con una VC^G.

In questo esercizio mentale l'anagrafe dello stato costituisce l'issuer^G, il cittadino costituisce il'holder^G e un tipico esempio di verifier^G potrebbero essere le forze dell'ordine ad un posto di blocco o un barista che vuole conoscere l'età di un sospetto minorenne. I dati di identità sarebbero contenuti nella claim^G che sarebbe firmata con il DID Document^G dell'anagrafe dello stato.

L'unico requisito è che sia holder^G e verifier^G abbiano uno smartphone o qualunque dispositivo abilitato. Paradossalmente infatti con una adeguata soluzione di *caching* dei DID Document^G non c'è nemmeno bisogno di una connessione internet per la DID Resolution^G

3.3.3 Possibilità aggiuntive

Zero klnowlege proof Metodi crittografici che permettono ad un holder^G di compilare una VP^G con i soli frammenti ci claim^G necessari provenienti da una VP^G che supporti la tecnologia. Ad esempio in una VC^G emessa dall'anagraafe civile potrebbe essere mostrato solo il nome tenendo altre informazini private come cognome data di nascita o residenza.

Quanto mostrato precedentemente riguardo la SSI^G è una mera panoramica tecnologica e la documenazione tecnica del *W3C* (ancora in fase di Draft) per il DID^G [10] e la VC^G [11] è molto più ampia e dettagliata riguardo alle infinite possibilità offerte.

3.4 UNIX PAM

Il modulo PAM[G] fornisce un'interfaccia per l'autenticazione delle credenziali degli utenti e per l'autorizzazione degli accessi alle risorse del sistema; permette di configurare politiche di autenticazione e autorizzazione personalizzate, consentendo l'uso di diversi metodi di autenticazione, come password, *certificati digitali*, *token* e, in questo caso, VC[G].

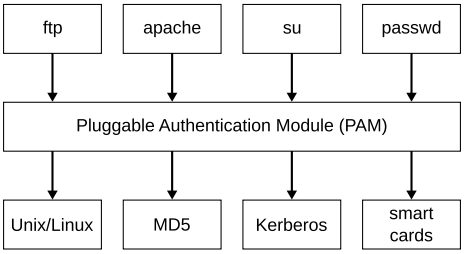


Figura 7: Funzionamento di PAM[G]

Il suo funzionamento generale è semplice. Come mostrato difatti in , questo si comporta come uno *switch* programmabile che a seconda dell'applicazione che richiede un processo di autenticazione, lo dirotta a un modulo appropriamene configurato che se ne prenda carico. Altro aspetto importante è che più moduli possono essere configurati per ogni applicazione e può essere configurato come il risultato di ogni modulo influenzi il processo formando così una catena di autenticazione personalizzata.

3.5 UNIX NSS

Il modulo NSS[G] è responsabile della risoluzione delle informazioni relative agli utenti, gruppi e altre entità di sistema all'interno del sistema operativo *UNIX*; permette di integrare nuovi servizi di autenticazione e autorizzazione nel sistema, consentendo l'accesso a tali servizi da parte delle applicazioni del sistema.

Similarmente a PAM, NSS agisce come uno *switch* per la risoluzione di una determinata risorsa tramite nome, utilizzando diversi *resolver*/tecnologie. I tipi di risorse risolvibili sono molteplici e la catena dei *resolver* è configurabile i maniera simile ma meno potente di PAM.

In particolare, la differenza consiste nell'impossibilità di creare configurazioni "per applicazione" e l'impossibilità di influenzare la catena di *resolver* con l'esito delle operazioni, infatti, un fallimento comporta lo scorrimento della catena mentre un successo causa un ritorno al chiamante. Di fatto la configurazione è quindi *system-wide*

3.6 SSH

SSH (*Secure SHell*)[G] è un protocollo che permette di stabilire una sessione remota cifrata e sicura alla famigerata porta 22 tra due host connessi in rete ed è il successore storico del meno sicuro e sofisticato *Telnet*.

È utilizzato principalmente come utility a linea di comando per la connessione a un host remoto ed è di fatto lo standard nel mondo della *system administration*.

Le sue due caratteristiche più significative sono la sicurezza e la flessibilità. Infatti supporta l'autenticazione tramite nome utente password (*basic auth*) e l'autenticazione tramite *SSH keys* che sono chiavi asimmetriche

La sua *feature* principale costituisce anche il suo secondo punto forte ed è il *tunnel SSH*, cioè la possibilità di incapsulare pacchetti di rete in pacchetti SSH, inviarli tramite il canale sicuro di comunicazione e successivamente dirottarli localmente ad una porta di rete. Questa possibilità è la caratteristica distintiva di questo protocollo poiché permette la creazione di un canale sicuro *multi-purpose* e affidabile.

Due esempi popolari nella *system administration* sono il *tunneling* del traffico *http(S)* per effettuare manutenzione senza aprire porte di rete o il tool *rsync* che permette di archiviare, sincronizzare e trasferire file in maniera sicura e veloce grazie al *tunnel SSH*

3.7 HMAC

HMAC^G è un codice crittografico utilizzato per autenticare e verificare l'integrità di un messaggio tramite l'utilizzo di una *funzione di hash* e una chiave segreta condivisa.

Il principale vantaggio rispetto ad altri tipi di firma è che HMAC^G abbatte la complessità derivante dall'uso di *crittografia simmetrica* e supporta una varietà di *funzioni di hash*.

Dati una *funzione di hash* e una chiave segreta il funzionamento è abbastanza semplice consistendo di una derivazione di chiave e due iterazioni della *funzione di hash*.

Dalla chiave segreta vengono derivate due chiavi denominate **inner key** e **outer key**. Dopo di ché viene effettuata una prima iterazione utilizzando la *funzione di hash* sull *plaintext* usando la *inner key* e generando un **inner hash**. Successivamente viene effettuata una seconda iterazione con la funzione di hash sull'*inner hash* usando la *outer key* per produrre l'**outer hash**. Quest'ultimo è l' il valore di hash finale utilizzato come firma del *plaintext*.

Nonostante questo metodo sia particolarmente resistente agli attacchi, è bene ricordare che non è un metodi crittografico ma genera solamente una firma che viene accodata al *plaintext*. Inoltre è necessario che la chiave venga condivisa in anticipo utilizzando un canale sicuro poiché si tratta di crittografia simmetrica.

Sapendo che la chiave e la *funzione di hash* sono concordati in anticipo, per verificare una firma HMAC la procedura è la medesima. Infatti il processo consiste nella generazione locale della firma HMAC e il confronto con la firma ricevuta assieme al *payload*.

Capitolo 4

Progettazione e codifica

In qeusto capitolo viene dettagliato lo svolgimento dello stage divisio in 3 parti: ricerche preliminari, progettazione dell'architettura e sviluppo del PoC[G]

Prima di iniziare con la progettazione era indispensabile comprendere sufficientemente a fondo le tecnologie con cui avrei dovuto avere a che fare. In particolare la richiesta era di sviluppare un architettura per login SSH[G] tramite VC[G] com provisioning[G] JIT[G] di utenti autorizzati, in caso non siano registrati localmente.

Per comprendere sufficientemente a fondo le variabili in gioco ho speso molto tempo a fare ricerche approfondite durante la fase di formazione e di implementazione e di seguito riassumo i punti salienti di ciò che ho imparato.

4.1 Ricerche

Iniziamo trattando i problemi uno per volta, iniziando dall'obiettivo principale di realizzare un sistema di login SSH[G]

Delle brevi ricerche esulta evidente che su Linux, SSH[G] si appoggia a Linux PAM (*Pluggable Authentication Modules*)[G], il sistema principale di autenticazione utilizzato su Linux. Approfondendo le ricerche scopriamo che come il nome suggerisce, il sistema è modulare e ciò ci punta verso la giusta direzione poiché se ciò fosse vero potremmo scrivere un modulo personalizzato che utilizza le tecnologie del paradigma SSI[G].

Successive ricerche sul sistema PAM[G] confermano i sospetti. Per sua natura infatti il sistema è modulare, proprio allo scopo di facile espansione delle possibilità di autenticazione utilizzando diverse tecnologie.

4.1.1 PAM

Esplorandone la documentazione online è evidente che mette a disposizione due set di API[G], un primo per l'utilizzo all'interno di un applicazione, e un secondo utile a sviluppare moduli personalizzati. Ricordiamo inoltre il suo funzionamento descritto in Sezione 3.4.

Innanzitutto viene creata una specifica configurazione PAM[G] per ogni PAM Client[G]. La configurazione adeguata viene scelta in base al nome del file stesso in modo che coincida con il nome con cui il *client* di identifica.

Le chiamate che un PAM Client[G] può effettuare sono divise in categorie:

Authentication management Si occupa di assicurarsi dell'identità dell'utente. Riceve le credenziali (username e auth_token) ed effettua controlli sulla base di quale decidere il suo stato di autenticazione.

Account management Si occupa di effettuare altri controlli una volta conosciuta l'identità dell'utente. Una volta che l'*Authentication management* ha successo, altri controlli possono essere necessari prima di confermare all'applicazione lo stato di successo di autenticazione, ad esempio l'account potrebbe essere scaduto o il sistema potrebbe non avere abbastanza risorse per supportare l'attività di un altro utente. Una volta finiti i controlli non relativi all'identità del'utente, viene restituito un risultato all'applicazione.

Credentials Permette l'inserimento di nuove credenziali nel PAM Context^G in modo che siano a disposizione di tutti i moduli per la consultazione se necessario.

Session management Si occupa di gestire la sessione dell'utente appena autenticato. Viene chiamato alla creazione e distruzione della sessione stessa e ha compiti come la configurazione delle variabili d'ambiente, caricamento della *home directory* e provisioning^G.

Authtoken management Si occupa della gestione del token di autenticazione, tipicamente in caso di scadenza delle credenziali. Può essere chiamato dall'utente o dall'applicazione in diversi punti del flusso di autenticazione.

I moduli sono solitamente chiamati in ordine seguente: Authentication, Account, Session (Authtoken e Credentials possono essere chiamati più flessibilmente) ma è necessaria prima una chiamata speciale per inizializzare il PAM Context^G, che chiede il nome dell'applicazione e, se noto, il nome utente. Il risultato di questa chiamate è il cosiddetto PAM Context^G che contiene tutti i dati associati al processo di autenticazione e la eventuale sessione che verrà creata, unitamente a una(o potenzialmente di più) funzioni di conversazione.

La funzione di conversazione è la principale componente di PAM^G, in quanto punto di collegamento tramite applicazione *client* e modulo di autenticazione. Come suggerisce il nome, è una funzione che vine utilizzata dall'applicazione per consentire la comunicazione diretta tra con il modulo. Le chiamate a questa funzione sono tipicamente nascoste dietro all'api *client* ma è possibile utilizzarla direttamente quando più flessibilità è necessaria.

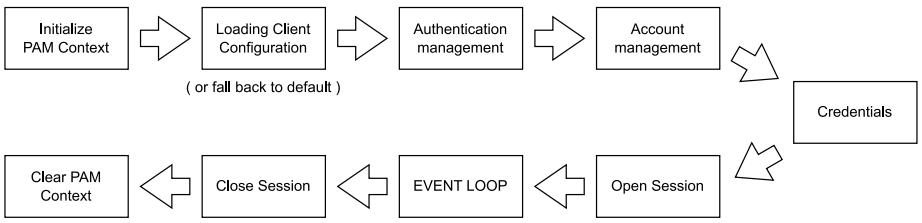


Figura 8: Tipico esempio di workflow PAM^G

PAM^G mette a disposizione una funzione di conversazione standard che fornisce un implementazione di base per i diversi tipi di interazioni che è possibile incontrare con il protocollo. È possibile però nei casi più complessi avere più flessibilità creando funzioni di conversazione personalizzate. In questo modo è possibile decidere, per ogni tipo di interazione, come PAM^G dovrebbe compor-

tarsi e quali dati dovrebbe fornire ai moduli. In pratica definisce in che modo PAM risponde alle richieste di dati proveniente dal modulo.

La seconda API[G], come spiegato prima, è utilizzata per creare moduli personalizzati. Esiste un set standard di funzioni utilizzate per comunicare con l'applicazione e richiedere/trasmettere dati, ma non sono altro che *helpers* per facilitare l'utilizzo della funzione di conversazione in una data maniera per le operazioni più comuni. In poche parole è possibile ottenere le informazioni utilizzando API[G] di più basso livello come la funzione di conversazione o l'interazione con il PAM Context[G].

Una volta caricata la configurazione, PAM[G] agisce come uno *switch* dirottando il flusso di autenticazione e autorizzazione tra più moduli garantendo un controllo molto fine dei quest'ultimo. È possibile indicare una catena di moduli da eseguire in un particolare ordine per ogni famiglia di operazioni (elencate precedentemente) e specificare condizioni per ogni combinazione modulo/categoria tramite "*flag*" e codici di ritorno per ogni categoria di modulo. Lo *switch* si occuperà successivamente, in base alle regole definite nel file di configurazione, di elaborare i "*flag*" restituiti dai vari moduli in risultati per ogni sezione da restituire all'applicazione.

4.1.2 Integrazione SSH di PAM e NSS

Una serie di prime prove suggerisce che l'implementazione di PAM[G] in SSH[G] sia differente da quella standard poiché sviluppato un modulo custom PAM[G], molto semplice par altro, il comportamento osservato nell'utilizzo in SSH[G] differisce significativamente da quello osservato in un applicazione di "debug" che utilizza PAM[G] nella maniera standard. La situazione si fa sempre più bizzarra più si scava a fondo, infatti, il comportamento di PAM[G] in SSH[G] risulta differente in diversi momenti del ciclo e addirittura inconsistente tra diverse esecuzioni.

L'analisi dei log SSH[G] non hanno portato a nessuna scoperta, infittendoli però, la situazione si è fatta interessante. Come mostrato in, tra diversi test effettuati in diverse situazioni "csuali" portano alla luce diciture diverse tra esecuzioni che, combinate con i comportamenti osservati, rivelano quelli che sembrano degli schemi. Pur essendo sulla buona strada, non siamo ancora in grado di fare delle affermazioni certe e l'unica opzione rimasta è osservare il codice sorgente di OpenSSH [12] tramite un mirror si GitHub.

Una volta compresa la struttura della *repository*, non è difficile notare la presenza di diverse funzioni di conversazione che vengono attivate in diverse circostanze; giustificando completamente le differenze in comportamento osservate.

Disattivare una delle opzioni di autenticazione (il tipo "mouse and keyboard") espone la funzione di conversazione universale lasciando libertà ai creatori dei moduli per la richiesta di dati e attivando una modalità interattiva. Altre funzioni di conversazione utilizzate sono la funzione nulla, una più permissiva e una che sopprime la comunicazione verso il client tranne in casi selezionati. Noi vogliamo quella più permissiva.

Con i dovuti aggiornamenti di configurazione, questo punto, il modulo funziona come atteso lasciando libertà si moduli di fare ciò che vogliono.

L'esplorazione ora continua verso la seconda *feature* richiesta: provisioning[G] di un utente autorizzato ma non registrato. il problema è che per costruzione, SSH[G] non è pensato per il login di un utente non registrato, restituendo perciò un errore se l'utente non è registrato nel sistema.

Tornati sul codice sorgente di OpenSSH, questa volta andiamo alla ricerca di una chiamata di sistema che restituisca le informazioni dell'utente . Una breve ricerca in rete suggerisce che il colpevole sia NSS[G]. Esso mette a disposizione diverse API[G] per chiedere al sistema informazioni riguardo agli utenti registrati. Filtrando le chiamate interessanti e cercandole nel codice sorgente risulta effettivamente una chiamata effettuata prime dell'inizializzazione del contesto PAM[G].

Seguendo il flusso dei dati nuovamente, vengono tracciati ed i risultati, confermato da test sul campo, rivelano che l'unica informazione effettivamente utile a questo stage, è l'identità (definita da GID[G] e UID[G]) dell'utente allo scopo di identificare il giusto livello di permessi per eseguire il processo di autenticazione PAM.

Successivi test confermano infatti che lo sviluppo di un modulo personalizzato appropriato può eludere il controllo e far proseguire il processo normalmente con rischi di sicurezza limitati.

4.1.3 NSS

Precedentemente ho glossato su NSS[G] che ci crea un problema molto importante perché l'utente non esiste e PAM[G] non può quindi eseguire con i permessi corretti, Per comprendere le possibilità per aggirare l'ostacolo è necessario comprenderne il funzionamento.

NSS (*Name Switch Service*)[G] ed è un servizio dei sistemi operativi GNU per la risoluzione delle risorse tramite identificativo. È organizzato, similmente a PAM, in moduli e categorie. NSS dichiara diversi tipi di informazione la cui risoluzione è supportata e per ogni tipo, sono disponibili più servizi e tecnologie diverse tra cui scegliere. Diversamente da PAM[G], la configurazione è *system-wide* e non consente un grado fino di controllo ma similarmente a PAM consente di dichiarare l'ordine delle operazioni, con l'eccezione che la lista viene scorsa iterativamente in *top to bottom order* e il primo servizio che ritorna un risultato ferma le iterazioni e verrà restituito. Ogni tipo di dato risolvibile ha una struttura definita e standardizzata e, nuovamente in maniera simile a PAM[G], è possibile creare moduli personalizzati per una specifica combinazione tipo-servizio.

Diversamente da PAM[G] non è consigliata l'implementazione di tutti i dati per un servizio personalizzato, ma è richiesta l'implementazione di tutti le API[G] offerte per ogni tipo che si vuole implementare.

Ciò ci dà la possibilità di creare un modulo NSS[G] personalizzato per alterare il *record* in modo da permettere la prosecuzione del processo PAM[G].

4.2 Progettazione

Il progetto può essere diviso in 2 parti differenti e indipendenti: l'**autenticazione** e il **provisioning** dell'account. Essendo i due concetti indipendenti si può procedere per ordine partendo arbitrariamente dall'autenticazione.

Analizzando le carte sul tavolo, conosciamo il metodo do login e il mezzo di login. Sappiamo che la credenziale è tipicamente utilizzata sotto forma di JWT^G cioè una stringa di caratteri *ASCII* con lunghezza variabile ma tipicamente compresa tra i 1500 e 2500 byte. Sappiamo però per costruzione di un JWT^G che esso è composto principalmente dal payload e da una firma e la sua dimensione è quindi destinata a scalare linearmente con la quantità di claim che contiene. In pratica bisogna tenere conto che la dimensione può aumentare all'aumentare dei dettagli richiesti per l'autenticazione ma non diminuire.

Il secondo dettaglio conosciuto è il metodo di login. Le credenziali devono in qualche modo raggiungere il modulo PAM sottostante l''unica maniera per riuscirci è attraverso il i campi "nume utente" e "password" richiesti da SSH.

Qui si pone la sfida più significativa dell'intero progetto in quanto l'idea più banale e intuitiva è di spedire la VC^G tramite uno dei due campi ma i test condotti suggeriscono in maniera inequivocabile che il limite di byte trasmissibili è di 1024 e risulta quindi impossibile spedire un intero *token*. Sapendo che la cosa vale per entrambi i campi si profilano due opzioni:

1. Siccome il *token* utilizzato per i test è minimale, ha una lunghezza minore di 2048 bytes ed è quindi possibile spezzarlo e spedirlo utilizzando entrambi i campi.
2. Delegare la fase di validazione del *token* a un bastion host^G che eliminerà quindi il *boilerplate* mantenendo solo i dati essenziali di dimensione ben minore di 1024 bytes.

Dopo un consulto con il tutor aziendale la decisione è stata di utilizzare il secondo approccio per evitare futuri problemi di lunghezza del *token* e in ogni caso è molto scomodo per un utente dividere il *token* in due parti.

Inoltre uno scenario di questo tipo era stato ipotizzato e sono quindi stato guidato verso Apache Guacamole^G, un software per remote desktop e il motivo è semplice: questo software supporta il protocollo SSH^G "*out-of-the-box*" e consente una possibilità che poi comunque avrebbero voluto esplorare. Cioè un volta effettuato il login da Monokee SSO^G, di *spownare una shell* virtuale sulla finestra del browser. Oltretutto il software è *open source* ed è possibile quindi modificare il processo di login perché inserendo una funzione personalizzata, si possano *sniffare* le credenziali dal flusso di autenticazione e modificarle come descritto nella seconda opzione.

La manipolazione del *token* descritta precedentemente avviene come descritto in Figura 9.

Inizialmente vengono effettuate delle chiamate al credential service^G di Monokee per il processo di DID Auth^G e verifica delle credenziali. Il servizio restituisce esito positivo, accompagnato dalle claim^G, se e solo se l'holder^G è certificato, la VC^G è verificata e il certificato è emesso da Monokee ed è del tipo corretto per accedere al servizio selezionato. In caso di fallimento l'intero processo abortisce.

Successivamente viene effettuato il *parsing* delle claim^G ottenute per eliminare il *boilerplate* e mantenere le informazioni essenziali. Queste informazioni devono essere quindi spedite salvando spazio ma essendo comunque autenticabili e riconducibili al bastion host^G. In questa maniera un attaccante con accesso

alla rete interna non può fabbricare una credenziale generica e prendere il controllo del sistema.

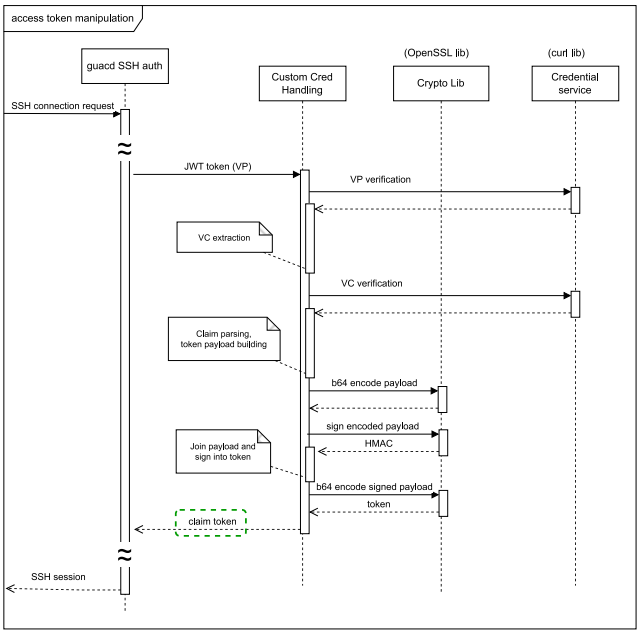


Figura 9: Architettura proposta per la manipolazione del token VC[G] in Apache Guacamole[G]

Come mostrato in Figura 10 i dati sono quindi collassati in un *payload* di dimensione minima e firmati con HMAC[G] dal bastion host[G]. La firma ottenuta è in *formato binary* ed è quindi convertita in *base 64* per essere spedita. La firma viene quindi accodata al *payload* e tutto quanto viene convertito in *base 64* per lo stesso problema di trasmissione.

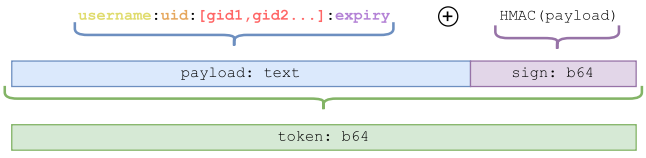


Figura 10: Layout del *token HMAC*[G] restituito da Apache Guacamole[G] e ricevuto da PAM[G]

Come emerge dalle ricerche effettuate sul funzionamento dell'autenticazione SSH[G], viene effettuata una chiamata ad NSS[G] prima di iniziare l'effettivo processo di autenticazione PAM[G] allo scopo di determinare il corretto set di permessi per eseguire lo stesso processo.

Ciò pone un problema nel caso del provisioning[G] di un nuovo utente perché quest'ultimo non esiste ancora nel sistema e il processo fallirebbe di conseguenza.

Progettare il modulo NSS[G] non è banale perché a differenza di PAM[G] non permette di scegliere applicazioni target ed il suo effetto è quindi *system-wide*.

Di conseguenza per intercettare i *token HMAC*[G] è necessario che sia posizionato in testa allo *stack dei resolver* ed deve essere in grado di distinguere uno username unix da un *token HMAC*[G] e fallire inoltrando il compito al prossimo *resolver*. Nel caso del *token* il modulo procederà alla decodifica e verifica della firma. In caso negativo i processo NSS personalizzato abortisce e l'intero processo di autenticazione termina ma in caso positivo viene forgiata una entry *passwd* come descritto precedentemente.

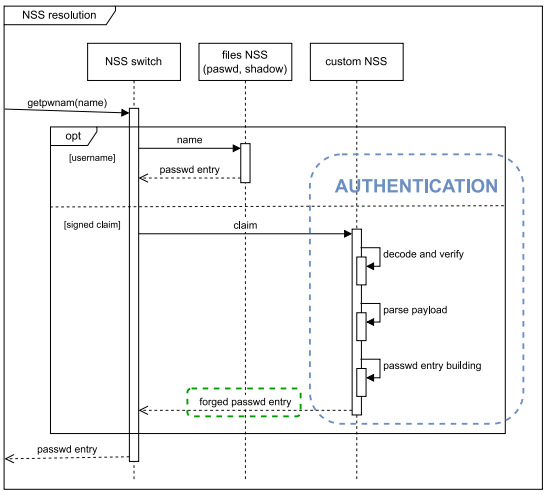


Figura 11: [4] Architettura proposta per il modulo NSS[G]

Successivamente il processo PAM viene inizializzato con i privilegi appena indicato da NSS.

Il processo PAM[G] è composto di 4 fasi ma solo 3 ci interessano per lo scopo: **Authentication**, **Account**, **Session**. Dai test eseguiti e l'analisi del codice sorgente di *OpenSSH* emerge che le fasi cono sempre chiamate in maniera ordinata e possiamo perciò sempre contare sull'ordine delle operazioni.

La prima fase è quella di *Authentication* dove viene controllata l'identità dell'utente.

A questo punto viene ricevuto il *token HMAC*[G] firmato dal bastion host[G] con HMAC[G]. Viene decodificato e dal *parsing del payload* vengono estratti i precedentemente contenuti nella claim[G] della VC[G]

Una volta analizzato il problema ci rendiamo conto che il minimo set di informazioni necessaire è quindi il nome utente seguito da GID[G] e UID[G] le informazioni necessarie

4.3 Codifica

Come appena visto l'architettura si compone di 3 componenti principali, il bastion host[G] e i due moduli PAM[G] e NSS[G].

4.3.1 Apache Guacamole

Per lo sviluppo della modifica custom di Apache Guacamole[G] sono stati necessari degli accorgimenti particolari derivanti principalmente dal fatto che il

linguaggio di sviluppo è il "C". Il linguaggio è noto per essere ostico e datato avendo una pessima *developer experience*.

La prima cosa da fare è stata di trovare un punto del codice dove le credenziali fossero esposte prima del processo di autenticazione SSH[G], in modo da poterle intercettare e modificare.

Figura 12: Snippet intercettazione *auth token* in AApache Guacamole[G]

Questo punto è stato localizzato subito prima del processo di autenticazione Figura 12. In quel punto entrambe le credenziali (nome utente e password) sono esposti e la prossimità all'utilizzo delle stesse promette che nessun'altra modifica verrà applicata alle credenziali una volta alterate.

Come accennato "C" non fornisce la migliore tra le *develpoer sxperience,* infatti per nessuna delle 4 azioni da compiere c'è bisogno di una libreria aggiuntiva.

Le operazioni in questione sono la comunicazione web, il *parsing* dei risultati, la firma e l'*encoding* del risultato. Rispettivamente sono state scelte le seguenti librerie:

Curl è principalmente un tool a linea di comando per il trasferimento di dati tramite *url*. Mette anche a disposizione una API[G] ben documentata per il linguaggio "C" che semplice è facile da usare. Inoltre **curl** è quasi il tool standard in Linux nella sua categoria testimoniando quindi a favore dell'affidabilità della sua API[G].

OpenSSL È la libreria standard per l'implementazione del protocollo TS[G]. Gode di una ottima reputazione come testimonia il suo status di *standard de facto*. L'API TLS che mette a disposizione è basata su una libreria crittografica *general-purpose* di qualità che può essere utilizzata in modalità *standalone* con un API[G] per il linguaggio "C".

Regex.h Fa parte della *C standard library* e offre una, seppur datata, API per la ricerca e sostituzione con *espressioni regolari*

Una volta scelte le librerie e testate singolarmente è necessario integrare al progetto. Siccome il linguaggio è "C" la compilazione è appoggiata a un *build*

system dove tutte le informazioni riguardo la compilazione sono configurate, quindi anche le librerie da includere.

Il *build_system* utilizzato è *cmake* ma data la dimensione spropositata della *codebase* vengono utilizzati anche *Apache Automake* e *Apache Autoconf*

Normalmente è sufficiente un *build system* per la configurazione della build ma quando il progetto diventa molto grande è spesso necessario di configurare il *build system* a sua volta e ciò è fatto con *Apache Automake* e *Apache Autoconf*. Questi due tool dono utilizzati per la generazione sistematica della configurazione del *build system* tramite file di configurazione adeguato.

Sfortunatamente per quanto inestimabili nel giusto contesto, questi tool sono noti per essere difficili da usare. Infatti il solo riuscire a compilare localmente il codice sorgente di Apache Guacamole[G] ha richiesto svariate ore di lavoro, cosa che si è ripetuta per l'inserimento delle librerie nella configurazione del processo di build. Un esempio è mostrato in Figura 13 per l'inserimento della libreria curl.

```
AC_CHECK_LIB([ssh2], [libssh2_userauth_publickey_frommemory],
                     [SSH_LIBS="$SSH_LIBS -lssh2 -lcurl"],
                     [have_libssh2=no])
```

Figura 13: Snippet aggiunta libcurl ad Apache Autoconf

Una volta riusciti a compilare i cambiamenti con le librerie adeguate, è stato implementato il *client web* di Guacamole. Il progetto mette a disposizione una API[G] javascript chiamata **guacamole-common-js** permettendo l'accesso al protocollo utilizzato da Apache Guacamole e il meccanismo di *tunneling* utilizzato per collegarsi al server.

L'interfaccia utente implementata è molto semplice e permette la scelta di un host di destinazione e una VC[G] da spedire. Se a questo punto il collegamento ha successo, viene visualizzato un terminale virtuale che viene poi terminato alla chiusura della sessione SSH[G].

A lato server il progetto mette a disposizione una api *Java* chiamata **guacamole-common** per il raccordo tra il vero e proprio demone server **guacd** ed il client. La scelta è però ricaduta sul progetto **guacamole-lite** che offre un *drop-in replacement* scritti in *Node.Js* per **guacamole-common**.

4.3.2 NSS e PAM

Per lo sviluppo dei due moduli personalizzati NSS[G] e PAM[G] è stato scelto il linguaggio *Rust* per più motivi. La motivazione principale è stata di evitare il linguaggio "C" poiché può essere ostico e il linguaggio *Rust* ha performance quasi equivalenti ed è la migliore tra le alternative per la sintassi e varietà di librerie di qualità ma soprattetto per la facilità con cui è possibile interfacciarsi con altri linguaggi tramite le Rust FFI[G]. In questo caso ce n'è bisogno poiché le API[G] di Linux, come ogni altro sistema operativo, sono disponibili per il linguaggio "C" ed è necessario adattarle.

Anche in questo caso sono necessarie delle librerie per il "parsing", la codifica in *base 64* e la firma HMAC[G] ma sono anche necessarie delle interfacce Rust FFI[G] che mi permettano l'accesso alle api PAM[G] e NSS[G]:

base64 libreria semplice e leggera con l'unico scopo di effettuare codifica/decodifica di dati in *base64*

ring utilizzata in questo caso per la firma HMAC[G]. È una libreria crittografica *general purpose* semplice, leggera e facile da usare. Implementa le operazioni crittografiche principali ma non è completa come potrebbe essere OpenSSL

regex È una libreria che fornisce il supporto per espressioni regolari. Lo sviluppo è concentrato sull'efficienza e manca quindi il supporto per operazioni per cui non è ancora stato scoperto un algoritmo efficiente. In questo caso in cui solo una semplice ricerca, le funzionalità offerte sono sufficienti

pam_rs È una libreria Linux che non è caricata ufficialmente su *crates.iO* e fornisce i binding per il set completi di API esposte da PAM. Ciò significa che è possibile utilizzare questa libreria per creare un client che supporti PAM[G] e un modulo personalizzato. Anche in questo caso non tutte le funzioni sono incluse ma tutte le funzionalità sono incluse e potrebbero comunque essere aggiunte manualmente.

libnss Fornisce *bindings* Rust FFI[G] per costruire moduli NSS[G] personalizzati in *Rust*. Non è completa in quanto non supporta tutti i database ci NSS ma anche questa volta è sufficiente allo scopo. Questa libreria non supporta funzionalità di client NSS[G] ma ciò è facilmente fattibile interrogando direttamente la shell con la libreria standard di *Rust* e le utility di lLinux.

Una volta effettuato il *parsing* dei dati è necessario verificare che l'utente sia presente nel sistema ed effettuare il provisioning[G]. Per effettuare queste due operazioni ci sono due strade percorribili:

1. Utilizzare l'interfaccia file IO di *Rust* e modificare direttamente i file di sistema per aggiungere l'utente.
2. affidarsi alle utility di sistema ed accederle tramite il *package* Command di *Rust*.

Il primo approccio è più puro ma necessita di una conoscenza estremamente approfondita del processo e c'è sempre il rachio di aver dimenticato qualche caso particolare, poi non è mai una cosa saggia implementare da zero dei protocolli di sicurezza. La scelta ricade quindi sulla seconda opzione.

Per comunicare don il sistema in questa maniera è necessario utilizzare il *package* Command di *Rust* che mette a disposizione funzionalità per interagire con la shell di sistema registrando gli stream stdout e stderr lo *status code*.

Capitolo 5

Verifica e validazione

In questo capitolo viene discussa e documentata la fase di testing del prodotto.

Premetto che trattandosi di un PoC[G], non ho ritenuto necessario considerare una suite di test approfondita dando priorità invece alla fattibilità tecnica e la dimostrazione delle potenzialità del sistema.

Praticamente possiamo quindi dividere il progetto in due fasi molto distinte dal punto di vista della codifica: le personalizzazione di Apache Guacamole[G] e la scrittura di moduli di sistema Linux (NSS[G] e PAM[G]). Entrambi hanno profili molto diversi e presentano problematiche diverse anche rispetto al *testing*.

Dal punto di vista del *testing* la fase più problematica è stata la personalizzazione di Apache Guacamole[G] e il motivo è semplice. La *codebase* è mastodontica ed è scritta nel linguaggio di programmazione "C" utilizzato a un livello sofisticato. Entrambi questi aspetti mi erano poco familiari (e per certi aspetti lo sono tutt'ora), infatti ho incontrato diversi *roadblock* durante il processo come discusso ampiamente in Sezione 4.3. La scelta fatta quindi in questo caso fatte le precedenti considerazioni è stata di procedere senza test automatici.

Spostandosi alla scrittura di moduli di sistema Linux (NSS[G] e PAM[G]), si è passati a una *codebase* più semplice e a un linguaggio come *Rust* che integra un *test framework*. Nonostante ciò e il supporto *builtin* per i test di *Rust* risulta comunque difficile introdurre test significativi per la velocità con cui il codice mutava. La maggior parte sono stati quindi effettuati a mano per provare il più possibile a completare gli obiettivi fissati nel tempo stabilito.

Per coprire tutte le funzionalità e assicurarne il funzionamento, sono state effettuate diverse verifiche che possono essere classificate in diverse categorie di test:

- Test di **connessione web** con la libreria *curl* per accertarsi che il credential service[G] risponda correttamente alle richieste del bastion host[G]
- Test di **regex** sui *token HMAC*[G] per assicurarsi che i *token HMAC*[G] e le VC[G]/VP[G] vengano processate e i dati vengano processati correttamente ritornando i dati attesi;
- Test di parsing tramite **regex** su VC[G] e VP[G] per assicurarsi che le claim vengano estratte con successo;
- Test di **encoding e decoding** in *base64* per verificare che le librerie *openssl* e *base64* siano utilizzate correttamente producendo il risultato atteso
- Test di **firma HMAC**[G] per verificare che le librerie *ring* e *opensssl* siano utilizzate correttamente e producano il risultato atteso.
- Test di generali **sistema**

Una volta ultimati i test e certificato il funzionamento di ogni "*building block*", sono stati eseguiti i test di sistema. Come era prevedibile questi si sono rivelati i più ostici e sono quelli che hanno richiesto la maggior dedizione per essere completati.

Tenendo a mente che si tratta di un PoC[G] è irragionevole pensare di coprire tutti gli *edge case*. È quindi necessario creare una *baseline* di assunzioni realistiche sul sistema che dobbiamo assumere vere in ogni istante per limitare la quantità di casi possibili di cui tenere conto nei test di sistema:

- nel bastion host[G] host che esegue Apache Guacamole[G] la comunicazione tra i processi *guacd* (Guacamole daemon) e *guacamole-lite* sia sempre attiva e affidabile.
- Non cada la connessione tra *client* e *server* durante il processo di autenticazione.
- L'*host SSH*[G] di destinazione sia sempre raggiungibile e adeguatamente configurato.
- La risposta del credential service[G] sia sempre in formato corretto

Non è difficile immaginare uno scenario in cui la mancanza di questa *baseline* porti quando meno a *unexpected behaviour* o al fallimento diretto del processo di autenticazione, sottolineando l'importanza di test accurati al di fuori dell'ambiente di sviluppo.

Sapendo che le *core feature* del sistema sono il login tramite VC[G], il provisioning[G] di un utente non registrato e l'applicazione di ulteriori restrizioni, possiamo identificare i casi interessanti da verificare.

Siccome i moduli creati e il bastion host[G] modificato sono già stati testati, ora è necessario verificare il funzionamento in un contesto reale, quindi con un vero *host SSH*[G] a cui connettersi che sia correttamente configurato e implementi i 2 moduli NSS[G] e PAM[G]. Perché il prodotto possa essere accettato è necessario che il sistema possa gestire la situazione in ogni stato possibile, inclusi quindi lo stato del sistema target e dell'input ricevuto(la VC[G]).

I test di sistema effettuati sono quindi stati i seguenti:

- Data una VC[G] valida, non scaduta e di tipo coretto e collegata a un utente sia registrato, il autenticazione sia permessa;
- Data una VC[G] non valida, l'autenticazione fallisca;
- Data una VC[G] valida ma scaduta, l'autenticazione fallisca;
- Data una VC[G] valida, non scaduta e collegata a un utente non registrato, venga fatto il provisioning[G] correttamente
- Dopo ogni autenticazione riuscita, controllare che l'utente realmente autenticato presso l'host sia quello atteso e che il sistema stia eseguendo, dal punto di vista dell'utente, con i privilegi coretti. In altre parole i dettagli dell'utente devono corrispondere alla VC[G] presentata;
- Nel caso di autenticazione riuscita con provisioning[G], controllare dopo la disconnessione che sia stato effettuato il *deprovisioning*
- In caso di autenticazione riuscita con provisioning[G], utilizzare un altro account con privilegi amministrativi per controllare che, dal punto di vista del sistema, l'utente sia stato correttamente *provisionato*, anche se solo temporaneamente.

Tutti i controlli sono stati eseguiti con 3 credenziali diverse per assicurare un minimo di ripetibilità e non affidare l'esito dei test al caso o a un *glitch*.

Successivamente sono stati effettuati degli altri test meno strutturati ma più specifici per assicurarsi che l'autenticazione venga rifiutata in caso le credenziali fornite fossero conflittuali. In altre parole il problema è che in caso la claim^G ottenuta contenesse credenziali tali per cui si verificasse un conflitto di UID^G , GID^G o username, significherebbe che il sistema si trova in uno stato anomalo e l'autenticazione non sarebbe sicura o possibile.

Una veloce analisi dello scenario identifica 2 fattori che contribuiscono all'esito dell'operazione:

1. presenza di un utente registrato a sistema;
2. validità delle credenziali fornite;
3. possibilità di login prima e dopo il primo accesso di un utente autorizzato;

Il passaggio della suite di test proposta verifica il funzionamento del login via VC^G ed il corretto provisioning^G dell'account quando necessario.

Capitolo 6

Possibili cambiamenti e miglioramenti

In questo capitolo so discute brevemente dei possibili miglioramenti apportabili al PoC[G] sia da un punto di vista architetturale che delle funzionalità.

Verso la fine dello stage è stato organizzato un incontro con tutto il team al completo (io ho lavorato quotidianamente solo con alcuni di loro)dove ho presentato li mio lavoro e sono giunti degli spunti interessanti.

Di seguito sono elencati gli spunti più interessanti provenienti dalla presentazione finale e varie interazioni con il tutor aziendale nell'intero corso dello stage, a cui seguono delle considerazioni.

6.1 Miglioramenti architetturali

Durante la presentazione sono state proposte principalmente due modifiche architetturali, una dividendo il e una seconda provando a fare leva sul meccanismo selle chiavi

6.2 Miglioramento delle funzionalità

- **Provisioning e deprovisioning on demand** Attualmente, a scopo dimostrativo, un account di cui è stato fatto il provisioning[G], viene eliminato successivamente alla sconnessione della sessione ma nulla vieterebbe di decidere che comportamento prendere con l'aggiunta di un flag. Ad esempio il provisioning[G] potrebbe essere deciso in base a parametri elaborati in locale o non essere effettuato se l'utente non è registrato in locale. Si potrebbe anche pensare di definire diverse modalità selezionabili dal *flag*. Una possibile implementazione potrebbe essere anche semplicemente il passaggio di un flag come primo carattere che definisca il comportamento da tenere riguardo al provisioning[G].
- **Fetch di una configurazione standard in provisioning** Un altra opzione interessante sarebbe di utilizzare la fase di **Create session** di PAM[G] per effettuare il *fetch* di una configurazione standard in caso di provisioning[G]. Questa potrebbe includere una serie di tool e impostazioni che potrebbero variare dal gruppo a gruppo o in base a claim[G] effettuate dalla credenziale verificabile.

Capitolo 7

Conclusioni

Questo capitolo chiude con le riflessioni personali sull'esperienza vissuta e una valutazione oggettiva sui risultati dello stage. Vengono discussi vari aspetti come l'aderenza alla pianificazione e il raggiungimento degli obiettivi pianificati tenendo conto delle sfide incontrate.

7.1 Consuntivo finale

Le prime due settimane di stage sono andate completamente secondo i piani riuscendo anche a risparmiare una giornata di formazione. È stato però chiaro fin dall'inizio che sarebbe stato necessario ricavare del tempo per un attività di cui non è stato tenuto completamente conto nella pianificazione, cioè l'introduzione del bastion host^G

Come dettagliato la questione della personalizzazione di Apache Guacamole^G ha rallentato considerevolmente il lavoro fino alla terza settimana che è stato in parte recuperato durante la quinta.

Il lavoro è proceduto dopo nei tempi stabiliti anche se invertito tra sviluppo NSS^G e PAM^G. Il tempo guadagnato durante la seconda e quinta settimana non sono però bastati a compensare la terza settimana di lavoro non programmato ed è stato perciò necessario aggiungere una giornata di stage nell'ottava settimana. Ciò però è comunque risultato in una fase documentativa molto stringata.

Settimana	Attività	Ore
1 - 2	Formazione	56 (-8)
2 - 3	Modifica Apache Guacamole^G	64 (+64)
5	Progettazione e sviluppo NSS	32 (-24)
4 - 5	Progettazione e sviluppo PAM	56
5 - 8	Esperimenti e testing	88
8	Documentazione stage	16 (-24)
	Totale	312 (+8)

Tabella 3: Consuntivo ore finale

7.2 Raggiungimento degli obiettivi

Dei 7 obiettivi prefissati 4 sono stati raggiunti completamente, uno solo parzialmente e 3 non sono stati raggiunti.

Tutti i requisiti obbligatori sono stati soddisfatti mentre solo alcuni dei requisiti desiderabili e facoltativi sono stati presi in considerazione. Di questi ultimi, uno desiderabile è stato raggiunto e uno facoltativo è stato parzialmente raggiunto.

Come evidente in Tabella 3 sono state riscontrate sfide impreviste durante lo svolgimento dell'attività di stage rendendo difficile il completamento in toto degli obiettivi.

Codice	Descrizione	Stato
OO01	Progettazione e sviluppo di un modulo NSS^G personalizzato per l'autenticazione basata su credenziali verificabili.	Raggiunto
OO02	Progettazione e sviluppo di un modulo PAM^G personalizzato per l'integrazione dell'autenticazione SSH^G con le credenziali verificabili e il provisioning^G JIT (*just-in-time*)^G degli account.	Raggiunto
OO03	Realizzazione di una PoC^G funzionante che dimostri l'autenticazione SSH^G basata su credenziali verificabili e il provisioning^G JIT (*just-in-time*)^G degli account.	Raggiunto
OD01	Gestione delle politiche di accesso e delle autorizzazioni tramite PAM^G.	Raggiunto
OD02	Integrazione con un sistema di gestione delle identità e degli accessi esistente.	Non raggiunto
OF01	Documentazione di integrazione per consentire ad altri sviluppatori di utilizzare il meccanismo di autenticazione SSH^G basato su credenziali verificabili.	Parziale
OF02	Implementazione di meccanismi di revoca delle credenziali verificabili.	Non raggiunto

Tabella 4: Raggiungimento dei requisiti di progetto

7.3 Conoscenze acquisite

Nel corso dello stage ho appreso molte nuove conoscenze, alcune delle quali inaspettate.

La principale competenza acquisita riguarda il mondo della SSI^G. È un paradigma nuovo ed entusiasmante ma che ambisce già a cambiare il mondo dell'identità digitale con il suo uso originale e creativo di una "vecchia" tecnologia come la *blockchain*.

Un altra competenza molto importante e molto tecnica che ho appreso è l'orientamento in grandi *codebase* e la comprensione del decide. Durante questi mesi infatti ho dovuto cimentarmi nell'analisi dettagliata e approfondita di SSH^G usando *GitHub* per carpire il suo funzionamento dettagliato. Inoltre ho avuto bisogno di comprendere Apache Guacamole^G abbastanza da poter modi-

ficare la funzionalità di autenticazione integrata. Ho imparato quindi a modificare un codice complesso scritto da altri.

Altro set di competenze ha a che fare con il processo standard di autenticazione di Linux che fa affidamento su PAM[G] e NSS[G] oltre che ai file passwd[G] e shadow[G]. L'analisi del protocollo SSH[G] mi ha anche fatto comprendere come funziona sotto il cofano l'autenticazione di questo protocollo.

Ultima delle competenze tecniche, ma per questo no meno importante è la conoscenza del linguaggio di programmazione *Rust* e delle sue librerie Rust FFI[G] e quindi la *keyworkd* unsafe

Per quanto riguarda invece le conoscenze e competenze non tecniche acquisite, ho imparato molte cose tra cui il time management il lavoro in team e l'autonomia di lavoro. In particolare la distinzione tra le due e come debbano coesistere in un buon bilanciamento per essere efficaci. Inoltre ho potenziato le mie abilità di *troubleshooting* e *problem solving* grazie a tutte le sfide affrontate.

La conoscenza forse più importante è però l'esperienza in un ambito lavorativo reale che mi insegna cosa aspettarmi quando entrerò effettivamente nell'industria.

7.4 Valutazione personale

Volendo fare un resoconto finale dell'esperienza è stata molto positiva. Non solo mi sono divertilo ma ho acquisito conoscenze e collegamenti di industria molto importanti che un giorno potrebbero aiutarmi molto.

Le conoscenze che ho acquistato sono anche spendibili direttamente sul lavoro e non possono essere insegnate, ma vanno provate quindi sono grato di averne avuto la possibilità.

Quest'esperienza mi ha dato un altro punto di vista e delle nuove consapevolezze riguardo alla carriera che sto intraprendendo e mi ha mostrato che per quanto possa tutto sembrare intimidatorio, la calma e il sangue freddo sono sempre la soluzione migliore.

Facendo della retrospettiva posso dire di essere moderatamente orgoglioso di ciò che ho fatto, imparato e dei passi avanti che ho fatto come persona e come professionista. Ciò che mi impedisce di essere completamente orgoglioso è la consapevolezza che avrei potuto fare qualcosa in più per sfruttare al meglio questa esperienza. Ad esempio avrei potuto sfruttare maggiormente i miei colleghi e il tutor aziendale che hanno sempre dato disponibilità piena. Io però ho preferito in più occasioni di prendere la via difficile convinto di potermi arrangiare. A onor del vero mi sono sempre arrangiato in quelle situazioni poiché conosco i miei limiti, ma devo imparare che a volte, specialmente con degli esperti a disposizione, chiedere aiuto può risultare in un risparmi o di tempo e in un inestimabile opportunità di apprendimento e miglioramento.

Glossario

Agile: Per sviluppo Agile del software si intende la disciplina dell'Ingegneria del Software derivata dall'Agile manifesto del 2001 [13] che colleziona le best-practice per uno sviluppo software ordinato e ordinato al processo piuttosto che al risultato. 1, 36

***API* – Application Programming Interface**. 10, 17, 19, 20, 24

bastion host: Host di rete il cui compito è isolare un altro host di rete agendo da frontend per il secondo host. 21, 22, 23, 27, 28, 31

clientless: Si riferisce ad una modalità di accesso a risorse chen non preveda l'installazione di software client sul disspositivo dell'utilizzatore. Tipicamente ciò avviene utilizzando un client web renderizzato "on-the-fly" sulla finestra del browser. 10

CMake: *Build system* utilizzato ormai come standard de-facto per la compilazione incrementale di codice "C" o "C++" 9, 35

credential service: Servisio di Monokee che verifica VP^G e VC^G rilasciate dall'azienda stessa. 21, 27, 28

daemon: In un sistema *multitasking* è un programam che esegue in background senza supervisione dell'utente e fornendo un servizio. Spesso noti impropriamente come "server" 10

***DID* – Distributed Identifier**: identificatore univoco associato ad un DID Document^G che costituisce una "identità astratta". L'unico modo per collegarlo ad un soggetto è il processo di DID Auth^G. Vedi Sezione 3.3.2 12, 13, 14, 34

DID Auth: Processo efettuato da un verifier^G per associare un DID^G a un soggetto che deve quindi dimostrarne il controllo. Vedi Sezione 3.3.2 13, 34, 36

DID Controller: Soggetto che può provare di essere in grado di modificare un DID Document^G. Vedi Sezione 3.3.2 13

DID Document: Documento digirale immagazzinato in una *blockchain* che contiene le informazioni utili al processo di DID Auth^G e a verificare la firma apposta a VC^G e VP^G- Vedi Sezione 3.3.2 13, 14, 34

DID Resolution: Processo che dato un DID^G restituisce il DID Document^G associato. Vedi Sezione 3.3.2 13, 14

DID Subject: Soggetto associato ad un DID^G e descritto da un DID Document^G. Vedi Sezione 3.3.2 13

***GID* – Group ID**: Nel contesto del sistema operativo Linux, è un identificativo assogciato univocamente ad un gruppo di utenti del sistema. Se associato ad un UID^G indica l'appartenenza al gruppo dell'utente ad esso associato 20, 23, 29, 35

Apache Guacamole: *Gateway desktop* remoto (o sulizione per desktop remoto) opensource sviluppato da *Apache Software Foudation*. Supporta diversi protocolli e opera in modalità clientless. Vedere Sezione 3.2 iv, 10, 22, 24, 27, 28, 31

***HMAC* – Keyed-Hash Message Authentication Code**: È un codice crittografico utilizzato per autenticare e verificare l'integrità di un messaggio tramite l'utilizzo di una *funzione di hash* e una chiave segreta condivisa. Vedere Sezione 3.7 iv, 16, 22, 23, 26, 27

***IAM* – Identity and Access Management**: Con IAM si intendono i sistemi integrati di tecnologie, criteri e procedure in grado di consentire alle organizzazioni di facilitare - e al tempo stesso controllare - gli accessi degli utenti ad applicazioni e dati critici, proteggendo contestualmente i dati personali da accessi non autorizzati. [14] 1, 3, 7

***JIT* – just-in-time**: Filosofia che prevede la produzione di una risorsa solo quando qeusta sia necessaria. 5, 6, 7, 8, 17, 32

***JWT* – Json Web Token**: È un formato per la trasmissione di informaizoni in formato JSON. Fornisce funzionalità di confidezialità (JWE) tramite crittografia e autenticazione (JWS) tramite firma crittografica 21

***LSP* – Language Server Protocol**: Nel contesto degli ambienti di sviluppo software, protocollo che permette ad un *language server* di comunicare con più client per controllare gli errori nel codice o fornire supporto specifico al linguaggio. 9

Monokee SSO: Monokee SSO è un prodotto di Monokee s.r.l sviluppato in collaborazione con Athesys. È una soluzione di SSO^G modulare il cui punto forte è la facile integrazione con nuove tecnologie. [7] 1, 21

Ninja: *Build system* focalizzato sulla veolcità di esecuzione. Pensato per essere eseguito da un *Build system* di più alto livello, tipicamente CMake^G [15] 9

***NSS* – Name Switch Service**: In un sistema Linux il modulo NSS^G è responsabile della risoluzione delle informazioni relative agli utenti, gruppi e altre entità di sistema all'interno del sistema operativo *UNIX*; permette di integrare nuovi servizi di autenticazione e autorizzazione nel sistema. Vedi Sezione 3.5) i, iv, 5, 6, 20, 22, 23, 25, 26, 28, 31, 32, 35

***PAM* – Pluggable Authentication Modules**: Il modulo PAM^G fornisce un'interfaccia per l'autenticazione delle credenziali degli utenti e per l'autorizzazione degli accessi alle risorse del sistema; permette di configurare politiche di autenticazione e autorizzazione personalizzate, consentendo l'uso di diversi metodi di autenticazione. Vedi Sezione 3.4 i, iv, 5, 6, 7, 17, 18, 19, 20, 22, 23, 25, 26, 27, 28, 30, 31, 32, 35

PAM Client: Un applicazione che si serve dei servizi di PAM. 17

PAM Context: L'insieme dei dati di contesto riguardanti riguardanti l'attuale sessione PAM^G. Ne fanno parte le funzoini di conversazione, le redenziali presentate e lo stato interno. 18, 19

passwd: File facente parte del istema di autenticazinoe di Linux. Contiene informazioni relative al'utente come username, GID^G e UID^G. Normalmente lavora in congiunzione con il file shadow^G 33, 36

piano di lavoro: Il piano di lavoro è un documento concordato a priori dello stage tra il candidato, il responsabile stage, il tutor interno e il tutor aziendale dove vengono descritte le condizioni e i vincoli del progetto. Vi sono dettagliati tema, pianificazione, requisiti, deliverable e i termini generali del contratto di stage come la eventuale retribuzione, orario etc… 3, 5

***PoC* – Proof of Concept**: In italiano si traduce con "dimostrazione di fattibilità" e si riferisce a una realizzazione incompleta o abbozzata il cui scopo è quello di dimostrare la fattibilità o confermare la validità di alcuni principi o concetti fondamentali. i, 5, 6, 27, 28, 30, 32

provisioning: Con provisioning si intende il processo di creazione e configurazione di un'infrastruttura IT, che comprende le procedure necessarie per

gestire l'accesso di utenti e sistemi a varie risorse. [16] In qeusto caso il provisioning è riferito all'utente descritto dalle VC. i, 5, 6, 7, 8, 28, 30, 32

***RDP* – Remote Desktop Protocol**: Protocollo per desktop remoto sviluppato da Microsoft 10

***Rust FFI* – Foreign Function Interface**: Parte della libreria standard del linguaggio di programmazione Rust. Permette un modo semplice per creare binding per le API di altri linguaggi di programmazione come ad esempio il "C" 25, 26

Scrum: Scrum è il modello di sviluppo Agile[G] più conosciuto. I suoi tratti più carattersitici sono gli *stand-up meeting* giornalieri, la divisione di periodi chiamati *Sprint* e la valutazione dei requisiti tramite *user stories*. [17] 1

shadow: File facente parte del istema di autenticazinoe di Linux. Contiene l'-hash delle password di tutti gli utenti e le informazioni relative. Normalmente lavora in congiunzione cin il file passwd[G] 33, 35

***SSH* – Secure SHell**: Protocollo che permette di stabilire una sessione remota cifrata e sicura alla famigerata porta 22 tra due host connessi in rete. Vedi Sezione 3.6) i, 5, 6, 15, 17, 19, 20, 21, 22, 24, 25, 28, 32, 33

***SSI* – Self-Sovereign Identity**: È un nuovo paradigma nella gestione dell'identità digitale e si propne come la soluzione definitiva al problema delel moderne souzioni di SSO e autenticazoine i ngenerale con prevenzione delle frodi utilizzando la blockchain ed il concetto di identità decentralizzata. Vedi Sezione 3.3 i, iv, 1, 3, 7, 11, 14, 17, 32

claim: Asserzione effettuata da un issuer nei confronti di un issuer[G] nei confronti di un suggetto ed immagazzinate in un VC[G] Vedi Sezione 3.3.2 13, 14, 21, 23, 29, 30, 36, 37

holder: Entità che riceve di conseguenza e detiene delle VC[G] che spesso (ma non necessariamente) la riguardano Vedi Sezione 3.3.2 12, 13, 14, 21, 37

issuer: Entità che emette una VC[G] contenente delle claim[G] riguardo un soggetto una proof[G]. Vedi Sezione 3.3.2 12, 13, 14, 36, 37

proof: Firma generata tramite la chiave privata di un issuer[G] apposta in coda ad una VC[G] o VP[G] ed utilizzata per verificarne l'autenticità tramite la chiave pubblica dell' issuer[G] stesso. Vedi Sezione 3.3.2 13, 14, 36, 37

verifier: Entità che effettua il processo di DID Auth[G] e verifica della/e claim[G] al fine di associare concertezza una claim[G] ad un soggetto. Vedi Sezione 3.3.2 12, 13, 14, 34

***SSO* – Single Sign-on**: Un sistema di Single Sign-on è un sistema di controllo dell'accesso centralizzato. Ciò significa che una sola autenticazione presso il sistema consente di essere autenticati presso tutti i servizi che lo utilizzano. Ex. "Google SSO" o "Facebook SSO". 1, 7, 35

system level programming: Termine ombrelloche comprende le attività di sviluppo a basso livello in linguaggi come "G" o *Rust*. Tipicamente è associato alla scrittura di codice *backend* molto specializzato o per un sistema operativo o vigeogame. 9

***TS* – Transport Layer Security**. 24

***UID* – User ID**: Nel contesto del sistema operativo Linux, è un identificativo assogciato univocamente ad un utente regitrato al sistema 20, 23, 29, 34, 35

***VC* – Verifiable Credential**: Credenziale a prova di manomissione la cui validità può essere verificata crittograficamente tramite una proof[G] che con-

tiene. Viene generatA da un issuer^G e contiene una o più claim^G. [11] Vedi Sezione 3.3.2 i, iv, 5, 6, 13, 14, 15, 17, 21, 22, 25, 27, 28, 29, 34, 36, 37

***VNC* – Virtual Network Computing**: Protocollo per desktop remoto 10

***VP* – Verifiable Presentation**: Credenziale a prova di manomissione la cui validitè può essere verificata crittograficamente tramite una proof^G che contiene. Viene generatA da un holder^G e contiene una o più VC^G. Una credenziale verificabile è una credenziale a prova di manomissione la cui paternità può essere verificata crittograficamente. [11] Vedi Sezione 3.3.2 o 13, 14, 27

***VPN* – Virtual Private Network**: Tecnologia che tramite tunneling e più in generale crittografia, consente di creare un canale di comunicazione sicuro tra due dispositivi. Tipicamente è utilizzato per accesso ad una rete privata (spesso aziendale) e i due dispositivi sono un gateway e un dispositivo fuori dalla rete. 3

***VPS* – Virtual Private Server**: Istanza virtualizzata di una macchina che permette di eseguire pattività in un ambiente controllato, scalabile e opaco , creando un sistema tipo *sandbpx* 3

***WASM* – Web Assembly**: Standard Web che per mette l'esecuzione di codice compilato all'interno di una pagina Web con performance simili a quelle native. 9

Bibliografia

[1] Apache Foundation. Disponibile su: https://guacamole.apache.org/doc/0.9.4/images/guac-arch.png

[2] github.com/vadimpronin. Disponibile su: ttps://cloud.githubusercontent.com/assets/5534215/25705792/3140af24-30e7-11e7-99a0-0f77c5bf2e73.png

[3] Daniel H Hardman via Wikimedia Commons. Disponibile su: https://commons.wikimedia.org/wiki/File:VC_triangle_of_Trust.svg

[4] Noerionderivative via Wikimedia Commons. Disponibile su: https://commons.wikimedia.org/wiki/File:PAM_Diagramm.svg

[5] «Pagina web di Athesys s.r.l.». Consultato: 1 dicembre 2023. [Online]. Disponibile su: https://athesys.it/title/

[6] «Pagina web di Monokee s.r.l.». Consultato: 1 dicembre 2023. [Online]. Disponibile su: https://monokee.com/azienda/

[7] «Pagina web di Monokee SSO». Consultato: 1 dicembre 2023. [Online]. Disponibile su: https://monokee.com/single-sign-on/

[8] «Rust is the most admired language». Stack Overflow. Consultato: 4 dicembre 2023. Disponibile su: https://survey.stackoverflow.co/2023/#section-admired-and-desired-programming-scripting-and-markup-languages

[9] «Guacamole implementation and architecture». Apache Foundation. Consultato: 4 dicembre 2023. Disponibile su: https://guacamole.apache.org/doc/gug/guacamole-architecture.html

[10] «W3C Recommendation | Decentralized Identifiers (DIDs) v1.0». W3C. Consultato: 4 dicembre 2023. Disponibile su: https://www.w3.org/TR/did-core/

[11] «W3C Working Draft | Verifiable Credentials Data Model v2.0». W3C. Consultato: 4 dicembre 2023. Disponibile su: https://www.w3.org/TR/vc-data-model-2.0/

[12] «openssh-portable». GitHub. Disponibile su: https://github.com/openssh/openssh-portable

[13] «Agile Manifesto». Disponibile su: https://agilemanifesto.org/

[14] «Identity Management». Wikipedia. Consultato: 4 dicembre 2023. Disponibile su: https://it.wikipedia.org/wiki/Identity_management

[15] «Ninja build system». [Online]. Disponibile su: https://ninja-build.org/

[16] «Cos'è il provisioning?». RedHat. Consultato: 4 dicembre 2023. Disponibile su: https://www.redhat.com/it/topics/automation/what-is-provisioning

[17] «Che cos'è Scrum e come iniziare». Atlassian. Consultato: 4 dicembre 2023. Disponibile su: https://www.atlassian.com/it/agile/scrum