REFLECTIONS ON DISREGARDING TRUST

Weaponization of CDP and MHTML in Headless Session Hijacking

SPEAKER BRIEFING

Gregory Disney-Leugers

Follow-up to Reflections on Trusting Trust (1984)

ACT I

THE FOLLOW-UP

Reflections on Trusting Trust · 1984 → runtime · 2026

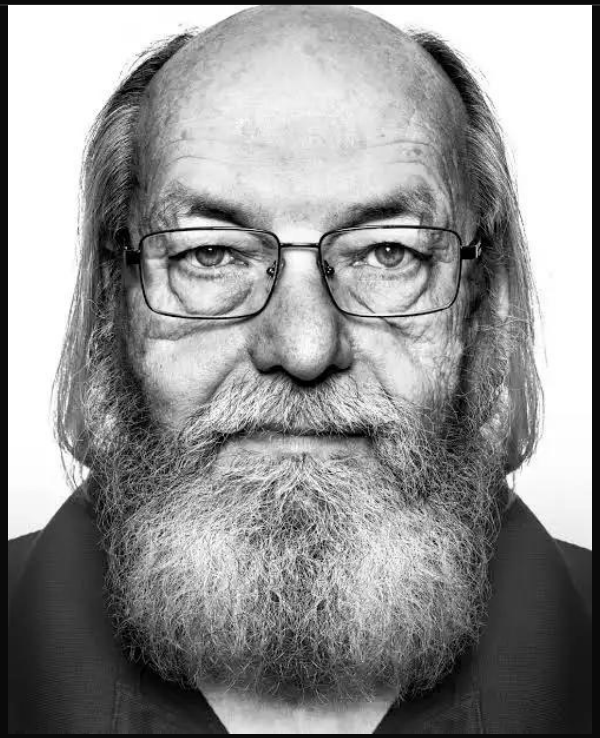
~10 min

// Historical

Forty years in the making

Reflections on Trusting Trust

"The moral is obvious. You can't trust code that you did not totally create yourself. (Especially code from companies that employ people like me.) No amount of source-level verification or scrutiny will protect you from using untrusted code. In demonstrating the possibility of this kind of attack" - *Ken Thompson*



// About the Speaker

Who am I?

Tech lead of offensive security

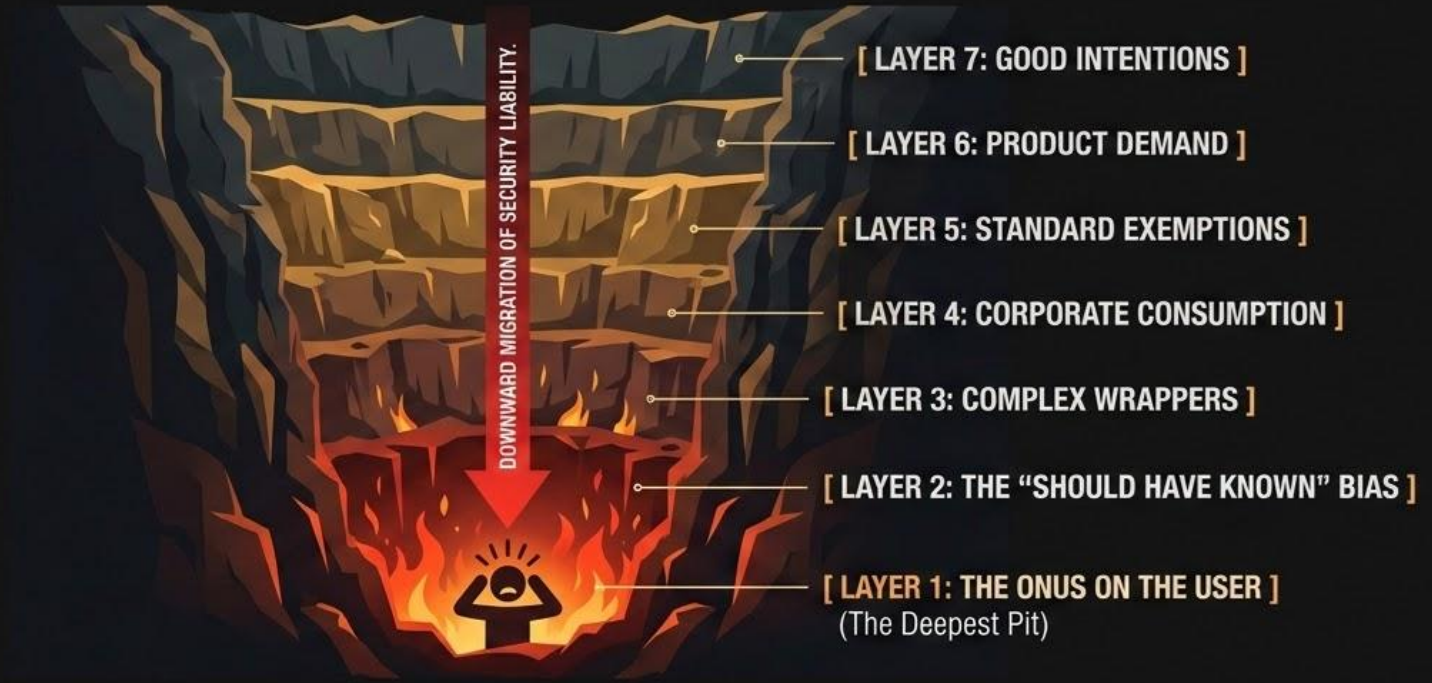
Developer of:

- TunnelTug
- Phisheries
- 0TrustCloud

// THEORETICAL FOUNDATION

Pillar 1: Dante's Inferno of Deflected Ownership

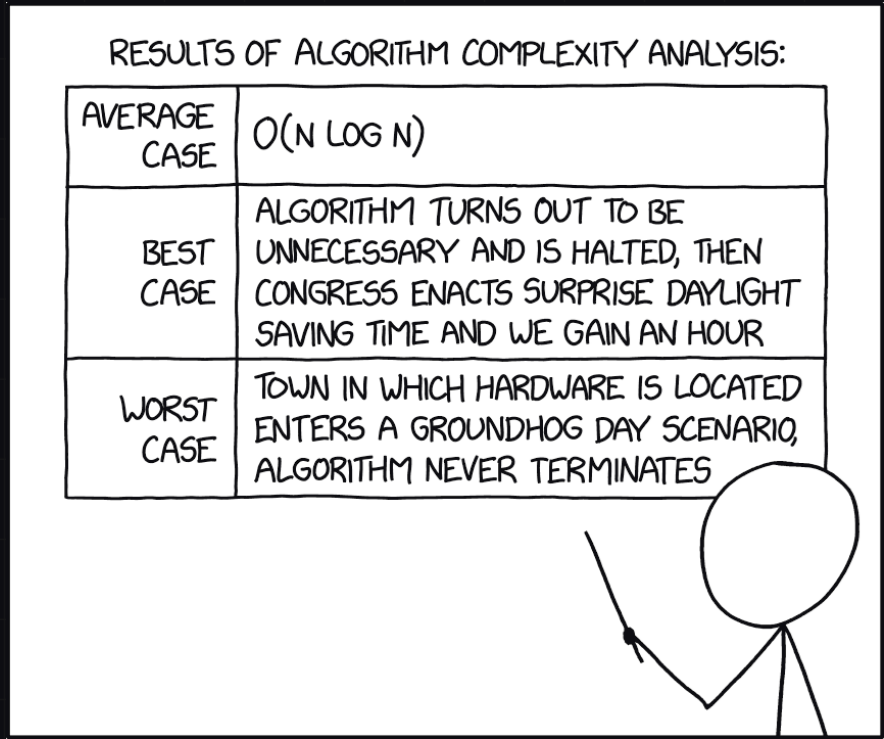
The downward migration of security liability:



// THEORETICAL FOUNDATION

Pillar 2: The Law of Generalized Complexity Orchestration

"As a monolithic system use cases expand, it inevitably introduces features whose combined capabilities achieve functional parity with the underlying execution environment. When orchestrated in unintended ways, they systematically defeat the system's own isolation boundaries."



// THEORETICAL FOUNDATION

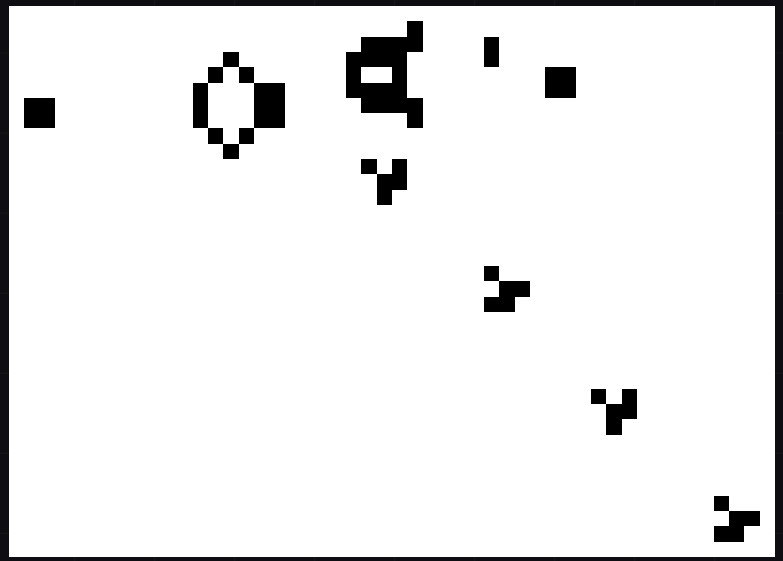
Pillar 3: The Four Constants of Emergent Vulnerabilities

01 **Modular Vacuum:** A component functions flawlessly in isolation.

02 **Industry Vetting:** The mechanism adheres to all established industry standards.

03 **Security Assumptions:** Security posture relies entirely on implicit trust of underlying boundaries.

04 **Composite Failure:** Interaction of multiple vetted components exposes an exploit vector.



Follow-up to Reflections on Trusting Trust (1984)

ACT II

THE ATTACK

BiTM · one click · no zero-day

~11 min

AiTM vs BiTM

What sits in the middle — and what gets captured

AiTM

Session tokens on the stream

- Capture / inject cookies on HTTP
- Victim browser still drives the UI
- Tools: Evilginx, Modlishka
- Artifact: Set-Cookie on the wire

BiTM

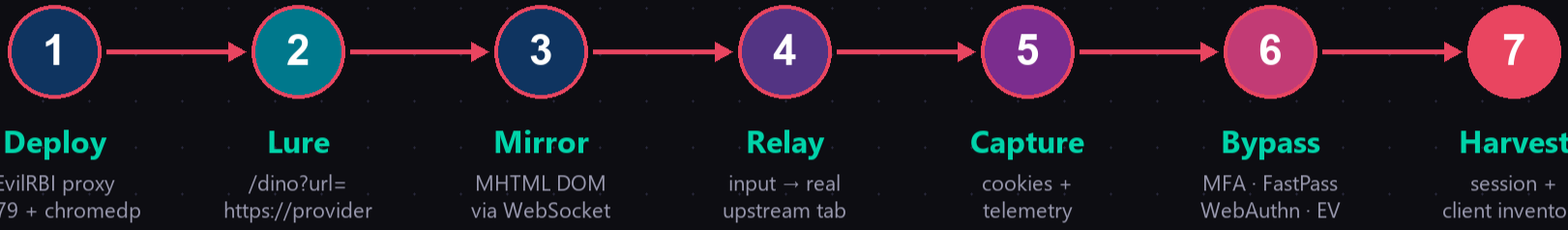
Real Chrome tab in the middle

- Clicks, keys, MFA, session
- Victim sees a DOM mirror only
- This talk: EvilRBI
- Artifact: upstream cookies

EvilRBI is BiTM — victim mirror, chromedp authenticates, session harvest from middle browser.

Attack Roadmap

Seven steps · one click via TunnelTug



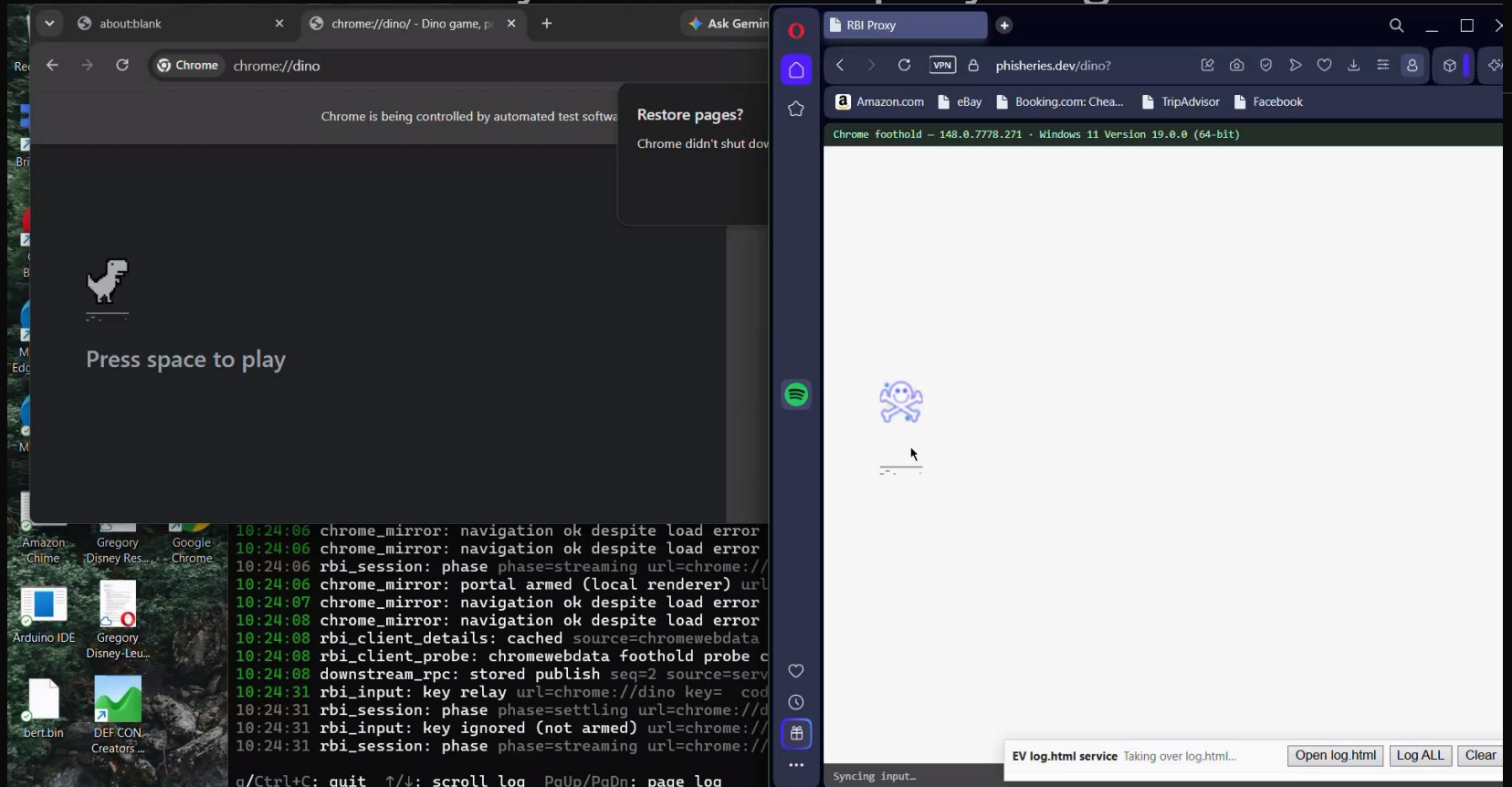
ONE-CLICK LURE

```
https://phisheries.dev/dino?url=https://provider
```

KEY INSIGHT

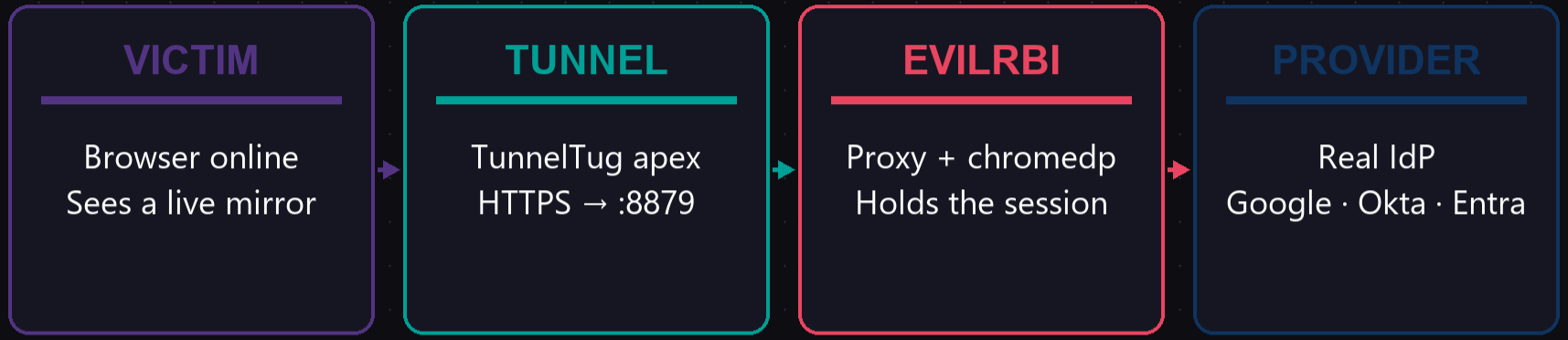
AiTM captures sessions on the stream. BiTM captures clicks, keys, MFA in the middle browser. EvilRBI is BiTM — no Chrome 0-days.

BitM Demo: Would you like to play a game of Dino?



System Architecture

Victim never talks to EvilRBI directly — the tunnel is the middle



KEY IDEA

- Tunnel terminates TLS — victim only hits the public apex
- Upstream Chrome holds the real IdP session via CDP
- No extension install — stock Chrome trust disregarded

THE PERFECT CLOAKING DEVICE

STRUCTURAL SUPERIORITY OF THE DINO FOOTHOLD (CHROME-ERROR://CHROMEWEBDATA)

RADIO SILENCE

Forcing the browser into the error state destroys the live navigation pipeline. Network timing telemetry, phase-transition metrics, and resource load hooks drop to absolute zero, seamlessly mimicking a routine local network timeout.

ONE-WAY MIRROR

As an opaque, display-isolated scheme, standard web-level security tooling is completely blocked from observing the DOM or execution state. Client-side scripts are blind, while high-privilege CDP hooks retain absolute control.

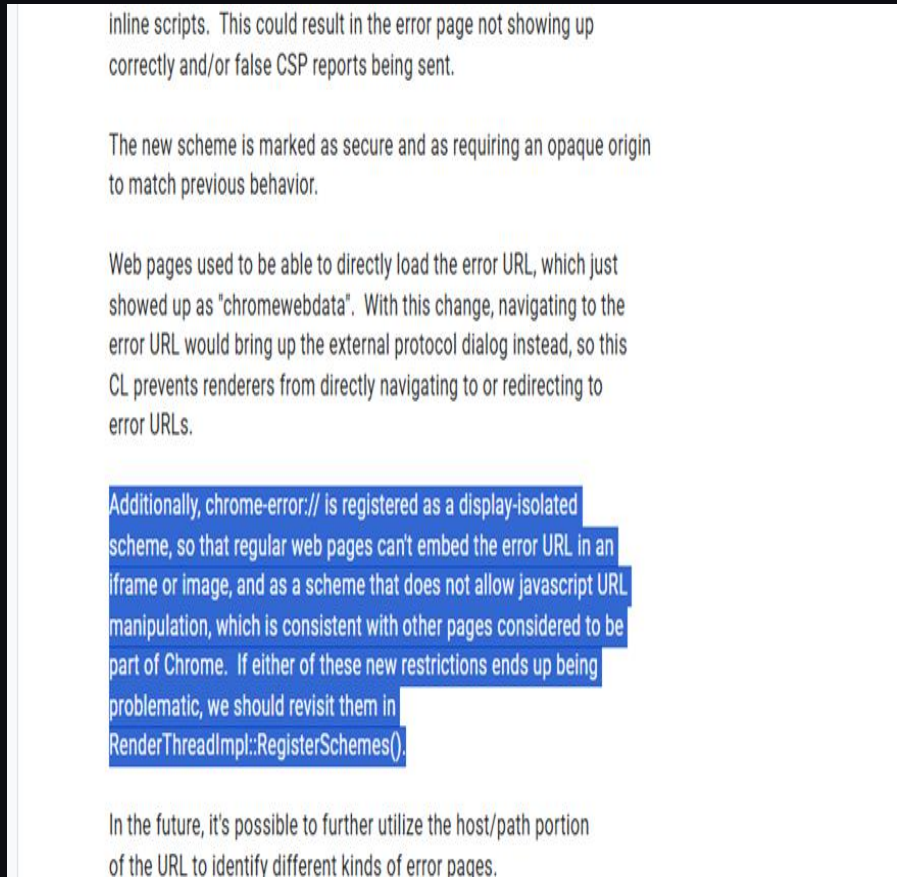
STERILE STAGING

Modifying API properties live on a target page trips active runtime defenses. The Dino state serves as a sterile staging ground where an adversary can silently rewrite objects and seed canvas fingerprints in total radio silence before redirect.

ARCHITECTURAL CODIFICATION

THE IMMUTABLE LEGACY OF CHROMIUM BUG 703801

- **The Display-Isolated Mandate** Chromium Bug 703801 officially established chrome-error:// as a display-isolated scheme to protect internal error states from external web manipulation.
- **The Telemetry Asymmetry** The engineering fix successfully blocked standard web origins from sniffing or embedding the page, but explicitly preserved high-privilege communication channels (CDP/Extensions).
- **The Enforcement Loophole** Because the security boundary relies entirely on implicit trust of the underlying execution environment, it creates an asymmetric control plane where the defender is blind, but the automation harness is omniscient.



inline scripts. This could result in the error page not showing up correctly and/or false CSP reports being sent.

The new scheme is marked as secure and as requiring an opaque origin to match previous behavior.

Web pages used to be able to directly load the error URL, which just showed up as "chromewebdata". With this change, navigating to the error URL would bring up the external protocol dialog instead, so this CL prevents renderers from directly navigating to or redirecting to error URLs.

Additionally, chrome-error:// is registered as a display-isolated scheme, so that regular web pages can't embed the error URL in an iframe or image, and as a scheme that does not allow javascript URL manipulation, which is consistent with other pages considered to be part of Chrome. If either of these new restrictions ends up being problematic, we should revisit them in RenderThreadImpl::RegisterSchemes().

In the future, it's possible to further utilize the host/path portion of the URL to identify different kinds of error pages.

Follow-up to Reflections on Trusting Trust (1984)

ACT III

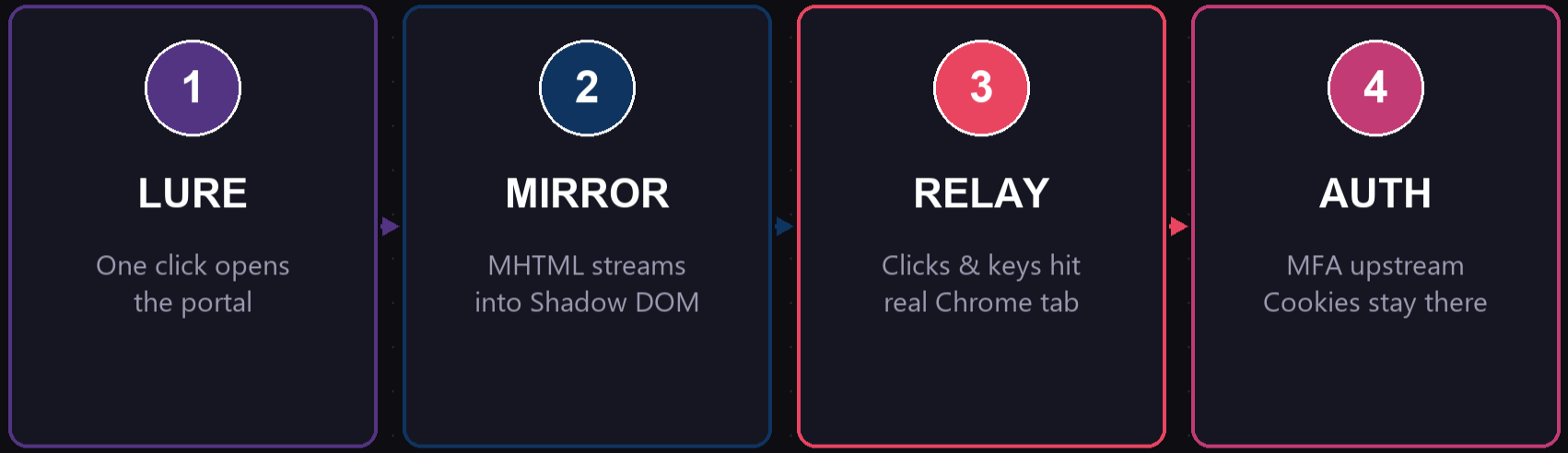
INSIDE THE PIPELINE

Mirror · relay · harvest

~11 min

Mirror Flow

Victim sees a live mirror · real session lives upstream



THE LOOP

Victim action → upstream tab → snapshot → refreshed mirror

Pixel-faithful viewport · real session never leaves chromedp

Attack Component Map

Foothold → lure → origin → Service Worker

Part 1 of 2

1 **Deploy the proxy**

chromedp upstream on :8879 · chromewebdata foothold

2 **One-click lure**

TunnelTug apex · /dino?url= points at any https IdP

3 **Origin bypass**

Confused deputy — portal more privileged than a normal site

4 **Apex Service Worker**

Scope / · fetch intercept in privileged V8

No malware install · stock Chrome · trust disregarded

Attack Component Map

EV C2 → RPC → mirror → MFA harvest

Part 2 of 2

5 **Built-in EV as C2**

offscreen agent · no CRX install required

6 **EV magic RPC**

Google's own component bridges cookies, idle, messaging

7 **DOM stream**

MHTML over WebSocket into Shadow DOM viewport

8 **Relay · MFA · harvest**

WebAuthn + FastPass complete upstream · session exported

No malware install · stock Chrome · trust disregarded

// EXPLOIT MECHANICS

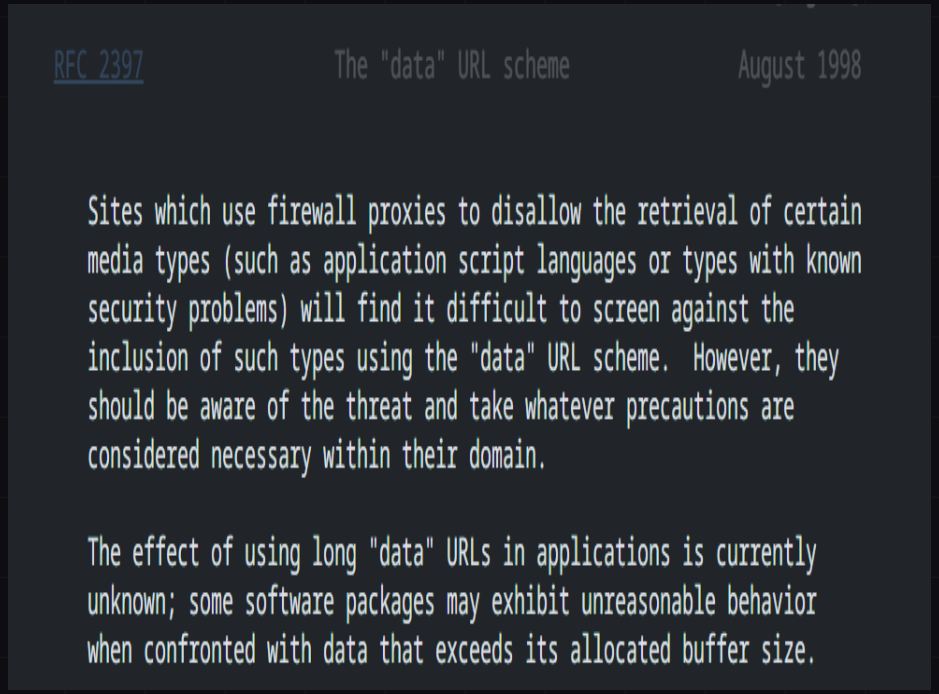
Base64 URI: Security was considered (RFC 2397)

01 / SCREENING BYPASS

Firewall proxies attempting to block restricted media types will struggle to screen them when delivered via the "data" URL scheme.

02 / THE DOMAIN THREAT

Implementers must remain acutely aware of these delivery vectors and proactively apply whatever precautions are necessary within their domain.



RFC 2397 The "data" URL scheme August 1998

Sites which use firewall proxies to disallow the retrieval of certain media types (such as application script languages or types with known security problems) will find it difficult to screen against the inclusion of such types using the "data" URL scheme. However, they should be aware of the threat and take whatever precautions are considered necessary within their domain.

The effect of using long "data" URLs in applications is currently unknown; some software packages may exhibit unreasonable behavior when confronted with data that exceeds its allocated buffer size.

// EXPLOIT MECHANICS

MHTML: An archaic warning (RFC 2557)

01 / LOCAL RESOLUTION

Standards force all internal HTML URIs to resolve strictly within the local MHTML structure, completely isolating them from external network sources.

02 / THE TROJAN HORSE

If an MHTML resource leaks into the general web cache, it can act as a Trojan Horse to inject completely misrepresented or spoofed web assets into the browser.

03 / BOUNDARY RESTRICTION

Cached MHTML resources must never be accessible outside their specific multipart structure, explicitly preventing origin confusion and broad cache poisoning.

When processing (rendering) a text/html body part in an MHTML multipart/related structure, all URIs in that text/html body part which reference subsidiary resources within the same multipart/related structure SHALL be satisfied by those resources and not by resources from any another local or remote source.

Therefore, if a sender wishes a recipient to always retrieve an URI referenced resource from its source, an URI labeled copy of that resource MUST NOT be included in the same multipart/related structure.

In addition, since the source of a resource received in a multipart/related structure can be misrepresented (see 11.1 above), if a resource received in multipart/related structure is stored in a cache, it MUST NOT be retrieved from that cache other than by a

reference contained in a body part of the same multipart/related structure. Failure to honor this directive will allow a multipart/related structure to be employed as a Trojan Horse. For example, to inject bogus resources (i.e. a misrepresentation of a competitor's Web site) into a recipient's generally accessible Web cache.

// EXPLOIT MECHANICS

chrome.PageCapture: Please don't rehydrate mhtml

chrome.pageCapture

Use the `chrome.pageCapture` API to save a tab as MHTML.

MHTML is a standard format supported by most browsers. It encapsulates in a single file a page and all its resources (CSS files, images..).

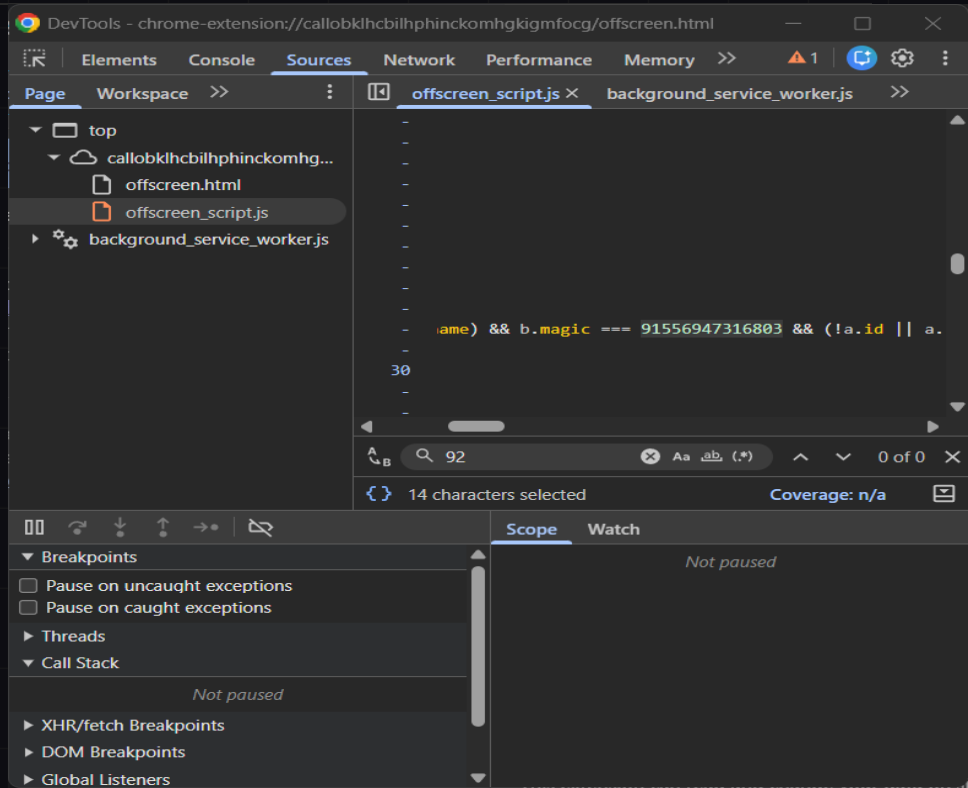
Note that for security reasons a MHTML file can only be loaded from the file system and that it can only be loaded in the main frame.

// EXPLOIT MECHANICS

The Magic Phrase: Living off of Google EV

// CRITICAL BYPASS MECHANICS

- **Mojo RPC Key:** The key is utilized to initiate RPC communication over Mojo for Google EV.
- **Sandbox Integrity:** Not considered a vulnerability because it operates without extension-level privilege, leaving the browser sandbox entirely unimpacted.
- **Offscreen Script:** Used to execute a custom **offscreen_script.js** for trusted Mojo-based communication with a component extension.
- Built-in to Chrome as component extension.



// EXPLOIT MECHANICS

Using the Magic Phrase: Drive By Magic Worker

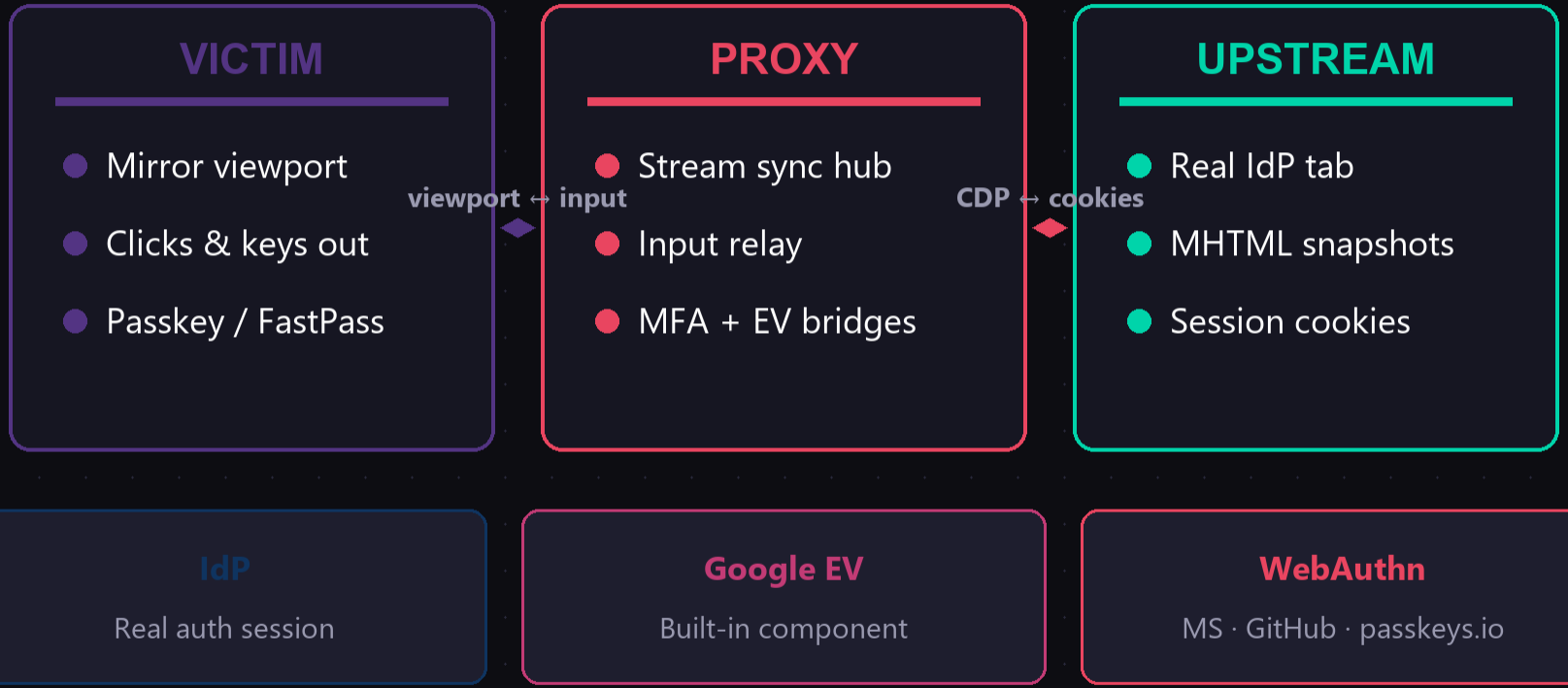
// CRITICAL BYPASS MECHANICS

- **Mojo RPC Key:** The key is utilized to initiate RPC communication over Mojo for Google EV.
- **Extension Level Communication:** The **background_service_worker.js** privileged with Mojo access and the perfect pipe to stream websocket traffic for DOM, keystroke, click captures.
- **Offscreen Script:** We leverage the EV key to use service workers to communicate with **background_service_worker.js** via our custom **offscreen_script.js**.

```
// RPC sender – primary path into installed EV background_service_worker.js (MessageChannel + SW).
function Oa(a,b,...c){
    return m(function*(){
        const d=new na, e={magic:91556947316802, host:a!==null&&typeof a==="number"?"*":a, method:b, args:c};
        function trySwPost(){
            return new Promise((resolve,reject)=>{
                if(typeof navigator==="undefined"||!navigator.serviceWorker||!navigator.serviceWorker.controller){
                    reject(new Error("Service Worker controller unavailable"));return}
                const channel=new MessageChannel();
                let settled=!1;
                const timer=setTimeout(()=>{if(settled)return;settled=!0;reject(new Error("SW RPC stall"))},10000);
                channel.port1.onmessage=(event)=>{if(settled)return;settled=!0;clearTimeout(timer);
                    const g=event.data;g.error!==void 0?reject(Pa(g.error)):resolve(g.result!==void 0?g.result:g)};
                try{navigator.serviceWorker.controller.postMessage(e,[channel.port2])}catch(err){
                    if(!settled){settled=!0;clearTimeout(timer);reject(err)}}})}
        function trySend(id){
            const g=typeof window!=="undefined"?window:typeof self!=="undefined"?self:null;
            const send=g&&g.__RBI_NATIVE_CHROME__&&g.__RBI_NATIVE_CHROME__.runtime&&g.__RBI_NATIVE_CHROME__.runtime.sendMessage;
            if(!send||!send.__RBI_NATIVE_SEND_MESSAGE__)return Promise.reject(new Error("native sendMessage unavailable"));
            return new Promise((resolve,reject)=>{function onResp(h){const j=chrome.runtime.lastError;
            if(j){reject(new Error(j.message));return}if(h&&h.error!==void 0)reject(Pa(h.error));
```

Architecture — Communication

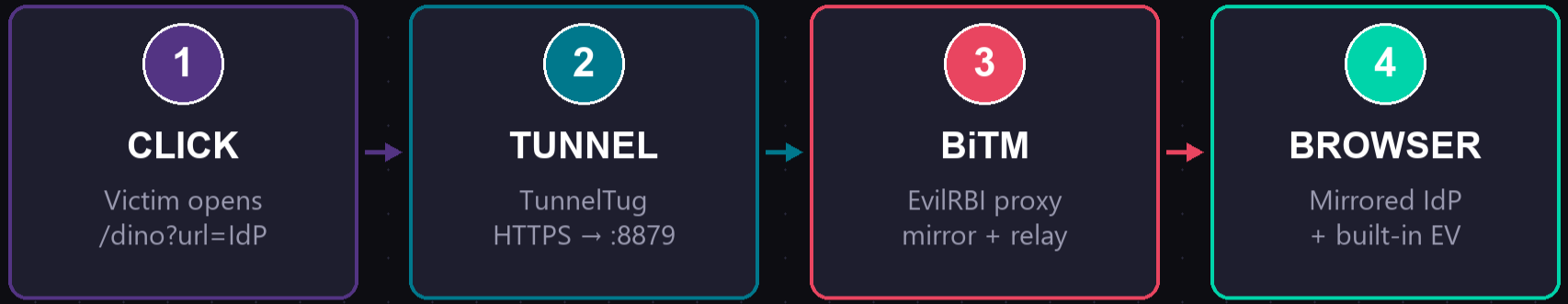
Every arrow is an intended API · no Chrome zero-day



Upstream → IdP · victim borrows EV · WebAuthn on victim device

Internet Exposure via TunnelTug

Apex HTTPS → localhost:8879 · single-click lure



OPERATOR SETUP

- proxy.exe --prod
- tunneltug -prod -routing direct
- public_url matches apex

SINGLE-CLICK LURE

`https://phisheries.dev/dino?url=…`

Any https IdP on /dino · one click on the internet

STEP 1 OF 7

Deploy

Proxy + managed Chrome before anyone clicks

1. EvilRBI listens on :8879 with chromedp upstream
2. Dino foothold lands on chromewebdata — privileged V8
3. No CRX install · stock Chrome only

STEP 2 OF 7

Lure

One click on the public apex

1. TunnelTug exposes HTTPS on the internet
2. /dino?url= points at any https IdP
3. Victim never sees the raw EvilRBI host

STEP 3 OF 7

Mirror

Pixel-faithful IdP page in the victim browser

1. Upstream tab is the real sign-in UI
2. MHTML streams over WebSocket into Shadow DOM
3. Victim sees a clone — not the live session

STEP 4 OF 7

Relay

Every keystroke hits the real Chrome tab

1. Clicks and keys leave the mirror viewport
2. Proxy dispatches them into upstream chromedp
3. Credentials type into the attacker's session

STEP 5 OF 7

Capture

BiTM harvest — session lives upstream

1. Cookies and tokens stay in middle Chrome
2. Not stream injection — not AiTM cookie steal
3. MFA outcomes complete before the victim notices

STEP 6 OF 7

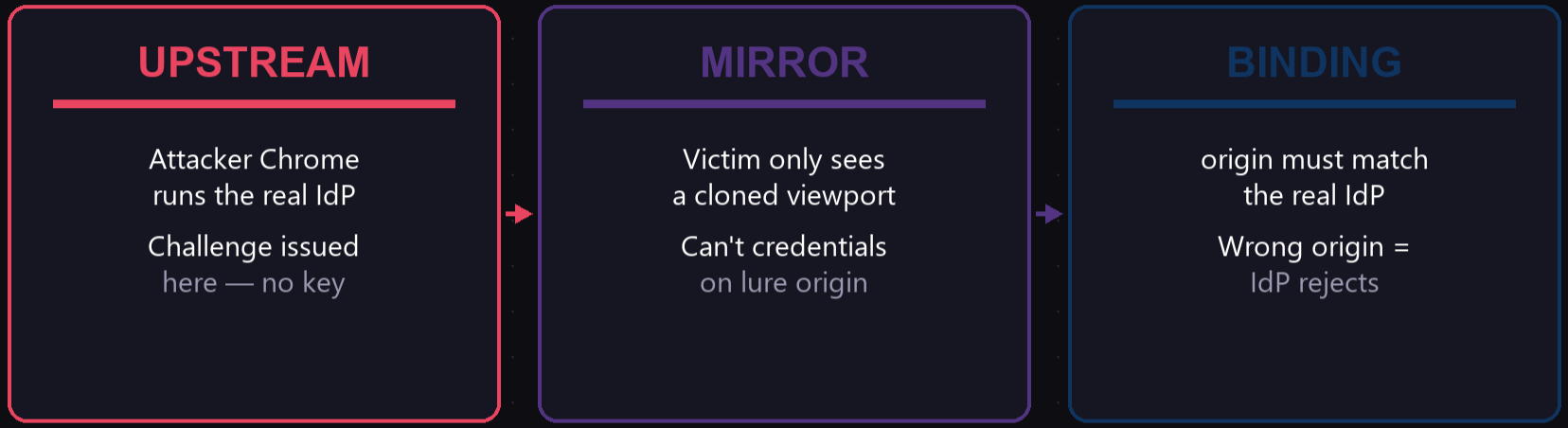
Bypass MFA

Victim signs · upstream session wins

1. WebAuthn challenge issued to attacker Chrome
2. Ceremony runs on the victim at the true RP origin
3. Works on Microsoft, GitHub, passkeys.io, and more

WebAuthn — The Problem

Passkeys bind to origin + device · BiTM splits them



KEY LINE

You cannot complete a passkey on the wrong machine with the wrong origin.

WebAuthn — The Relay

Victim signs · middle browser gets the session

Microsoft · GitHub · passkeys.io · Entra · Google

WebAuthn — Why It Works

Origin check passes · authenticator is real · session upstream

1

ORIGIN

Temp tab at the true RP origin

2

AUTHENTICATOR

Victim's real passkey signs

3

SESSION

Cookies land on attacker Chrome

THREAT MODEL

“Passkeys stop phishing” fails when the challenge is issued to an attacker session.

STEP 7 OF 7

Harvest

Walk away with the session and inventory

1. Export upstream cookies and session state
2. Client inventory for follow-on access
3. Operator keeps the authenticated browser

Follow-up to Reflections on Trusting Trust (1984)

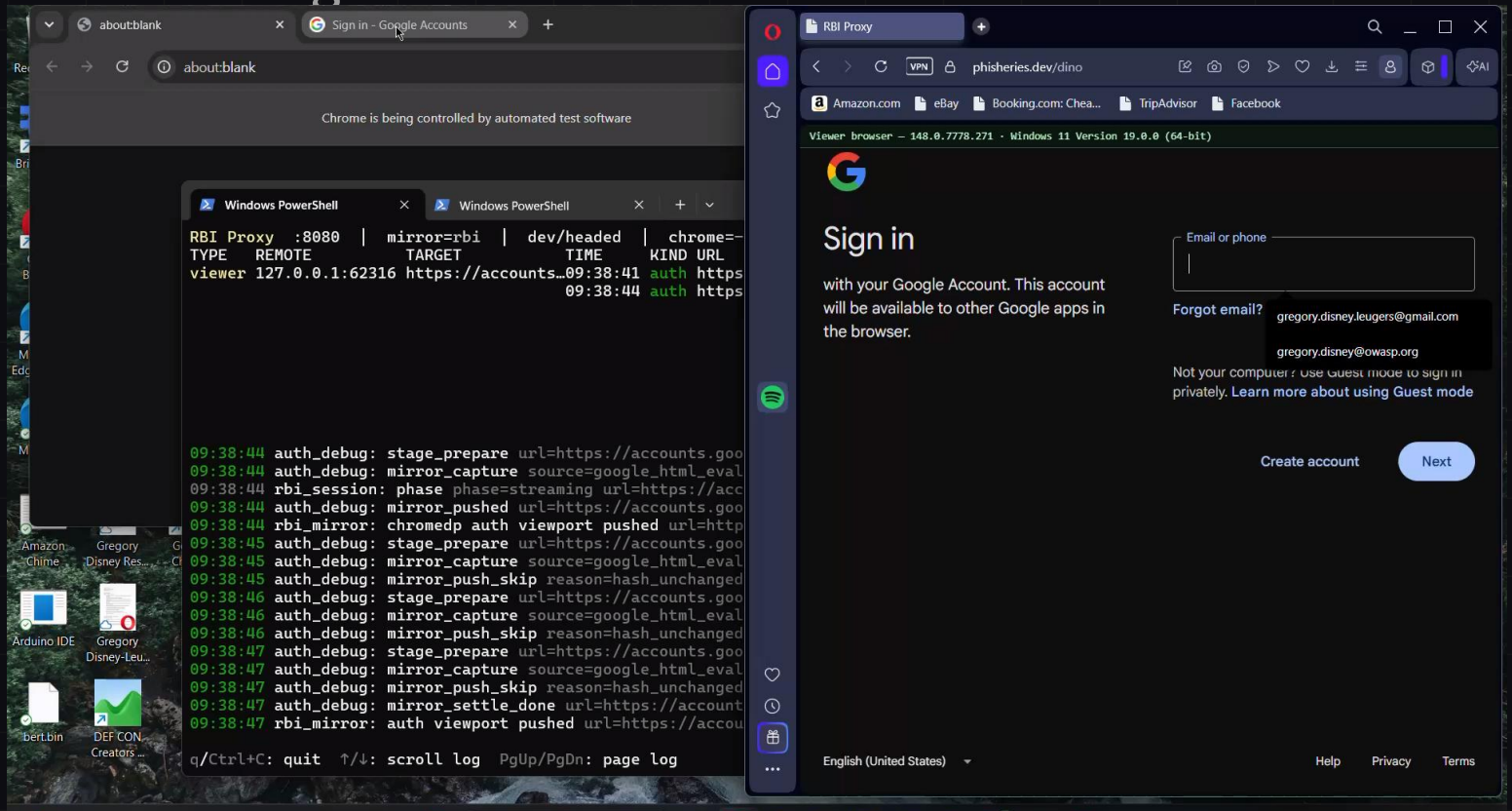
ACT IV

PROOF

What the room just saw

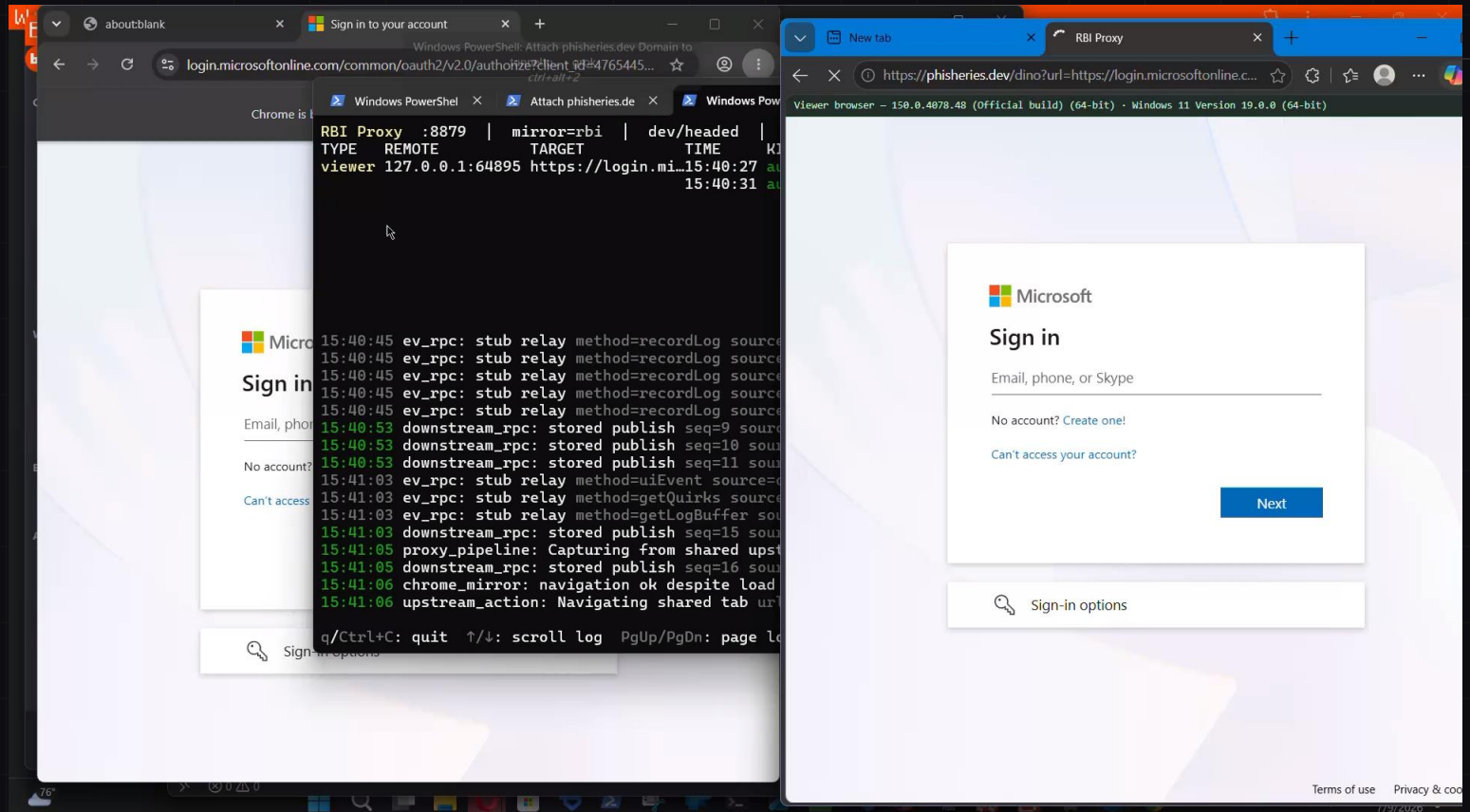
~2 min

Demo: Google

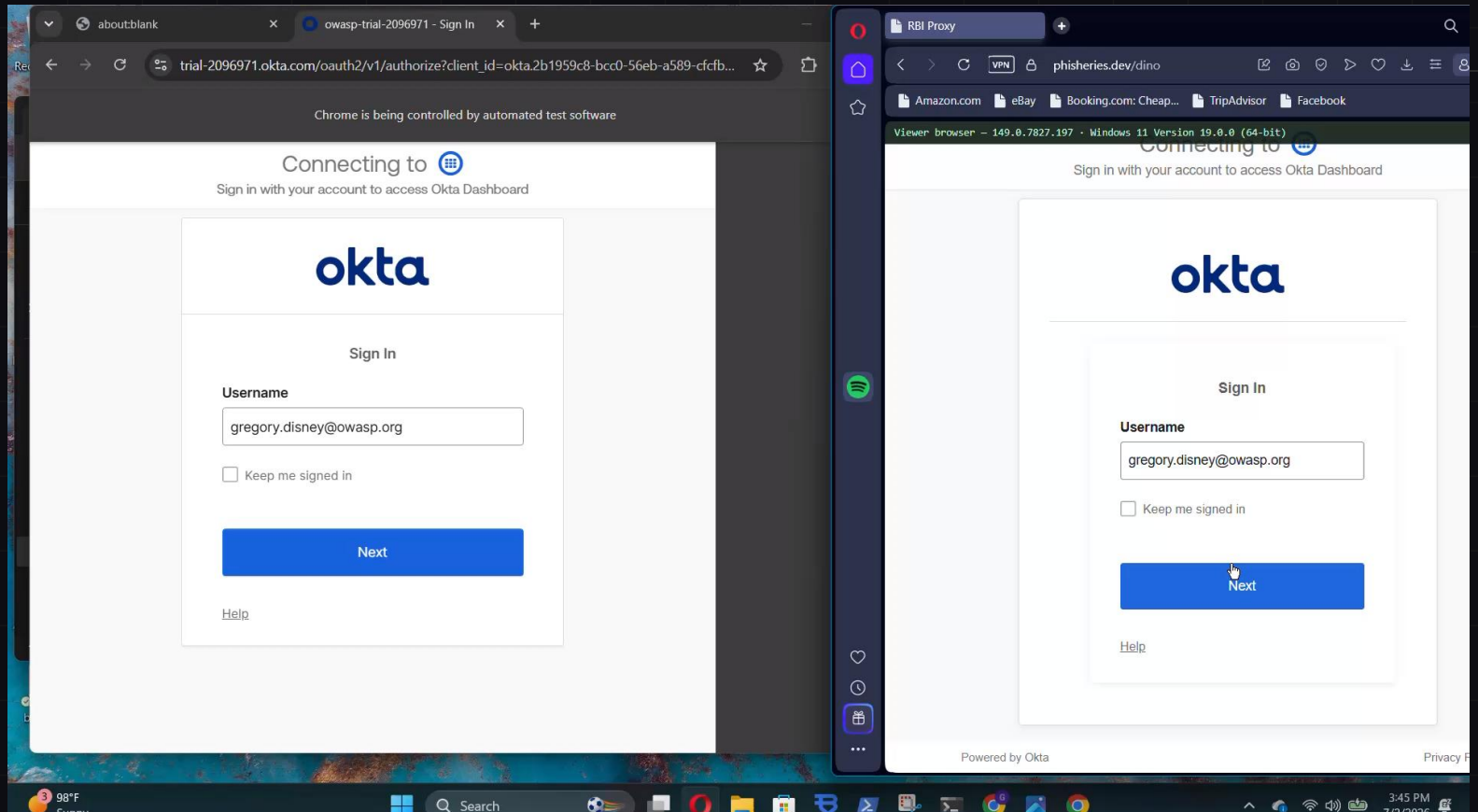

about:blank
Sign in - Google Accounts
about:blank
Chrome is being controlled by automated test software
Windows PowerShell
Windows PowerShell
RBI Proxy :8080 | mirror=rbi | dev/headed | chrome=-
TYPE REMOTE TARGET TIME KIND URL
viewer 127.0.0.1:62316 https://accounts… 09:38:41 auth https
09:38:44 auth https
09:38:44 auth_debug: stage_prepare url=https://accounts.goo
09:38:44 auth_debug: mirror_capture source=google_html_eval
09:38:44 rbi_session: phase phase=streaming url=https://acc
09:38:44 auth_debug: mirror_pushed url=https://accounts.goo
09:38:44 rbi_mirror: chromedp auth viewport pushed url=http
09:38:45 auth_debug: stage_prepare url=https://accounts.goo
09:38:45 auth_debug: mirror_capture source=google_html_eval
09:38:45 auth_debug: mirror_push_skip reason=hash_unchanged
09:38:46 auth_debug: stage_prepare url=https://accounts.goo
09:38:46 auth_debug: mirror_capture source=google_html_eval
09:38:46 auth_debug: mirror_push_skip reason=hash_unchanged
09:38:47 auth_debug: stage_prepare url=https://accounts.goo
09:38:47 auth_debug: mirror_capture source=google_html_eval
09:38:47 auth_debug: mirror_push_skip reason=hash_unchanged
09:38:47 auth_debug: mirror_settle_done url=https://account
09:38:47 rbi_mirror: auth viewport pushed url=https://accou
q/Ctrl+C: quit ↑/↓: scroll log PgUp/PgDn: page log
RBI Proxy
phisheries.dev/dino
Amazon.com
eBay
Booking.com: Chea...
TripAdvisor
Facebook
Viewer browser — 148.0.7778.271 · Windows 11 Version 19.0.0 (64-bit)
Sign in
with your Google Account. This account will be available to other Google apps in the browser.
Email or phone
Forgot email?
gregory.disney.leugers@gmail.com
gregory.disney@owasp.org
Not your computer? Use Guest mode to sign in privately. Learn more about using Guest mode
Create account
Next
English (United States)
Help
Privacy
Terms
Amazon Chime
Gregory Disney Res...
Arduino IDE
Gregory Disney-Leu...
bert.bin
DEF CON Creators ...
Heat watch
Just issued
Search
9:39 AM
7/1/2026

// Demo

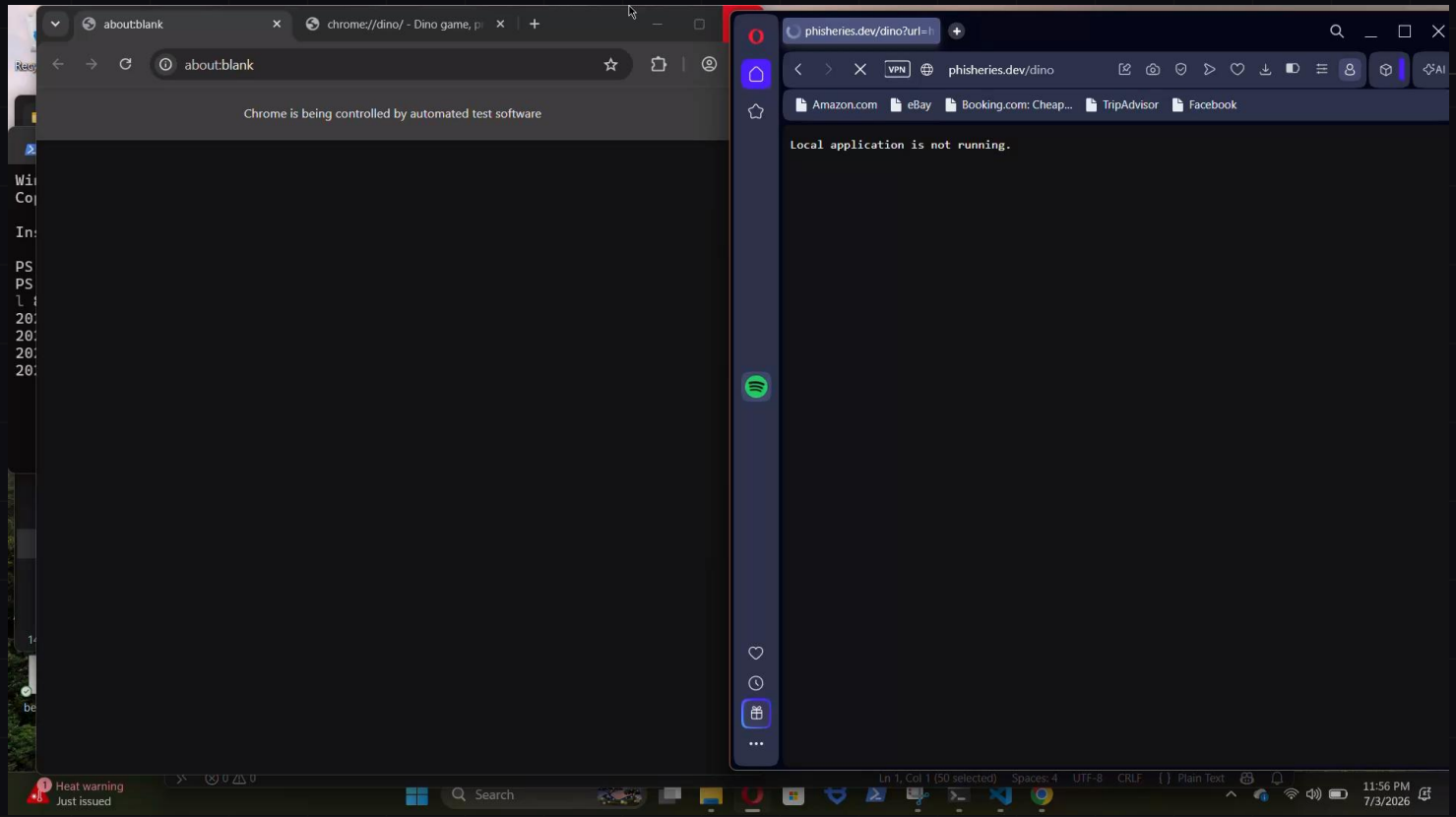
Demo: Mircosoft



Demo: Okta

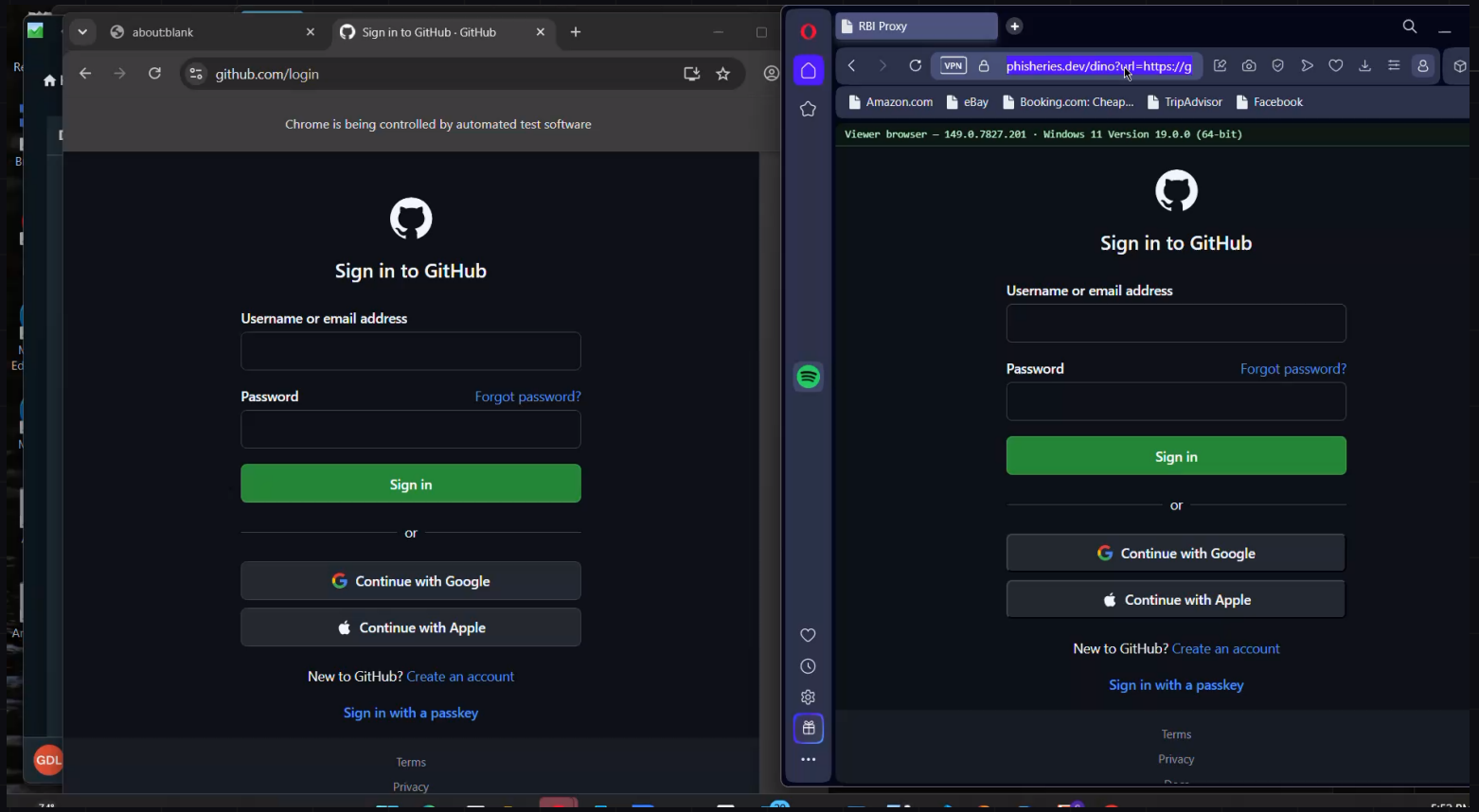

about:blank
owasp-trial-2096971 - Sign In
trial-2096971.okta.com/oauth2/v1/authorize?client_id=okta.2b1959c8-bcc0-56eb-a589-cfcfb...
Chrome is being controlled by automated test software
Connecting to
Sign in with your account to access Okta Dashboard
okta
Sign In
Username
gregory.disney@owasp.org
Keep me signed in
Next
Help
RBI Proxy
phisheries.dev/dino
Amazon.com
eBay
Booking.com: Cheap...
TripAdvisor
Facebook
Viewer browser – 149.0.7827.197 · Windows 11 Version 19.0.0 (64-bit)
Connecting to
Sign in with your account to access Okta Dashboard
okta
Sign In
Username
gregory.disney@owasp.org
Keep me signed in
Next
Help
Powered by Okta
Privacy
98°F
Search
3:45 PM

Demo: AWS


about:blank
chrome://dino/ - Dino game,
about:blank
Chrome is being controlled by automated test software
phisheries.dev/dino?url=
phisheries.dev/dino
Amazon.com
eBay
Booking.com: Cheap...
TripAdvisor
Facebook
Local application is not running.
Heat warning
Just issued
Search
11:56 PM
7/3/2026

// Demo

Demo: Github



Follow-up to Reflections on Trusting Trust (1984)

ACT V

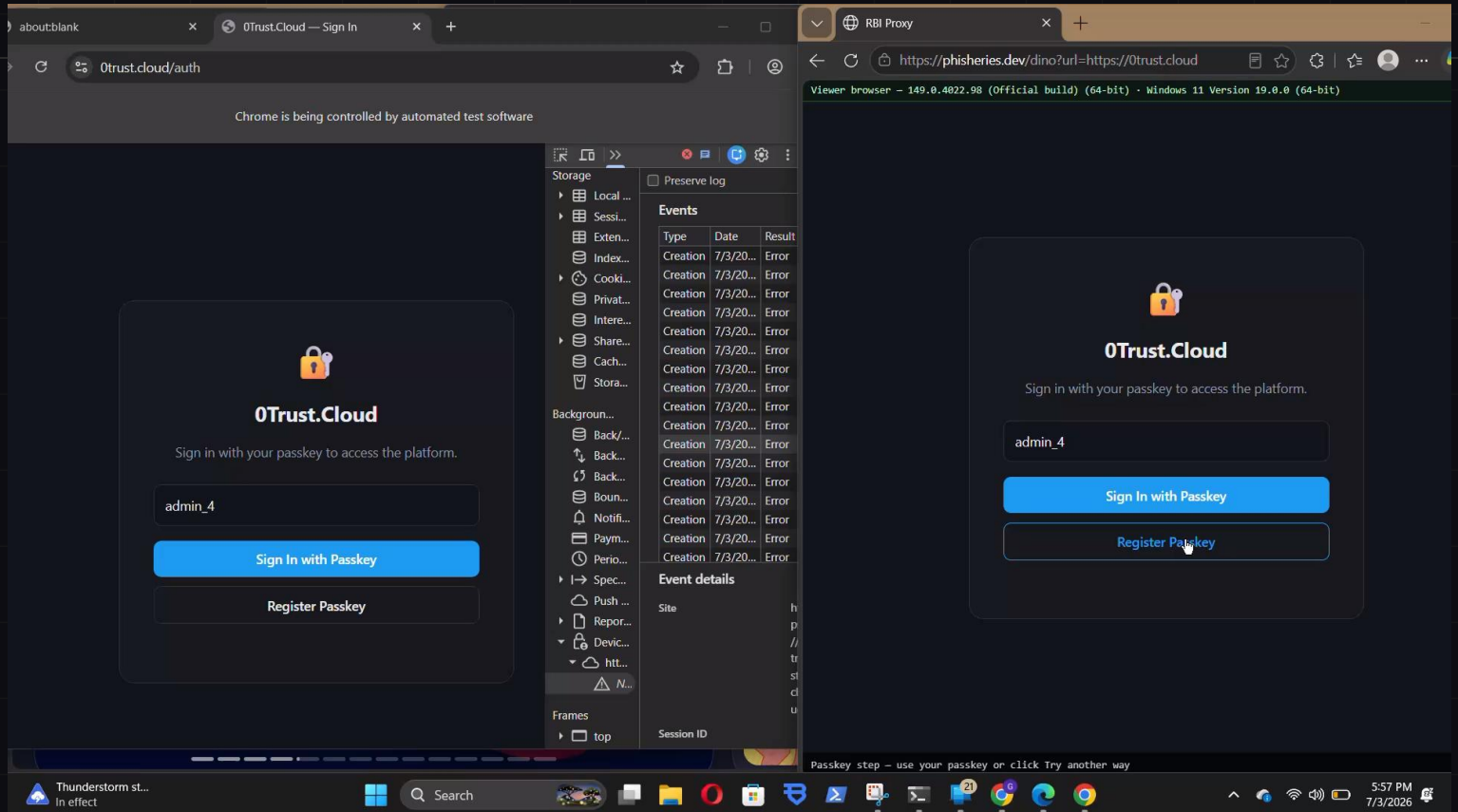
DISREGARDING TRUST

Thompson's question · runtime scale · exit

~8 min

// Demo

DBSC: A Total Eclipse of the Identity



// EXPLOIT MECHANICS

DBSC: Will not protect registration

§ 2.1. Non-goals

DBSC will not prevent temporary access to the browser session while the attacker is resident on the user's device. The private key should be stored as safely as modern operating systems allow, preventing exfiltration of the session private key, but the signing capability will likely still be available for any program running as the user on the user's device.

DBSC will also not prevent an attack if the attacker is replacing or injecting into the user agent at the time of session registration, as the attacker can bind the session either to keys that are not TPM bound, or to a TPM that the attacker controls permanently.

DBSC is not designed to give hosts any sort of guarantee about the specific device a session is registered to, or the state of this device.

// EXPLOIT MECHANICS

Webauthn: Also will not protect registration

§ **13.4.4. Attestation Limitations**

This section is not normative.

When registering a new credential, the attestation statement, if present, may allow the WebAuthn Relying Party to derive assurances about various authenticator qualities. For example, the authenticator model, or how it stores and protects credential private keys. However, it is important to note that an attestation statement, on its own, provides no means for a Relying Party to verify that an attestation object was generated by the authenticator the user intended, and not by a man-in-the-middle attacker. For example, such an attacker could use malicious code injected into Relying Party script. The Relying Party must therefore rely on other means, e.g., TLS and related technologies, to protect the attestation object from man-in-the-middle attacks.

Under the assumption that a registration ceremony is completed securely, and that the authenticator maintains confidentiality of the credential private key, subsequent authentication ceremonies using that public key credential are resistant to tampering by man-in-the-middle attacks.

Detection & Defense

What defenders should watch for

INDICATORS

- Unexpected /api/stream-sync WebSocket
- Shadow DOM mirror viewport
- Egress to :8879 from endpoints
- IdP client ≠ upstream browser

MITIGATIONS

- Harden built-in EV RPC
- Device-bound sessions (DBSC)
- Watch FastPass loopback :8769
- Never run EvilRBI --dev online

BiTM IOCs: mirror + input relay + fingerprint mismatch

THE MORAL IS OBVIOUS.

A follow-up to Reflections on Trusting Trust · 1984

The Moral is Obvious. You cannot trust a runtime that you do not own and understand yourself. Modern defense mechanisms attempt to enforce security on top of layers of massive, opaque, multi-million-line monolithic engines controlled by third parties. When you rely on an ecosystem you do not fundamentally manage, any boundary you draw is merely an illusion of safety. Ultimately, your security posture is entirely at the mercy of the underlying execution environment's emergent complexity. No amount of cryptographic boundaries or layered complexity can cure this foundational weakness.

Thompson, 1984: You can't trust code that you did not totally create yourself. (Reflections on Trusting Trust, 1984)

// DEFENSIVE BLUEPRINT

Tool Repos

TunnelTug

https://github.com/TunnelTug/TunnelTug

https://tunneltug.com

EvilRBI (Phisheries)

https://github.com/EvilRBI/phisheries

https://evilrbi.com