Aalto University

UNIVERSITY OF HELSINKI

MAN-IN-THE-MACHINE: EXPLOIT ILL-SECURE COMMUNICATION INSIDE THE COMPUTER

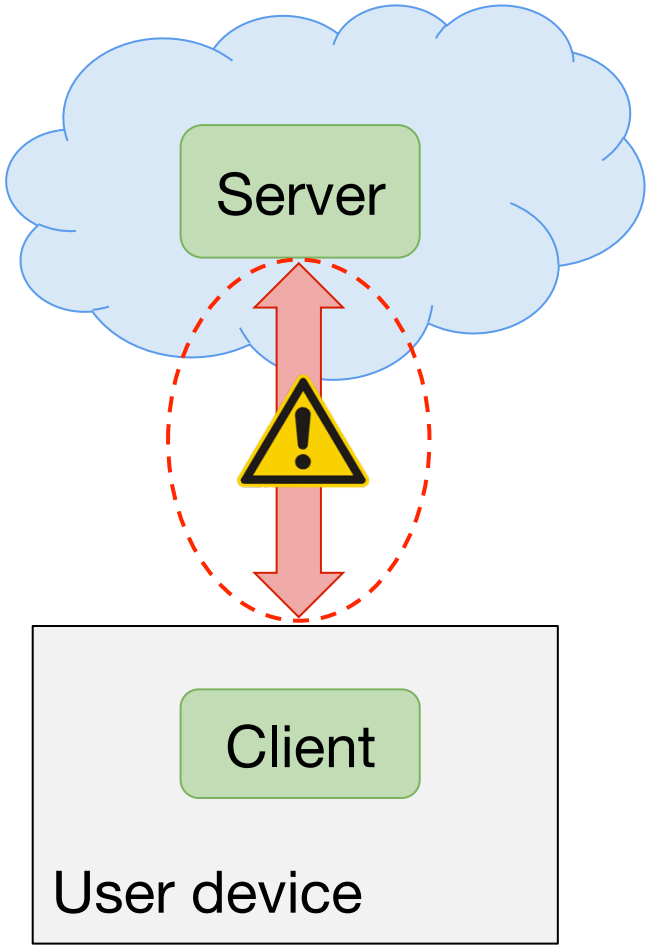
Thanh Bui*, Siddharth Rao*, Markku Antikainen†,

Viswanathan Bojan*, Tuomas Aura*

*Aalto University, Finland †University of Helsinki, Finland

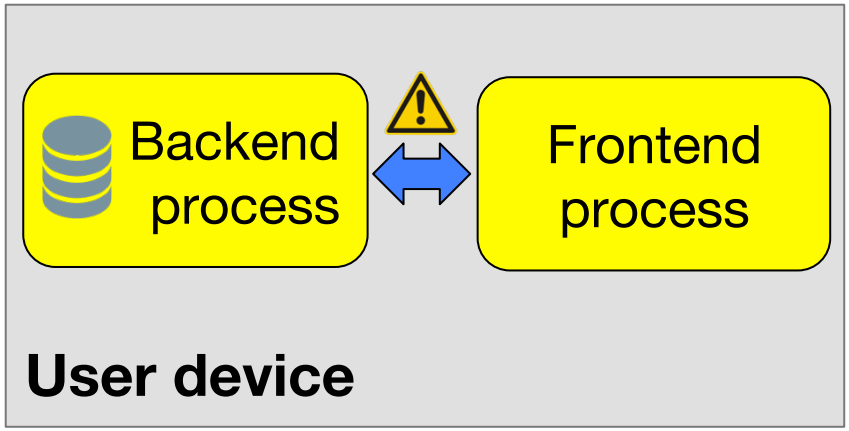
USENIX Security 2018

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)

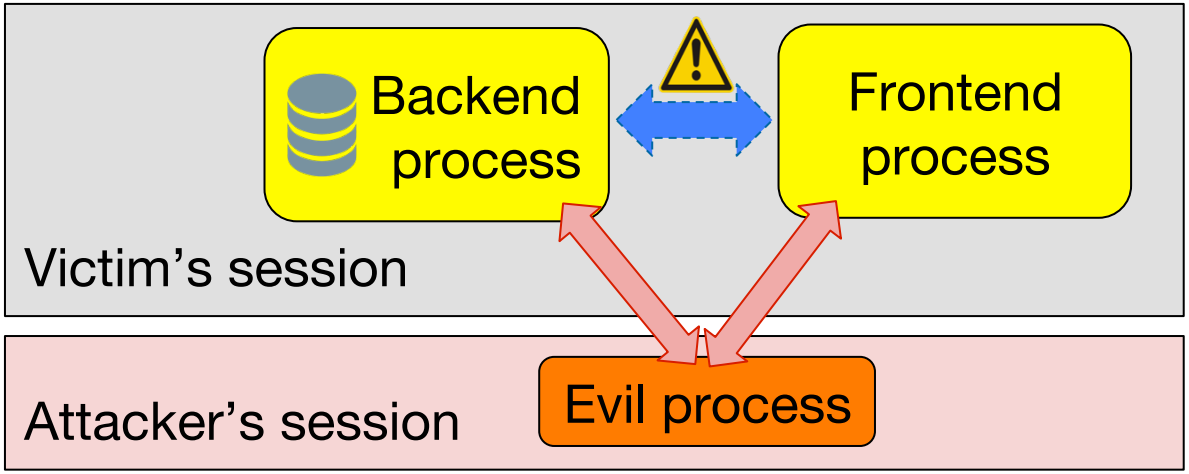


- Not all communication goes over the network
- Software consists of multiple local processes that need to communicate

We try to understand security of communication inside the computer

Man-in-the-Machine (MitMa)

Man-in-the-Machine (MitMa)



- **Attacker**: Unprivileged user, e.g. coworker, guest user
- **Target:** Multi-user computers
- **Method**: Intercept IPC from the attacker's login session
  - Fast user switching, `nohup`, remote access (SSH and remote desktop)

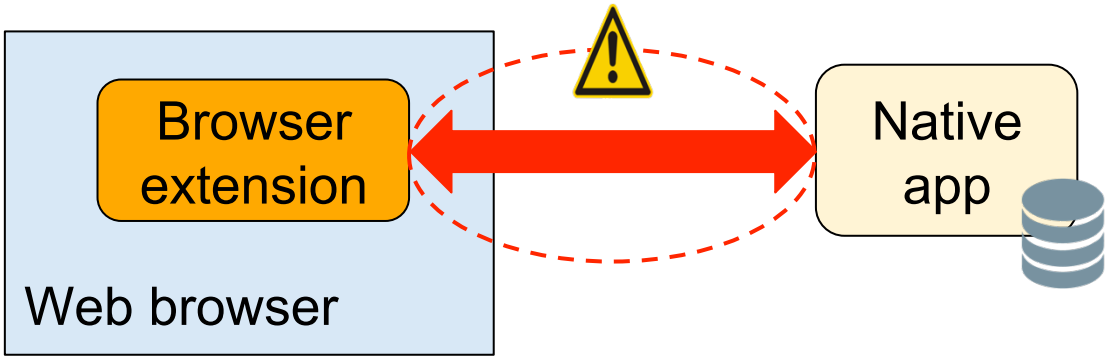
What makes IPC vulnerable

- **Vulnerable IPC methods**: Server binds to a specific identifier or name and waits for client communication
  - Client and server impersonation possible:
    - Network socket on localhost `127.0.0.1:<port>`
    - Named pipe on Windows `\\.\pipe\`
  - Unauthorized access to Windows USB HID devices (e.g. security keys)
- **Secure IPC methods**: No server waiting for clients
  - Socket pairs
  - Unnamed pipes

Case studies

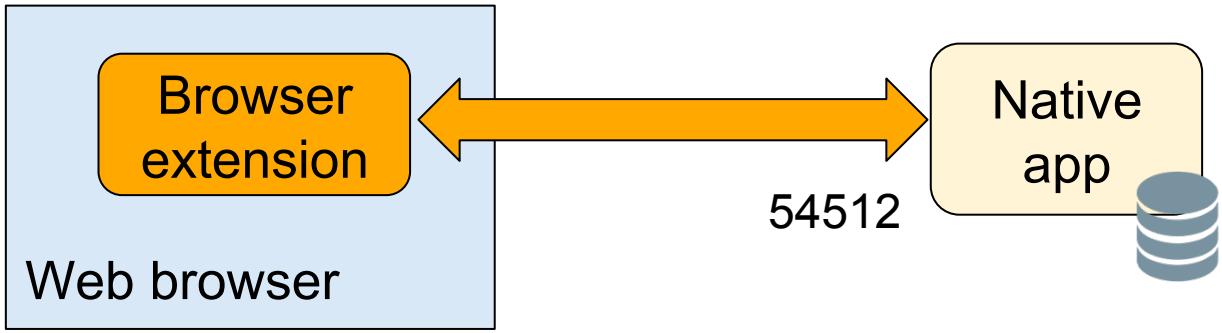
Standalone password managers

- **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



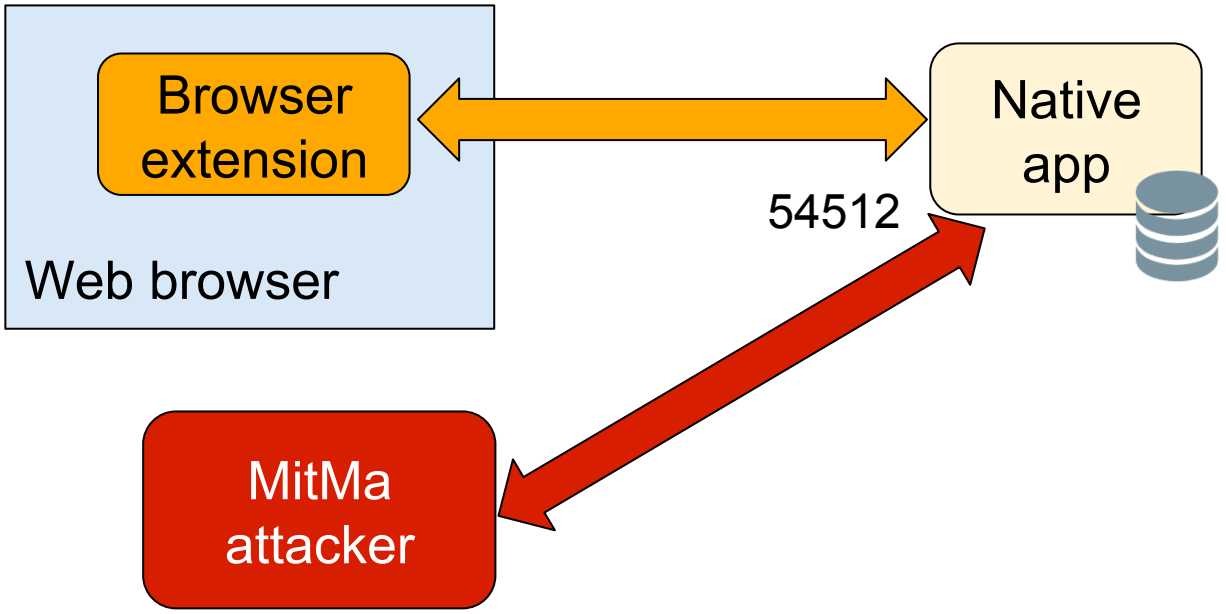
Case 1: RoboForm

- 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 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 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")

1Password - Key derivation protocol

1. C → S: "hello"
2. C ← S: code (random 6-digit string)
3. C → S: hmac_key
4. [illegible]ode
5. [illegible]
6. [illegible]
7. [illegible]
8. [illegible]nce$_C$)
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")

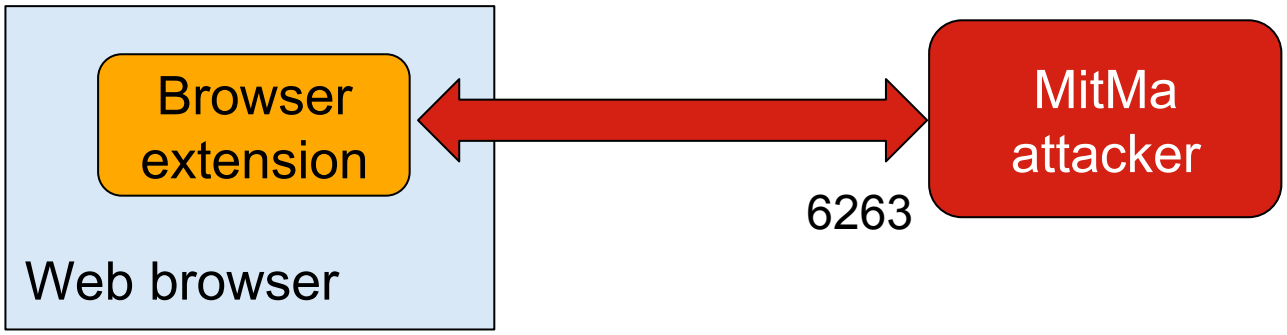
- Insecure protocol
- No server verification

✓ **Server impersonation**

Server impersonation on 1Password

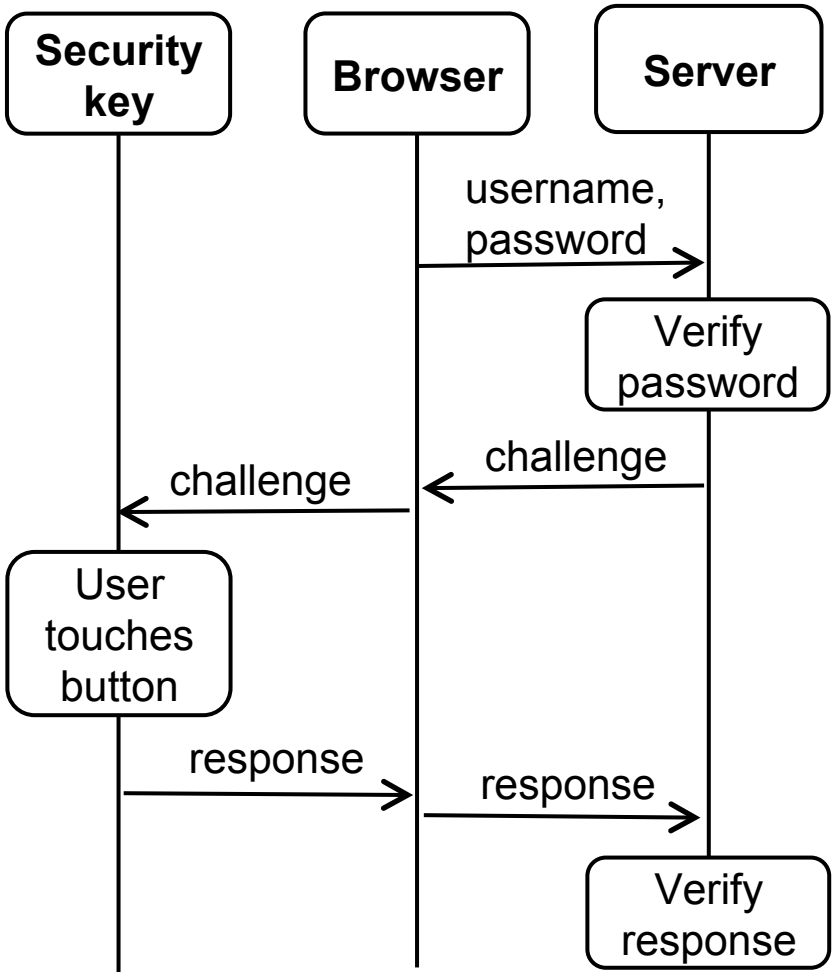
1. Run WebSocket server on port 6263, and benign server silently fails
2. Run the protocol with the browser extension but skip user confirmation
3. Send "`collectDocuments`" to the browser extension

→ Attacker obtains: web form data including login credentials



Case 3: 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 to only 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

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 have access a computer
  - Disable remote access: SSH, Remote desktop
- Attack detection easier in IPC than in network
  - Compare owner of client and server processes with OS APIs
- Cryptographic protection
  - User-assisted pairing vs TLS and PKI
  - Avoid self-made crypto!

Conclusion

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