Digital Identity Explained

What session tokens, refresh tokens, cookies, API keys, and wallet keys actually are, why attackers target them, and how Cyber Crucible's kernel behavioral engine protects them without ever reading them.

Do you collect any of identity data?

The short answer is no, we do not.

The longer answer, is, no we do not, but in two different ways:

First, we never transmit passwords, cookies, oauth tokens, or anything else that would be considered identity data to our Cyber Crucible servers. That is true whether the server is customer-hosted, or Cyber Crucible hosted.

Second, we learned to conduct behavioral analysis of identity information, without actually processing identity information. That means that our analytics are not actually processing or exposing sensitive data such as actual passwords, session tokens, etc.

How Is This Different Than Normal Identity Monitoring?

Cyber Crucible’s digital identity theft capability is proactive and preventative. It is preventing access to the information needed to conduct blackmail, extortion, and modern identity theft operations.

Traditional identity theft protection deals with passwords and other data in fraudsters' hands, and monitoring for things like fraudulent credit card usage, taking out fraudulent loans, or passwords eventually being found in data breaches. This is very reactive, and is difficult for customers to manage in their daily lives. It also opens the opportunity, which Cyber Crucible has observed, to blackmail individuals, and even coerce employees and executives into assisting the attackers to helping attack their employers.

There is a concerted effort by attackers to go after cookies, logins, messaging platforms, and other forms of authentication and private data to enable their attacks.

Cyber Crucible’s identity theft focused analytics prevent that, in a manner which provides authoritative behavioral decision making without user involvement, against modern extortion tactics and cybercriminal file-less attacks.

What identity data is protected?

Core Windows Operating System

- NTDS (Active Directory)
- Incubating feature that will continue to improve

VPN client credentials

- usernames
- passwords
- key-based authentication

Web Browsers

- cookies
- refresh tokens (“remember me” authentication tokens)
- oAuth tokens (session tokens, access tokens)
- passwords
- usernames
- browser history

Collaboration & Messaging Applications

- contacts
- userrnames
- chat histories
- saved chat contents
- saved chat attachments
- private messaging encryption keys
- passwords
- oauth tokens (session tokens, access tokens, refresh tokens)

Crypto Wallets

- wallet contents
- transaction records

- passwords
- encryption keys

Gaming:

- Incubating - currently only Steam
- passwords
- purchases
- credentials
- usernames
- oAuth tokens (session tokens, access tokens, refresh tokens)

File Sharing:

- Incubating - currently only FileZilla
- passwords
- purchases
- credentials
- usernames
- key-based authentication
- servers & server settings

How is identity data protected?

Short answer: Cyber Crucible identifies the locations where identity data is stored and protects those locations at the kernel level. When a program reaches for one, its access is traced and analyzed — **without the passwords, tokens, or encryption keys themselves ever being accessed or collected**. Based on that analysis the program is allowed, given fake data, denied, or suspended.

Protecting access, not the secret

Most identity protection works on the secret: encrypt it, vault it, or watch for it appearing in a breach dump. Cyber Crucible works on the **act of reaching for it**.

Automated attacks are predictable in one specific way — they must go to known locations to find credentials. Browser cookie stores, VPN credential paths, the Windows credential store, wallet files, and Active Directory data all live where attackers already know to look. That predictability is the defensive opportunity.

Cyber Crucible identifies those identity data stores and then protects them. Any program touching one is evaluated, along with its parent and child processes and the libraries it has loaded.

The secrets are never read

This is a deliberate design constraint, not a side effect: **the access is traced and analyzed without the credential itself being accessed.** Cyber Crucible does not read, process, collect, or transmit passwords, session tokens, cookies, or encryption keys — to any server, whether Cyber Crucible-hosted or customer-hosted.

Behavioral analysis of identity access turns out not to require the identity data. What matters is *which program is reaching where, in what context, having done what beforehand* — none of which requires opening the secret.

The consequence is that the protection cannot itself become the breach, and there is no central store of your credentials to subpoena, leak, or lose.

What the behavioral engine evaluates

The decision isn't "is this program on a list?" — it's "what is this code actually doing right now?" Signals include:

- **Memory state and behavior** — tracked across the program's whole lifecycle, including in-memory changes that never touch disk. This is the platform's foundational sensor layer, shared with data protection and FortressAI rather than specific to identity.
- **Process lineage** — what launched this, and what it launched in turn.
- **Access pattern** — is this a normal application reading its own credential, or a sweep across many identity locations at once?
- **Library and injection activity** — has this trusted process been injected into or altered in memory?

The response is graduated, not binary

Not every program reaching for identity data is an attacker, so the response depends on what the program is and how it is behaving:

Situation	Response
A known application, not exploited, accessing what it legitimately needs	Allowed
A known application, not exploited, but **exceeding** its appropriate access	**Given fake data, or access denied** — governed by privacy behavioral rules
An **unknown** application	**Rejected or suspended**, per the behavioral engine and your settings
A **compromised** application — process or libraries exploited	**Rejected or suspended**, per the behavioral engine and your settings

The middle case matters more than it first appears. Plenty of legitimate software reaches further into identity stores than it needs to. Suspending it would break the user's day for no security gain — but handing it real credentials is an unnecessary exposure. Returning fake data satisfies the program while the real secret stays put.

The distinction the engine is drawing is between a trusted program *overreaching* and a program that is unknown or has been *taken over*. Only the latter is treated as an attack.

Why this happens on the endpoint

An automated infostealer can infiltrate, collect, and delete its own in-memory tooling within seconds. Sending telemetry to a cloud service and waiting for a verdict cannot beat that. The full detect-decide-respond loop therefore runs locally in the kernel, which is also why it works with no connectivity at all.

> See also: "*Do you collect any of identity data?*", "*What identity data is protected?*", and "*What happens when a program tries to access identity data it shouldn't?*"

What cryptocurrency wallets are protected?

- Coinomi
- Armory (“Bitcoin Armory”)
- Electrum
- Exodus
- Guarda

What happens when a program tries to access identity data it shouldn't?

Short answer: It depends on whether the program is trusted and whether it has been compromised. A legitimate application that simply overreaches is given fake data or quietly denied. An unknown or compromised application is rejected or suspended. The real credential is never handed over in any of those cases.

Why a single response would be wrong

The obvious design is "block anything that touches a credential store." It fails immediately in practice, because legitimate software touches those stores constantly — browsers, password managers, VPN clients, sync tools, and backup agents all have real reasons to be there.

Blocking everything breaks the machine. Allowing everything trusted means an exploited browser gets whatever it asks for. The useful question is narrower: **is this program behaving the way it should be?**

The four outcomes

1. Allowed. A known, unexploited application accessing what it legitimately needs. Nothing changes.

2. Fake data returned. A known, unexploited application that exceeds its appropriate access. Privacy behavioral rules apply: the program receives plausible but false data. It continues running normally, and the real credential never leaves storage. From the program's perspective nothing failed — which is precisely why this works without breaking workflows.

3. Access denied. The same situation, where returning fake data isn't appropriate. The request is refused; the application keeps running.

4. Rejected or suspended. The program is unknown, or a known program whose process or libraries show signs of exploitation. Here the concern isn't overreach, it's that the program is no longer acting as itself. Whether it's rejected or suspended depends on the behavioral engine's assessment and your configured settings.

Why deception rather than blocking

Returning fake data is a deliberate choice. A blocked request tells an attacker they've been detected and prompts them to try another route. Fake data that looks real does not — the tooling proceeds with credentials that unlock nothing, and the operator may not learn anything is wrong for some time.

This is the same principle behind canary files in ransomware defense: give the attacker something convincing to grab that costs you nothing.

Configurability

The boundary between rejection and suspension is tunable. Environments with low tolerance for interruption — clinical systems, production lines — can weight toward denial and fake data over suspension, while high-security environments can be more aggressive.

What is a session token, and why is stealing one worse than stealing a password?

Short answer: A session token is the credential your browser or app holds *after* you log in, proving you're already authenticated. Stealing one is often worse than stealing a password because it typically bypasses both the password and multi-factor authentication — the attacker simply resumes your existing session.

Why it exists

Re-entering a password on every request would be unusable, so after a successful login the service issues a token. Every later request carries that token instead of your credentials. The service trusts it as proof you already authenticated.

Why attackers prefer it

- **It skips MFA.** MFA is checked at login. A token issued *after* that check represents an already-passed authentication, so replaying it usually doesn't re-trigger the second factor.
- **It looks legitimate.** To the service, a valid token is a valid user. There's no failed-login signal and nothing anomalous to alert on.
- **It's portable.** Many tokens work from a different machine, network, or country.

Why this changed attacker behavior

Attackers increasingly avoid lingering in a network. Staying resident raises the odds of detection. Instead they take identity material and leave — then return at will as an authenticated user. The token is a legitimate-looking back door that the authentication system itself issued.

How Cyber Crucible addresses it

Token theft is stopped at the moment of access. When a program reaches for the browser or application storage where tokens live, its access is assessed in under 200 milliseconds. An

overreaching but legitimate program is given fake data or denied; an unknown or compromised one is rejected or suspended. Either way the real token is never handed over.

What is a refresh token, and why is losing one especially damaging?

Short answer: A refresh token is a long-lived credential used to silently obtain new access tokens without the user logging in again. It's what "remember me" relies on. Because it can keep minting fresh access indefinitely, a stolen refresh token can give an attacker durable access long after your session ends.

Access token vs. refresh token

- **Access token (session token):** short-lived, used on each request, expires in minutes or hours.
- **Refresh token:** long-lived, used only to request new access tokens. Sometimes valid for weeks or months.

A stolen access token eventually expires. A stolen refresh token can be used to generate new access tokens repeatedly — which turns a one-time theft into persistent access.

Why "remember me" is the trade-off

Staying signed in requires storing something durable on the device. That convenience is exactly what makes the stored token valuable to an attacker. Browser and app credential stores are standard, predictable locations, so automated tools go straight to them.

Reducing the damage

Revoking a refresh token invalidates the access it can mint — which is why "sign out of all devices" and token revocation matter after any suspected compromise. But revocation is a response *after* theft.

Cyber Crucible's approach is to prevent the read. The credential store is a protected location: a legitimate program overreaching there receives fake data or is denied, while an unknown or compromised one is rejected or suspended. The refresh token itself is never surrendered.

What are browser cookies, and how do attackers abuse them?

Short answer: Cookies are small pieces of data a website stores in your browser, and some of them are authentication cookies that keep you logged in. Stealing those is functionally equivalent to stealing an active session — the attacker can load them into their own browser and appear to be you.

Not all cookies matter equally

- **Preference cookies** — language, theme, layout. Low value.
- **Analytics cookies** — usage measurement. Low value.
- **Authentication / session cookies** — prove you're logged in. **High value.**

Attackers care almost exclusively about the third category.

Why cookie theft is effective

A stolen authentication cookie usually doesn't trigger a login event, a password prompt, or an MFA challenge. The session already exists; the attacker is simply presenting proof of it. From the application's perspective this looks like ordinary continued activity.

Browsers store cookies in known, predictable file locations on every operating system, so automated tools can locate them without knowing anything about the specific machine.

Why the browser is a focal point

The browser holds cookies, saved passwords, OAuth tokens, and history in one place, which makes it the single richest identity target on most endpoints. Cyber Crucible has seen this directly: from Q4 2023 onward, automated responses against Chrome, Edge, and Chromium activity climbed from zero into the thousands, with related CVEs later published by Google and Microsoft.

Protection

Browser credential storage is one of the protected identity locations. A process attempting to read it is assessed in context — an exploited browser is treated very differently from a healthy one, and in neither case does an overreaching request receive the real cookie.

What is an API key, and what happens if one is stolen?

Short answer: An API key is a secret string that identifies and authorizes a program — rather than a person — when it calls a service. A stolen key lets an attacker make requests as your application, often with no user interaction, no MFA, and no obvious sign anything is wrong.

Why they're attractive targets

- **No human in the loop.** Keys are built for automated, unattended use, so there's no MFA prompt to interfere.
- **Often broadly scoped.** Keys are frequently issued with more permission than the task requires.
- **Long-lived.** Many are never rotated, so a key stolen today may still work months later.
- **Stored in predictable places.** Configuration files, environment variables, and credential stores — all locations automation can enumerate.

Typical consequences

Depending on what the key authorizes: reading or exporting customer data, sending messages as your organization, spending money against a cloud account, or pivoting into other connected systems.

Because the requests are properly authenticated, they usually appear in logs as normal application traffic. Detection commonly happens well after the fact, via a bill or an audit.

Reducing exposure

Scope keys narrowly, rotate them regularly, and keep them out of source control. Those are sound practices, but they limit blast radius rather than prevent theft.

Cyber Crucible targets the theft itself. Credential and configuration locations are protected, and a program reading them is evaluated in context: fake data or denial for a legitimate overreach, rejection or suspension for an unknown or compromised process. The key is not handed over either way.

Where does Windows store passwords and credentials?

Short answer: In several predictable places — the Windows Credential Manager, browser password stores, VPN client configuration, and on domain controllers the Active Directory database (NTDS). "Predictable" is the key word: automated attacks rely on those locations being the same on every machine.

The main stores

- **Windows Credential Manager** — saved logins for network resources and applications.
- **Browser password stores** — Chrome, Edge, and Firefox each keep saved credentials in known profile paths.
- **VPN client credentials** — usernames, passwords, and key-based authentication material.
- **NTDS (Active Directory)** — on a domain controller, the credential data for the entire domain. The highest-value target on the network.
- **Application-specific stores** — messaging and collaboration tools, file transfer clients, and gaming platforms each maintain their own.

Why predictability is the whole point

Attackers know nothing about your specific environment. Their automation is written against paths that exist everywhere — `C:\Users\[Username]`, standard application data folders, standard drive letters. A script doesn't need to understand your network; it just walks known paths.

That's a weakness in their method. If those exact locations are the ones being watched, the attacker's own reliability requirement becomes the trigger that catches them.

What Cyber Crucible does with that

These stores are the monitored identity-theft entry points. A program reading them has its intent assessed in under 200 milliseconds — and is suspended if that intent is malicious, before any credential is taken.

What are cryptocurrency wallet keys, and why is theft irreversible?

Short answer: A wallet key is the private cryptographic key that authorizes spending from a cryptocurrency address. Whoever holds it controls the funds. Theft is effectively irreversible because blockchain transactions can't be reversed and there's no institution to appeal to.

Why this differs from a bank

A fraudulent card charge can be disputed and reversed by an issuer. A cryptocurrency transaction signed with your private key is valid by definition — the network cannot distinguish a theft from a legitimate transfer, and there is no central authority to unwind it.

That makes wallet keys unusually attractive: the payoff is immediate, final, and hard to trace.

What attackers look for

- **Private keys and seed phrases** stored in wallet application files
- **Wallet contents and transaction records** useful for targeting the highest-value victims
- **Passwords protecting the wallet**, often reused elsewhere

Desktop wallet software stores this material in predictable application-data locations — the same pattern that makes every other credential store reachable by automation.

Protection

Wallet files are among the identity locations Cyber Crucible monitors. Because the response is suspension at the moment of malicious access, the theft is prevented rather than detected afterward — which matters more here than almost anywhere else, since there is no recovery path once funds move.

> See also: "*What cryptocurrency wallets are protected?*" for the specific applications currently covered.

What is key-based authentication, and why do stolen keys grant lasting access?

Short answer: Key-based authentication proves identity with a cryptographic private key instead of a password — used by SSH, many VPNs, and file transfer clients. A stolen private key grants access without any password or MFA prompt, and because keys are rarely rotated, that access can persist for a very long time.

How it works

You hold a private key; the server holds the matching public key. At connection time the server issues a challenge only the private key can answer. The key itself never crosses the network.

This is genuinely stronger than passwords — it resists guessing, phishing, and reuse. Its weakness is different: **everything depends on the private key file staying private.**

Why a stolen key is so durable

- **No password prompt.** The key *is* the credential.
- **Frequently no MFA.** Key auth often stands alone, particularly for automated systems.
- **Rarely rotated.** Passwords expire on a schedule; keys commonly live for years.
- **Predictable storage.** Standard directories and VPN client configuration paths.

A stolen key is a quiet, durable back door that produces no failed logins and looks entirely legitimate.

Protection

VPN and key-based authentication material are explicitly among the protected identity categories. Access is evaluated at the kernel level, so a program harvesting key files is stopped — denied or fed fake data if it is a trusted program overreaching, rejected or suspended if it is unknown or

compromised — before the key leaves the machine.

Why does protecting identity data require kernel-level access?

Short answer: Because the theft happens in the space between a program requesting credential data and the operating system delivering it. Only a defense operating at the kernel — below the applications and libraries an attacker can manipulate — can see that request, judge its intent, and stop it in time.

The three requirements

1. See the access as it happens. Identity theft isn't a file appearing on disk; it's a read. Tools scanning for malicious files see nothing, because in a modern infostealer nothing is written. The event to catch is the access attempt itself.

2. Decide fast enough. Automated tooling can infiltrate, collect, and self-delete within seconds. Any architecture that ships telemetry to a cloud service and waits for a verdict has already lost the race. The decision has to happen locally, in milliseconds.

3. Not depend on what the attacker controls. Security tools that rely on operating system libraries can be blinded when an attacker compromises those libraries in memory — the agent keeps running and reports nothing. Cyber Crucible was deliberately decoupled from Windows libraries so a compromised OS cannot silence it.

The consequence

Kernel-level operation is what allows all three at once: visibility into the access attempt, a sub-200-millisecond local decision, and independence from a potentially compromised operating system.

It's also why the protection works with no connectivity — air-gapped, offline, or at sea — since nothing has to leave the device for a decision to be made.