Last mile authentication problem

Exploiting the missing link in end-to-end secure communication

DEF CON 26

Our team

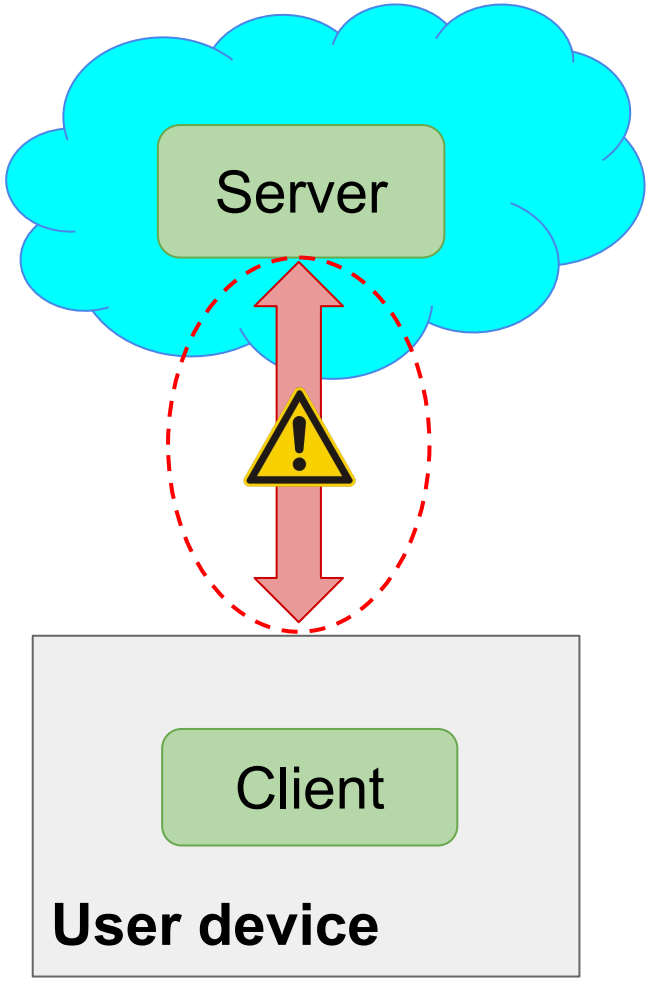
Sid Rao
Doctoral Candidate
Aalto University
Finland

Thanh Bui
Doctoral Candidate
Aalto University
Finland

Dr. Markku Antikainen
Post-doc researcher
University of Helsinki
Finland

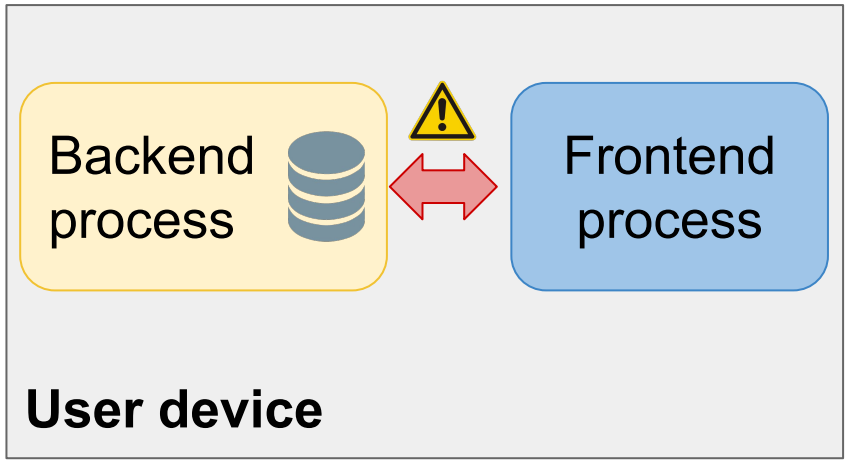
Prof. Tuomas Aura
Professor
Aalto University
Finland

Traditional network threat model



- Server and user device are trusted
- Untrusted network:
  - “man in the middle”
- Solution: crypto (TLS and web PKI) to protect communication

Our focus: Inter-process communication (IPC)



- Not all communication goes over the traditional network
- Software consists of multiple local processes that need to communicate

We try to understand security of communication inside the computer

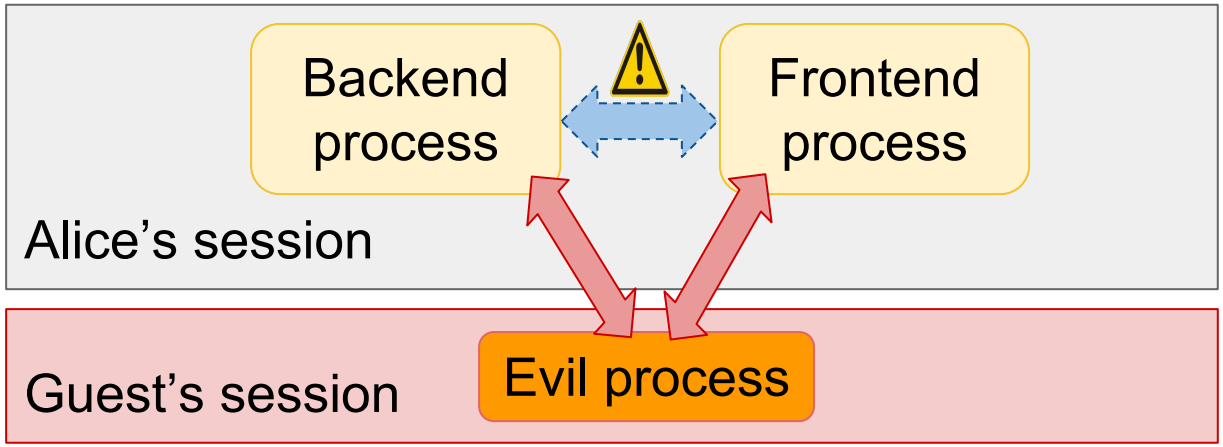
This talk

- We exploit “last mile communication” inside the computer
- Structure
  - **Man-in-the-Machine**
  - IPC and attack vectors
  - Case studies
  - Mitigation
  - Conclusion

Credentials and **second authentication factors**
can be compromised from inside the computer

Man-in-the-Machine (MitMa)

Man-in-the-Machine (MitMa)



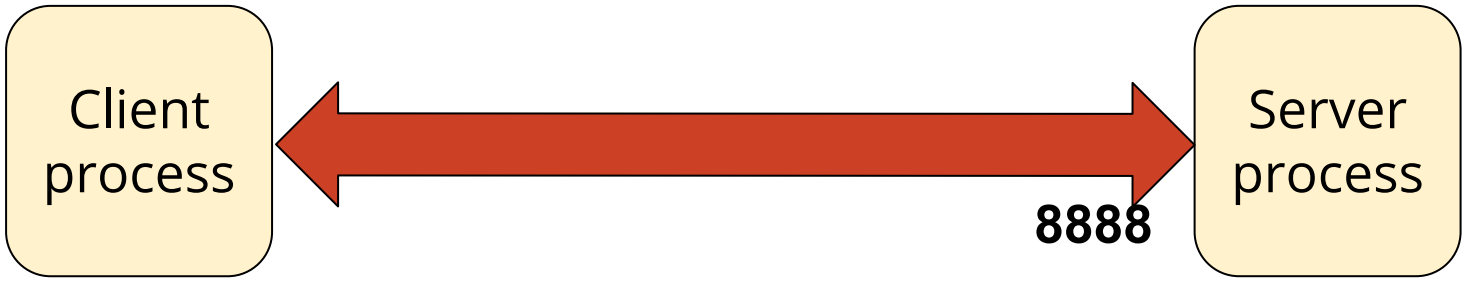
- **Attacker**: Unprivileged user, including guest
  - e.g. co-workers, family members
- **Method**: Exploit IPC from the attacker's login session
  - Fast user switching, `nohup`, remote access (SSH, remote desktop)
  - No privilege escalation required

Inter Process Communication

IPC methods

- File system
- Signal and semaphore
- **Network socket** and Unix domain socket
- Message queue
- Anonymous pipe and **named pipe**
- Shared memory
- Clipboard
- Remote Procedure Call (RPC)
- **USB** (though not strictly an IPC method)
- ...

Network sockets on localhost



- Over loopback interface: `127.0.0.1:<port>`
- Server listens on a specified port and waits for client connections
  - Only a single process can bind to a port at a time
  - Any process regardless of its owner can listen on ports > 1024
- No built-in access control

Network sockets: Client impersonation

- Any local process can connect to any server on `localhost`
- If server accepts only one client, connect before the legitimate client

Network sockets: Server impersonation

- Bind to the port before the legitimate server does

Man-in-the-Middle?

Legitimate and malicious servers cannot bind to a port at the same time

Network sockets: Man-in-the-Middle

- If primary port is taken, the server might failover to another port from a predefined list
- MitMa attacker can
  - Listen on the primary port to receive client connection
  - Connect himself to the legitimate server on the secondary port

Windows named pipe

- Client-server architecture similar to network sockets
- Named pipes placed in a special path `\\.\pipe\`
  - Every user (including guest) has access to the path
- A named pipe can have multiple **instances** that share the same name
  - Each instance connects exactly one **pipe server** and one **pipe client**
  - New pipe clients connected to the pipe servers in round-robin order

Windows named pipe: Access control

- Unlike network sockets, named pipes have a built-in DACL
- If a named pipe does not exist,
  - any user can create the first instance and set DACL of all pipe instances
- If it exists,
  - Only users with `FILE_CREATE_PIPE_INSTANCE` permission can create new instances

Windows named pipe: Client impersonation

- Any process can connect to any open pipe instance but subject to access control check

Windows named pipe: Server impersonation

- **Pipe name hijacking**: create the first instance to control the pipe's DACL
  - Set DACL to allow everyone to create new instances
- Victim process may create the second instance without noticing that another user is the owner of the pipe

Windows named pipe: Man-in-the-Middle

Client + Server impersonation =

Man-in-the-middle

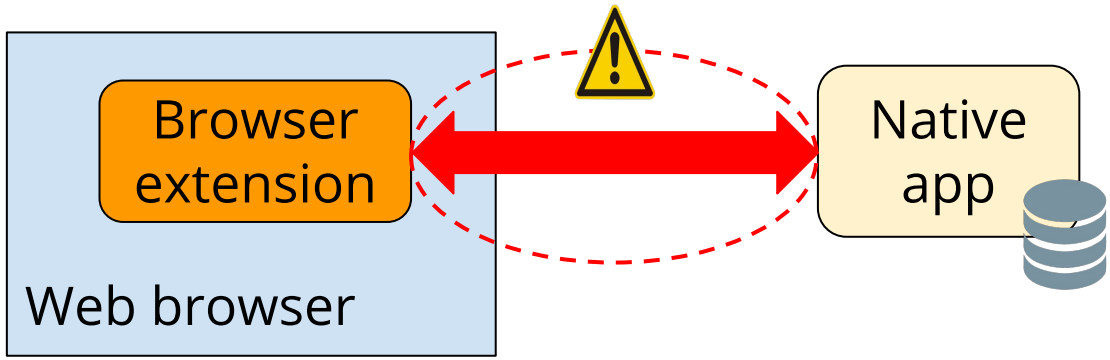
USB HID devices

- E.g. hardware security tokens
- On Linux and macOS:
  - USB HIDs can be accessed from only current interactive user session on console
- On Windows:
  - USB HIDs can be accessed from any user session, including those in the background
  - Security of the devices depends on application-level security mechanism implemented in the hardware or software

Case studies

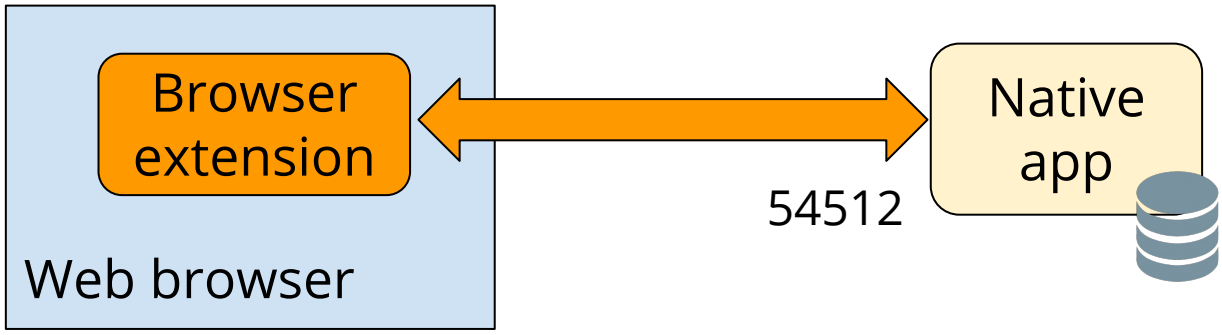
Standalone password managers

- **Native desktop app** manages the password vault
- **Browser extension** enters passwords into login pages and stores new ones in the vault
- Native app and browser extension communicate via IPC



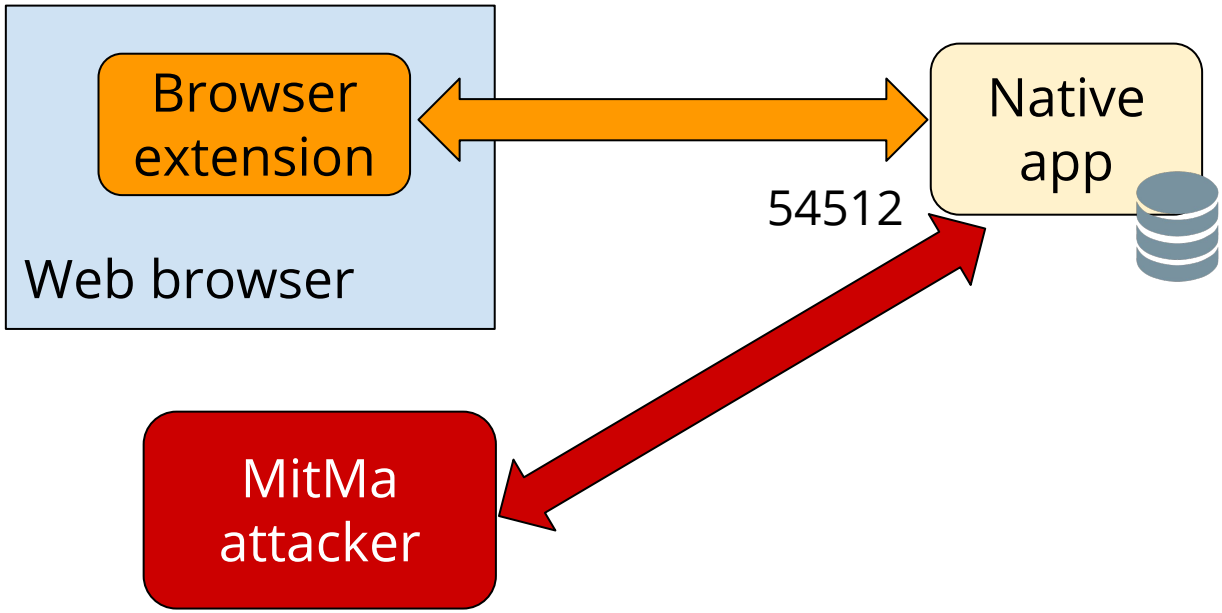
Case 1: RoboForm

- Desktop app runs a HTTP server on a port 54512
- Browser extension connects as a client to the server
- No authentication



Client impersonation on RoboForm

1. Connect to the app as a client
2. Query all passwords managed by the app



Case 2: 1Password

- Desktop app runs a WebSocket server on port 6263
- Server verifies client by checking:
  - Browser extension ID
  - Code signing signature
  - Server and client processes owned by the same user
- Client does NOT verify the server
- Server and client run a cryptographic protocol to agree on a shared key, but its ad-hoc design is insecure

1Password - Key derivation protocol

1. C → S: "hello"
2. C ← S: code (random 6-digit string)
3. C → S: hmac_key
4. Both browser extension and app display the code
5. User confirms to the app whether they match
6. C ← S: "authRegistered"
7. C → S: $nonce_C$
8. C ← S: $nonce_S$, m_S=HMAC(hmac_key, $nonce_S$||$nonce_C$)
9. C → S: m_C=HMAC(hmac_key, m_S)
10. C ← S: "welcome"
11. Both sides derive encryption key
    K=HMAC(hmac_key, m_S||m_C||"encryption")

C: Browser extension

S: Desktop app

1Password - Key derivation protocol

1. C → S: “hello”
2. C ← S: code (random 6-digit string)
3. C → S: hmac_key
4. Both br… owser extension
5. User co… sktop app
6. C ← S:
7. C → S:
8. C ← S:
9. C → S: m_c=HMAC(hmac_key, m_s)
10. C ← S: “welcome”
11. Both sides derive encryption key
    K=HMAC(hmac_key, m_s||m_c||”encryption”)

- Insecure protocol
- No server verification

✓ **Server impersonation**

Server impersonation

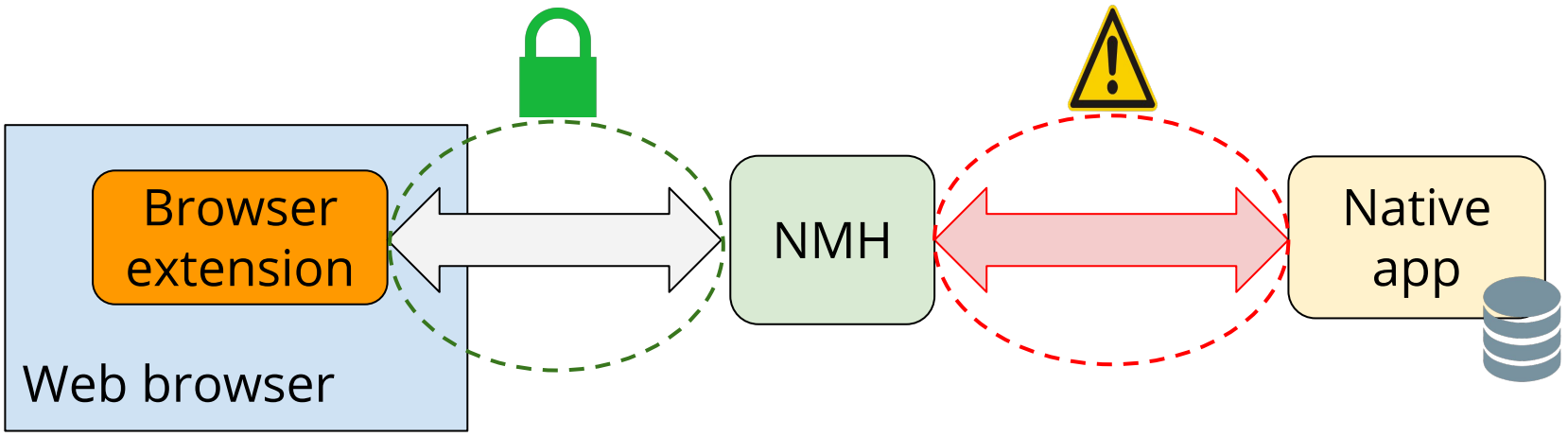
on

1Password

Demo

Password managers with Native messaging

- Native app registers with the browser an executable, called **Native Messaging Host (NMH)**
- Browser starts the NMH in a child process and lets the browser extension communicate with it
- NMH and the app use IPC to communicate with each other



Case 3: Password Boss

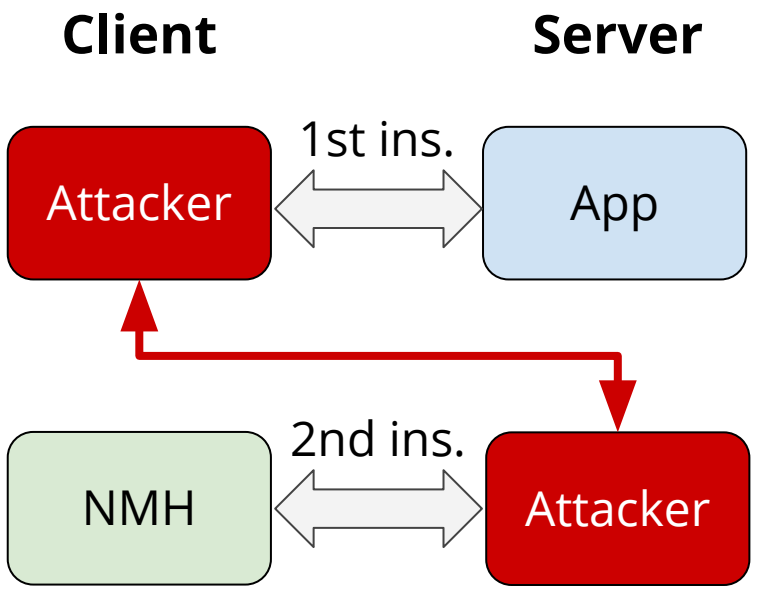
- On Windows, named pipe is used for IPC between NMH and the native app
- The app creates a named pipe that allows all authenticated to have FULL access
- NMH connects to the named pipe as a client and forwards messages between browser extension and native app
- All messages are sent in plaintext

✓ **Man-in-the-Middle**

Man-in-the-Middle on Password Boss (1)

By authenticated user:

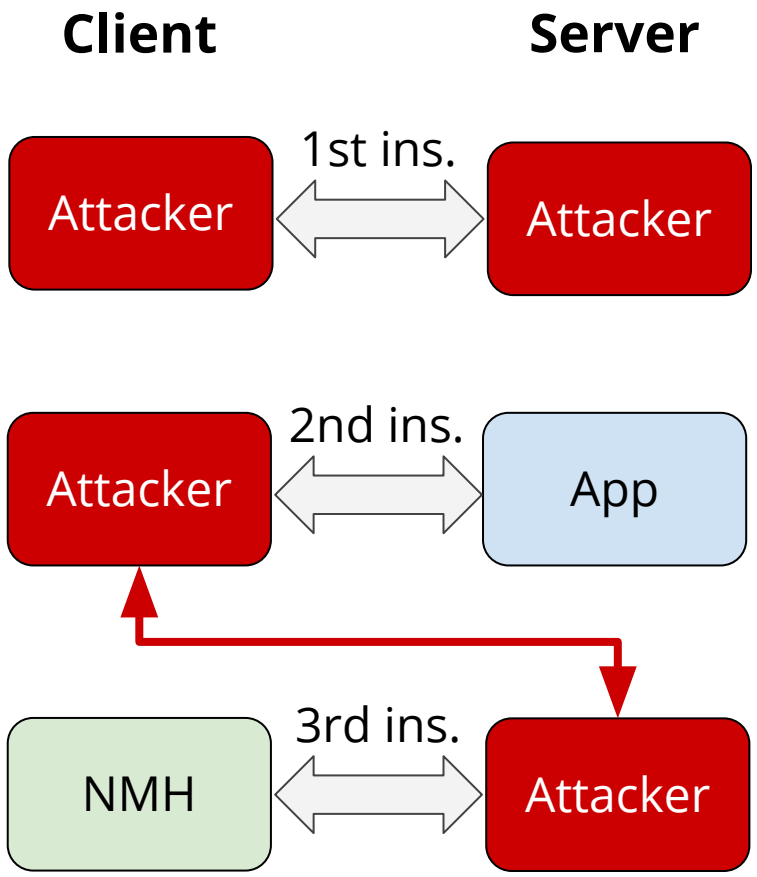
1. Connect as a client to the app's pipe instance
2. Create another instance and wait for the NMH
3. NMH connects to the attacker's instance
4. Forward messages between the two pipe instances



Man-in-the-Middle on Password Boss (2)

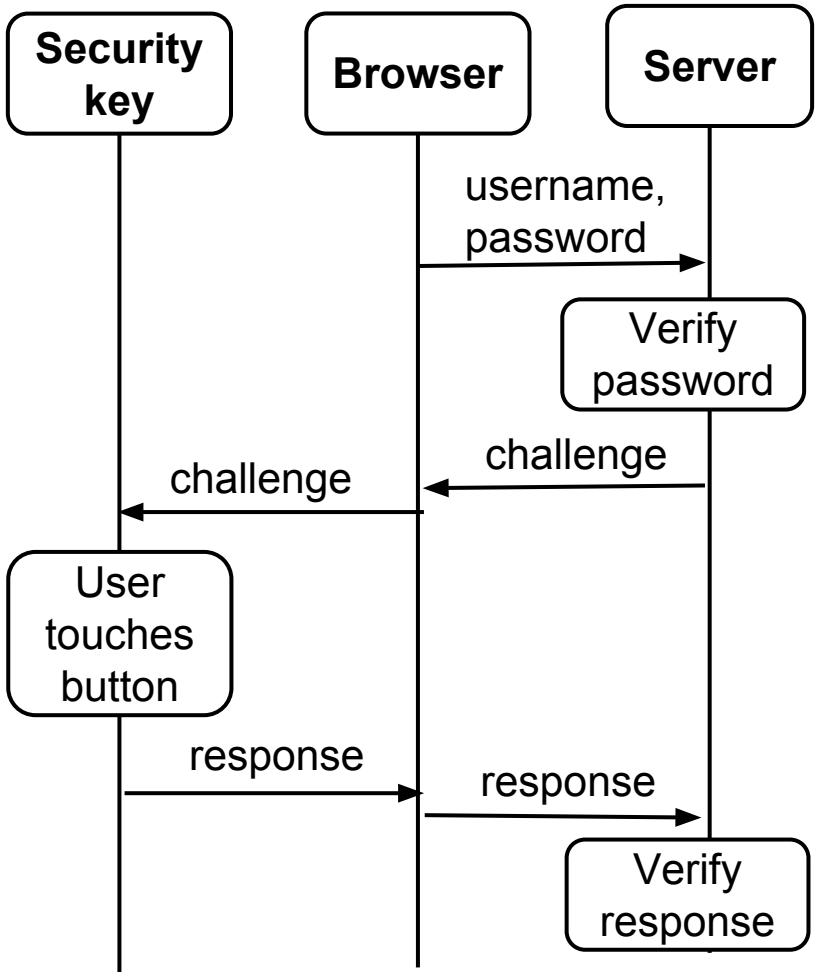
By guest:

- Guest cannot access the app's pipe instance or create a new one
- Solution: Pipe name hijacking
  - Create the first instance
  - Set FULL access to all users
- Rest is same as the attack by authenticated user



Case 4: FIDO U2F security key

- 2nd authentication factor based on public key crypto
- **Challenge-response protocol**
  - Browser keeps sending the challenge to the device
  - User activates the device by touching a button on it
  - The device responds only to the first request after the touch



Unauthorized access of FIDO U2F key

On Windows, USB HIDs can be accessed from any user session

Assumption: Attacker has obtained the 1st authentication factor

Attack steps:

1. Attacker signs in using the 1st factor and receives a challenge
2. Attacker keeps sending the challenge to the device at a high rate
3. Victim signs in to ANY service using the same security key and touches the button on the device

→ Attacker receives the response with high probability

Unauthorized access

of

FIDO U2F Security Key

Demo

Summary of vulnerabilities

Application		OS	IPC Channel	Attack
Password managers	**Roboform**	macOS	Network socket	Client imp.
	Dashlane	macOS, Windows	Network socket	Server imp.
	1Password	macOS	Network socket	Server imp.
	F-Secure Key	macOS, Windows	Network socket	Client imp. Server imp.
	Password Boss	Windows	Named pipe	MitM
	Sticky Password	macOS	Network socket	Client imp. Server imp.
Hardware tokens	**FIDO U2F Key**	Windows	USB	Unauthorized access
	DigiSign	macOS, Windows, Linux	Network socket	Client imp.
Others	**MySQL**	Windows	Named pipe	MitM
	Transmission	macOS, Windows, Linux	Network socket	Client imp.
	Spotify	macOS, Windows, Linux	Network socket	Client imp.
	Blizzard	macOS, Windows	Network socket	Client imp.
	Keybase	Windows	Named pipe	Server imp.

Mitigation

- Spatial and temporal separation of users
  - Limit the number of users that can access a computer
  - Disable remote access (SSH and Remote desktop)
- Attack detection easier in IPC than in network
  - Compare owner of client and server processes
  - Query client/server binary
- Cryptographic protection
  - User-assisted pairing vs TLS and PKI
  - Avoid self-made crypto!

Conclusion

Software developers beware, IPC is not inherently secure!

- IPC client-server architecture may be vulnerable to client and server impersonation and man-in-the-middle attacks
- Unprivileged user or process can attack IPC of other users on the same computer

Contact

Thanh Bui
Siddharth Rao
Markku Antikainen
Tuomas Aura

Read more

”Man-in-the-Machine: Exploiting ill-secured communication inside the computer”, USENIX Security 2018