Scientific Research Publishing

Journal of Software Engineering and Applications, 2026, 19(7), 266-299
https://www.scirp.org/journal/jsea
ISSN Online: 1945-3124
ISSN Print: 1945-3116

Why Credentials Are Not Hard to Hunt: A Critical Review of Credential Gathering Attacks

Jiya Patel[1], Peng Liu[2]

[1]School of Electrical Engineering and Computer Science, The Pennsylvania State University, State College, USA
[2]College of Information Sciences and Technology, The Pennsylvania State University, State College, USA
Email: pxl20@psu.edu

How to cite this paper: Patel, J. and Liu, P. (2026) Why Credentials Are Not Hard to Hunt: A Critical Review of Credential Gathering Attacks. *Journal of Software Engineering and Applications*, **19**, 266-299.
https://doi.org/10.4236/jsea.2026.197012

Received: June 16, 2026
Accepted: July 21, 2026
Published: July 24, 2026

Abstract

Credential theft remains one of the most persistent vectors of cyberattack because credentials are heterogeneous: different types are stored, stolen, and exploited through distinct mechanisms, yet most defenses are applied uniformly across them. This paper combines a critical review with a targeted design proposal. We first examine how major credential types, including passwords, hashes, Kerberos tickets, cookies, payment-card data, and cryptographic keys, are stored and extracted from memory, disk, and network traffic, and how attackers exploit them once stolen. We then survey existing protection mechanisms, from hardware isolation to FIDO2/WebAuthn, and identify two unresolved gaps in FIDO2: its reliance on a trustworthy browser, and the absence of a cryptographically sound account-recovery mechanism. To address these, we propose an extension combining zero-knowledge browser attestation, built on Direct Anonymous Attestation and TPM-measured platform state, with a Shamir Secret Sharing-based recovery architecture distributed across trusted contacts. We argue, and illustrate through this extension, that credential-specific defenses are more tractable and effective than general-purpose ones.

Keywords

Credential Theft, FIDO2, Browser Attestation, Shamir Secret Sharing

1. Introduction

The security of modern computing systems rests on a deceptively simple foundation: the ability to verify that a user is who they claim to be. Credentials, the tokens, secrets, and cryptographic artifacts by which identity is established, are

therefore among the most consequential assets in any networked environment. They are also among the most persistently targeted. Credential theft underlies a significant proportion of real-world cyberattacks, from targeted corporate intrusions to large-scale financial fraud, and its prevalence shows no sign of declining despite decades of defensive investment.

This paper undertakes a systematic examination of credential gathering attacks, with the dual aim of understanding how and why credential theft succeeds at such scale, and of identifying where current defenses fall short. The central argument motivating this work is that credential theft is not a monolithic problem. Different credential types, ranging from Windows NTLM hashes and Kerberos tickets to payment card magnetic stripe data and browser-stored passwords, are stored in different locations, stolen through different techniques, and exploited in different ways. Yet prevailing defense strategies tend to treat them uniformly. This mismatch between the heterogeneity of the attack surface and the uniformity of the defense is a significant and underappreciated gap in the field.

The paper is organized into three parts. Part 1 establishes the technical foundation by surveying the major credential types in active use, the storage mechanisms through which they are held, the techniques by which malware locates and extracts them, and the ways in which attackers leverage stolen credentials once obtained. Part 2 surveys existing protection mechanisms, evaluating their effectiveness and identifying their limitations, with particular attention to the FIDO2 authentication standard, which represents the most significant recent advancement in credential security. Part 3 builds on this analysis by proposing a targeted extension to FIDO2 that addresses two specific, previously unresolved limitations: the vulnerability of the FIDO2 authentication flow to a compromised browser, and the absence of a cryptographically sound account recovery mechanism.

This work does not attempt a universal defense against all credential theft. Such a defense does not exist, and claiming otherwise would misrepresent the nature of the problem. Instead, the paper demonstrates through careful analysis that targeted, credential-specific defenses, grounded in a precise understanding of how each credential type is stored and stolen, are both more tractable and more effective than general-purpose approaches. The FIDO2 extension developed in the last section is offered as one such targeted defense, and as an illustration of the methodology this paper advocates.

2. Understanding Credentials in Security Contexts

For the purposes of this paper, a credential is any artifact, secret, or token that a system accepts as proof of identity or authorization, whether possessed by a user (a password), issued by a system (a Kerberos ticket or OAuth token), or physically encoded (payment card track data). A credential gathering attack refers to any technique by which an adversary obtains such an artifact without legitimate authorization, whether through extraction from memory or disk, interception in transit, or social engineering. This paper treats these credential types within a sin-

gle scope because, despite their structural differences, they share a common attack lifecycle: storage in a vulnerable location, extraction by an adversary, and reuse or resale for unauthorized access. This section establishes the technical foundation for the remainder of the paper by examining three interconnected dimensions of credential security: the diverse types of credentials in active use, the locations and mechanisms through which they are stored, and the techniques by which malicious actors hunt and extract them. Understanding these dimensions in depth is a prerequisite for evaluating the adequacy of existing defenses, which is the subject of the next section.

2.1. Types of Credentials

2.1.1. Username and Password Combinations

These are the most common form of authentication credentials used in everyday life, which is why they are also highly targeted by the attackers' malware. A username is a unique identifier of the user's account for a particular service/application. A password is something only the user knows ideally, which can be a sequence of letters (lowercase and uppercase), numbers, and special characters. A strong password is not too short and is a combination of all of them and is difficult to guess.

2.1.2. Multi-Factor Authentication Credentials

Multi-factor authentication (MFA) is an additional layer of verification after the password. This is a One-Time Password (OTP) generated by authenticator application or sent via SMS or email, hardware token codes (a physical device that generates OTPs), and biometric data (like fingerprint or retina scan). Another form of MFA is the security questions and their answers.

2.1.3. Cookies

Cookies are shared by banking or web servers with the authenticated clients so that there is no need of authenticating the same clients at every use. For example, if you are on a shopping website, you don't want to keep logging in every time you click on an object. In that case you are authenticated through Cookies. Malware prefer targeting these because once they get a hold of these, they can login as a legitimate user easily. Some cookies carry protections that limit this exposure: cookies marked HttpOnly cannot be read by client-side JavaScript, which mitigates theft via cross-site scripting, though this does not prevent extraction by malware with direct access to browser memory or disk. The Secure flag further limits a cookie to HTTPS transmission, reducing exposure to network interception.

2.1.4. OAuth Tokens and SAML Assertions

OAuth Tokens allow user to give an application limited access to their accounts in another service without sharing password. SAML Assertions are digital certificates mostly used in enterprises for Single Sign On (SSO) across multiple internal systems. For example, if you login to your PSU Canvas, you don't have to login again to check Outlook emails or open Starfish.

2.1.5. Linux Password Hashes

Linux stores hashes of its passwords in its systems. Modern Linux distributions use various hashing algorithms, with yescrypt being the current default for major distributions like Debian 11, Fedora 35+, and Ubuntu 22.04+, though SHA-512, SHA-256, and MD5 are also supported [1].

2.1.6. Windows NT Hash and LM Hash

Just as Linux Password Hashes, Windows also stores password as the LAN Manager (LM) hash and the Windows NT hash (NT hash). The Windows NT hash is based on the Unicode character set, is case-sensitive, and can be up to 128 characters long. It is computed using the MD4 hash function, which computes a 16-byte digest of the clear-text password [2]. The LM hash is relatively weak and prone to fast brute force attacks, which is why the NT hash is preferred for security [3].

For network logins, the client is given a 16-byte challenge and computes a challenge response by encrypting it with the 16-byte Windows NT hash. The server authenticates the user by querying the Windows NT hash from the database, encrypting its own challenge using that stored hash, then comparing it with the client's challenge response to verify identity without the need to share passwords [2]. For interactive logins, the clear-text password is converted to the NT hash and compared directly against the stored hash in the database (similar to Linux Password Hashes) [2]. After a user changes their password using NTLM, the old NT hash can still be used for network access for a configurable time period, defaulting to five minutes, to allow background services to continue running [4]. Because NTLM hashes can be used directly for authentication without knowledge of the underlying plaintext password, their theft enables pass-the-hash attacks, discussed in Section 2.4.2, in which an attacker authenticates as a legitimate user without ever cracking the credential.

2.1.7. Windows Hello PIN

When one sets up a Windows PC, one uses 4 - 6 digit PIN instead of Microsoft password, that is the Windows Hello PIN. It is a single-factor authentication that is used instead of passwords. Unlike a password, the PIN is not transmitted to or compared against a remote server. It is verified locally and used to unlock a private key that is generated and held inside the device's TPM. This means a stolen PIN is useless without the specific physical device it was set up on, and the TPM additionally enforces anti-hammering lockouts that limit the number of incorrect PIN attempts, making brute-force guessing impractical even with physical access to the device.

2.1.8. Kerberos Tickets

A Kerberos ticket is an encrypted authentication credential that proves a user's identity to a service providing server without transmitting passwords over the network. This involves two tickets which are the Ticket-Granting Ticket (TGT) and service ticket. First, the user logs in and interacts with the Authentication Server (AS) which verifies the user and generates a TGT which contains a session key encrypted by the Ticket Granting Server's key along with another copy of session

key encrypted for the user. Next, the user interacts with the Ticket Granting Server (TGS) to get the specific service ticket to interact with the respective server. The server ticket issued by TGS is encrypted by that server's key and it contains identity information, network address, validity period, and a new client-server session key. The TGS also shares the client-server session key with the user. Finally, when the user sends this server ticket along with its identity information, the server verifies the information given by user with the information decrypted from the server ticket to authenticate the user and fulfill the service. Because Kerberos tickets grant authenticated access to network services for the duration of their validity period, their theft enables pass-the-ticket attacks and, in the case of the KRBTGT account, domain-wide Golden Ticket attacks, both examined in detail in Sections 2.4.2 and 2.4.7.

2.1.9. Payment Card Information

Payment card information, governed by PCI-DSS (Payment Card Industry Data Security Standard), consists of two main categories, which are, cardholder data and sensitive authentication data ([5], p.~4). The cardholder data includes the Primary Account Number (PAN), cardholder name, expiration date, and service code (3 digits in the magnetic stripe which indicates the usage restrictions of the card). The Sensitive Authentication data includes the verification codes (CVV/CVV2/CVC2/CID) or PIN Verification Value (which the user enters in the chip-and-pin transactions).

Payment card has a magnetic stripe and the data is stored on it in two tracks with overlapping but differently formatted information. Track 1 contains the cardholder's name (2 - 26 characters), PAN (16 - 19 digits), expiration date in "YYMM" format, service code (3 digits), and discretionary data that may include card verification codes (CVV/CVV2/CVC2/CID) or PIN Verification Value ([5], p.~4; [6], p.~6). Track 2 contains similar data in a shorter, BCD (Binary-Coded Decimal) encoded format for faster processing by older terminals, including the PAN, expiration date, service code, and discretionary data, but excludes the cardholder name ([6], pp.~6-7). The plaintext availability of this data during transaction processing creates the vulnerability exploited by POS RAM scraping malware, the earliest and most extensively documented form of credential gathering malware, examined in Section 2.3.1.

2.1.10. Cryptographic Keys and Certificates

Digital certificates and their associated private keys provide strong authentication for secure communications and transactions. These credentials are used in SSL/TLS connections, code signing, email encryption, and secure banking authentication.

2.2. Storage Mechanisms for Credentials

2.2.1. System Memory (RAM)

All of the credentials mentioned in the previous section are stored in RAM during

transaction and authentication processes, making it a prime target for various malware. Credentials must temporarily exist in plaintext in RAM for processing, creating a window of opportunity for malware exploitation regardless of network-level encryption ([6], p.~4).

• Point-of-Sale System Memory

The Payment Card data, specifically the magnetic stripe data, is stored in plaintext in the Point-of-Sale (POS) system RAM during transaction processing. Windows-based POS systems, which "represent the 88% of current POS systems" ([6], p.~4), are the primary targets, with malware specifically looking for payment-related data.

The fundamental vulnerability stems from the requirement that "data manipulations are carried out by the payment application when processing an authorization or a settlement, and thus, payment card data remain in memory of the processing machine" ([6], p.~4).

• Browser Process Memory

When users log into their online (banking) accounts, the usernames, passwords, and second-factor authentication codes (MFA) are temporarily stored in browser memory, a portion of system RAM allocated to the web browser process [7]. Just like in POS, credentials are stored here in plaintext.

• LSASS (Local Security Authority Subsystem Service) Process Memory

The Local Security Authority Subsystem Service (LSASS.exe) is a critical Windows system process and it holds passwords (in some cases), Windows hashes, Windows Hello PIN, and Kerberos tickets in LSASS process memory [8] [9]. "This critical system process enforces security policies on Windows machines. LSASS is responsible for performing essential tasks related to security such as verifying user logins, managing passwords, and creating access tokens." [8] According to Ah-Fat *et al.*, credentials are cached locally inside LSASS process memory when a user performs an authenticated action on that machine, such as logging on using Remote Desktop, scheduling a task, or executing a process using "RunAs" [8].

Keys and credentials held in LSASS are attractive to attackers as they enable them to perform operations such as decrypting sensitive data or impersonating users that have a higher level of privilege on the domain [8]. As such, this process contains a considerable amount of very sensitive data in its memory, making LSASS a prime target on Windows to any internal attacker that seeks to steal credentials [8]. Tools like Mimikatz specifically target LSASS to extract these credentials [9]. The concentration of multiple high-value credential types in a single process makes LSASS the single highest-value target for credential extraction on Windows systems, and the primary target of the extraction techniques examined in Section 2.3.4.

2.2.2. Disk Storage

All the credentials mentioned in the previous section are stored temporarily or for a longer period of time in disks. So, just like malware targets RAM, it also targets Disk Storage of various end-point devices of network.

Attackers have reportedly "stolen full databases containing sensitive credit card data in several PoS system breaches" ([10], p.~13). The TJX Companies breach of 2008 exemplifies this, where "a weak encryption scheme used in the wireless network of some stores was successfully exploited to gain access to the internal network. From there, hackers obtained access to company servers that stored customer card details. It is estimated more than 40 million of credit-card records were actually stolen" ([6], p.~2). This demonstrates that credentials stored on disk remain vulnerable even after the initial transaction or authentication event has completed.

• Windows Security Accounts Manager (SAM) Database

Windows NT hashes and LM hashes are stored in the Security Accounts Manager (SAM) database on local Windows systems at the file location C:\windows\system32\config\SAM [2] [3] [11]. The SAM database maintains local user account information and their corresponding password hashes. Upon successful authentication, the workstation loads the password hash into memory and into the SAM file [11]. By default, the password hash is locked into memory, and a direct attempt to read it will lead to a Blue Screen of Death (BSOD) [11]. This database is a primary target for offline password attacks when attackers gain access to system files. By default, Windows systems cache the last 10 password hashes for authentication purposes, which remain accessible to attackers who compromise the system.

• Windows Active Directory Database

In domain environments, NT hashes, LM hashes, Kerberos tickets, and the critical KRBTGT account credentials are stored in the Active Directory (AD) database on domain controllers [2] [3] [9]. The KRBTGT account is used for encrypting all Kerberos authentication tokens for the domain controller, and the password and username for this account never changes [9]. Compromising this account allows attackers to create Kerberos Golden Tickets that provide access to all AD-connected systems, making it one of the most dangerous abilities available to post-exploitation tools [9].

• Linux Shadow File

Linux password hashes are stored in the /etc/shadow file, which holds the hashed passwords of users listed in /etc/passwd [1]. The password field in /etc/shadow uses a dollar-delimited format where the initial characters identify the encryption algorithm, followed by optional parameters, a salt value, and the actual hashed password [1]. A salt is a random supplemental value used during the hashing process to introduce complexity, reducing the odds of successful cracking [1].

When a user attempts to log in, the system applies the same hashing algorithm and salt specified in their /etc/shadow entry to the password, and compares the resulting hash with the stored one. The /etc/shadow file is readable only by root, making it significantly more secure than the older practice of storing password hashes in the world-readable /etc/passwd file [1].

While the restricted permissions provide some protection, attackers who gain root access or exploit privilege escalation vulnerabilities can access this file to extract password hashes for offline cracking attempts. The shadow file's security relies on proper system hardening and access controls, making it vulnerable when these protections are bypassed.

• Browser Credential Stores

Web browsers have credential stores where they store encrypted usernames and passwords associated with websites, including banking portals. On Windows systems, browsers use the Data Protection API (DPAPI) to encrypt stored usernames and passwords. However, malware running with user-level privileges can use the same DPAPI functions to decrypt these stored credentials. Banking Trojans often target browser credential stores because they contain high-value financial authentication information that can be extracted without requiring real-time interception during login sessions.

• Windows Credential Manager and LSA Secrets

Windows Credential Manager stores username, passwords, and digital certificates. LSA (Local Security Authority) Secrets, which is a protected registry location in Windows that stores sensitive information needed by the operating system, stores cryptographic keys, service account passwords, and digital certificates. Both of these storage locations are accessible through tools like Mimikatz, which can extract credentials even when they are meant to be protected by the operating system [9]. The lsadump module in Mimikatz is specifically designed to access these credential stores for privilege escalation and lateral movement attacks [9].

2.3. Credential Hunting Techniques

Malware looks for credentials in one of the places listed in the previous section. Any credential data is accessed during three key vulnerability points: "1) in memory: data manipulations are carried out by the payment application when processing an authorization or a settlement, and thus, payment card data remain in memory of the processing machine; 2) at rest, when the payment application stores data, either temporarily or for long term, on a disk device; or 3) in transit, when payment data are received and sent to and from other application and devices within the system" ([6], p.~4).

2.3.1. POS RAM Scraping Technique

Point-of-Sale (PoS) RAM scraping malware specifically targets the magnetic stripe data of payment card information as it temporarily resides in system memory during transaction processing [6] [10], so it exploits the first of the three vulnerable points mentioned above. In this technique, attacker is "retrieving the list of processes on execution. Each process is then discarded or selected (depending on the type of search performed). For each selected process, its allocated memory is scrapped looking for memory patterns that match card data format" ([10], p.~3). The key here is selecting the processes whose RAM can have card data. Attacker uses blacklisting and whitelisting for it, where blacklist

has processes that are well known to not contain track data, and whitelist has processes that are known to contain track data [10]. As mentioned earlier, card data is present as Track 1 and Track 2 data which follow a certain format. After segregating the processes, regular expressions (regex) are used to look for track data patterns [10]. A lot of the malwares also use custom algorithms to search for the track data patterns in the processes' memories [6]. According to Rodríguez, two specific RAM scraping malwares LogPOS and BernhardPOS are stealthier because they insert the scraping function directly into the target process so that the search function is performed from the own process' memory space which helps it escape the process monitoring tools [6]. Once the data is found, attacker "performs a Luhn algorithm check to validate if the data that matched the regular expression are valid card numbers" ([10], p.~3). Luhn Algorithm, also known as "mod 10 algorithm" is a formula used to check identification numbers, since most credit card numbers and government identification numbers use this algorithm to distinguish between valid numbers and mistyped or incorrect numbers [12].

"Historically, the earliest evidence of POS RAM scraping was reported in a Visa Data Security Alert issued on 2 October 2008" ([6], p.~1), when cybercriminals attempted to install debugging tools on POS systems to dump Track 1 and Track 2 credit card data from RAM. The threat has only intensified, with the "biggest data leakages occurred at 2013 and 2014 were caused by POS RAM scraping malware" ([6], p.~3), notably the "BlackPOS malware…behind data breach occurred at Target in 2013, which exposed 40 million of customer's credit and debit card information in only three weeks" ([6], p.~3).

2.3.2. Web Injection Techniques

Trojans can be used to steal the usernames, passwords, and MFA a user may use to authenticate themselves on a webpage. This is done using Trojans equipped with a web inject functionality, that silently modifies webpage on infected device which lets them quietly intercept the login information the user types in [7]. Here are the different web injection techniques:

1) Form Grabbers: According to Kiwia *et al.*, the attacker can use a malware called form grabber which sits between the browser rendering engine, which is responsible for displaying the webpage and its interaction with the user, and the API function, which is responsible for encrypting the data collected from the webpage before sending it to the server [7]. When a user enters their username/password/MFA, it is stored in the memory allocated to the browser in plaintext until the API function encrypts it, and that is when form grabber steals it.

2) API Hooking: Another way to inject malware to steal username/password/MFA is through API hooking. "API hooking is a technique by which the behavior and flow of applications can be influenced and modified through inserting memory break point and JMP (jump) instructions ([7], p.~8)." In this case, whenever API functions are called, the code is redirected to some malicious code which runs and intercepts the user data before the API function can encrypt it.

3) Keystroke Logging: "In Keystroke logging an adversary covertly records user's keystroke as they are being typed either through a software program or a hardware device or even by monitoring electromagnetic emissions ([7], p.~7)." Adversary inserts a malware called keylogger by patching memory tables or injecting directly into the process memory using Direct Memory Access (DMA) [7]. Only user level privilege is needed to do this, so one successful phishing attack and the system becomes prone to keystroke logging.

Beyond immediate theft, cybercriminals employ Man-in-the-Browser (MITB) and Man-in-the-Middle (MITM) techniques, where "in MITB a Trojan redirects the victim to a phishing website that is controlled by the adversaries to harvest credentials," while "in MITM the adversary goes further by intercepting communications from victims and responses from server and establishing an interactive process for collecting users data" ([7], p.~11).

2.3.3. Password Hash Extraction

Retrieving password hashes can be done through different approaches [11]. Free applications are available for download on the Internet that any novice could use to obtain password hashes if they have local administrator privileges on the machine, but if the local machine has antivirus software, it is very likely that downloaded applications will be blocked or deleted by the antivirus, which is why professionals use scripts to accomplish hash extraction [11]. Scripts have the same functionalities as applications but are very hard to detect or block by antivirus software since these are simple commands that do not harm the operating system or delete files, making them very difficult to detect even by antivirus heuristic scanners or monitors [11]. Code is executed in a split second and hashes become available to the attacker [11]. From this point forward, the attacker possesses the password hashes of all users who authenticated on the workstation since the last reboot, including the local administrator and local system account [11].

2.3.4. LSASS Memory Extraction

Tools like Mimikatz are used to steal passwords, NTLM hashes, and Kerberos tickets from LSASS. Mimikatz is an open source tool that can extract plaintext passwords, hashes, PIN codes, and Kerberos tickets from memory [9]. According to Ah-Fat *et al.*, Mimikatz performs multiple cross-process reads of the LSASS address space using Windows API calls such as ReadProcessMemory [8]. To access LSASS memory, attackers need debugger privileges, which allow the tool to interact with specific processes [9]. The specific Mimikatz command used to steal passwords from active logon sessions is sekurlsa::logonpasswords [8]. Malicious tools must perform multiple reads because the LSASS address space layout is dynamic and unpredictable, making it impossible to know in advance where sensitive information resides [8]. To overcome this challenge, Mimikatz uses fixed reference points like the Process Environment Block (PEB) and follows pointers through memory structures to locate passwords, NTLM hashes, and Kerberos tickets [8].

The extraction process works as follows:

1) Mimikatz first queries the PEB to find where msv1_0.dll is loaded in memory. This DLL is the Windows authentication package that handles NTLM authentication and stores credential information for logged-in users.

2) Mimikatz then reads the entire msv1_0.dll into its own memory buffer and searches for specific "magic bytes", hardcoded byte sequences that appear in predictable locations within the DLL's data structures [8]. These magic bytes serve as landmarks, one set is located near a pointer to the logon session list, while another set is located near the AES encryption key and Initialization Vector (IV) stored in LSASS memory [8].

3) Once the logon session list pointer is found, Mimikatz follows it to access each active user session. For each session, it performs individual reads to extract metadata such as username, domain, and SID, along with a pointer indicating where that user's credentials (passwords, NTLM hashes, Kerberos tickets) are stored [8].

4) Following this credential pointer, Mimikatz performs additional reads to pull various data structures from LSASS memory, with the final structure containing the encrypted passwords, NTLM hashes, and Kerberos tickets stored as a Unicode string [8]. Finally, Mimikatz uses the AES key and IV retrieved earlier to decrypt this credential material, acquiring the plaintext passwords and authentication tokens [8].

2.3.5. Network Interception

According to Huq, credit card details can be stolen when sent over LAN/WAN since PCI DSS does not mandate it being encrypted [10]. Only data transmitted through public network needs to be encrypted. This can lead to network sniffing attacks. Encrypting all data transmitted over networks can prevent this [10]. Similarly, banking credentials and cookies can be intercepted during transit if proper encryption is not implemented across all network segments. Man-in-the-middle attacks can capture credentials as they traverse internal networks, particularly in environments where TLS/SSL is not enforced for internal traffic or where certificate validation is not properly implemented.

2.4. Applications of Compromised Credentials

The following sections examine how attackers leverage credentials once obtained.

2.4.1. Exfiltration Methods

Once credentials are harvested from compromised systems, they have lost their protection, but the attackers still need to transfer them to themselves, in other words, they need to be exfiltrated to the attackers. The exfiltration methods vary depending on the type of malware and the operational security needs of the attackers. Here are two examples of exfiltration methods.

• Payment Card Data Exfiltration

Once harvested from infected POS systems, stolen card data is exfiltrated

through "eight categories" of techniques ([10], p.~6), which are:

1) Direct Exfiltration to C&C servers via HTTP POST/GET/Header requests: C&C (Command & Control) servers are controlled by attackers and they are used for data exfiltration by making it look like a HTTP server. So, the compromised system where the Payment Card details are stolen from sends data to C&C servers camouflaged as a HTTP request, and the C&C servers respond back letting it know to send the next batch or stop sending.

2) FTP servers: The payment card details from the compromised systems is stored in files and sent to FTP(File Transfer Protocol) servers that are in control of attackers.

3) Compromised or fake hosted websites: The payment card details from the compromised systems can be posted on compromised or fake hosted websites for attackers to collect it from there.

4) Email: The payment card details from the compromised systems can be emailed to attacker controlled accounts which would look like regular business emails to an outsider.

5) Tor network circuits: The payment card details from the compromised system is routed through The Onion Router (Tor) network so that it can not be traced back to the attacker.

6) RDP/backdoor access: RDP (Remote Desktop Protocol) connections or backdoor access to the compromised system can be used by attacker to retrieve payment card details from the compromised system.

7) Manual removal by insiders: An insider can physically obtain payment card details from the compromised system and hand it to the attacker.

8) Dumping to USB devices: Payment card details can be physically copied into a USB and obtained by attacker.

In cases involving millions of credit card numbers such as "Target" breaches, "attackers use multistage data exfiltration" ([10], p.~6), where they use an already-compromised system on the LAN/WAN as an internal dump server for large batches of stolen credit card data, then upload the data from dump servers to remote FTP servers and delete all traces. Some older POS RAM scraper families "did not have data-exfiltration functionality. Instead, they dropped the data gathered into text files that were then manually removed or remotely collected" ([10], pp.~5-6). Specifically, "rdasrv, GetmypassPOS, and MalumPOS" generate "a file inside the compromised machine with the scrapped data" ([10], p.~18), requiring additional methods for exfiltration.

• Exfiltration after Web Injection

Once banking credentials (username, password, MFA) are captured through web injection, they are exfiltrated to cybercriminals through some of the methods discussed in the previous section:

1) Command and Control (C&C) Infrastructure: The stolen credentials are "saved into a file and sent back to the criminals using a command & control server over HTTP" ([7], p.~10). Banking Trojans commonly use HTTP-based backdoors

for this exfiltration, with "112 out of 127 Trojan samples" utilizing "backdoors for C&C communication via GET/POST HTTP requests" ([7], p.~10).

2) Encryption: More sophisticated variants employ encryption to evade detection, as "banking Trojans like Carberp had gone to a level of implementing randomly generated cryptographic cipher to encrypt data obtained from the victim while sending back to its C&C" ([7], p.~10), ensuring that stolen credentials (banking details such as username, password, and MFA) bypass security controls such as data loss prevention devices.

These exfiltration methods are not only used to obtain different credentials such as payment card details, usernames, passwords, and MFA as mentioned above, these can also be used to exfiltrate sensitive data from file shares and network resources that were previously inaccessible [9]. The combination of access to certain compromised workstations and lateral movement capabilities allows attackers to systematically access file servers, databases, and other data repositories across the compromised network, extracting valuable intellectual property, financial information, and personally identifiable information.

2.4.2. Pass-the-Hash and Pass-the-Ticket Attacks

To access a resource in a workstation or a server, it is sufficient to know the password for a specific user, but if non-interactive authentication is used to access this resource, it is enough to know the password hash of a specific user rather than the plaintext password itself [11]. That means only knowing password hashes such as NTLM hash and Kerberos tickets is sufficient to perform authentication attacks that do not require plaintext passwords.

Pass-the-hash attacks involve extracting the Windows NTLM hash string from a target and using it to log in as that user [9]. Attackers use this technique because they do not need to crack passwords and only need the hash string for authentication [9]. Similarly, pass-the-ticket attacks involve extracting a user's Kerberos ticket and using it for authentication, working essentially the same as pass-the-hash but for newer versions of Windows [9]. These attacks allow attackers to authenticate to network resources while bypassing traditional password-based security controls, making detection more challenging.

Single sign-on (SSO) mechanisms that bring significant benefits to the user experience also increase the risk of pass-the-hash attacks if an operating system is compromised, as credentials must be stored or cached to allow the operating system to perform actions on behalf of the user [11].

2.4.3. Network Scanning and Harvesting

This is done once the password hashes are hunted. After obtaining password hashes, the attacker initiates a new session in the context of the user whose password hash has been stolen [11]. The next step is what is called "harvesting", the entire network is scanned for workstations and servers where the compromised user might have access [11]. Depending on the purpose of the attack (data theft or infrastructure damage), the attacker's subsequent steps may vary [11]. If the

purpose is data theft, the attacker performs lateral movement across the network, harvesting as many workstations as possible and collecting as much data as possible [11]. If the goal is to compromise the organization's service or infrastructure, the attacker seeks privilege escalation by capturing password hashes of more privileged users such as server administrators or domain administrators [11].

2.4.4. Privilege Escalation

Privilege escalation is established when a password hash of a more privileged user is captured [11]. One way this can be done is wait for the privileged user to access a resource on a workstation or a server already compromised during harvesting and network scanning [11]. Attackers may even deliberately attract a high privileged user's attention by simulating a problem or abnormal activity on a workstation or server, then wait for an administrator or technician to log in and investigate the root cause of the problem [11]. If this ambush is successful, password hashes of the administrator or technician are stored on the compromised machine and the attacker can obtain and reuse them [11].

2.4.5. Financial Fraud with Payment Cards

"Payment cards are used or accessed using four different interfaces which are by physical access (holding the card physically), by the magnetic stripe (swiping the card), by a chip reader (inserting the card in the machine), or by a Near-Field Communication (NFC) reader (short-range contactless communication technology)" ([6], p.~5).

There are two ways stolen track 1 and track 2 payment card information can be used:

1) Counterfeiting: Copying the track 1&2 information in a blank card's magnetic stripe and use it as a regular card. No one can tell the difference between the real and the fake one since they both have same information in them and when accessed through one of the interfaces discussed above, they are checked the same way too.

2) Transactions where Physical Cards are Not Needed: Beyond counterfeiting where track 1&2 information is used to clone payment cards, the stolen track1&2 information can be used to commit fraudulent "card-not-present" transactions such as online transactions [10]. For these transactions, The data obtained through physical access, "Name: cardholder's name, limited up to 26 characters long; Expiration date: 'YY/MM' format…; Credit Card Number, or Primary Account Number (PAN)…; Card Verification Value (CVV/CVV2)" ([6], p.~5), is sufficient for online transactions where the physical card is not required.

2.4.6. Direct Financial Theft from Bank Accounts

When banking credentials (username, password, MFA) are stolen, they are often used to steal money directly from the victim's bank accounts. Moore reports that in the case of Dridex, "the stolen credentials were used by criminals to hack into the user's account and steal money" ([13], p.~3), with "over £20 million ($30 million)" stolen "from UK bank accounts by these Dridex hackers" ([13], p.~2).

Additionally, infected computers become part of botnets that "forward spam emails containing malware to other users in the network" ([13], p.~3), expanding the criminal infrastructure and enabling further credential theft campaigns.

2.4.7. Kerberos Golden Ticket and Silver Ticket Attacks

Mimikatz can build Kerberos Golden Tickets and Kerberos Silver Tickets using stolen credentials (password hashes, the domain name, and domain security identifier (SID)) [9]. A successful Kerberos Golden Ticket compromise gives an attacker access to all Active Directory-connected systems which is why it is one of the most dangerous abilities of Mimikatz [9]. Active Directory is a centralized database which has information about all devices and resources on the network.

To create a Golden Ticket, attackers need access to a privileged account that can access the Active Directory domain controller and has access to the KRBTGT account [9]. The KRBTGT account is used for encrypting all Kerberos authentication tokens for the domain controller, and the domain controller uses this account to decrypt and verify the tokens it receives too. The password and username for this account never changes [9]. Once attacker extracts password hashes, the domain name, and domain security identifier (SID) for the KRBTGT account, they can create a Kerberos Golden Ticket to perform a pass-the-ticket attack, which provides access to the entire domain controller [9]. Once attacker compromises one domain controller, they can further escalate and compromise the entire active directory.

Kerberos Silver Tickets exploit a Windows feature that allows them to use network services [9]. Kerberos uses the ticket granting service to issue tickets to access services on a network, but Windows does not always check these tickets, so attackers can use them to gain further network access [9].

2.4.8. Domain Replication Abuse

A domain replication is a process of copying and synchronizing domain information among the multiple domain controllers (DCs) within a network. Once the attacker gets access to a privileged account with domain replication rights, Mimikatz can perform DCSync attacks where legitimate domain replication protocols are exploited and the attacker pretends to be a domain controller to extract password hashes, domain name, domain security identifier (SID), and Kerberos Ticket Generating Ticket (KRBTGT) account details from Active Directory [9].

Mimikatz uses Microsoft's Directory Replication Service Remote Protocol and the GetNCChanges API function to mimic the behavior of domain controllers and ask other domain controllers for copies of information [9]. A successful DCSync attack provides an attacker with administrative access to all information from the domain controller, effectively compromising the entire Active Directory forest [9]. This technique is particularly dangerous because it uses legitimate Windows functionality, making it difficult to distinguish from normal domain controller synchronization activities. The situation can become worse if the compromised domain has active Trust Relationships with other forests or domains, as the dam-

age can extend to other domains and forests depending on the trust directions [11].

In DCShadow attacks, attackers create a fake domain controller to replicate malicious changes to the legitimate domain controller [9]. The lsadump module of Mimikatz is used to register the compromised workstation as a domain controller in Active Directory [9]. Then using the compromised workstation now registered as domain controller, attacker replicates malicious changes to the legitimate domain controllers.

2.4.9. Exploitation of Shared Administrator Passwords

In many organizations, to achieve minimal administrative effort, a single password is assigned to the local administrator of all workstations in the environment [11]. In this case, if a potential attack is successful, the attacker will gain full access to all workstations in the domain environment where the local administrator is set with the same password [11]. This practice significantly amplifies the impact of a single successful credential theft, allowing attackers to compromise entire networks through a single stolen credential.

2.4.10. Selling on Dark Web

Often stolen credentials, may it be payment card details, usernames, passwords, hashes, or kerberos tickets, are sold on dark web marketplaces. The more privileged the credentials, the more valuable they are. "From botnet services and malware, to user data such as credit card information and passwords, darkweb marketplaces offer ease of use, product variety, and most importantly effective anonymity to both buyers and vendors" [14].

This section has demonstrated that credentials are not a monolithic category. They vary significantly in their structure, storage location, and exposure window, and these differences directly shape the techniques that adversaries use to steal them. Password hashes stored in the Windows SAM database are targeted differently than Kerberos tickets cached in LSASS memory, which are in turn stolen through different means than payment card data scraped from POS system RAM. This observation, that credential theft is fundamentally a heterogeneous problem, is central to the argument developed in the sections that follow. If the attack surface is diverse, the defense must be equally precise.

3. Credential Protection Mechanisms

Having established how credentials are stored and stolen, this section surveys the landscape of existing defenses. The mechanisms examined range from hardware-level isolation technologies such as Microsoft Credential Guard and Trusted Platform Module, to cryptographic approaches including secret splitting and threshold cryptography, to behavioral and network-based detection strategies. While many of these defenses are effective within their intended scope, a recurring pattern emerges: most existing protections are designed as general-purpose solutions applied uniformly across credential types, despite the heterogeneity documented

in the previous section. This section concludes by introducing FIDO2, a modern authentication standard that represents the most significant recent advancement in credential security, and examines both its strengths and its remaining limitations.

3.1. Existing Defenses against Credential Gathering

There are several existing defenses that are used in real world to prevent credential theft:

3.1.1. Microsoft Credential Guard

Microsoft Credential Guard is a security feature that uses Virtualization-based Security (VBS) to isolate and protect NTLM password hashes, Kerberos Tickets, and application-stored domain credentials from theft attacks [15]. It isolates the LSASS memory in a protected virtualized environment. So, even if attacker's malware has administrative privileges, they can not access the LSASS memory [15].

Credential Guard splits the LSA (Local Security Authority) into two processes [16]:

1) LSASS.exe: The Windows operating system that runs normally which is mentioned previously.

2) LSAIso.exe: This runs in the Virtual Secure Mode (VSM) with its own memory space, minimized kernel, and CPU-level enforcement.

Now, when authentication is required, the request is made to LSASS.exe in the normal OS, which communicates to LSAIso.exe through a controlled Remote Procedure Calls (RPC) [16]. LSAIso.exe processes the request in the protected VSM environment and sends the results back through RPC.

3.1.2. Credential Caching Management

Disabling or minimizing cached credentials through group policy reduces the attack surface since it is one of the popular attack spots [17].

3.1.3. Hardware-Based Protections

Additional protection layers can be implemented in the form of Data Execution Prevention, which prevents executing malicious code in data region of the memory, and Address Space Layout Randomization (ASLR), which randomizes memory locations for important program components every time an application is run [17]. The VBS mentioned in the Microsoft Credential Guard section uses Virtual Secure Mode (VSM) for isolation and protection. Trusted Platform Module (TPM 2.0) is a hardware component that protects the VSM master key so that the data stored in VSM can only be accessed in a secure environment [16].

3.1.4. Anti-Keylogging Defenses

Sajid *et al.* [18] demonstrated that API hooking can be used to intercept input-related API calls invoked by keyloggers at runtime and realistic decoy keystrokes can be injected so that attacker fails to capture real keystrokes. This deception based approach leads to feeding false information to the adversaries disrupting

their decision making process.

3.1.5. Authentication Hardening

There are several Authentication Hardening means that are used to further reduce the risk of credential theft [19].

1) Device Fingerprinting: Identifying known and trusted devices and adding an extra layer of authentication when logging into a new device.

2) Behavioral Analysis: Recognizing baseline user patterns and taking precautions if the patterns show something abnormal.

3) Geolocation Analysis: Detecting impossible travel scenarios and blocking them.

4) Session Risk Scoring: Evaluating continuously the legitimacy of access requests.

5) Access Controls: Access to credential storage locations must be restricted and least privilege principles must be used to limit giving administrative access to sensitive credential stores [17]. Application whitelisting and execution control also prevents credential theft by using tools like AppLocker or Windows Defender Application Control to create whitelists that only allow authorized applications to execute, blocking malicious credential-theft tools like Mimikatz or keyloggers from running, and configuring script blocking mechanisms to prevent unauthorized PowerShell or JavaScript execution that could be used to dump credentials [20].

6) Zero Trust Architecture: Zero Trust Architecture eliminates implicit trust and requires continuous verification throughout user sessions rather than relying on initial login authentication [19]. This architecture grants privileges only when needed and then revokes them immediately after use, monitors user activities for anomalies, and requires extra layers of authentication if user tries to escalate privileges or access sensitive information.

The authentication framework must encompass multiple factor categories such as knowledge factors like strong passwords, possession factors such as hardware tokens or mobile devices, and inherence factors including biometric verification and MFA to ensure that credential compromise alone cannot facilitate unauthorized access [19].

3.1.6. Network Monitoring

Security Information and Event Management (SIEM) systems provide network monitoring capabilities by gathering logs from various sources such as network devices, servers, and security systems, giving a centralized source of security related information [21]. SIEM helps detect potential threats and vulnerabilities and can also correlate events from different sources, identifying patterns and sending alerts in case of a cyber attack [21]. A Human Risk Management Platform can help monitor and identify suspicious activities such as unusual login attempts or access patterns further protecting sensitive accounts from potential breaches, and this continuous monitoring approach complements the need for organizations to em-

ploy Multi-Factor Authentication (MFA) and Zero Trust policies [22].

3.1.7. Endpoint Security

Endpoint Security includes anti-malware scans, allowlisting for application control, process controls, and behavioral analysis which can be offered by tools ranging from basic antivirus to sophisticated Endpoint Detection & Response (EDR) [23]. EDR offers 24/7 monitoring of endpoint activities such as process execution, file changes, and network connections, and utilize behavioral analysis with advanced technologies such as machine learning and artificial intelligence to identify suspicious activities that deviate from the norm which can indicate a cyber attack that was or was not previously known [21]. EDR can identify when attackers attempt to access credential stores, intercept authentication processes, or use tools like Mimikatz for credential harvesting [24]. Implementing monitoring systems to detect credential dumping tools like Mimikatz or Windows Credential Editor allows organizations to identify suspicious process behavior, and this is one of the strategies mentioned by MITRE ATT&CK framework [17]. EDR can also enable automated response actions such as isolating a compromised endpoint device to prevent further spread of threats, kill malicious processes, and roll back changes made by malware [21]. EDR has forensic capabilities that let security teams trace the root cause of an attack and understand the full attack chain which can help prevent future attacks [24].

3.1.8. Browser Isolation (Sandboxing)

Web sessions and open links are run in secure containers in cloud-based isolated platforms so that if a phishing page includes malicious scripts, credential fields, or malware payloads, they cannot reach the user's machine or network because input control policies can disable form submissions or file downloads when isolation is triggered [25].

3.1.9. Anti-Phishing Measures

Real time link analysis, behavioral scoring, and DNS filtering, verifying sender authenticity to block spoofed messages, and using hardware security keys to prevent credential theft by replacing passwords with device bound cryptographic credentials are all anti-phishing measures [25].

3.1.10. Decoy Credentials/ Honeytokens

Honeytokens are decoy data artifacts designed to look like legitimate and valuable information such as fake login credentials, documents, or API keys [26]. When an attacker interacts with a honeytoken, an alert is generated, providing early warning of a potential breach or misuse [27]. Credential honeytokens include dummy usernames, passwords, or API keys stashed in a system [26]. These will never be used by a legitimate user, so if they are used, it is a strong indication of an attack attempt [26] [28]. Honeytokens can be placed in applications or configuration files, signaling possible malicious intent when they are used [29]. Attackers may target LSASS (in Windows) where password hashes are stored in system memory,

so honeytokens can be seeded in memory using tools like RunAs or New-HoneyHash.ps1.

When adversaries authenticate using honeytokens, they land in sandbox environments that divert them from real infrastructure, allowing security teams to know within seconds what is compromised and what actions were performed. In enterprise environments, honeytokens placed in Active Directory have revealed credential harvesting and privilege escalation attempts [27]. So, honeytokens can help detect honeytoken stealing instantly. The drawback to this approach is that it can not guarantee prevention of real credentials' theft.

Even with these defenses, no system is completely secure. One small misconfiguration or a zero-day vulnerability is all it takes to compromise the network.

3.2. Secret Splitting Techniques

One thing observed was that the entire authentication material was stored in a single place, for example LSASS memory in Windows, which made it an easy target for the attackers. So, why not divide this authentication material into several parts and store them separately? Shamir's Secret Sharing (SSS) is an efficient secret sharing algorithm in which a secret is mathematically divided into parts (shares) from which the secret can be reassembled only when a certain minimum (threshold) number of shares are combined [30]. The algorithm provides information-theoretic security, meaning that even with unlimited computational power, possessing fewer than the threshold number of shares reveals nothing about the secret [30]. This approach is used in enterprise systems like HashiCorp Vault for secrets management [31], cryptocurrency wallets for recovery phrase backup [32], and DNSSEC key ceremonies [33]. For authentication systems, threshold cryptography extends this concept by ensuring credentials are never fully reconstructed. Instead, each device computes a signature share using only its secret piece, and these shares combine to generate valid authentication without revealing the complete credential [34] [35]. This architecture provides protection against credential theft by requiring attackers to compromise multiple devices simultaneously, while allowing legitimate authentication to succeed even when some devices are lost or stolen [34].

The drawback to this method is its single point of failure. The secret must exist in one place when it is split into shares, and again in one place when it is reassembled. These are attack points, and other schemes including multisignature eliminate at least one of these single points of failure [30]. Multisignature (multisig) is a security mechanism that requires multiple independent private keys to authorize a transaction, rather than relying on a single key that could be stolen or compromised. The technology is heavily used in cryptocurrency applications, with 9 million enterprise-grade multisig wallets deployed in 2025 and a market value of $1.27 billion in 2024 [36]. The difference between multisig and secret sharing algorithm is that in secret sharing algorithm, a single key is divided into multiple shares whereas in multisig, each part has its own key, so they do not need to be

brought together to reassemble the key and verify. Instead, each key is verified separately in multisig. The key strength of multisig is that it eliminates single points of failure because an attacker must compromise multiple separate keys to steal funds, and each key holder's signature is transparent and accountable [36] [37]. Although this looks great in theory and is used profoundly in real world, it has its limitations. Not all blockchain protocols support it consistently, poor implementations have led to major security breaches including a $30 million Ethereum theft from the Parity Wallet [38], transactions require coordination among multiple parties which creates operational delays, and the technology is primarily limited to blockchain applications rather than general-purpose authentication systems [38] [39].

3.3. Cryptographic Hardware for Secure Credential Management

When the operating system or the software is compromised, traditional defenses fail. Passwords, encryption keys, kerberos tickets, or any other authentication credentials which are stored in system memory can be accessed and the malware can just read them. So, why not try to find hardware-based authentication solutions where hardware is in charge of authentication so even if the software is compromised, credential theft can not take place. Several hardware-based solutions address this gap:

1) Trusted Platform Module (TPM): There is a secure hardware vault called Trusted Platform Module (TPM) that was mentioned previously which protects the VSM master key. Similarly, it can also generate other cryptographic keys, and protect passwords along with them [40]. The private portion of key pairs is decrypted inside the TPM, making it only accessible by the TPM hardware or firmware [40].

2) Chain of Trust (CoT) - Verifying Every Step: Before the computer boots up and before any credentials are used, the hardware checks every piece of software (firmware, bootloader, operating system) to make sure nothing has been tampered with. Each component verifies the next one in the chain. If any component has been modified by malware, the chain breaks and credentials are not released. It's like a series of security checkpoints where each guard verifies the next guard's identity before handing over the keys. The CoT can be extended further into the application domain, allowing for files, directories, devices, peripherals, etc. to be measured and attested [40].

3) Trusted Execution Environments (TEEs) - Isolated Safe Zones: A trusted execution environment is an area protected by a system processor where sensitive secrets like cryptographic keys and authentication strings can be preserved within the TEE, and operations involving these secrets can be performed within the TEE, thereby eliminating the need to extract the secrets outside of the TEE [40]. The difference between TPM and TEE is that TPM is a hardware vault for secrets whereas TEE is a panic room inside the processor where the sensitive operations happen in complete isolation. A TEE helps ensure that operations performed

within it and the associated data cannot be viewed from outside, not even by privileged software or debuggers [40].

TEE can be used to prove a system's integrity before releasing any credentials to it. In this remote attestation, TEE generates a verifiable cryptographic security certificate (quote of the enclave) which is sent to the attesting client who checks if the signature is valid and concludes if the computer's software is genuinely running [40]. The quote may contain the public key part of an enclave key pair which the client can use to encrypt secrets before sending them so that they can only be decrypted inside TEE [40].

4) Hardware Security Modules (HSMs): An HSM is a separate physical device (like an external hard drive, but for security) attached to a server can be used by workloads to store keys and perform cryptographic operations, resulting in keys being protected at rest and in use [40]. Because of HSMs, there is no need to store keys on disk or load them into RAM which are two most frequently attacked places for credential theft [40].

5) Hardware-Protected Browsers: Trusted hardware enclaves integrated into browsers enable protection of user secrets during web browsing sessions, even if the entire underlying browser and OS are fully controlled by a malicious attacker [41]. So, even if you type passwords into web browser, hardware protection ensures that they go straight to the hardware protected area.

While hardware-based authentication mechanisms provide strong protection against credential theft, their adoption faces significant practical barriers including performance overhead, cost, implementation complexity, and the need to rewrite existing applications. Despite these challenges, critical sectors like banking and cloud computing are increasingly deploying these technologies, suggesting that hardware-enabled security represents an important evolution in credential protection, even if full widespread adoption remains aspirational.

3.4. FIDO2 Authentication Standard

FIDO2 represents a modern authentication standard that eliminates the fundamental vulnerabilities inherent in password-based systems. The FIDO2 framework consists of two complementary protocols that work together to provide passwordless authentication:

• WebAuthn (Web Authentication API): This is the browser-side protocol defined by the World Wide Web Consortium (W3C). It enables web applications to trigger authentication or registration events via JavaScript, handling the secure communication between the browser (or operating system) and the web service. WebAuthn manages the transmission of challenges, receipt of responses, and secure data exchange with the relying party (the website or service requesting authentication) [42].

• CTAP (Client-to-Authenticator Protocol): This is the device-side protocol developed by the FIDO Alliance. It handles the communication between the browser or operating system and the authenticator device which can be through a USB

security key, smartphone, or built-in hardware module such as a Trusted Platform Module (TPM) used by Windows Hello [42].

Together, these protocols form the complete FIDO2 standard, ensuring secure and phishing-resistant communication throughout the authentication chain.

FIDO2 uses asymmetric cryptography. The system uses two mathematically linked keys which are the public key and the private key. The private key is securely stored on the user's authenticator device and never leaves it, while the public key is shared with the online service during registration [42]. To authenticate, the private key is used to create a digital signature that proves it knows the credential without revealing the key itself. The server verifies this signature using the stored public key. Passkey is the user-friendly name of the asymmetric key pair.

Here are the layers of protection provided by FIDO2's architecture:

1) Elimination of Shared Secrets: Since FIDO2 uses asymmetric cryptography, it does not rely on shared secrets such as passwords or verification codes that attackers can steal and use. The private key is never shared so as long as it is stored securely, it can not be stolen. This will be further discussed in the latter points.

2) Domain-Bound Credentials: During both registration and authentication, the authenticator validates the origin of requests using the Relying Party ID (RP ID), which is derived from the website's domain. Each credential is cryptographically bound to the specific domain where it was registered. The authenticator stores the hashed RP ID along with the private key and will only respond to authentication requests that match this domain [42]. This makes phishing attacks technically impractical, assuming the browser correctly reports origin information to the authenticator since phishing websites can replicate the visual design of legitimate sites, but they cannot fake the origin information sent by the browser to the authenticator.

3) Replay Attack Prevention: Once the private key is used to encrypt the signature, what if the attacker steals or sniffs it and tries to reuse it? To prevent this replay attack, a nonce is used. Nonce is a one-time cryptographic challenge generated by the server using the private key. Since it is a fresh challenge every time, it is only valid for a specific authentication attempt and can not be reused [42].

4) Hardware-Level Key Protection: FIDO2 authenticators store the private keys using hardware-level security mechanisms described in the previous section. Devices such as USB security keys contain dedicated secure elements, smartphones use trusted execution environments, and computers equipped with TPM chips leverage these cryptographic processors to protect key material [42]. So, the keys are isolated in tamper-resistant hardware that is specifically designed to resist both physical and software-based extraction attempts.

The security of FIDO2 is established through two distinct processes: registration and authentication. During registration [42], the authenticator generates an asymmetric key pair bound to a specific relying party (RP ID) and user, storing the private key in secure hardware while transmitting only the public key and a

signed challenge to the server. This ensures that credential material never leaves the authenticator in usable form. During authentication [42], the server issues a fresh nonce which the authenticator signs with the stored private key, and the server verifies the signature against the previously registered public key. Because each credential is scoped to a single RP ID and each challenge is unique, the protocol is inherently resistant to phishing and replay attacks.

FIDO2 also supports cross-device authentication in which users can login in a different device using passkey in a different device. This is done using a short-range Bluetooth Low Energy (BLE) connection where the device without passkey displays a QR code that the device with passkey scans and goes through the authentication process. This way, user can log into a different device without having to share any cryptographic key material itself [42].

Although passkeys are becoming the default login method for large tech players like Apple, Google, and Microsoft and FIDO2 is even supported by Chrome, Edge, or Safari [42], this authentication standard has some limitations. Since browser validates the website and reports the identity correctly to the authenticator, if the browser is fully compromised, it could be a problem. "WebAuthn is designed to stop remote attackers, fake sites, and credential replay. It is not designed to survive a hostile OS or a malicious browser" [42]. Another issue can be account recovery in case the device with the authenticator is lost. "Since the loss of authenticators prevents users from accessing web services, usable recovery solutions preserving the original WebAuthn design choices and security objectives are urgently needed" [42].

4. A New Approach: An Extension of FIDO2 Authentication Standard

It is evident that there are various credentials, and the process of hunting them varies widely based on where they are stored, how they are used, and when. Since there are no universal malware that can be used to steal any or all credentials, nor is there a universal attack plan, it would be inaccurate to claim that there is a universal defense against all credential thefts. Instead of trying to form a shield against diverse malware, it is necessary to now shift the focus on each attack individually and work on the different mitigation strategies for them.

This section examines one such scenario. The following develops a defense plan to protect the authentication credentials, which are the usernames, passwords, and cookies, used for websites all over the world. It is already understood that one can use the FIDO2 Authentication Standards to secure the registration and login process, and it is becoming popular with time, which is why it is crucial to investigate its limitations thoroughly.

Section 4.1 formalizes the trust and threat models within which the proposed extension operates. Section 4.2 introduces a browser attestation mechanism using zero-knowledge proofs and Direct Anonymous Attestation. Section 4.3 proposes a recovery architecture combining trusted contacts, Shamir Secret Sharing, and

browser attestation, concluding with demonstration of how these two mechanisms function as a unified, mutually reinforcing system.

4.1. Security Model of FIDO2 Authentication Standard

4.1.1. Trust Model

FIDO2 Authentication Standard trusts both the client and the server. For the client, it considers the hardware responsible for generating and storing keys, and the authenticating based on the stored information benign. It also considers the OS or the browser as safe, which mediates the communication between the client and the server using the WebAuthn and CTAP protocols. On the server's side, it is trusted for storing keys and information, generating random challenges and verifying legitimately.

The verification process, the protocols, and generation of keys and challenges are cryptographically secure. The storage of the keys is taken care of using the hardware level key protections.

4.1.2. Threat Model

The threat model for the Extension of FIDO2 authentication standard considers that the adversary has achieved code execution in the user's browser or operating system but has not yet compromised the authenticator hardware itself. The adversary either wants to impersonate a legitimate user during FIDO2 authentication or recover an account by manipulating the account recovery process. It is assumed that the adversary does not have access to hardware-level key since its extraction requires physical access to the device.

FIDO2 is primarily created to defend against:

1) Phishing Attacks: Since keys are specific to the domain and each key is stored with the unique RP ID, phishing attacks are prevented under the threat model defined above. For example, hell0.com can not get a valid signature for hello.com.

2) Replay Attacks: The server generates a fresh challenge for every authentication and nonce and signature counters are also used, which makes replay attacks computationally infeasible under standard cryptographic assumptions.

3) Database Breaches: Since only public keys are well known and no shared secrets or password hashes exist, so even if a server is breached, there are no credentials to steal.

4) Man-in-the-Middle Attack: TLS protects the transportation, signatures protect authentication data, and the origin binding prevents credential forwarding.

Since FIDO2 fully trusts both the client and the server, it does not provide a defense against a compromised operating system or browser. Malware can inject malicious JavaScript or manipulate WebAuthn calls, potentially compromising the entire system. Additionally, if a device is lost or physically stolen, verifying the legitimate user becomes challenging because the authentication process depends on the device itself, which could then be misused by an unauthorized individual.

The proposed browser attestation mechanism detects and blocks the adversary that tries to imitate a legitimate user, while the Shamir-based recovery scheme

resists the latter goal, that is unauthorized recovery initiated by an adversary who has stolen the user's device.

4.2. Browser Attestation

Before any sensitive information is shared between the server and the client, and before the client is attested, the browser can be remotely attested as well. A browser can prove that it meets the security standards without revealing its exact configuration information, which can be used by the attackers to fingerprint and can become a liability, by producing a zero-knowledge proof that shows that the browser has a valid attestation signature from an approved authority, without disclosing the specific key that was used [43]. This idea builds on a cryptographic technique Direct Anonymous Attestation (DAA) which is adopted by the Trusted Computing Group in TPM specifications that is used for remote authentication of trusted hardware while preserving user privacy. The browser's TPM holds an Attestation Identity Key (AIK) provisioned during device manufacturing and registered with the server during initial setup [44]. At boot time, the TPM measures each software component in the chain (firmware, bootloader, OS, browser), during the Chain of Trust verification, and stores cryptographic hashes in Platform Configuration Registers (PCRs) [45]. When a server wants to verify that the browser is not modified, the browser will send its PCR values signed by its AIK which is on server's approved list, which proves to the server that the PCR values are unmodified and it can check that the PCR values match the known-good configurations [46]. The server can learn that the browser is secure and unmodified but will not get any other information like browser version, installed extensions, or hardware model [47].

This method can be extended for the present purpose through temporal attestation. For temporal attestation, the system does the browser verification only during high risk situations like right before the FIDO2 authentication execution instead of continuously monitoring the browser which raises computational overhead, privacy concerns, battery drain, and network overhead. By performing the browser attestation immediately before FIDO2 authentication, the attack window to compromise the browser for the browsing session is successfully reduced from the entire session to mere seconds, ensuring the browser is verified as secure at the exact moment when credential theft would be most damaging. The proof expires after use and contains no linkable identifiers across sessions, so it maintains privacy while still providing security assurance when needed most.

This mechanism assumes that the TPM's Attestation Identity Key has not been exfiltrated and that the TPM hardware is uncompromised. It also assumes that the zero-knowledge proof scheme used is sound and zero-knowledge under standard cryptographic assumptions. If these assumptions hold, the scheme prevents a malicious browser from successfully completing FIDO2 authentication on behalf of a user, since the attestation step will fail prior to credential use. **Figure 1** illustrates the complete browser attestation flow described above.

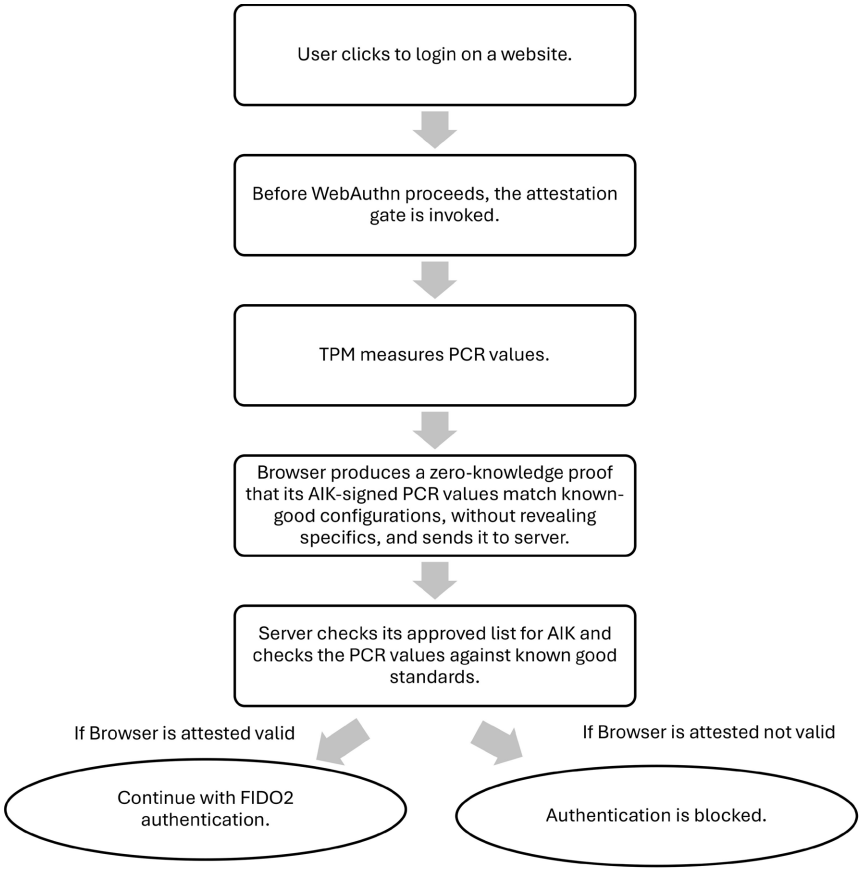

Figure 1. Browser attestation flow prior to FIDO2 authentication.

4.3. Account Recovery Mitigations

The solution to FIDO2's account recovery limitation when devices are lost or stolen is a recovery system that combines trusted contacts with Shamir Secret Sharing, browser attestation, and optional in-person verification. During setup, the user selects N trusted contacts (family members, close friends) and the system generates a recovery secret that is cryptographically split into N shares using the Shamir Secret Sharing, where any K shares (threshold) can reconstruct the secret [30], for example, N = 5 trusted contacts with a threshold K = 3. Each trusted contact gets one share and they register their own FIDO2 credentials with the browser attestation enabled. When the account recovery is needed, K contacts must go through the browser attestation immediately followed by the FIDO2 authentication to authenticate themselves. This provides a strong defense against a single point of failure or security questions. The system can further provide safeguard against social engineering and coercion attacks by implementing a mandatory time delay (24 - 72 hours depending on the user chosen preference at the time of initial setup) after the K contacts are authenticated, only after which the account can be recovered. During this delay, the legitimate user can get notifications to cancel unauthorized recovery attempts. Geographic distribution analysis can also be used to flag suspicious activities like multiple contacts authenticating from a

single location. For high-security accounts (banking, healthcare, enterprise), contacts must additionally verify their identity in-person at federated help desks, such as partner banks, post offices, or notary services, with government-issued identification before their share is released.

By combining trusted contacts (social trust), Shamir Secret Sharing (cryptographic security), browser attestation (device security), and optional in-person verification (identity assurance), the result is a recovery mechanism that scales across different risk profiles while maintaining the unphishable properties of FIDO2 authentication.

While the proposed recovery scheme offers substantial improvements over existing account recovery mechanisms, several residual risks warrant acknowledgment.

• Collusion risk: If any K of the N trusted contacts coordinate to initiate unauthorized recovery, the time-delay mitigation only partially addresses this threat, as colluding contacts could simply wait out the delay period together. This risk can be reduced by ensuring that the identities of the other trusted contacts remain mutually anonymous.

• Social engineering of contacts: An adversary who cannot directly compromise the account holder may instead target the trusted contacts themselves. While this attack surface is considerably harder to exploit than targeting a single user, particularly when contacts are geographically distributed and unknown to one another, it cannot be entirely eliminated. User education and the mutual anonymity property described above serve as partial mitigations.

• In-person verification scalability: Since it can be difficult to scale in-person help desks for account recovery, mandatory in-person verification is scoped to high-security account categories such as banking, healthcare, and enterprise environments.

Figure 2 depicts the resulting recovery architecture, showing how trusted contacts, Shamir Secret Sharing, browser attestation, and the optional in-person verification step combine into a single recovery flow.

Integration with Browser Attestation

If an adversary has achieved code execution in the victim's browser and then attempts to initiate account recovery from the compromised machine, the recovery flow will be triggered and the system will reach out to each of the trusted contacts to complete browser attestation immediately before their FIDO2 authentication step. The attestation gate will verify that each contact's browser is unmodified before their share is released, this also applies to the account holder's device at the point of recovery completion. An adversary operating from a compromised browser therefore cannot complete recovery even if they have successfully socially engineered K contacts, because the final delivery step will fail attestation on their end. This integration ensures that the two mechanisms are not merely complementary but mutually reinforcing. Attestation closes the gap that the recovery scheme alone leaves open.

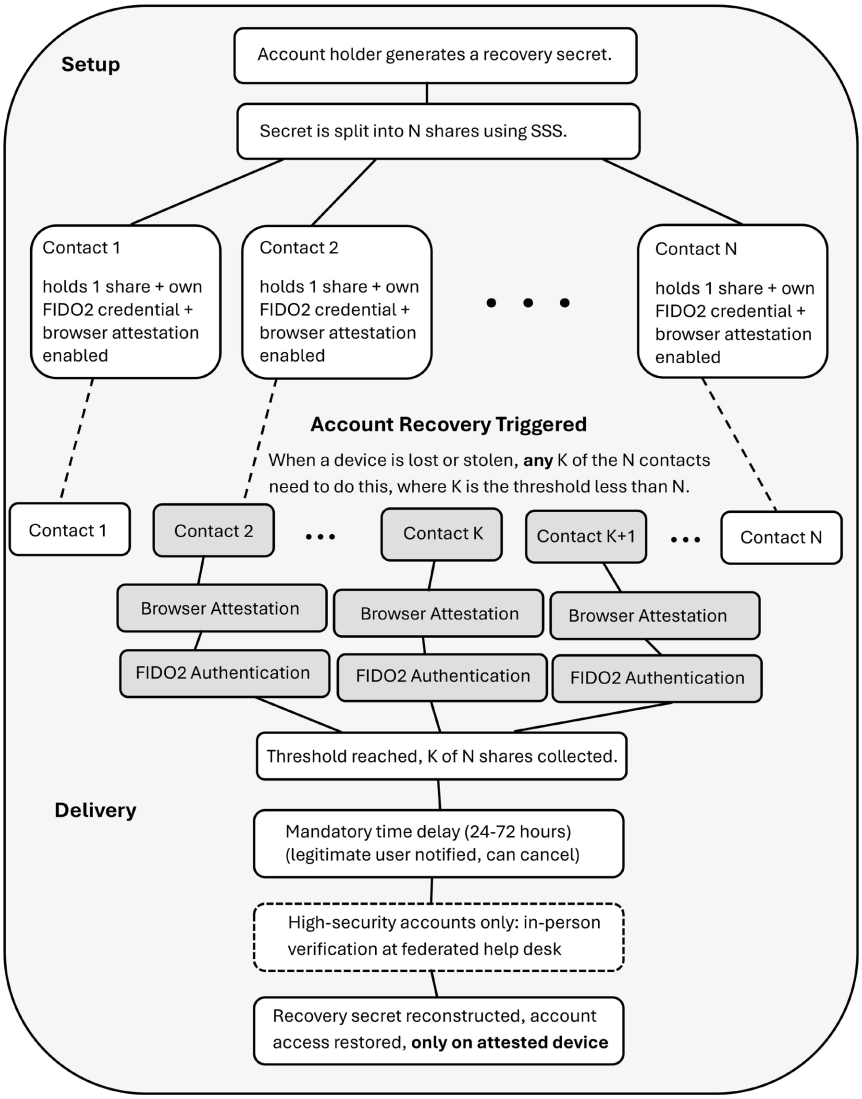

Figure 2. Proposed Shamir Secret Sharing recovery architecture for FIDO2 account recovery.

5. Conclusions

5.1. Summary of Findings

This paper set out to examine credential gathering attacks from three interconnected perspectives: how credentials are stolen, how existing defenses address this problem, and where a more precise, targeted approach can improve upon them. Across the sections, a consistent finding emerges: each credential is attacked differently, so having a uniform defense will leave significant attack surfaces inadequately protected.

The Understanding Credentials in Security Contexts section documented the breadth of this diversity. Credentials span a wide range of types, from plaintext passwords and NTLM hashes to Kerberos tickets, payment card magnetic stripe data, and cryptographic keys, each stored in different locations, accessible through

different means, and exploitable in different ways. The techniques attackers use reflect this variety: POS RAM scraping targets payment card data during the narrow window it exists in plaintext memory, LSASS extraction tools like Mimikatz target Windows authentication credentials cached in process memory, web injection techniques intercept banking credentials at the browser level, and pass-the-hash attacks leverage stolen hashes to authenticate without ever knowing the underlying password. The applications of stolen credentials are equally varied, ranging from direct financial fraud and lateral network movement to domain-wide compromise through Golden Ticket attacks.

The Credential Protection Mechanisms section surveyed the existing defensive landscape and found that while meaningful protections exist, none addresses the full breadth of the attack surface. Hardware-based mechanisms such as Microsoft Credential Guard and TPM-backed key storage provide strong protections in environments where they are deployed, but face adoption barriers. Cryptographic approaches such as Shamir Secret Sharing and multisignature schemes offer promising models for eliminating single points of failure but are primarily confined to specific application domains. FIDO2 represents the most significant recent advancement, eliminating shared secrets, binding credentials to specific domains, and leveraging hardware-level key protection. However, it makes a critical assumption: that the browser and operating system mediating authentication are trustworthy. It also lacks a robust account recovery mechanism. These two gaps motivated the contribution of the final section.

A New Approach section proposed a targeted extension to FIDO2 addressing both limitations. The browser attestation mechanism, grounded in Direct Anonymous Attestation and zero-knowledge proofs, verifies browser integrity immediately before authentication using TPM-measured Platform Configuration Register values. This reduces the window during which a compromised browser could successfully impersonate a legitimate user to mere seconds, while preserving user privacy through the zero-knowledge construction. The Shamir-based recovery architecture distributes a recovery secret among N trusted contacts, requiring any K of them to authenticate through both browser attestation and FIDO2 before shares are released, with a mandatory time delay and optional in-person verification for high-security accounts. Critically, the two mechanisms are mutually reinforcing: browser attestation closes the gap that the recovery scheme alone leaves open by requiring the account holder's device to pass attestation before the reconstructed secret is delivered.

5.2. Contributions

The primary contributions of this paper are as follows. First, a systematic characterization of credential theft as a heterogeneous problem, demonstrating that the attack surface varies significantly across credential types and that this variation is insufficiently reflected in current defensive frameworks. Second, a browser attestation scheme that extends the FIDO2 trust model to include verification of the

browser itself, using temporal zero-knowledge proofs based on Direct Anonymous Attestation, without requiring continuous monitoring or revealing device-specific information. Third, a threshold-based account recovery architecture for FIDO2 that combines social trust, cryptographic secret sharing, and hardware-based attestation to provide recovery guarantees that are resistant to both device theft and social engineering. Fourth, a demonstration that these two mechanisms function as a unified system whose combined security exceeds the sum of its parts.

5.3. Limitations

Several limitations of this work warrant acknowledgment. The browser attestation and recovery proposals are architectural, and formal security proofs for the combined protocol remain future work. The attestation mechanism assumes that the TPM hardware and Attestation Identity Key are uncompromised; an adversary with physical access to the device and the capability to extract the AIK falls outside the defined threat model. The federated help desk model required for high-security in-person verification assumes infrastructure that does not yet exist at scale.

5.4. Future Work

Several directions for future work follow naturally from this paper. Formal security proofs for the browser attestation scheme and the combined attestation-recovery protocol would significantly strengthen the theoretical foundations of the proposed extension. Empirical evaluation of the computational overhead introduced by the mandatory time delay step in real browser environments is needed to assess practical deployability. The design of a scalable federated identity verification infrastructure for high-security account recovery represents an important systems challenge. More broadly, the methodology advocated here, of developing targeted, credential-specific defenses grounded in precise analysis of storage and theft mechanisms, merits application to other credential types not addressed in this paper, including OAuth tokens, cryptographic certificates, and hardware security key credentials.

Acknowledgements

Peng Liu was partially supported by NSF CNS-2019340 and NSF ECCS-2140175.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Baeldung (2024) */Etc/Shadow* and Creating Yescrypt, MD5, SHA-256, and SHA-512 Password Hashes. https://www.baeldung.com/linux/shadow-passwords

[2] Microsoft (2025) NTLM user Authentication. https://learn.microsoft.com/en-us/troubleshoot/windows-server/windows-security/ntlm-user-authentication

[3] Microsoft (2025) How to Prevent Windows from Storing a LAN Manager (LM) Hash of the Password in AD and Local SAM Databases. https://learn.microsoft.com/en-us/troubleshoot/windows-server/windows-security/prevent-windows-store-lm-hash-password

[4] Microsoft (2025) NTLM Network Authentication Changes. https://learn.microsoft.com/en-us/troubleshoot/windows-server/windows-security/new-setting-modifies-ntlm-network-authentication

[5] PCI Security Standards Council, LLC (2024) Payment Card Industry Data Security Standard: Requirements and Testing Procedures, Version 4.0.1. Technical Report v4.0.1. https://www.pcisecuritystandards.org/document_library/

[6] Rodríguez, R.J. (2017) Evolution and Characterization of Point-of-Sale RAM Scraping Malware. *Journal of Computer Virology and Hacking Techniques*, **13**, 179-192. https://doi.org/10.1007/s11416-016-0280-4

[7] Kiwia, D., Dehghantanha, A., Choo, K.R. and Slaughter, J. (2018) A Cyber Kill Chain Based Taxonomy of Banking Trojans for Evolutionary Computational Intelligence. *Journal of Computational Science*, **27**, 394-409. https://doi.org/10.1016/j.jocs.2017.10.020

[8] Ah-Fat, P., Huth, M., Mead, R., Burrell, T. and Neil, J. (2020) Effective Detection of Credential Thefts from Windows Memory: Learning Access Behaviours to Local Security Authority Subsystem Service. 23*rd International Symposium on Research in Attacks*, *Intrusions and Defenses* (*RAID 2020*), San Sebastian, October 2020, 181-194. https://www.usenix.org/system/files/raid20-ah-fat.pdf

[9] Multi-State Information Sharing & Analysis Center (MS-ISAC) (2025) Mimikatz: The Finest in Post-Exploitation, Part 2 in a Series on Malware. https://www.cisecurity.org/insights/blog/mimikatz-the-finest-in-post-exploitation

[10] Huq, N. (2015) Defending against POS RAM Scrapers: Current and Next-Generation Technologies. Technical Report, Trend Micro, Forward-Looking Threat Research Team, White Paper. https://documents.trendmicro.com/assets/wp/wp-defending-against-pos-ram-scrapers.pdf

[11] Dimov, D. and Tzonev, Y. (2017) Pass-the-Hash—One of the Most Prevalent Yet Underrated Attacks for Credentials Theft and Reuse. *International Conference on Computer Systems and Technologies* (*CompSysTech*'17), Ruse, 23-24 June 2017, 149-154. https://doi.org/10.1145/3134302.3134338

[12] Hans Peter Luhn (1960) Computer for Verifying Numbers. US Patent 2,950,048A. https://patents.google.com/patent/US2950048A/en

[13] Moore, A. (2016) Research Note—Banking Malware Explained: The Case of Dridex. SSRN Working Paper, 7 p. https://doi.org/10.2139/ssrn.2797341

[14] Georgoulias, D., Yaben, R. and Vasilomanolakis, E. (2023) Cheaper than You Thought? A Dive into the Darkweb Market of Cyber-Crime Products. *Proceedings of the* 18*th International Conference on Availability*, *Reliability and Security*, New York, 29 August 2023-1 September 2023, 1-10. https://doi.org/10.1145/3600160.3605012

[15] Microsoft (2025) Credential Guard Overview. https://learn.microsoft.com/en-us/windows/security/identity-protection/credential-guard/

[16] Microsoft (2025) How Credential Guard Works. https://learn.microsoft.com/en-us/windows/security/identity-protection/credential-guard/how-it-works

[17] MITRE (2024) Credential Access Protection, Mitigation M1043. https://attack.mitre.org/mitigations/M1043/

[18] Sajid, M.S.I., Ahmed, S. and Sosnoski, R. (2025) Secure Development of a Hooking-Based Deception Framework against Keylogging Techniques. 2025 *IEEE Secure Development Conference* (*SecDev*), Indianapolis, 14-16 October 2025, 82-91. https://doi.org/10.1109/secdev66745.2025.00019

[19] Sarika Sharma (2025) 5 Ways to Defend against Credential Theft Attacks: A Technical Defense Framework. https://fidelissecurity.com/threatgeek/threat-detection-response/defend-against-credential-theft/

[20] MITRE Corporation (2019) Execution Prevention. https://attack.mitre.org/mitigations/M1038/

[21] Exabeam (2025) SIEM vs. EDR: Key Features, Differences, and How to Choose. Industry White Paper. https://www.exabeam.com/explainers/siem/siem-vs-edr-key-features-differences-and-how-to-choose/

[22] Keepnet Labs (2025) Understanding and Preventing Credential Theft, 2025. Industry White Paper. https://keepnetlabs.com/blog/understanding-and-preventing-credential-theft

[23] SentinelOne (2025) What Is Endpoint Security? Key Features, Types & Threats. Industry White Paper. https://www.sentinelone.com/cybersecurity-101/endpoint-security/what-is-endpoint-security/

[24] Vectra AI (2025) Endpoint Detection and Response (EDR): The Complete Security Guide. Industry White Paper. https://www.vectra.ai/topics/endpoint-detection-and-response

[25] Palo Alto Networks (2026) What Is Phishing? https://www.paloaltonetworks.com/cyberpedia/what-is-phishing

[26] SentinelOne (2025) What Are Honeytokens in Cybersecurity? https://www.sentinelone.com/cybersecurity-101/cybersecurity/honeytokens/

[27] Acalvio (2025) Understanding Honeytokens: Functions and Different Types. https://www.acalvio.com/resources/glossary/honeytoken/

[28] Huntress (2025) What Is a Honey Token? A Cybersecurity Trap for Catching Intruders. https://www.huntress.com/cybersecurity-101/topic/what-is-honey-token

[29] CrowdStrike (2025) What Are Honeytokens? https://www.crowdstrike.com/en-us/cybersecurity-101/identity-protection/honey-tokens/

[30] Shamir, A. (1979) How to Share a Secret. *Communications of the ACM*, **22**, 612-613. https://doi.org/10.1145/359168.359176

[31] HashiCorp (2026) Seal/Unseal. Vault Documentation. https://developer.hashicorp.com/vault/docs/concepts/seal

[32] Ledger Academy (2023) What Is Shamir's Secret Sharing? https://www.ledger.com/academy/topics/security/shamirs-secret-sharing

[33] Privy (2026) Sharing the Secret: A Peek under the Hood of Privy's Shamir Secret Sharing Library. https://privy.io/blog/shamir-secret-sharing-deep-dive

[34] Abidin, A., Aly, A., Mustafa, M.A. (2020) Collaborative Authentication Using Threshold Cryptography. In: Saracino, A. and Mori, P., Eds., *Emerging Technologies for*

Authorization and Authentication, Springer, 122-139.
https://doi.org/10.1007/978-3-030-39749-4_8

[35] COSIC (2022) Threshold Crypto and Its Application to Collaborative Authentication.
https://www.esat.kuleuven.be/cosic/blog/threshold-crypto-and-its-application-to-collaborative-authentication/

[36] BitGo (2026) What Are Multi-Signature Wallets? Crypto Wallet Security.
https://www.bitgo.com/resources/blog/what-is-a-multi-signature-wallet/

[37] Bitcoin Wiki (2025) Multi-Signature. https://en.bitcoin.it/wiki/Multi-signature

[38] Fireblocks (2025) MPC vs. Multi-Sig, October. Industry White Paper.
https://www.fireblocks.com/blog/mpc-vs-multi-sig

[39] Cointelegraph (2026) What Is a Multisignature Wallet, and How Does It Work?
https://cointelegraph.com/tags/multisignature

[40] Bartock, M., Souppaya, M., Savino, R., Knoll, T., Shetty, U. Cherfaoui, M., *et al.* (2021) Hardware-Enabled Security: Enabling a Layered Approach to Platform Security for Cloud and Edge Computing Use Cases. NIST Interagency Report NIST IR 8320 (2nd Draft), National Institute of Standards and Technology.
https://doi.org/10.6028/NIST.IR.8320

[41] Eskandarian, S., Sethi, T.K., Subbiah, V., Backes, M., Pellegrino, G., Boneh, D., *et al.* (2019) Fidelius: Protecting User Secrets from Compromised Browsers. 2019 *IEEE Symposium on Security and Privacy* (*SP*), San Francisco, 19-23 May 2019, 264-280.
https://doi.org/10.1109/sp.2019.00036

[42] Michael Waterman (2026) How FIDO2 Works, a Technical Deep Dive. Blog Post Providing Technical Explanation of FIDO2/Webathons Authentication Protocols.
https://michaelwaterman.nl/2025/04/02/how-fido2-works-a-technical-deep-dive/

[43] Adam Langley (2019) Zero-Knowledge Attestation. ImperialViolet Blog.
https://www.imperialviolet.org/2019/01/01/zkattestation.html

[44] Brickell, E., Camenisch, J. and Chen, L. (2004) Direct Anonymous Attestation. *Proceedings of the* 11*th ACM Conference on Computer and Communications Security*, New York, 11 February 2004, 132-145. https://doi.org/10.1145/1030083.1030103

[45] Yang, K., Chen, L., Zhang, Z., Newton, C.J.P., Yang, B. and Xi, L. (2021) Direct Anonymous Attestation with Optimal TPM Signing Efficiency. *IEEE Transactions on Information Forensics and Security*, **16**, 2260-2275.
https://doi.org/10.1109/tifs.2021.3051801

[46] Cloudflare (2022) Introducing Zero-Knowledge Proofs for Private Web Attestation with Cross/Multi-Vendor Hardware. Cloudflare Blog.
https://blog.cloudflare.com/introducing-zero-knowledge-proofs-for-private-web-attestation-with-cross-multi-vendor-hardware/

[47] ETSI (2023) Zero Knowledge Proof—European Digital Identity. Technical Report, European Telecommunications Standards Institute.
https://www.etsi.org/deliver/etsi_tr/119400_119499/119476/01.02.01_60/tr_119476v010201p.pdf