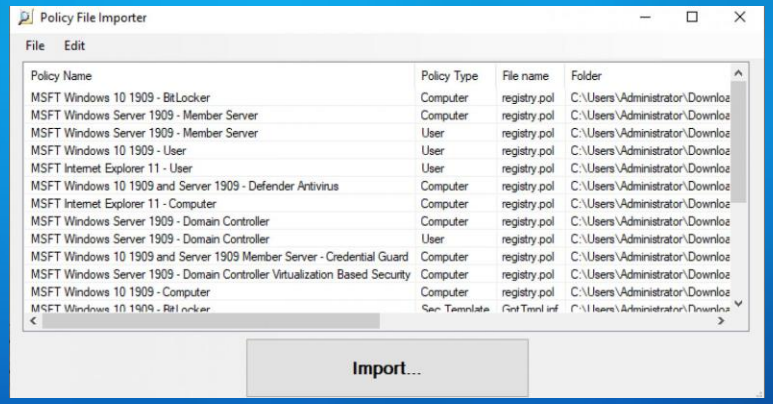
Windows 11: The journey to security by-default

David “dwizzzle” Weston
Director of OS Security
@dwizzzleMSFT

BlueHat IL 2023

Windows 11: Security Strategy

Hardware Security Baseline



Security by-default

Attack big rocks

“Big Rocks”

Running as admin

Unrestricted Platform

Memory Safety

The road to adminless Windows

Adminless

Admin by default one of the core issues in Windows

Removing admin has immense security value

Many security features in Windows (PPL, VBS) compensate for unnecessary Admin privileges. Admin->kernel not a boundary.

Significant OS "debt" with admin

Running as admin introduces user friction through many elevations for Common scenarios

Many win32 apps over privileged

Windows sandboxing focused on UWP, means many "classic" apps
Do not run with least privilege

Win32 App Isolation Goals

Containment

Make it **significantly harder** for attackers to cause **big** damage

Developer Simplicity

Reduce Developer effort to onboard apps

User Transparency

Provide frictionless end user experience for isolated Apps

Win32 App Isolation in Windows 11

App Container

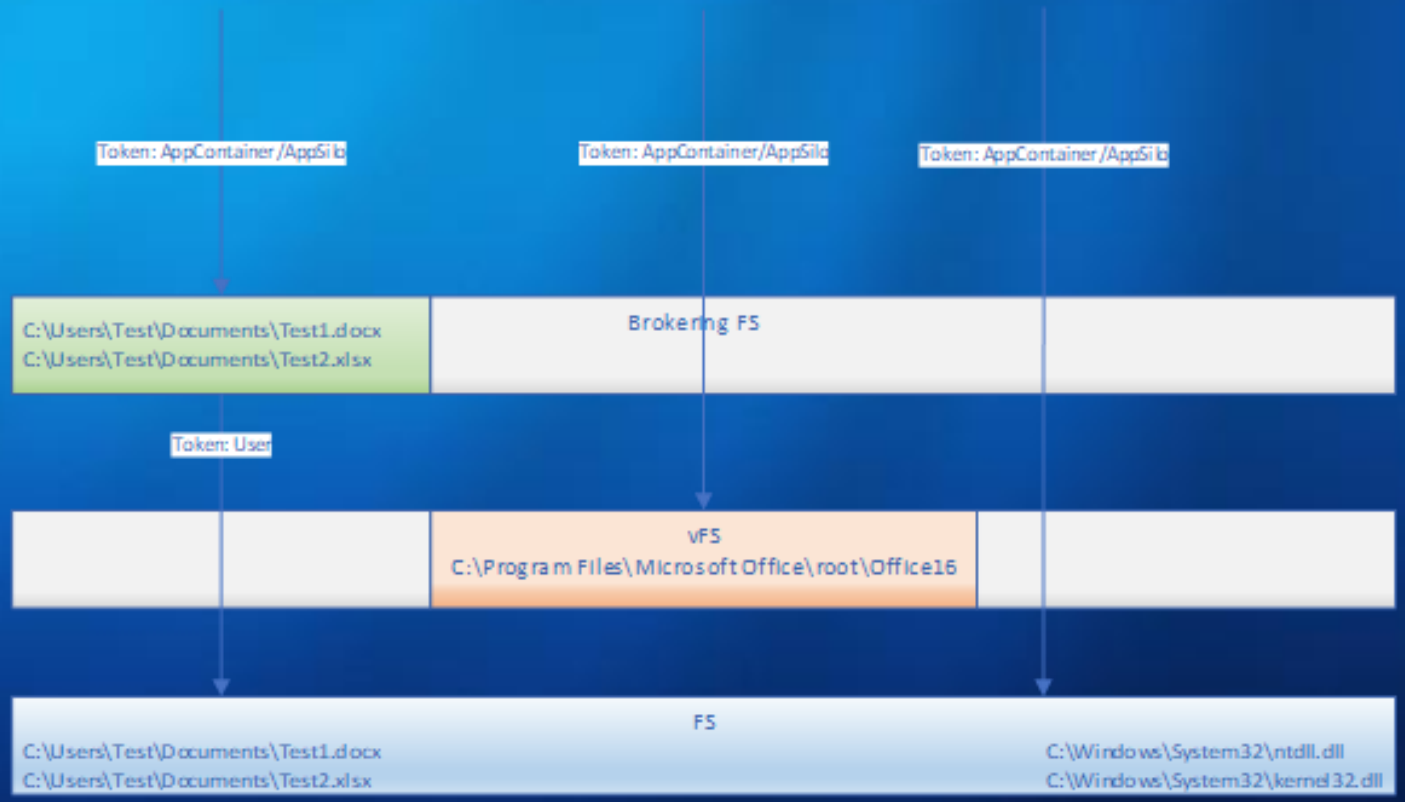
- Execute application at low IL
- Provide isolation

Helium Silos

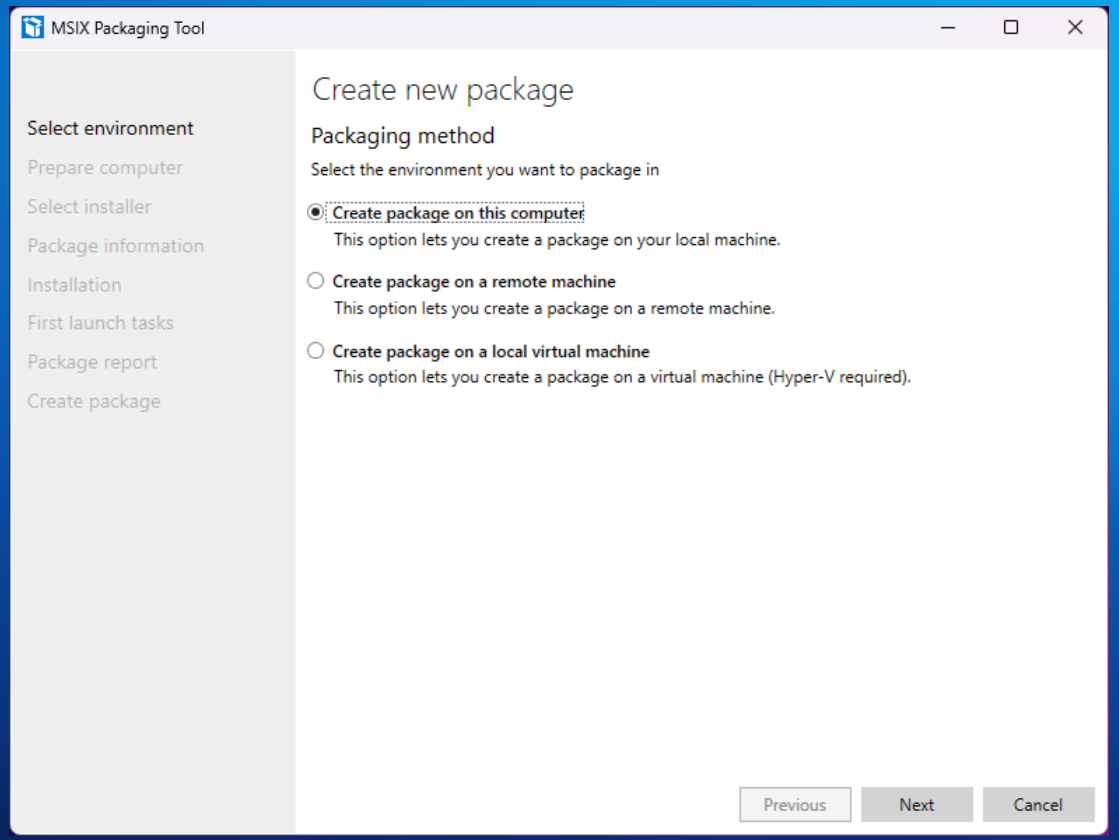
- Used by MSIX
- File system & registry virtualization
- Simplify installation & uninstallation

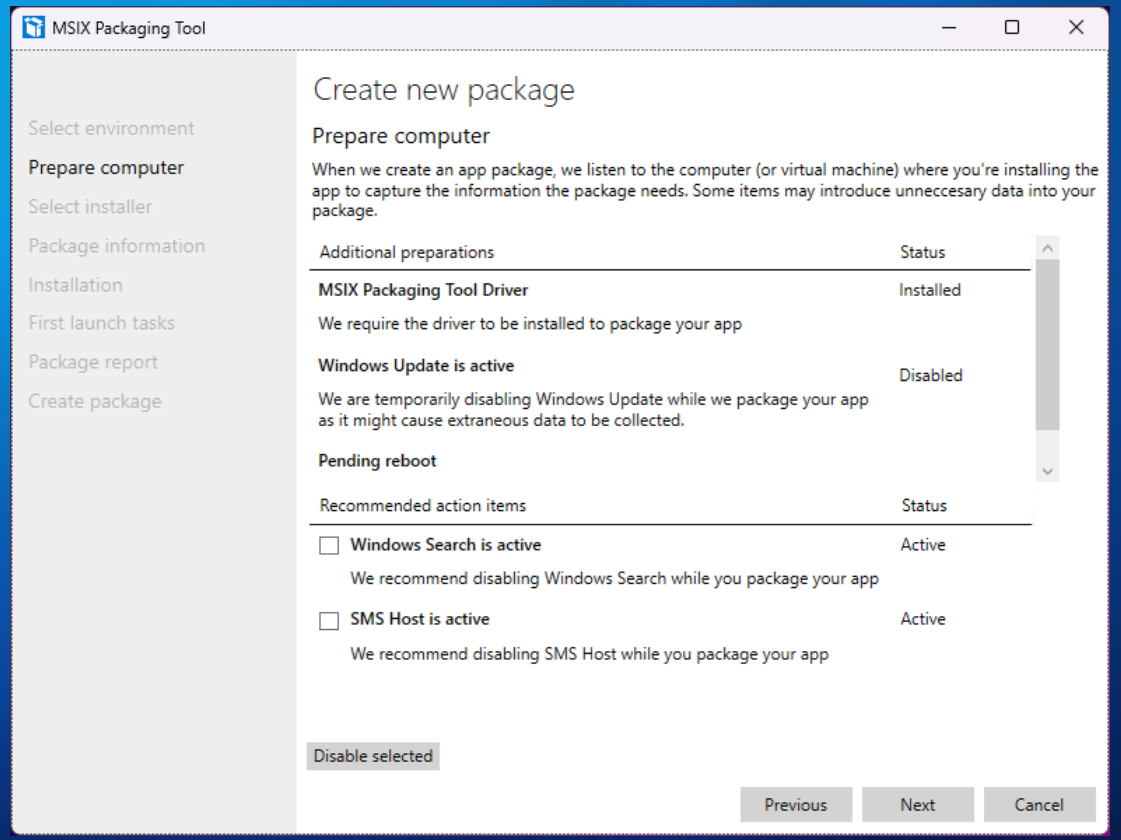
Brokering File System

- Mini-filter driver
- Manage access to user files per App

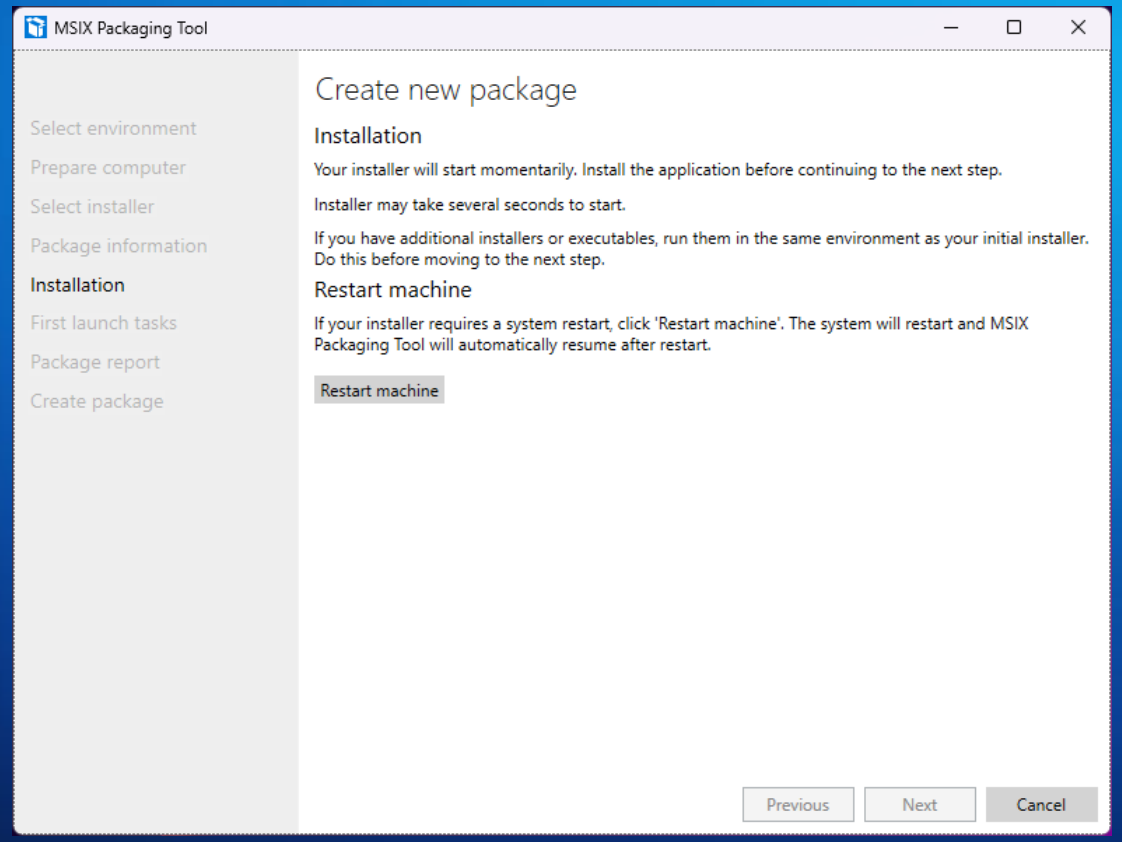


Packaging your Existing win32





Packaging your Existing win32 cont.



Application package manifest – before profiling

```
<Package xmlns="http://schemas.microsoft.com/appx/manifest/foundation/windows10" xmlns:previewsecurity2="http://schemas.microsoft.com/appx/manifest/preview/windows10/security/2"
xmlns:uap="http://schemas.microsoft.com/appx/manifest/uap/windows10" xmlns:uap10="http://schemas.microsoft.com/appx/manifest/uap/windows10/10"
xmlns:desktop7="http://schemas.microsoft.com/appx/manifest/desktop/windows10/7" xmlns:desktop9="http://schemas.microsoft.com/appx/manifest/desktop/windows10/9"
xmlns:rescap="http://schemas.microsoft.com/appx/manifest/foundation/windows10/restrictedcapabilities" xmlns:com="http://schemas.microsoft.com/appx/manifest/com/windows10" Ignorabl
desktop7 desktop9 rescap com previewsecurity2">
  <!-- Package created by MSIX Packaging Tool version: 1.0.0.1 -->
  <Identity Name="AppSilo-NotepadPP" Publisher="CN=Fabrikam Corporation, O=Fabrikam Corporation, L=Redmond, S=Washington, C=US" Version="1.0.0.0" ProcessorArchitecture="x64"/>
  <Properties>
    <DisplayName>AppSilo Notepad++</DisplayName>
    <PublisherDisplayName>Fabrikam Corporation</PublisherDisplayName>
    <Description>Notepad++ AppSilo</Description>
    <Logo>Assets\StoreLogo.png</Logo>
    <uap10:PackageIntegrity>
    ...
    </uap10:PackageIntegrity>
  </Properties>
  <Resources>
  ...
  </Resources>
  <Dependencies>
    <TargetDeviceFamily Name="Windows.Desktop" MinVersion="10.0.22622.0" MaxVersionTested="10.0.22622.0"/>
  </Dependencies>
  <Applications>
    <Application Id="NOTEPAD" Executable="VFS\ProgramFilesX64\Notepad++\notepad++.exe" uap10:TrustLevel="appContainer" previewsecurity2:RuntimeBehavior="appSilo">
      <uap:VisualElements BackgroundColor="transparent" DisplayName="Notepad++" Square150x150Logo="Assets\NOTEPAD-Square150x150Logo.png" Square44x44Logo="Assets\NOTEPAD-Square44x4
      Description="Notepad++">
        <uap:DefaultTile Wide310x150Logo="Assets\NOTEPAD-Wide310x150Logo.png" Square310x310Logo="Assets\NOTEPAD-Square310x310Logo.png" Square71x71Logo="Assets\NOTEPAD-Square71x71L
      </uap:VisualElements>
      <Extensions>
      ...
      </Extensions>
    </Application>
  </Applications>
  <Capabilities>
    <rescap:Capability Name="runFullTrust"/>
  </Capabilities>
</Package>
```

Application Capability Profiler (ACP)

Problem

Application will lose access to standard user resources once packaged

- Files
- Registry items
- Camera, microphone, location

Solution

Some resources allow access based on capabilities that can be declared by a packaged application

- Find the necessary capabilities
- Declare them in the MSIX package manifest

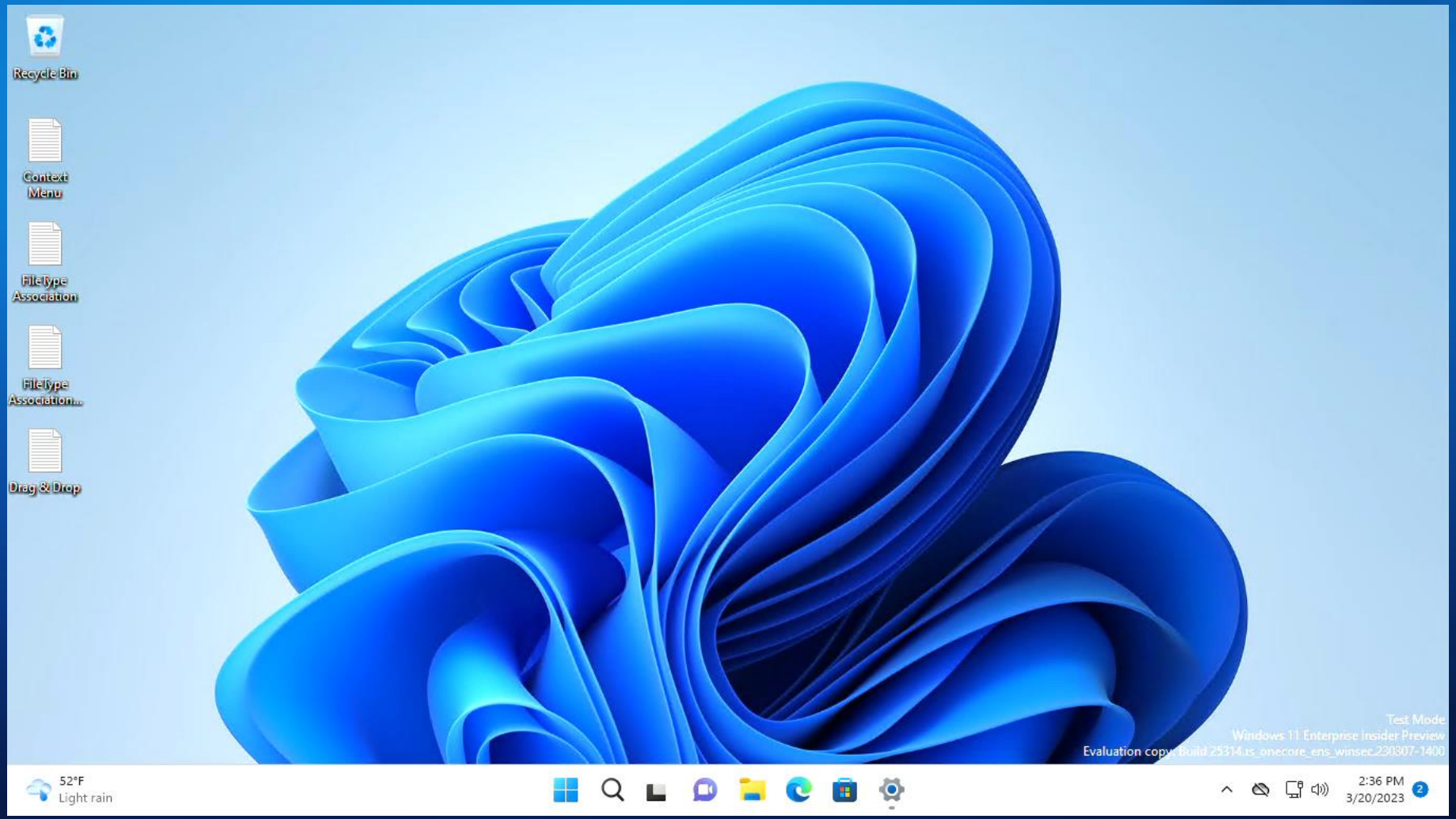
Capability-based access protects resources that are unnecessary to the application package.

Application experience

Win32 Feature Parity

- Implicit Brokering
- Manifest Extension Support
  - FileType Association
  - ComServers
  - Modern and Classic Context Menus
- Drag & Drop
- Printing
- Systray Icons/Shell Notifications
- Revoke Permissions through settings

Win32 Feature Parity Demo



Administrator

Users on Windows Tomorrow

Standard User
Does not have admin rights Cannot elevate and need intervention from IT admin, PAM Protects from malware & user mistakes Used by enterprise /w PAM

Admin-less user
Does not have persistent admin rights and cannot login as admin. Can elevate by itself with just in time, non-persistent admin rights. Uses passwordless strong auth for secure elevation Protects from malware but DOES NOT protects from user mistakes Used by consumers and enterprises

No Privilege User

Least Privilege User

Adminless approach

"Shadow" non-interactive local admin account

Reduce admin privs across OS (e.g. changing color profile)

Windows Hello creds use for elevation

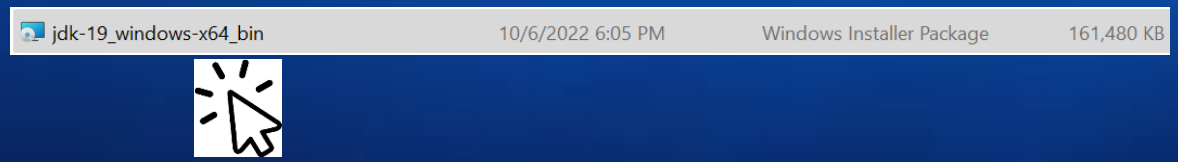
3rd party apps remove unnecessary privs

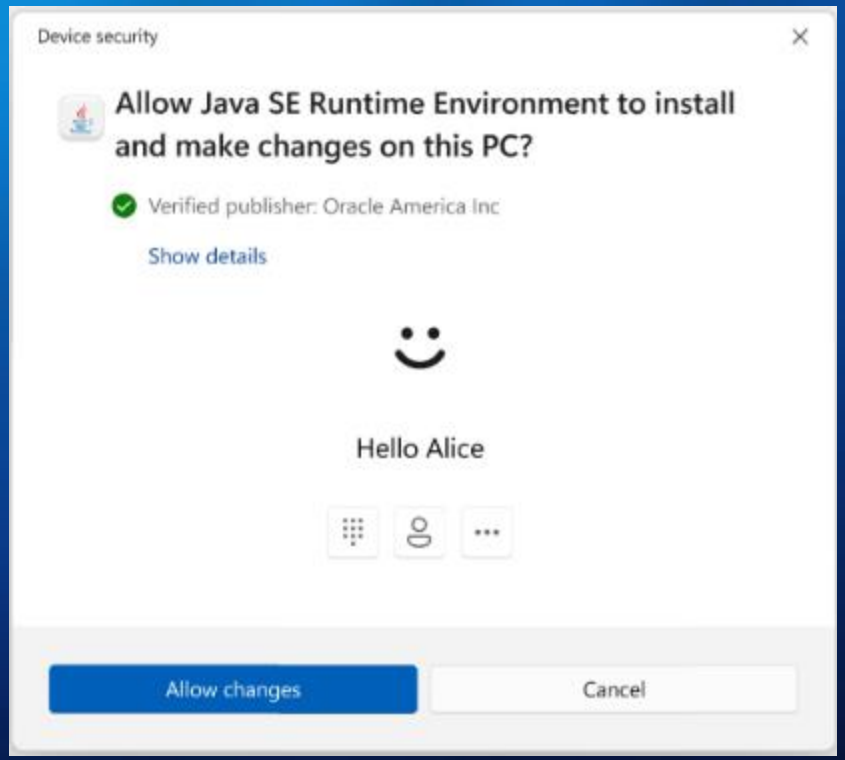
Elevation User Experience

Defaults to Passwordless auth to improve experience

End-user simply launches the app

- The app elevates with secure passwordless experience
- Least privilege admin is used to secure elevation
- User continues as least privilege after task is completed





Adminless

Capabilities Coming to Windows

Win32 App Isolation Preview

Releasing at //BUILD 2023, SDK and tools releasing on GitHub

Adminless Windows

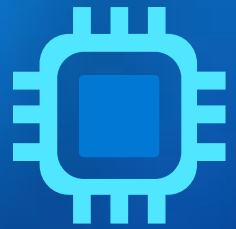
Coming in a Future Windows Release

Protecting the platform

Major Platform Issues

App signing not required on Windows

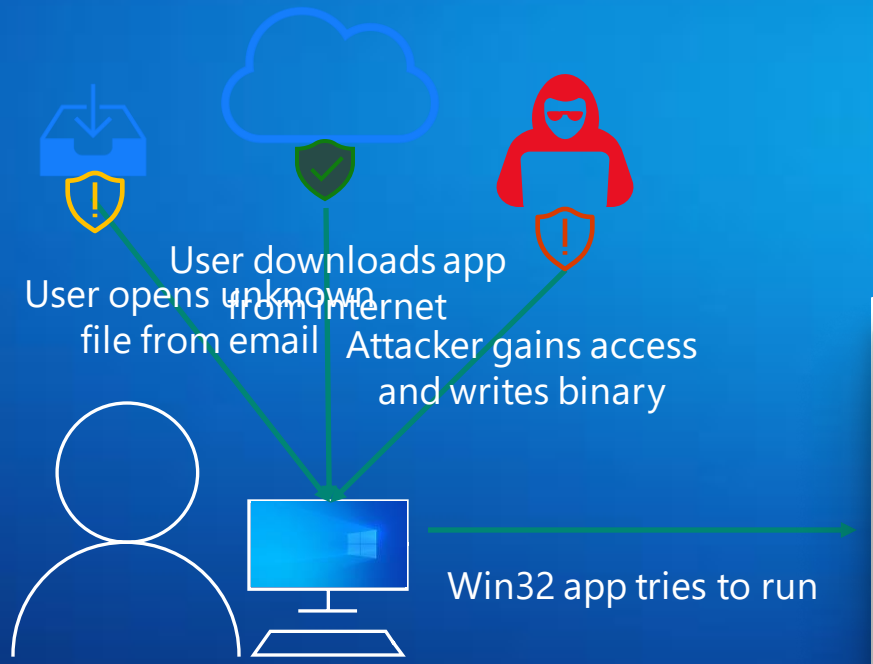
Comprehensive credential protection



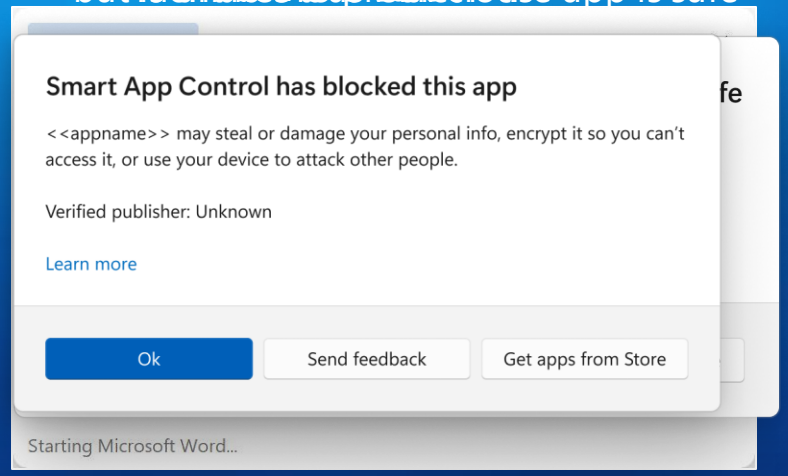
Security processor fundamentals

Smart App Control

AI-driven consumer app control stops malware



Smart App Control evaluates app and but is not able to identify if the app is safe



Windows 11 only runs "safe" apps by default

- Microsoft has validated it e.g. Store-signed apps, Drivers
- Microsoft's cloud AI/ML predicts app is safe
- It is signed by a certificate in the Microsoft root program

Untrusted apps, scripts, and file-based executables, such as malware, are unable to run

Users can opt-out to desktop mode and rely on protection solely from traditional AV

Secure by Default

and disrupting common attack patterns

Smart App Control

App blocked. Full protection day zero. No infections

Defender

Partial protection

5k devices infected

Full protection. No new infections

EXPLOIT LAUNCHED!
TH3KEN.DLL

Hour 0

Backdoor:MSIL/Vigorf.A
Defender Signature updated

BLOCKED BY DEFENDER

Hour 36

Controlled rollout

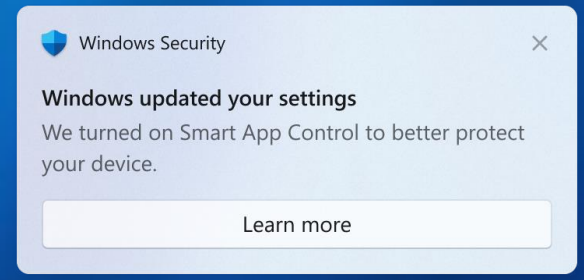
Maintaining the user's positive experience

Hardware Features

Download History

Performance History

NW Performance on similar users



Learning & Personalization Mode

Smart App Control enabled on clean installs in evaluation mode

Cloud AI model determines if the user's device is a good candidate for enforcement based on app usage

Enforcement Mode

Smart App Control enforcement optimized for users most likely to have a positive experience

Pluton Security Processor

Robust Hardware

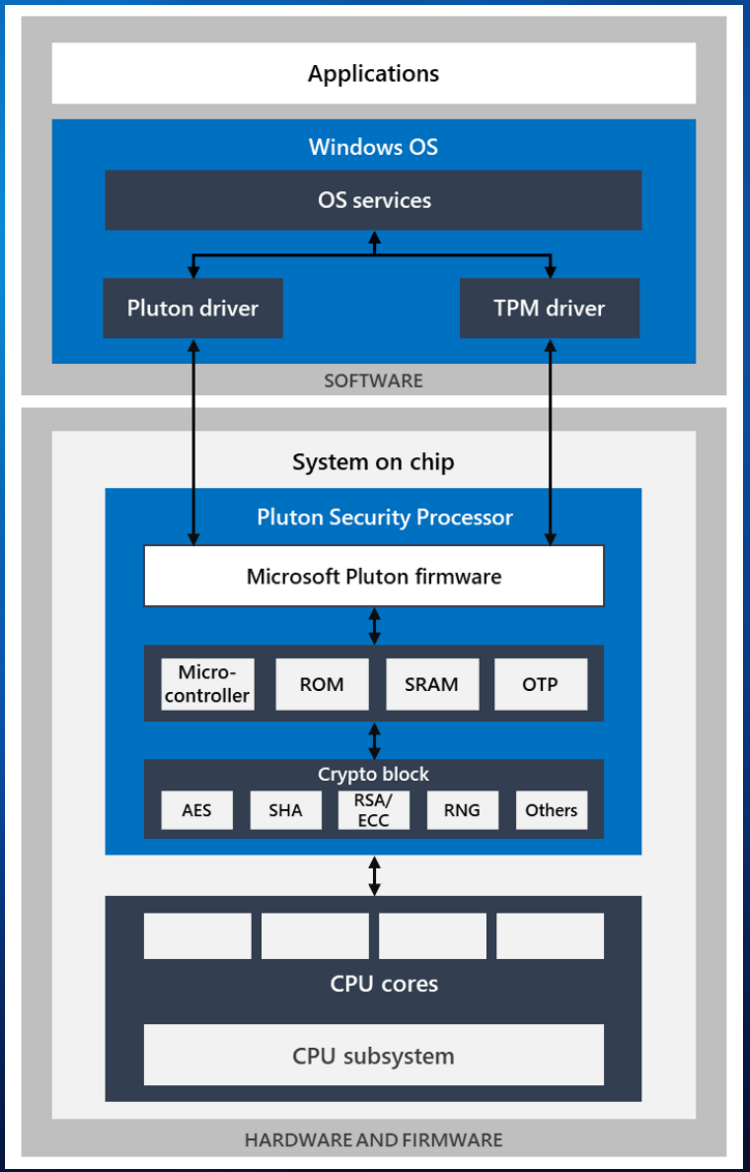
- Dedicated security processor that is embedded inside the larger system-on-chip (SoC) integrated circuit
- Pluton has its own microcontroller core, which boots from its own ROM and then loads the integrity-verified Pluton firmware into dedicated SRAM
- Architecture provides isolation from main CPU cores – side channel attacks against main system DRAM cannot extract data from Pluton SRAM
- Main CPU cores communicate with Pluton using dedicated security hardened hardware interface

Firmware renewability

- OS can load new validly signed firmware from OS disk
- Allows new firmware to be delivered like OS file updates in addition to supporting typical firmware update mechanisms like UEFI capsule updates
- Provides faster method to address security issues

Investments in memory-safe languages

- Aim for high bar for software correctness through proper architecture, careful design, exhaustive testing and internal\external code reviews
- Work underway to incorporate Rust into future Pluton firmware implementations to gradually transition to memory-safe languages



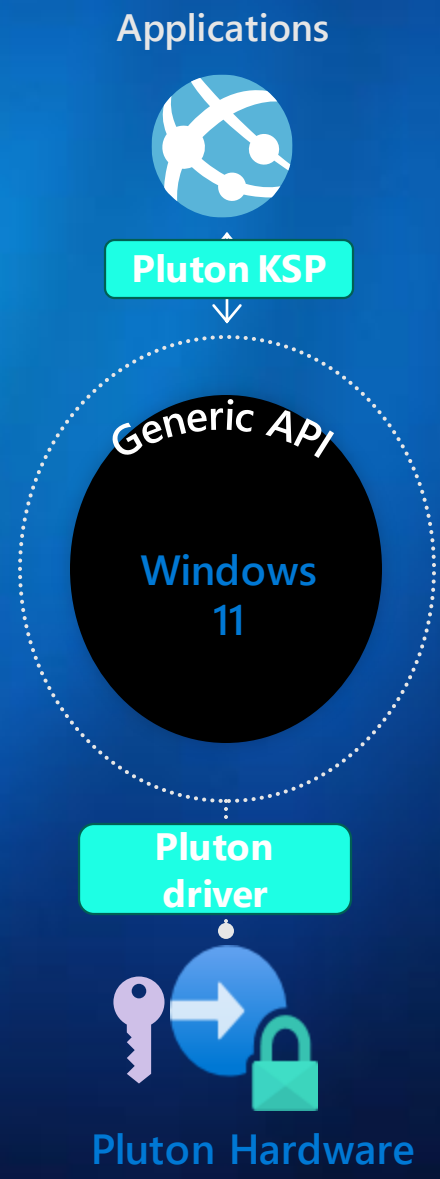
Pluton as a Key Storage Provider

Current protection of credentials is often not adequate against sophisticated attacks as they are not reliable or resilient to attacks (i.e. protected by software or reliability issues or attack vector identified etc.)

Credentials are safeguarded by Pluton security processor which is integrated into silicon

Apart from hardening, Pluton provides a reliable infrastructure for a better user experience

Future OS releases will incorporate a new Pluton Key Storage Provider and integrations planned with Azure Active Directory and Intune

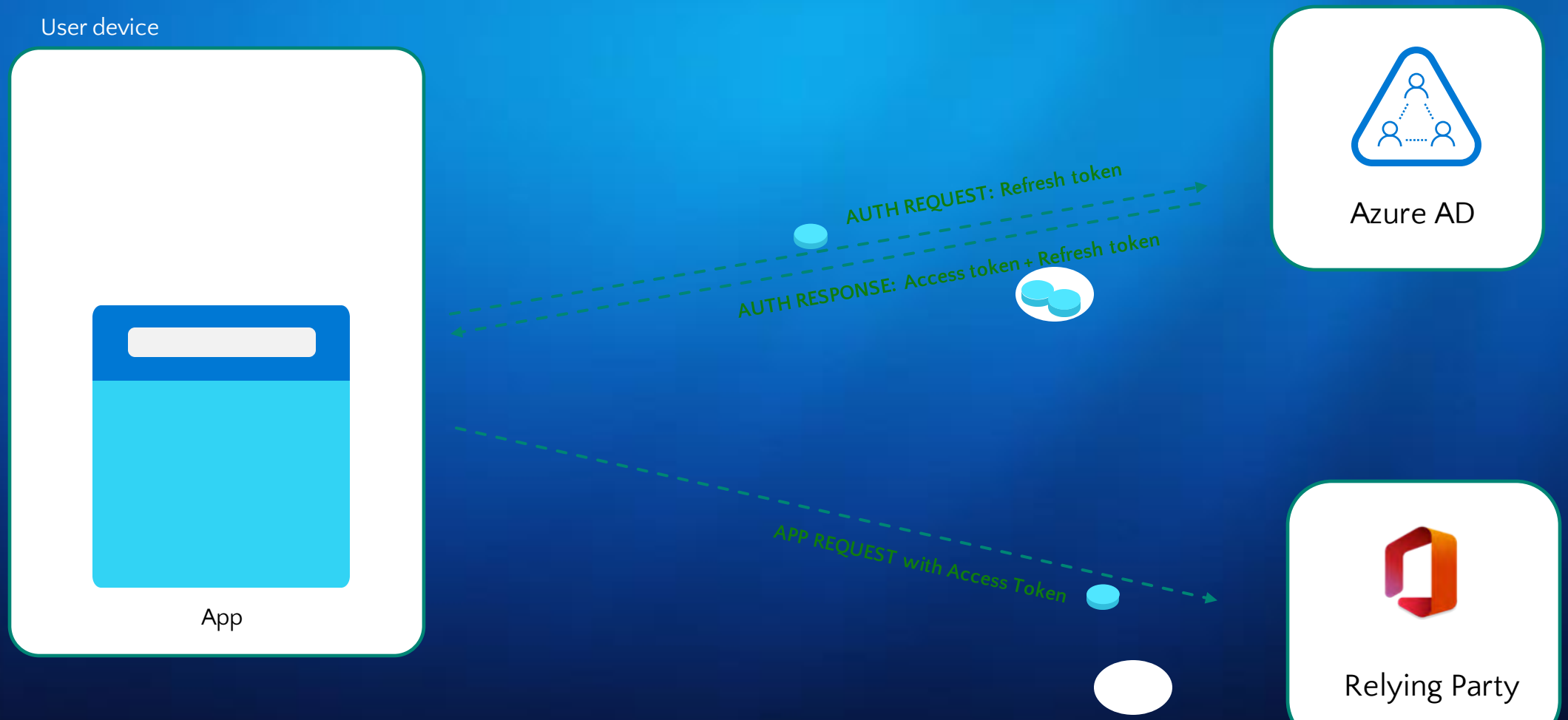


By design, OAuth artifacts are bearer tokens, meaning they are vulnerable to post-breach token theft and replay.

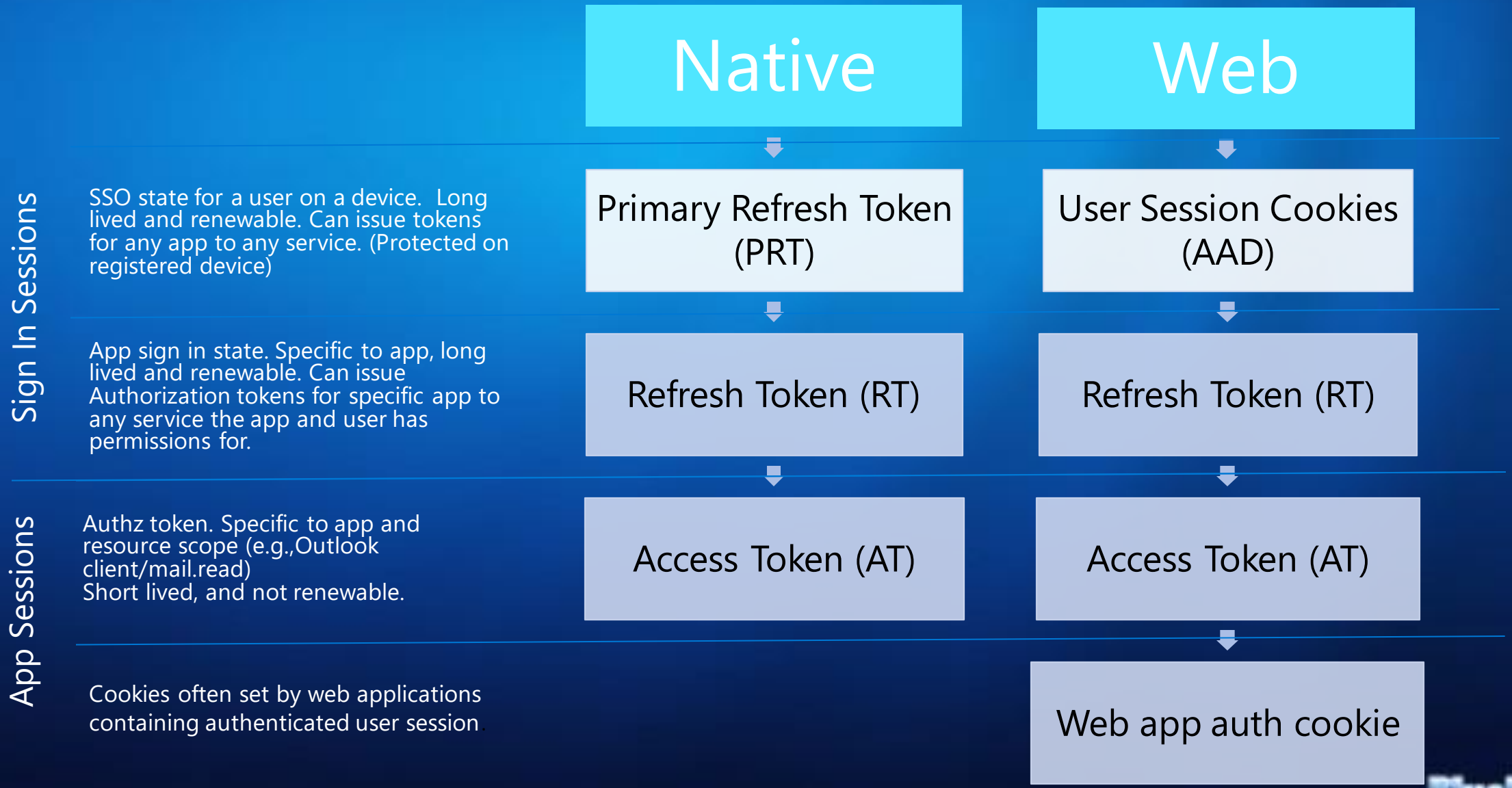
- Relatively rare but growing
  - **Estimate ~60k token replay attacks in Feb 23, up 100% YoY.**
- Damage: full user impersonation, bypassing **CA, strong auth, device and network controls, detection.**

- Huge concerns for **highly regulated customers (FinServ, gov, healthcare)**, esp. in the light of the latest security breaches.

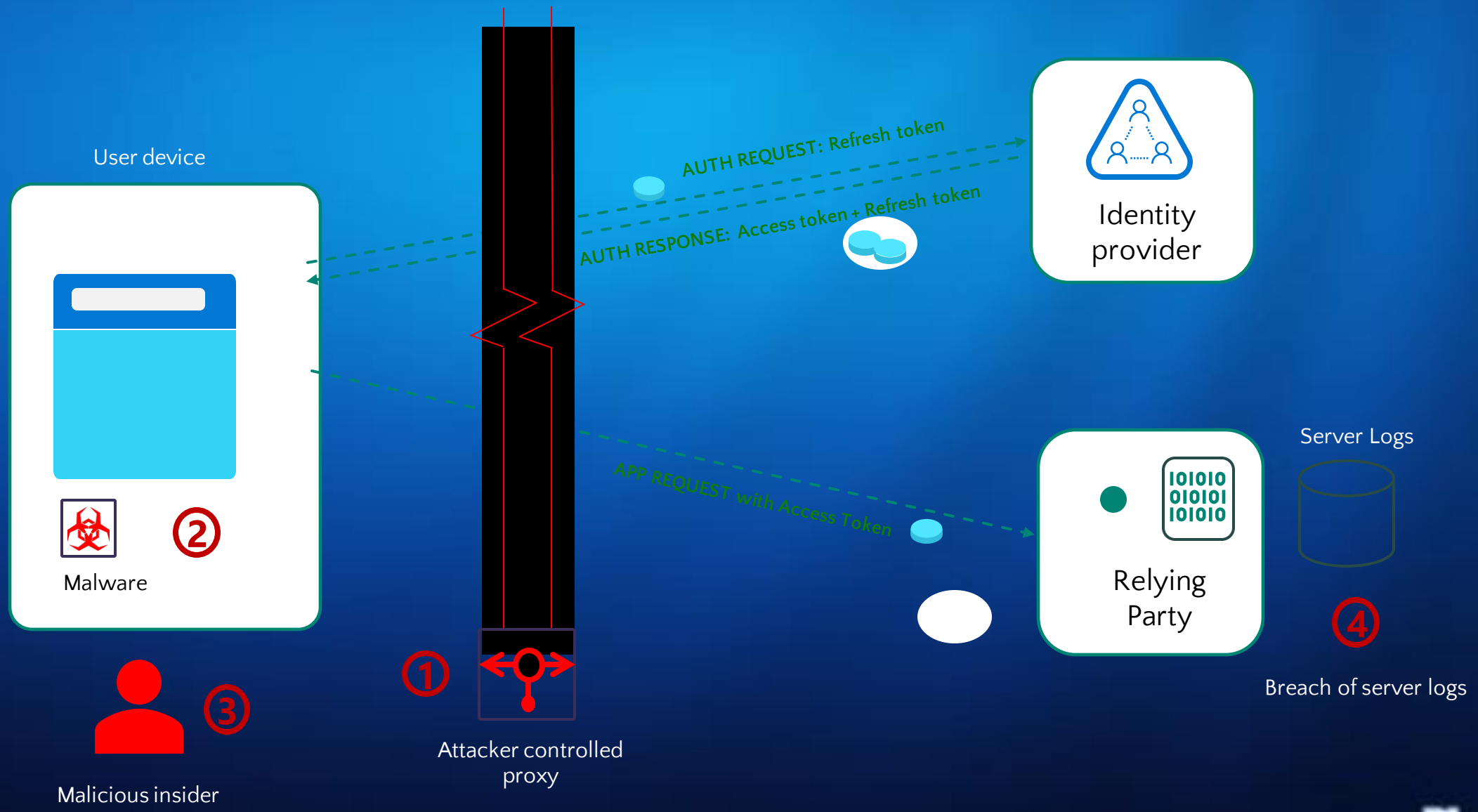
Bearer token request



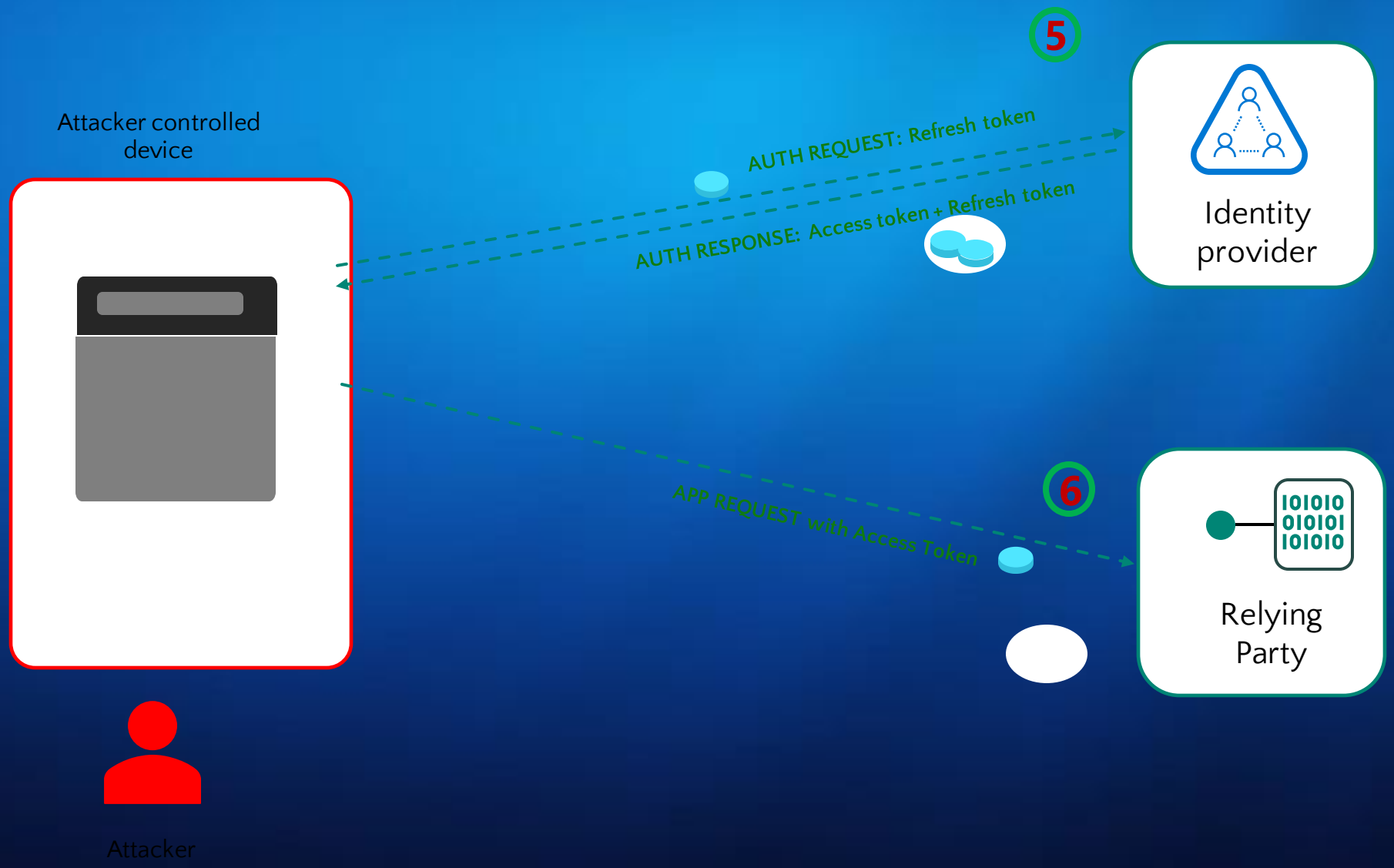
Authentication artifacts



Token theft


User device
Malware
②
Malicious insider
③
①
Attacker controlled proxy
AUTH REQUEST: Refresh token
AUTH RESPONSE: Access token + Refresh token
APP REQUEST with Access Token
Identity provider
Relying Party
Server Logs
④
Breach of server logs

Token Replay



What is Token Protection?

Token Protection makes resources resistant to access from **devices other than to which the user signed in**

- Resists access to a resource using a session artifact stolen from a user's device and replayed from an attacker-controlled device
- Token is cryptographically bound to a device with a binding key and cannot be used without the binding key
- Strength of Token protection depends on how the key is established and how well it's protected.

Solution

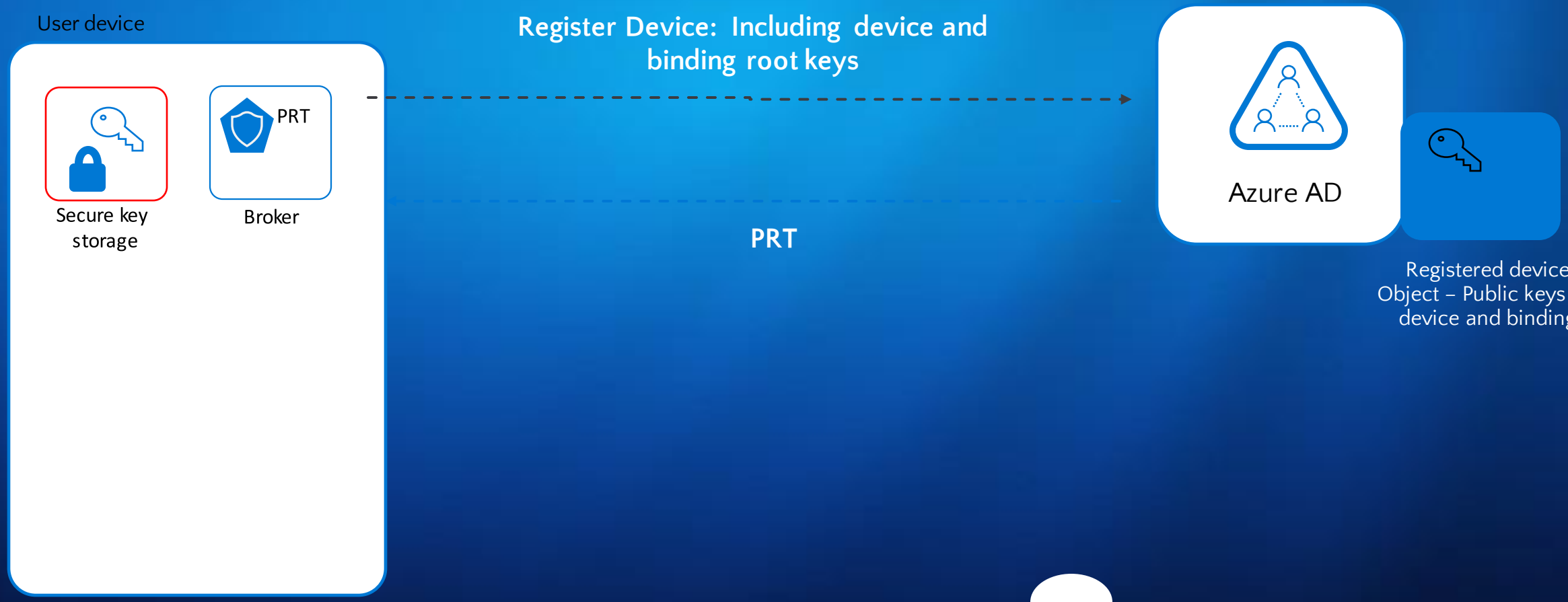
Azure AD Tokens are bound using application proof of possession

Using Azure AD Conditional Access a customer can:

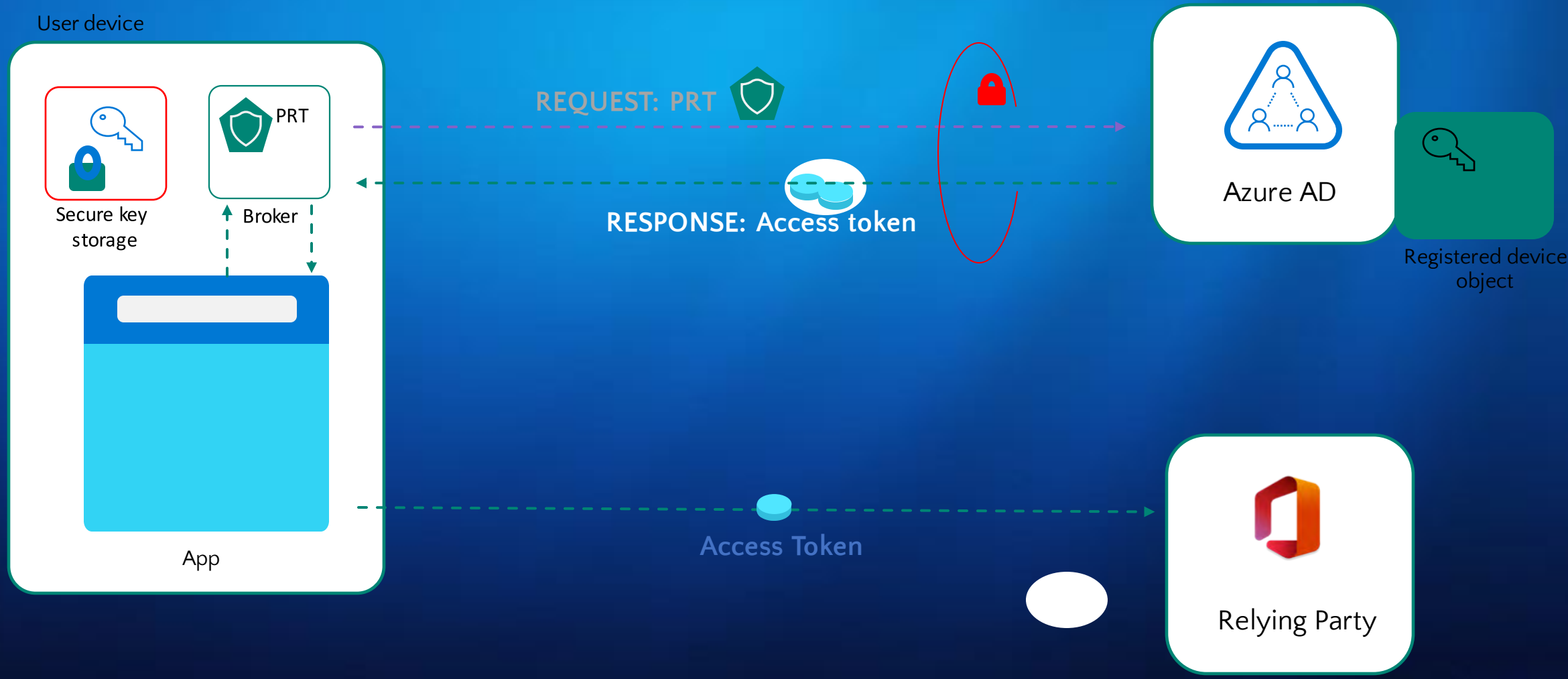
1. Require sign in sessions to be bound to issue access tokens (unbound refresh tokens won't be accepted)
2. Only issue bound access tokens
3. Require that workload auth session cookies are bound to device

If access token or workload cookie is bound, resource validates binding only accepts from the device they are issued to

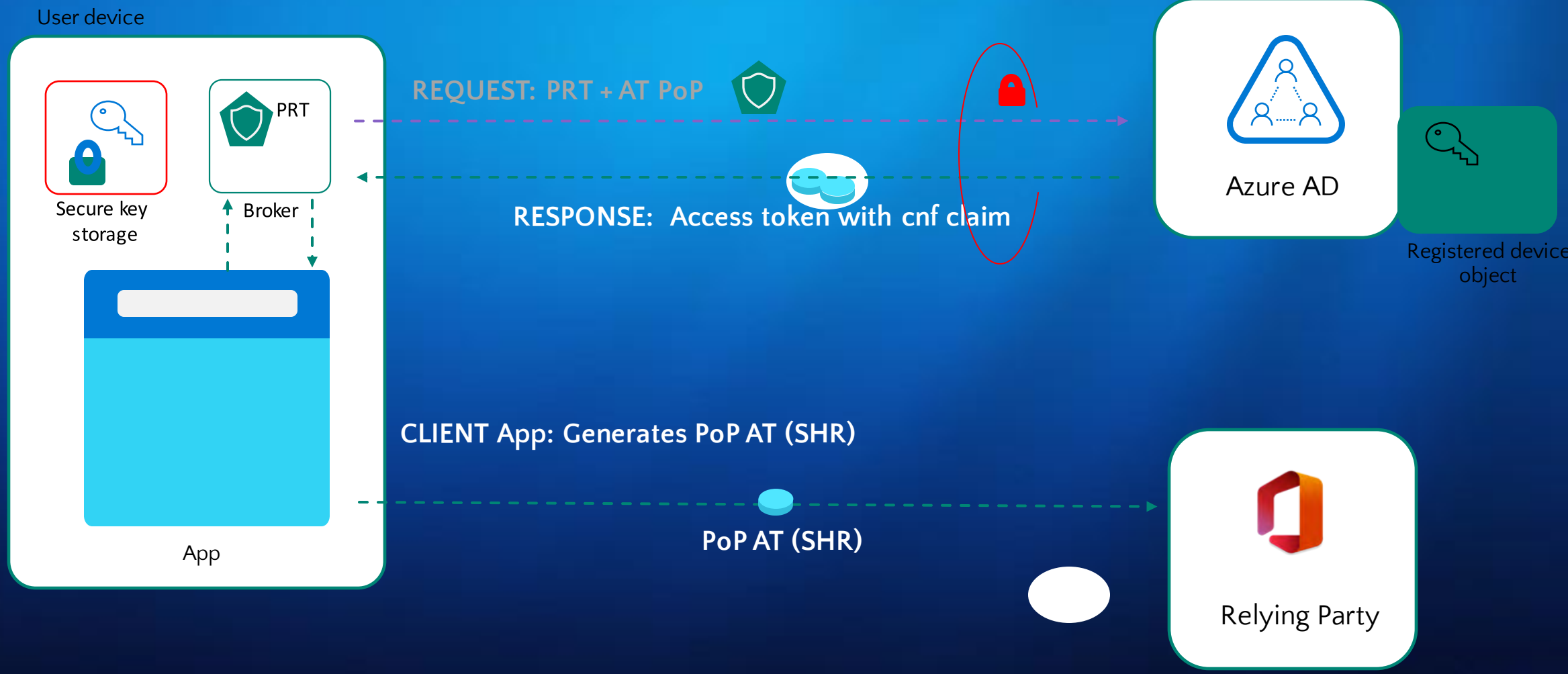
Device Registration



Token protection for Sign In Sessions



Token Protection for Sign In and App



Token Binding

Keyguard and OS support in latest Windows 11

Virtualization based security used to provide secure key storage

Azure AD support in private preview

Broad available for app and sign in sessions in the near futre

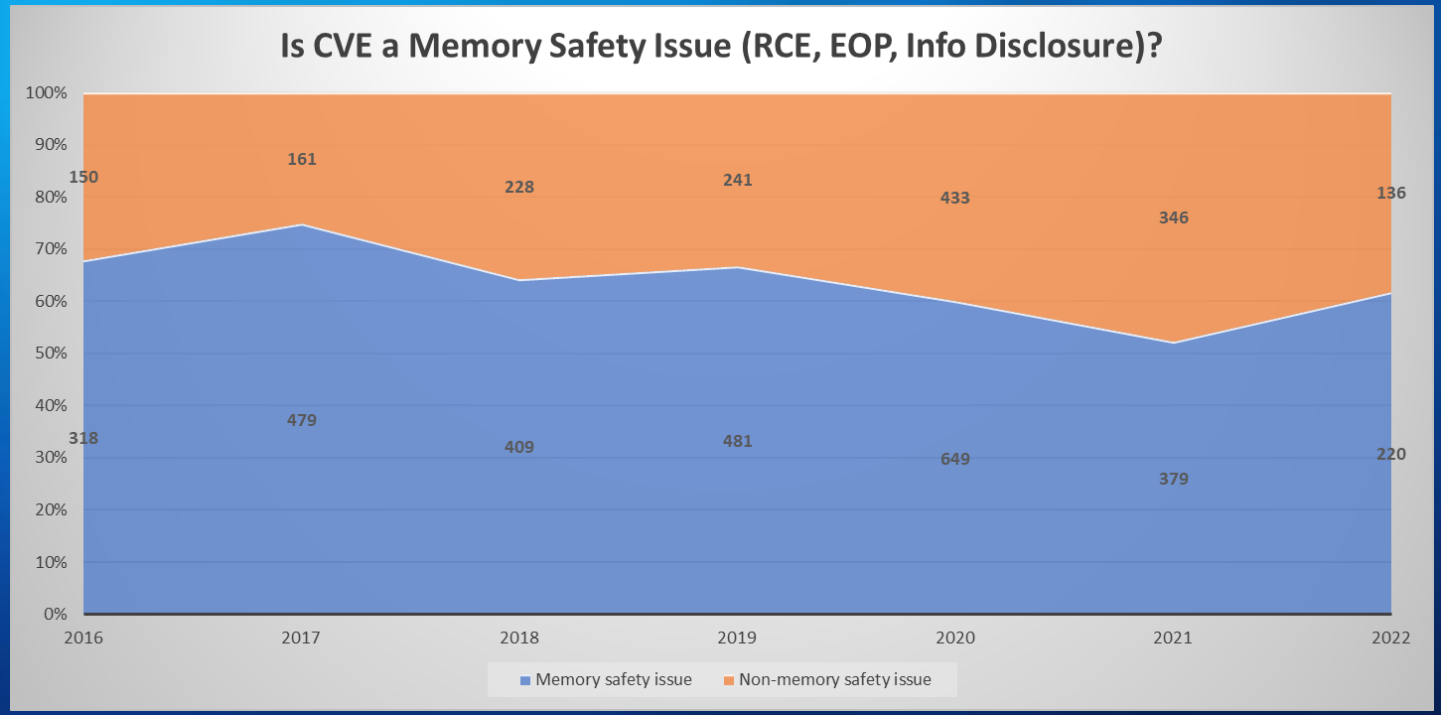
Memory safety in Windows

Microsoft and Memory Corruption

Reduced investment in mitigate exploit (CFG/XFG)

Increased investment in bug class elimination

Incremental Progress

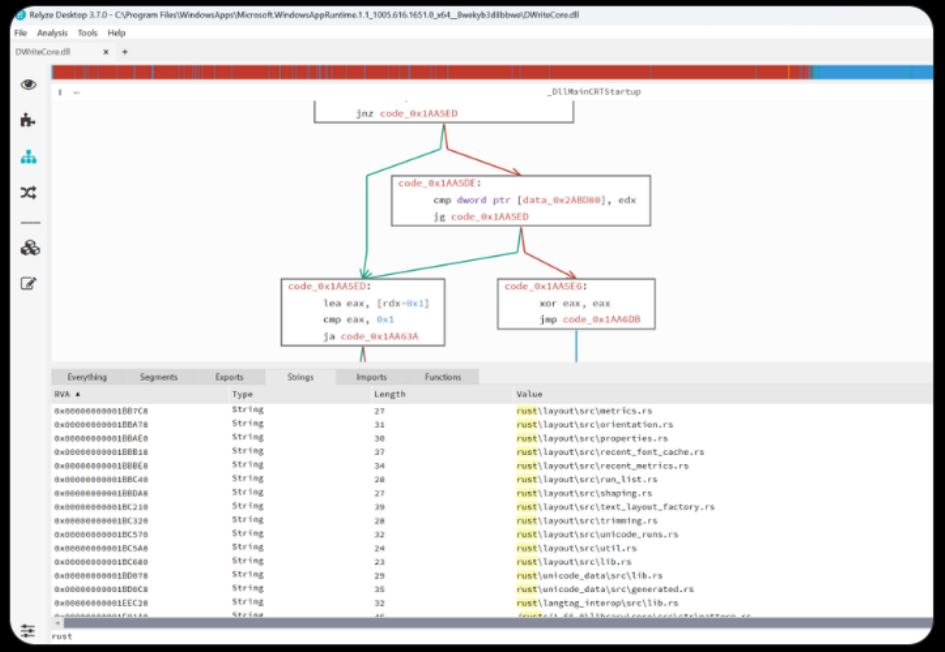


1. Memory Safe Languages
2. CPU Architectural Changes
3. Safer Language Subset

1

David Weston (DWIZZZLE)
@dwizzzleMSFT

dwrite font parsing ported to Rust? 😯😯😯
learn.microsoft.com/en-us/windows/...



2:46 AM · Oct 8, 2022

2

David Weston (DWIZZZLE)
@dwizzzleMSFT

Windows is putting Rust in the kernel 🤯 learn more at my @BlueHatIL talk.

9:06 PM · Mar 16, 2023 · **106.6K** Views

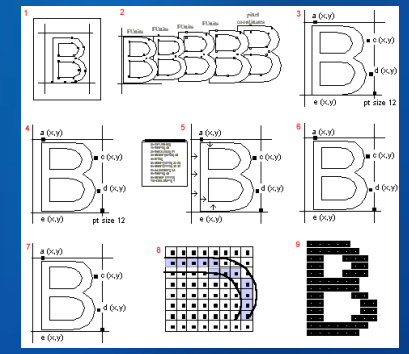
Arden White, Christopher Leung, and many others are to thank for this work

Rust in Windows: Crawl

- Learn by doing: Exploration → Flighting → Production (crawl → walk → run)
- Direct impact: Improve security
- Indirect impact: Gain experience with transitioning to Rust in production
  - Costs of learning Rust?
  - Costs of porting Rust?
  - Costs of writing new Rust?
  - Is the full pipeline of Rust tooling ready? Debugging, perf, cross-platform, POGO, etc.
  - Costs of maintaining a hybrid C++/Rust codebase?

What is DWrite? What is DWriteCore?

- Full stack for text analysis, layout, and rendering
  - Ships in Windows (dwrite.dll)
  - Handles all major languages and scripts
  - Huge amount of inherent complexity: complex scripts, complex glyph descriptions
- DWriteCore is DWrite "undocked" from Windows
  - Builds outside of Windows repo
  - Cross-platform: Windows, Linux, Android, iOS, Mac OS
  - Office contains an old fork (dwrite10), is migrating to DWriteCore for some platforms
    - All new feature development in DWrite has shifted to DWriteCore
- Collaboration between Rust team and DWrite team begin in 2020
- DWriteCore is now ~152 KLOC of Rust, ~96 KLOC of C++



DWrite Internals

Layout (10 KLOC)

- Line layout, justification
- Text run management: bold, italics, font face, underline, etc.
- Font fallback: Most fonts don't contain all glyphs (e.g. emoji)

Shaping (36 KLOC) + OTLS (18 KLOC)

- Complex script-specific layout: Thai, Indic, Arabic, Hebrew, Hangul, etc.
- Mandatory for complex scripts
- Many are driven by hand-written FSMs
- Complex transformation rules stored in font files (OpenType)
- Transforms sequences of glyphs, e.g. ligatures, connected scripts

Unicode Analysis (6 KLOC)

- Very large property tables
- Defined by Unicode standard

Glyph Data + Glyph Rendering (24 KLOC)

- Computes vector curves, runs bytecode programs (!!) from font files to adjust them
- Rasterizes vector curves to bitmaps
- Provides metrics (advance width, x-height, side bearings)
- Scales bitmaps for high-density scripts (e.g. Chinese)

Total ported code ~= 152 KLOC (some modules not shown). (Precise counts are complicated, due to test code.)

All code is 100% safe code, except at C++ boundary

Not all parts of DWrite are shown; just those relevant to port

Integrating C++ and Rust

- DWriteCore internally uses COM-like interfaces. These were a good integration point for C++/Rust, and provided natural boundaries for incremental porting.
- DWriteCore public APIs are all COM. In some cases, Rust code is directly callable from app code, through COM interfaces.

```
DWRITE_BEGIN_INTERFACE(INumberSubstitution,
   "9d5d67e0-7bde-4f6d-a073-360c5c381dd6") : IDWriteNumberSubstitution
{
   virtual NumberSubstitutionMode GetMode() const = 0;
   virtual NumberSubstitutionChars const& GetChars() const = 0;
   virtual uint32_t GetScript() const = 0;
};
```

```
com::interfaces! {
   #[uuid("9d5d67e0-7bde-4f6d-a073-360c5c381dd6")]
   pub unsafe interface INumberSubstitution : IDWriteNumberSubstitution {
      pub fn GetMode(&self) -> NumberSubstitutionMode;
      pub fn GetChars(&self) -> *const NumberSubstitutionChars;
      pub fn GetScript(&self) -> u32;
   }
}
```

- In other places, we statically link Rust and C++ code.

```
extern "C" IDWriteInlineObject* Rust_Layout_CreateInlineObject(
   IDWriteTextLayout *layout,
   InlineLayoutBoundMode boundMode,
   bool adjustBaseline);
```

```
#[no_mangle]
pub extern "C" fn Rust_Layout_CreateInlineObject(
   layout: IDWriteTextLayout,
   bound_mode: InlineLayoutBoundMode,
   adjust_baseline: bool,
) -> IDWriteInlineObject {
   ...
}
```

Performance

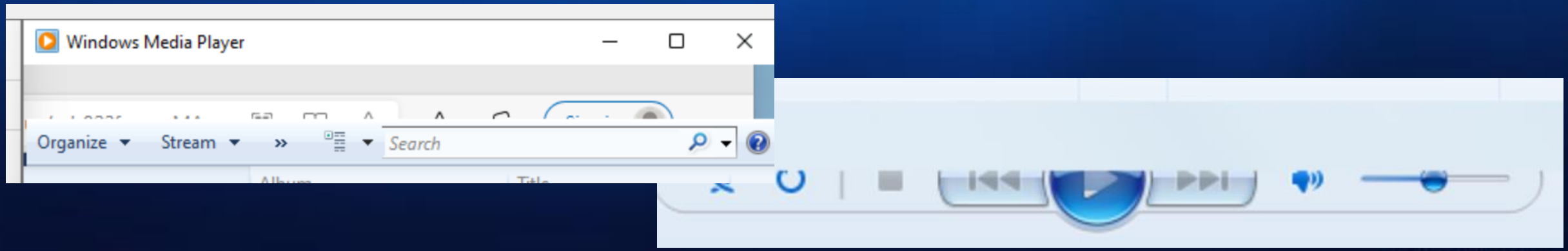
- Performance analysis and improvement in Rust is *amaaaaaaaazing*
  - Achieving perf parity was usually easy.
  - Achieving perf superiority was often easy. e.g. Shaping (OTLS) is 5% to 15% faster in Rust.

How much time did porting take? Regressions?

- TrueType:
  - ~2 months (1 dev, experienced in Rust) for the core functionality
  - ~2 months for exhaustive comparison testing and regression fixing
- Shaping + OTLS
  - ~2 months
  - ~1 month for comparison testing and regression fixing
  - ~2 weeks for performance improvements
- Layout
  - ~1.5 months
  - ~2 weeks for testing / regression fixing
- Unicode analysis
  - ~2 weeks
  - Low rate of regressions; very data-oriented

Win32k GDI port to Rust

- Ported the REGION data type and functions
  - Models overlapping controls (e.g., windows) in GDI.
  - “Leaf node” data type: few dependencies, many dependents.
  - Old (late 80s, early 90s), and perf critical (designed for a 286/386).
  - Maintenance nightmare: open-coded vector resizing and ref-counting.
- Currently disabled via a feature-flag.
- Windows boots with the Rust version, and all GDI tests pass.



Progress so Far

- Able to use standard Rust APIs (e.g., Vec and Result)
  - Much easier to write and understand than the C++.
  - Unlocks benefits from tooling: e.g., warnings for not consuming a Result.

C++

```
private:
    ULONGSIZE_T     sizeScanAlloc;

    // fields above here are untouched by vCopy.

    PSCAN           pScan;

    // pscnTail gets updated in vCopy

    PSCAN           pscnTail;

    // fields below here get copied in vCopy, fie

    ULONGSIZE_T     sizeScan;
    COUNT           cScans;
    RECTL           rcl;
```

Rust

```
    scan_count: usize,
    scan_data: Vec<i32>,
    bounds: RECTL,
```

Progress so far

- ~36 KLOC in the Rust port.
- Perf of the ported code has been excellent
  - No perf difference in Office apps (as measured by PCMark 10).
  - Micro-benchmarks show mostly no differences, with some wins for Rust.
- Has driven changes upstream in Rust
  - More try_ methods for Vec that don't panic on OOM: https://github.com/rust-lang/rust/pull/95051
- Calls to extern functions means there's a lot of "unsafe" code
  - Currently 163 unsafe functions (~10%) and 271 unsafe blocks.
  - But as we port more code, these have been disappearing.
  - We've even been able to write a SysCall is completely safe code.

Next steps with memory safety

Driven by community engagement

DWriteCore shipping
Included in Winapp SDK

GDI regions coming soon to insider preview
“Crawl” phase for win32k, prove viability

CPU architecture
CheriIOT, Memory Tagging, and other approaches being investigated for broad memory safety strategy

Windows continues to evolve

Tackling the oldest and largest challenges

More work happening to kill legacy attack surface (NTLM, SMBv2)

Killing bug classes is our focus

THANKS!!!!