

PASTDOOR

After BlastDoor: State Transformation Across Trusted System Services

Jeffery Kimbrow

Independent Research

research@silicongremlin.io

Abstract

PASTDOOR traces message-derived state beyond the BlastDoor parser and into the trusted services that consume, reconstruct, synchronize, allocate, render, and transmit it. BlastDoor confines parsing authority while accepted state continues into protobuf and XPC materialization, identity and device-state handling, image and graphics pipelines, networking telemetry, pairing, and accessory-update services. Each boundary replaces the original representation with consumer-local state and enforces a new set of invariants.

Recovered implementation diagnostics expose the exact validation and failure surfaces: protobuf detects concurrent mutation and byte-count divergence; XPC rejects missing receivers, missing protocols, and wire/local signature mismatches; identity services deserialize and validate device prekeys before message sealing; CMITileEngine and FigMetalAllocator enforce texture, region, resource-lifetime, IOSurface, and allocation state; and UARP reports protocol rejection, timeout, unsupported messages, and endpoint-removal failure. Runtime records independently capture CarPlay, accessory, memory-pressure, watch, networking, pairing, and process-resource effects. The completed result is a demonstrated post-BlastDoor capability for semantic state to survive representation changes and remain actionable across trusted process, device, graphics, accessory, and lifecycle boundaries.

The unresolved transition is object-level attribution: the runtime artifacts do not carry the identifier required to bind each downstream effect to a specific decoded input. PASTDOOR therefore separates parser containment from state containment.

1 Introduction

BlastDoor moves complex message parsing away from the primary messaging process and into a restricted service. That architecture limits the immediate authority of a parser compromise. The accepted state then re-enters the platform through ordinary trusted mechanisms: serialization, XPC, identity resolution, synchronization, image decoding, IOSurface sharing, Metal allocation, rendering caches, networking telemetry, pairing, and accessory management. The exploit-relevant object after parsing is therefore not limited to the original bytes or the BlastDoor process.

The persistent unit is semantic state: an identifier, account relation, device association, pass record, image dimension, resource descriptor, pairing state, or other value treated as authoritative by a downstream consumer. Each consumer materializes that value in a different representation and checks different invariants. A single upstream condition can therefore terminate as an XPC decode rejection, a prekey or version failure, a nil texture, an invalid output region, an IOSurface-backed allocation, an accessory endpoint error, or a resource-pressure event without preserving the initiating parser phenotype.

PASTDOOR demonstrates semantic-state continuation beyond parser containment: accepted state remains actionable as trusted consumers reconstruct it under new representations, invariants,

and authority. Exact implementation diagnostics establish the reachable mechanisms and local failure conditions. Runtime records establish concrete downstream effects in the corresponding operational domains.

BlastDoor is the entry boundary, not the end of the exploit surface. PASTDOOR analyzes the downstream path along three dimensions:

Reachability	Which trusted consumer can receive state derived from the decoded message?
Transformation	How does that consumer reconstruct or materialize the state?
Authority	What can that consumer change, synchronize, allocate, render, transmit, or otherwise act upon?

These dimensions define the PASTDOOR analysis model used throughout the paper.

1.1 Claim Architecture

PASTDOOR assigns every claim to one of four evidence classes. *I* denotes an implementation mechanism or local failure condition established by recovered implementation material. *R* denotes an instantiated runtime observation established by a runtime artifact. *C* denotes a causal join established by an identifier, observed state transition, timestamped transfer, schema instance, or independent runtime artifact connecting two endpoints. *U* denotes an unresolved causal join where the endpoints are established but the transition is not.

1.2 Attribution Boundary

Implementation and runtime evidence do not compose into object-level causality without a joining artifact:

$$I(a \rightarrow b) \wedge R(b) \not\Rightarrow C(a \rightarrow b). \quad (1)$$

Where no object identifier, transfer, state transition, schema instance, or independent runtime artifact closes the edge, PASTDOOR marks the join *U*. This Attribution Boundary applies to per-input runtime attribution; the underlying *I* mechanisms and *R* observations remain independently established.

2 Related Work

Groß’s reverse engineering of iMessage in iOS 14 and Apple’s platform-security documentation define the BlastDoor entry and isolation boundary [1,2]. Project Zero’s FORCEDENTRY analyses document the remote-code-execution and sandbox-escape stages, while Citizen Lab establishes in-the-wild delivery and victim-side forensic effects [3–5]. Their relationship to PASTDOOR is entry and immediate exploit progression; PASTDOOR begins where accepted state leaves that parser-local representation.

PWNYOURHOME and FINDMYPWN demonstrate cross-service composition: HomeKit or Find My activity precedes a second iMessage/BlastDoor phase in a two-process zero-click chain [6]. BLASTPASS demonstrates a distinct delivery relationship in which PassKit attachments carrying malicious images reached the victim through iMessage [7]. These works provide comparators for multi-service delivery and execution before or during the BlastDoor stage.

Apple’s platform-security guide, pointer-authentication guidance, Lockdown Mode documentation, and security advisories define the mitigation context [8–13]. PACMAN crosses a mitigation boundary by combining a software corruption primitive with speculative execution to bypass pointer authentication [14]. Spectre establishes the broader cross-abstraction result that architecturally discarded execution can still produce security-relevant microarchitectural state [15]. They provide comparators for mitigation crossing and cross-abstraction security, respectively.

Operation Triangulation provides the multi-stage-depth comparator. Its technical analyses recover a zero-click attachment, userland and kernel exploitation, hardware-assisted protection bypass, payload staging, in-memory implant behavior, and post-exploitation cleanup across multiple processes and abstraction layers [16–18]. Amnesty International’s Pegasus methodology, trace appendix, and Mobile Verification Toolkit establish the complementary forensic problem of reconstructing compromise from partial device artifacts [19–21].

Kimbrow’s systems models address cross-boundary fault propagation, observability loss, multi-boundary interference, causal admissibility, state evolution, and remount-oriented persistence analysis [22–28]. PASTDOOR applies those concerns to concrete post-BlastDoor consumers. Its novelty is the analysis target: prior exploit work primarily resolves delivery, RCE, sandbox escape, mitigation bypass, or implant behavior; PASTDOOR tracks accepted semantic state after parser containment through trusted consumers that reconstruct it under new representations, invariants, and authority.

3 BlastDoor Entry Point and Downstream Trust Path

Figure 1 shows the post-entry trust path. BlastDoor produces structured state under constrained authority. Protobuf and XPC materialize that state for other processes. Identity, pairing, and synchronization services bind it to account or device context. ImageIOXPC, IOSurface, Metal, and MPS convert media state into shared surfaces and GPU resources. Networking and accessory services translate host state into telemetry, transport, firmware, and endpoint representations. Every transition changes representation and moves the state into a consumer with different authority.

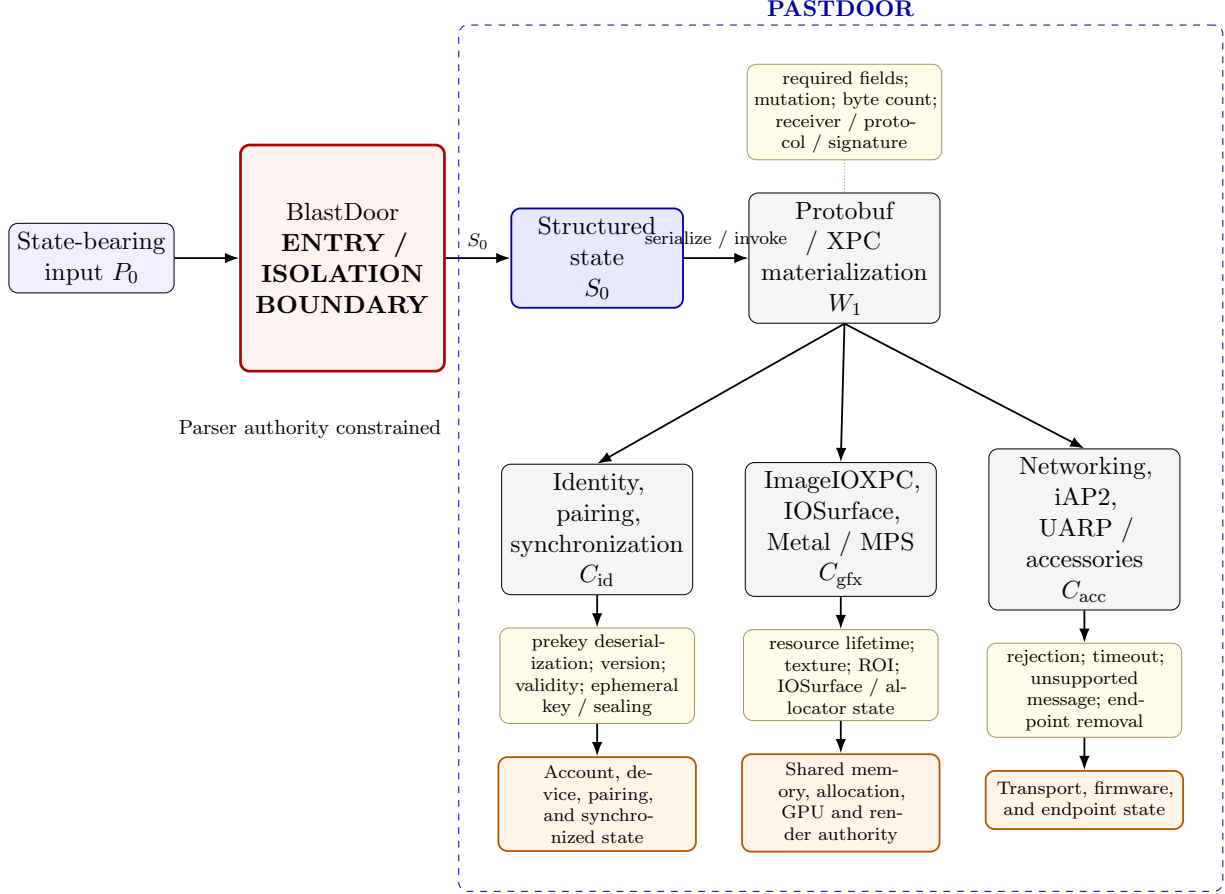


Figure 1: Full post-BlastDoor architecture. State changes representation from P_0 to S_0 , wire or invocation state W_1 , and consumer-local forms C_i . Yellow boxes mark recovered invariant-enforcement points; orange boxes identify the authority exercised by each consumer class. Solid arrows are implementation-supported edges (I), not a claim that all branches execute for one input.

The first downstream boundary is structured IPC. Protobuf enforces required fields, field types, mutation stability, and agreement between computed and emitted sizes. XPC enforces receiver availability, allowed protocols, selector and invocation state, and compatibility between wire and local signatures. The receiver does not copy the sender’s object; it reconstructs and validates a new object under the receiving interface.

Identity and device-state services then provide reconstruction across process and lifecycle boundaries. The implementation handles device identities, public prekeys, ephemeral P-256 keys, authenticated-encryption state, shared-secret computation, pass synchronization, companion relationships, and protected state. Remote-pairing, watch-registry, preference-synchronization, companion, placeholder, and icon-representation services are present in the captured runtime contexts. These mechanisms rebuild semantically related state from account, device, and remote sources after the originating process-local representation is gone.

The graphics path introduces another set of transformations. ImageIOXPC decodes an encoded object into an image representation. IOSurface can then provide shared backing across process boundaries. Metal and Metal Performance Shaders convert that backing into resources with allocator, dimension, heap, texture, kernel, matrix, and lifetime constraints. Rendering caches and framebuffer state add further local representations. A malformed or inconsistent property can therefore emerge

as a decoder error, an allocation failure, a nil resource, an incompatible texture, a cache inconsistency, or a later memory-pressure event, depending on where the state is observed.

Networking and accessory services form parallel translation paths. Link-quality measurement and analytics materialize interface and session state. iAP2 and UARP move state between host, transport, firmware, and accessory endpoints. Car-accessory, accessory-update, UARP, and asset-management services are present in the captured contexts. The recovered material identifies UARP as an accessory-update surface.

3.1 Boundary Depth

PASTDOOR operates within a deeper platform boundary stack. Apple documents privacy mediation, sandboxing, signed entitlements, code signing, pointer authentication, and kernel-integrity controls; public exploit analyses provide precedent for sandbox, control-flow, and kernel transitions [2, 4, 8–10, 14, 17]. Figure 2 organizes these boundaries by analytical depth. It is not a universal execution trace.

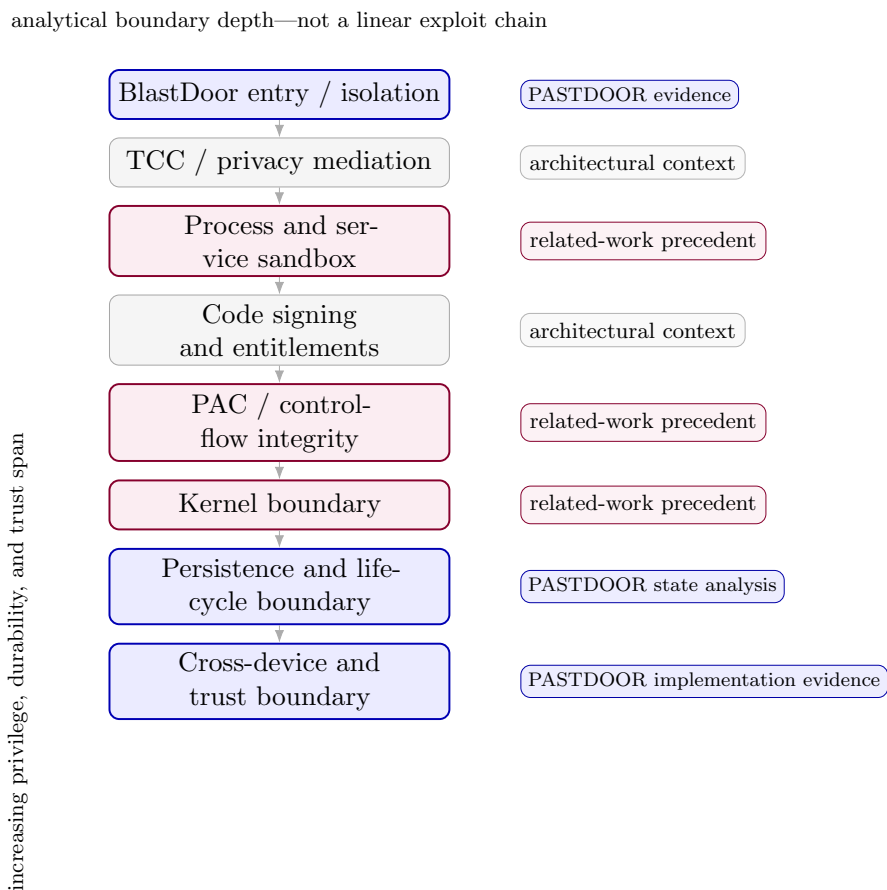


Figure 2: Boundary-depth map. Blue levels are directly represented in PASTDOOR evidence or state analysis, purple levels are demonstrated by cited exploit precedent, and gray levels are platform-architecture context. Dashed arrows order boundaries by analytical depth; they do not assert one universal execution path.

4 Post-BlastDoor Path Reconstruction

The path begins with a state-bearing message entering BlastDoor. The parser decodes it under restricted authority and emits structured state. That state then crosses into IPC materialization, identity or device-state association, media and graphics allocation, and networking or accessory translation. Figure 3 correlates representation, semantic state, and evidence class across the path.

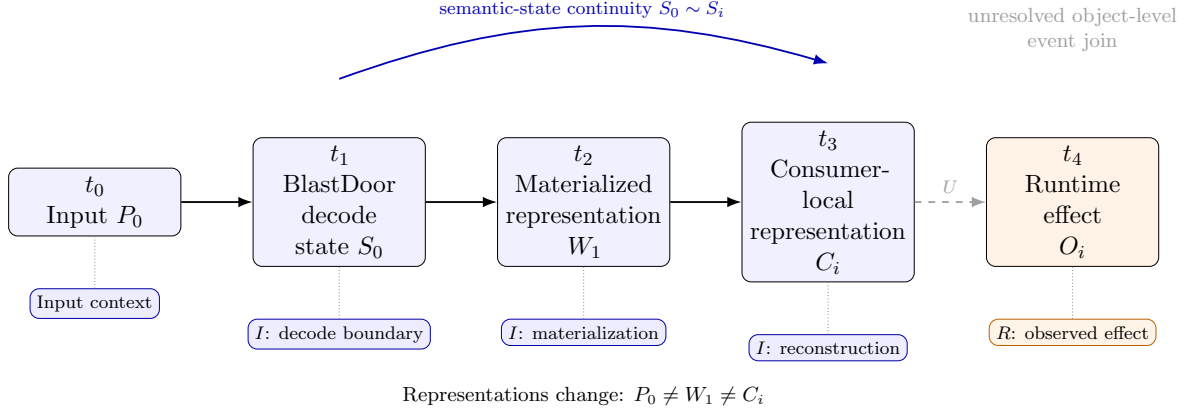


Figure 3: Evidence correlation across the post-BlastDoor path. Implementation evidence (I) establishes decoding, materialization, and reconstruction mechanisms while runtime evidence (R) establishes the final observation. Semantic continuity does not require representation identity. The dashed U edge marks the unresolved object-level join defined by the Attribution Boundary.

At t_0 , the provider supplies encoded representation P . At t_1 , BlastDoor decodes P into structured state S_0 . At t_2 , protobuf, XPC, identity, graphics, networking, or accessory code serializes, transfers, stores, or associates S_0 . At t_3 , the receiving service constructs consumer-local representation C_i from the transferred state and its local context. At t_4 , the consumer commits the state or exposes a local failure or resource effect:

$$P \xrightarrow{\text{BlastDoor}} S_0 \xrightarrow{\text{boundary}} C_i \xrightarrow{I_i} O_i. \quad (2)$$

Byte equality is not required across this path. Decode, normalization, serialization, reconstruction, and allocation deliberately replace the representation. Derivation is established by a shared identifier, observed transfer, matching schema instance, state transition, or independent runtime artifact—not by identical bytes or subsystem-name overlap.

At t_4 , the runtime records described in Section 7 establish R .

5 State Transition Semantics

The downstream path operates on semantic state rather than representation identity [27,28]. Let P_i be the provider representation presented at boundary B_i , and let S_i be retained, synchronized, or reconstructed state available to the consumer. The consumer applies local transformation T_i :

$$C_i = T_i(P_i, S_i), \quad (3)$$

where C_i is the consumer-local representation. The consumer evaluates an invariant I_i under local context Γ_i and exposes an observer-dependent result:

$$O_i = \Omega_i(C_i, I_i(C_i, \Gamma_i)). \quad (4)$$

Security-relevant state remains semantically related across boundaries even when the representations differ:

$$S_i \sim S_j \quad \text{while} \quad C_i \neq C_j. \quad (5)$$

The relation $\sim$ denotes semantic correspondence, not byte identity. It covers the same device, account, pass, image property, resource relationship, or accessory state reconstructed in different local forms.

Retention and reconstruction are distinct mechanisms. Retention carries a local object or equivalent state across a lifecycle transition. Reconstruction builds the later object from identity, synchronization, pairing, remote, or other authoritative sources. A hybrid path uses both:

$$S_{t+1}^{\text{retain}} = R_{\text{local}}(S_t), \quad (6)$$

$$S_{t+1}^{\text{reconstruct}} = R_{\text{remote}}(A_t, D_t), \quad (7)$$

$$S_{t+1}^{\text{hybrid}} = R(S_t, A_t, D_t), \quad (8)$$

where A_t and D_t represent account and device state. Figure 4 shows where each path rematerializes consumer-local state after a lifecycle boundary.

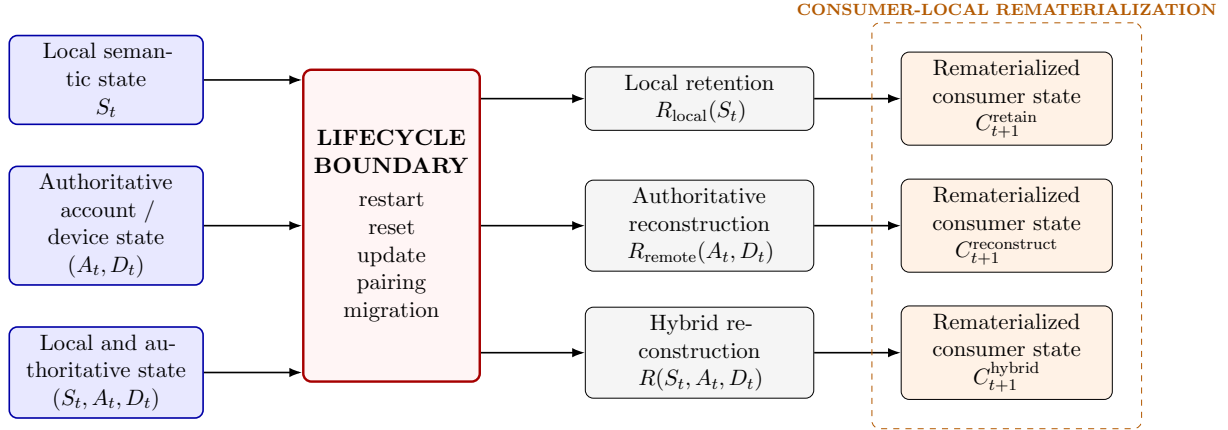


Figure 4: State reconstruction across lifecycle boundaries. Local retention carries S_t forward; authoritative reconstruction rebuilds state from account or device sources; hybrid reconstruction combines both. Each route produces a new consumer-local representation after restart, reset, update, pairing, or migration.

Recurrence after a lifecycle transition is therefore compatible with retained state, authoritative reconstruction, or a hybrid of both. The reconstruction source distinguishes the mechanisms.

6 Implementation Mechanisms

Recovered shared-cache material preserves symbols and embedded diagnostics rather than complete source bodies. The following listings reproduce the recovered strings exactly. They expose concrete validation branches, state assumptions, and failure paths without substituting reconstructed pseudocode for unavailable source.

6.1 Serialization and XPC

The serialization path contains explicit protections against structurally inconsistent messages. Protobuf diagnostics cover required-field validity, field-type validity, concurrent modification during serialization, and disagreement between calculated and serialized byte counts. If $M = (F_1, \dots, F_n)$ is a structured object and W is its wire representation, correct serialization requires at least

$$\text{type}(F_k) = \tau_k \quad (9)$$

for each field and

$$L_{\text{computed}}(M) = L_{\text{serialized}}(M). \quad (10)$$

The receiver then applies a decoder $D(W) \rightarrow M'$, creating a new local object rather than preserving the sender's in-memory representation.

Extracted protobuf diagnostics (verbatim):

```
CHECK failed: (byte_size_before_serialization) ==
(byte_size_after_serialization):
Protocol message was modified concurrently during serialization.

CHECK failed: (bytes_produced_by_serialization) ==
(byte_size_before_serialization):
Byte size calculation and serialization were inconsistent.
```

These checks compare the message before serialization, after size calculation, and after byte production. They matter because the wire object is accepted only if its measured state remains internally consistent throughout serialization. A mutation or size divergence is converted into an explicit failure at the IPC boundary rather than silently propagated.

The XPC material adds checks for undecodable messages, selector decoding, invocation state, and compatibility between wire and local signatures. These are downstream enforcement points. They reject undecodable messages and incompatible signatures and terminate invalid invocation state after the original content has already been accepted elsewhere.

Extracted XPC diagnostics (verbatim):

```
received an undecodable message for proxy %lld
(no exported object to receive message). Dropping message.

received an undecodable message
(no protocol set to define allowed messages on exported object).
Dropping message.

received an undecodable message
(incompatible reply block signature for %s
(wire: %@ vs local: %@))
```

The first two paths reject messages that cannot be bound to an exported receiver or allowed protocol. The third compares the signature reconstructed from the wire with the receiver's local reply-block signature. These diagnostics directly support the claim that downstream XPC consumers reinterpret and validate message state under local interface rules.

These branches establish the IPC mechanism (I): structured state leaves the parser, acquires a wire representation, and is reconstructed under the receiver's protocol and signature constraints.

6.2 Identity, pairing, and synchronization

Identity and synchronization services provide the clearest mechanism for state to reappear after its original local representation is gone. The implementation includes device identity, public prekeys, ephemeral public keys, authenticated encryption, shared-secret computation, pass synchronization, companion relationships, and errors associated with unavailable or invalid identity material. These components can associate content-derived state with durable account or device relationships.

Extracted identity and prekey diagnostics (verbatim):

```
Failed to deserialize the device public prekey.  
Tetra version mismatch. Versions: Theirs: %u, Ours: %ld  
Failed to encrypt to prekey that is no longer valid: %f.  
Failed to generate an ephemeral P-256 key for message sealing.  
The encryption of an outgoing message failed.
```

These failure paths establish the identity transition: the service deserializes a device prekey, enforces protocol-version compatibility, checks prekey validity, and generates an ephemeral P-256 key before sealing an outgoing message. The identity layer therefore reconstructs and acts on cryptographic state beyond the original parsing process.

Captured process contexts place remote pairing, watch registry, preference synchronization, companion, placeholder synchronization, and icon representation services in the recorded environments. The implementation establishes their reconstruction and synchronization capability.

The demonstrated capability is cross-boundary reconstruction. A downstream process or companion device can create semantically related local state from synchronized identity and device records. State recoverable from another authoritative source can therefore reappear after deletion of one process-local representation.

6.3 Image and graphics processing

The image and graphics path contains multiple independent representation and resource transitions.

ImageIOXPC converts encoded media into an image representation. IOSurface supplies shareable backing. Metal allocation and Metal Performance Shaders create resources governed by size, format, heap, dimension, kernel, matrix, and lifetime constraints. Rendering caches and framebuffer operations then produce further state.

The implementation exposes checks involving missing or nil resources, IOSurface backing, texture compatibility, allocation and reference lifetime, heap state, matrix dimensions, kernel dimensions, cache state, framebuffer completeness, and rendered output. CMITileEngine and FigMetalAllocator constraints appear in one of the examined system-cache sources. These findings identify concrete locations where a property accepted upstream can be rejected or manifested differently downstream.

Figure 5 expands the graphics branch of Figure 1 from encoded media through rendered output.

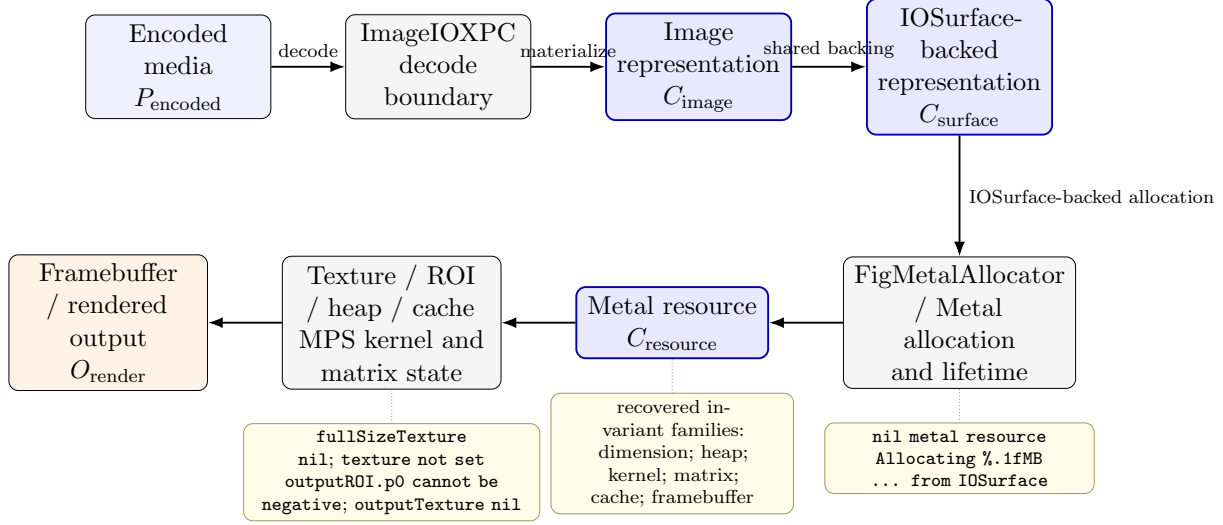


Figure 5: Graphics memory and resource architecture. Encoded media is decoded into an image representation, backed by IOSurface, allocated as a Metal resource, checked under texture, region, heap, cache, kernel, matrix, and lifetime invariants, and consumed by framebuffer or rendering operations. Annotations reproduce recovered diagnostics or invariant families; the graph does not add addresses, object layouts, or event-level execution claims.

Extracted graphics and allocator diagnostics (verbatim):

```

TileEngineError: (%s,%s:%d) %@ "allocated fullSizeTexture nil"
TileEngineError: (%s,%s:%d) %@ "texture not set"
TileEngineError: (%s,%s:%d) %@ "outputROI.p0 cannot be negative"
TileEngineError: (%s,%s:%d) %@ "outputTexture nil, internal error"

FigMetalAllocator >>>> %s:
Cannot increase the ref count of a nil metal resource
FigMetalAllocator >>>> %s: Allocating %.1fMB metal buffer from IOSurface

```

The tile-engine checks enforce the existence of intermediate and output textures and reject a negative output-region origin. The allocator diagnostics separately expose resource-lifetime handling and the conversion of IOSurface backing into a Metal buffer. Together they show two distinct downstream failure boundaries: image-derived geometry and texture state in the tile engine, and ownership or allocation state in the Metal allocator.

The graphics path establishes that image-derived state reaches texture, region, IOSurface, allocator, and resource-lifetime checks (*I*).

6.4 Networking and accessory services

The networking implementation includes link-quality triggers, interface and session state, and analytics representation. The accessory implementation includes iAP2-facing state, UARP restore and update diagnostics, and services for car accessories, accessory updates, and asset management. These mechanisms translate state across host, transport, firmware, and accessory boundaries.

Extracted UARP failure diagnostics (verbatim):

```
Timed out or aborted
Rejected
Not supported message received
UARP Restore: Failed to remove endpoint, status 0x%08x
```

These strings identify protocol-level rejection, timeout, unsupported-message, and endpoint-removal failures in the UARP restore path. They support an accessory-update failure surface with explicit endpoint state. They do not expose the command handler, the endpoint structure, or the call sequence, so no source-level transfer chain can be reconstructed from them alone.

The implementation establishes host, transport, firmware, and endpoint failure surfaces. UARP and accessory-update behavior remain distinct from RTKit because the recovered material contains no RTKit identifier.

7 Recorded Runtime Consequences

The runtime records capture concrete failures and resource effects after the initial message-processing context. The affected domains include Apple platform accessory services, memory termination, watch applications, Wi-Fi link quality, pairing performance, and per-process resource use. The longitudinal system log supplies the surrounding temporal context. These are instantiated runtime effects in downstream platform regions, not static implementation possibilities.

These records establish R . The object-level edge from an individual effect to a specific decoded input remains U ; report-level cause remains unresolved until that edge is closed.

8 Post-Exploitation Capability

PASTDOOR demonstrates four post-exploitation capabilities.

First, structured state crosses the BlastDoor boundary through normal serialization and IPC mechanisms. The receiver reconstructs the object under a new schema and invocation context. Security-relevant meaning survives without preserving the original bytes or process-local object.

Second, identity, pairing, and synchronization services associate that state with authoritative account and device records. A later process or companion device can reconstruct semantically related local state across restart, reset, migration, and cross-device boundaries.

Third, media-derived state moves through ImageIOXPC, IOSurface, Metal, MPS, and rendering caches. Each stage transforms the representation and exposes different invariants. The resulting graphics or resource failure is consumer-local and need not preserve the parser phenotype.

Fourth, networking and accessory services translate host state into transport-, firmware-, and device-local representations. The exploit surface therefore extends to trusted consumers and observers outside the original message process.

9 Security Impact

BlastDoor limits parser authority. The post-parser security boundary includes services with control over identity, synchronization, graphics resources, companion devices, transports, firmware updates, and accessories. Applying the Reachability–Transformation–Authority model identifies the state each service accepts, the transformation it performs, and the authority it exercises.

Recurrence after deletion or restart must be analyzed at the reconstruction boundary. Synchronization, pairing, identity recovery, cache reconstruction, or remote authoritative state can reproduce the visible condition without retaining the original local object. Endpoint recurrence alone cannot distinguish persistence from reconstruction.

Exploit instrumentation must therefore preserve semantic identifiers across boundaries. A complete trace records the decoded object, serialized form, XPC endpoint, identity or device association, IOSurface and graphics resource identifiers, accessory transfer state, and consumer-local outcome. Those joins convert the mechanism demonstrated by PASTDOOR into per-event attribution.

10 Conclusion

PASTDOOR establishes that BlastDoor confines parsing authority but not the lifetime of accepted semantic state. The recovered implementation exposes concrete paths through protobuf and XPC, identity and synchronization, ImageIOXPC and IOSurface, Metal and MPS, networking telemetry, pairing, and accessory management. Each path reconstructs state under new consumer semantics and new authority.

Runtime records capture faults and resource effects in the corresponding downstream domains. The completed result is the mechanism and capability: message-derived state can outlive the BlastDoor representation, cross trusted boundaries, and remain actionable after reconstruction.

Post-exploitation analysis must therefore follow state rather than bytes and authority rather than process names. After BlastDoor, the decisive questions are which trusted consumer received the state, how that consumer reconstructed it, which invariant it enforced, and what authority it exercised afterward.

Acknowledgments

Security research rarely begins at the boundary where the interesting question starts. This work owes much to the researchers who documented BlastDoor, modern iOS exploitation, and the platform mechanisms surrounding both. Their work made it possible to ask a different question here: not whether the door held, but what happened to accepted state after it passed through.

References

- [1] S. Groß, “A look at iMessage in iOS 14,” [Online]. Available: <https://projectzero.google/2021/01/a-look-at-imessage-in-ios-14.html>, 2021.
- [2] Apple, “BlastDoor for messages and IDS,” Apple Platform Security. [Online]. Available: <https://support.apple.com/guide/security/sec3c881cee/web>.
- [3] I. Beer and S. Groß, “A deep dive into an NSO zero-click iMessage exploit: Remote code execution,” Google Project Zero. [Online]. Available: <https://projectzero.google/2021/12/a-deep-dive-into-nso-zero-click.html>, 2021.
- [4] —, “FORCEDENTRY: Sandbox escape,” Google Project Zero. [Online]. Available: <https://projectzero.google/2022/03/forcedentry-sandbox-escape.html>, 2022.

- [5] B. Marczak, J. Scott-Railton, B. A. Razzak, N. Aljizawi, S. Anstis, K. Berdan, and R. Deibert, “FORCEDENTRY: NSO group iMessage zero-click exploit captured in the wild,” The Citizen Lab, University of Toronto. [Online]. Available: <https://citizenlab.ca/research/forcedentry-nso-group-imessage-zero-click-exploit-captured-in-the-wild/>, 2021.
- [6] B. Marczak, J. Scott-Railton, B. A. Razzak, and R. Deibert, “Triple threat: NSO group’s pegasus spyware returns in 2022 with a trio of iOS 15 and iOS 16 zero-click exploit chains,” The Citizen Lab, University of Toronto. [Online]. Available: <https://citizenlab.ca/research/nso-groups-pegasus-spyware-returns-in-2022/>, 2023.
- [7] The Citizen Lab, “BLASTPASS: NSO group iPhone zero-click, zero-day exploit captured in the wild,” University of Toronto. [Online]. Available: <https://citizenlab.ca/blastpass-nso-group-iphone-zero-click-zero-day-exploit-captured-in-the-wild/>, 2023.
- [8] Apple, “Apple platform security,” Apple Platform Security Guide. [Online]. Available: <https://support.apple.com/guide/security/welcome/web>.
- [9] —, “Preparing your app to work with pointer authentication,” Apple Developer Documentation. [Online]. Available: <https://developer.apple.com/documentation/security/preparing-your-app-to-work-with-pointer-authentication>.
- [10] —, “Improving control flow integrity with pointer authentication,” Apple Developer Documentation. [Online]. Available: <https://developer.apple.com/documentation/apple-silicon/improving-control-flow-integrity-with-pointer-authentication>.
- [11] —, “Lockdown mode security for apple devices,” Apple Platform Security. [Online]. Available: <https://support.apple.com/guide/security/sec2437264f0/web>.
- [12] —, “About the security content of iOS 14.8 and iPadOS 14.8,” Apple Security Releases, CVE-2021-30860 and CVE-2021-31010. [Online]. Available: <https://support.apple.com/103150>, 2021.
- [13] —, “About the security content of iOS 16.6.1 and iPadOS 16.6.1,” Apple Security Releases, CVE-2023-41064 and CVE-2023-41061. [Online]. Available: <https://support.apple.com/106361>, 2023.
- [14] J. Ravichandran, W. T. Na, J. Lang, and M. Yan, “PACMAN: Attacking ARM pointer authentication with speculative execution,” in *Proc. 49th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2022. [Online]. Available: <https://doi.org/10.1145/3470496.3527429>
- [15] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, “Spectre attacks: Exploiting speculative execution,” in *Proc. IEEE Symp. Security and Privacy*, 2019. [Online]. Available: <https://doi.org/10.1109/SP.2019.00002>
- [16] Kaspersky Global Research and Analysis Team, “How kaspersky obtained all stages of operation triangulation,” Securelist. [Online]. Available: <https://securelist.com/operation-triangulation-catching-wild-triangle/110916/>, 2023.
- [17] B. Larin, “Operation triangulation: The last (hardware) mystery,” Securelist. [Online]. Available: <https://securelist.com/operation-triangulation-the-last-hardware-mystery/111669/>, 2023.

- [18] G. Kucherin, L. Bezvershenko, and I. Kuznetsov, “Dissecting TriangleDB, a triangulation spyware implant,” Securelist. [Online]. Available: <https://securelist.com/triangledb-triangulation-implant/110050/>, 2023.
- [19] Amnesty International Security Lab, “Forensic methodology report: How to catch NSO group’s pegasus,” Amnesty International. [Online]. Available: <https://www.amnesty.org/en/latest/research/2021/07/forensic-methodology-report-how-to-catch-nso-groups-pegasus/>, 2021.
- [20] Amnesty International Security Lab / MVT Project, “Mobile verification toolkit,” [Online]. Available: <https://github.com/mvt-project/mvt>.
- [21] Amnesty International Security Lab, “Appendix e—pegasus forensic traces per target identified in the aftermath of the pegasus project revelations,” Amnesty International. [Online]. Available: <https://www.amnesty.org/en/latest/research/2021/08/appendix-e-pegasus-forensic-traces-per-target-identified-in-the-aftermath-of-the-revelations-of-pegasus-project/>, 2021.
- [22] J. Kimbrow, “The DIFFRACTION model: Case study on cross-boundary fault propagation in runtime systems,” Zenodo. [Online]. Available: <https://doi.org/10.5281/zenodo.20800364>.
- [23] —, “Observability collapse via node progression: A behavioral analysis of constraint-driven intake systems,” Zenodo. [Online]. Available: <https://doi.org/10.5281/zenodo.21115115>.
- [24] —, “DIFFRACTION—spectral conditioning and n-boundary interference modeling in modern runtime systems,” Zenodo. [Online]. Available: <https://doi.org/10.5281/zenodo.21283421>.
- [25] —, “DIFFRACTION: Abstract asymmetrical systems modeling,” Zenodo. [Online]. Available: <https://doi.org/10.5281/zenodo.21342631>.
- [26] —, “Causal layer integration: An admissibility framework for cross-representation dynamics,” Zenodo. [Online]. Available: <https://doi.org/10.5281/zenodo.22114782>.
- [27] —, “Boundary-constrained state evolution,” Zenodo. [Online]. Available: <https://doi.org/10.5281/zenodo.22259421>.
- [28] —, “RTmount—read-write-remount,” Zenodo. [Online]. Available: <https://doi.org/10.5281/zenodo.22759856>.