SIMPITY

BUILT DEEP FOR SECURITY

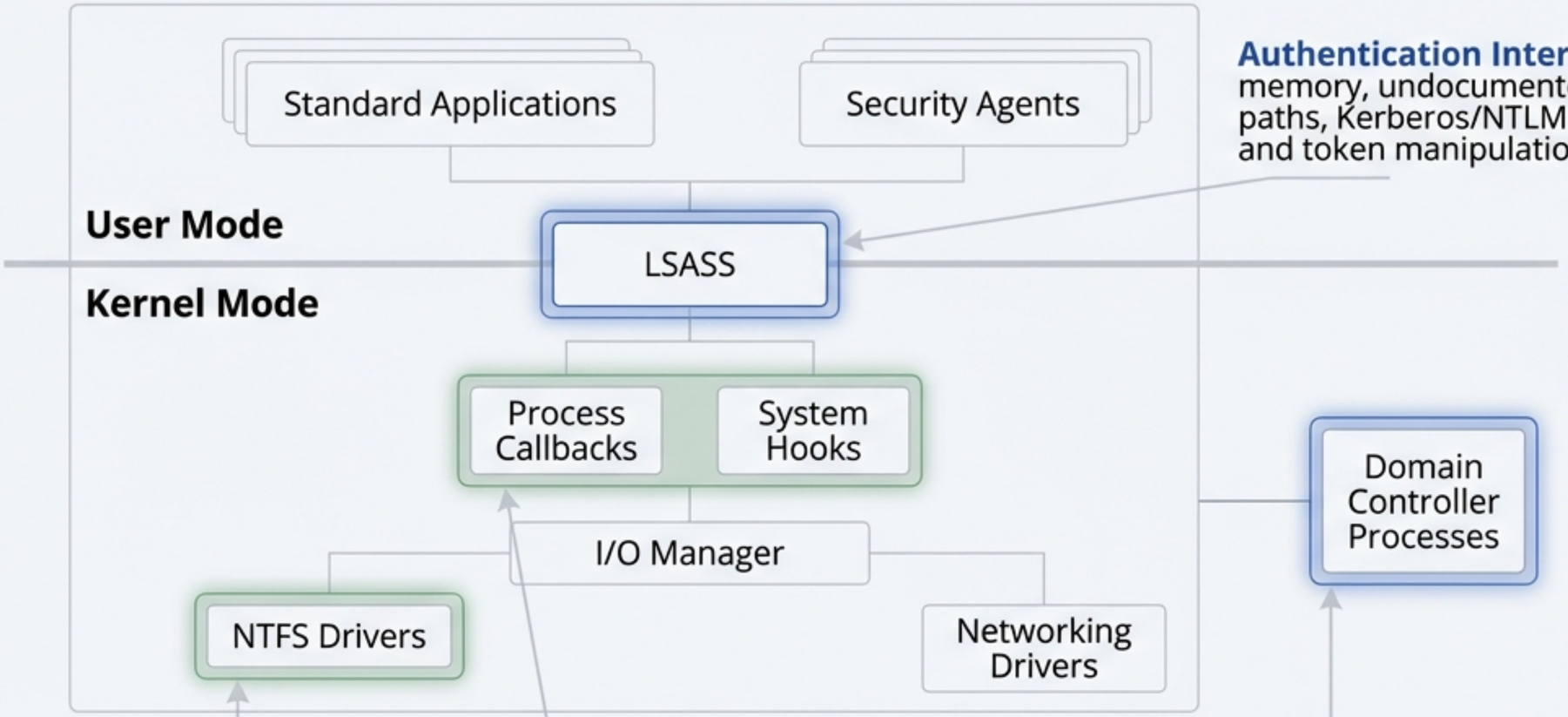
Simpity Security Engineering

Deep technical capabilities for security vendors that need engineering at the undocumented level of Windows.

Trusted by teams that build EDR, ITDR, AD security, DSPM, DLP, PAM, and endpoint agents.

The Layer Where Most Security Products Break

We solve problems at the depth where documentation ends, and reverse engineering begins. Our work starts where others cannot operate.



Authentication Internals: LSASS memory, undocumented authentication paths, Kerberos/NTLM credential logic, and token manipulation.

File System & Storage Drivers: NTFS minifilters and driver-level operations for both local disks and NAS appliances (NetApp/EMC).

Kernel-Mode Execution: Intercepting system behavior via callbacks, hooks, and filters before the OS executes commands.

Active Directory Internals: Monitoring and intervention within domain controller processes at massive scale (500+ DCs, 600K+ users).

Core Engineering Capabilities for Mission-Critical Products

Kernel-Level Interception & Enforcement

We intercept and block malicious operations inside the OS execution path, providing true pre-execution prevention.

Authentication Flow Control

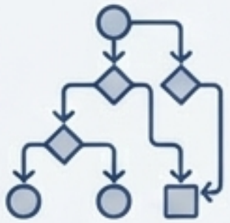
We safely instrument LSASS to gain real-time visibility and block credential abuse like Golden Ticket, DCSync, and Pass-the-Hash.

Patch Tuesday Resilience

We provide a 24-48 hour recovery cycle for product compatibility after Microsoft updates by reverse-engineering changes.

File System & NAS Security

We build driver-level monitors to detect and stop ransomware and anomalous file activity in its earliest stages.



Behavior-Based Detection Engines

We design detection logic based on sequences of action and behavioral anomalies, not brittle signatures.

Active Directory at Scale

We build stable, zero-lag monitoring solutions for high-load environments with hundreds of domain controllers.

Our Foundation: Reverse Engineering Undocumented Windows

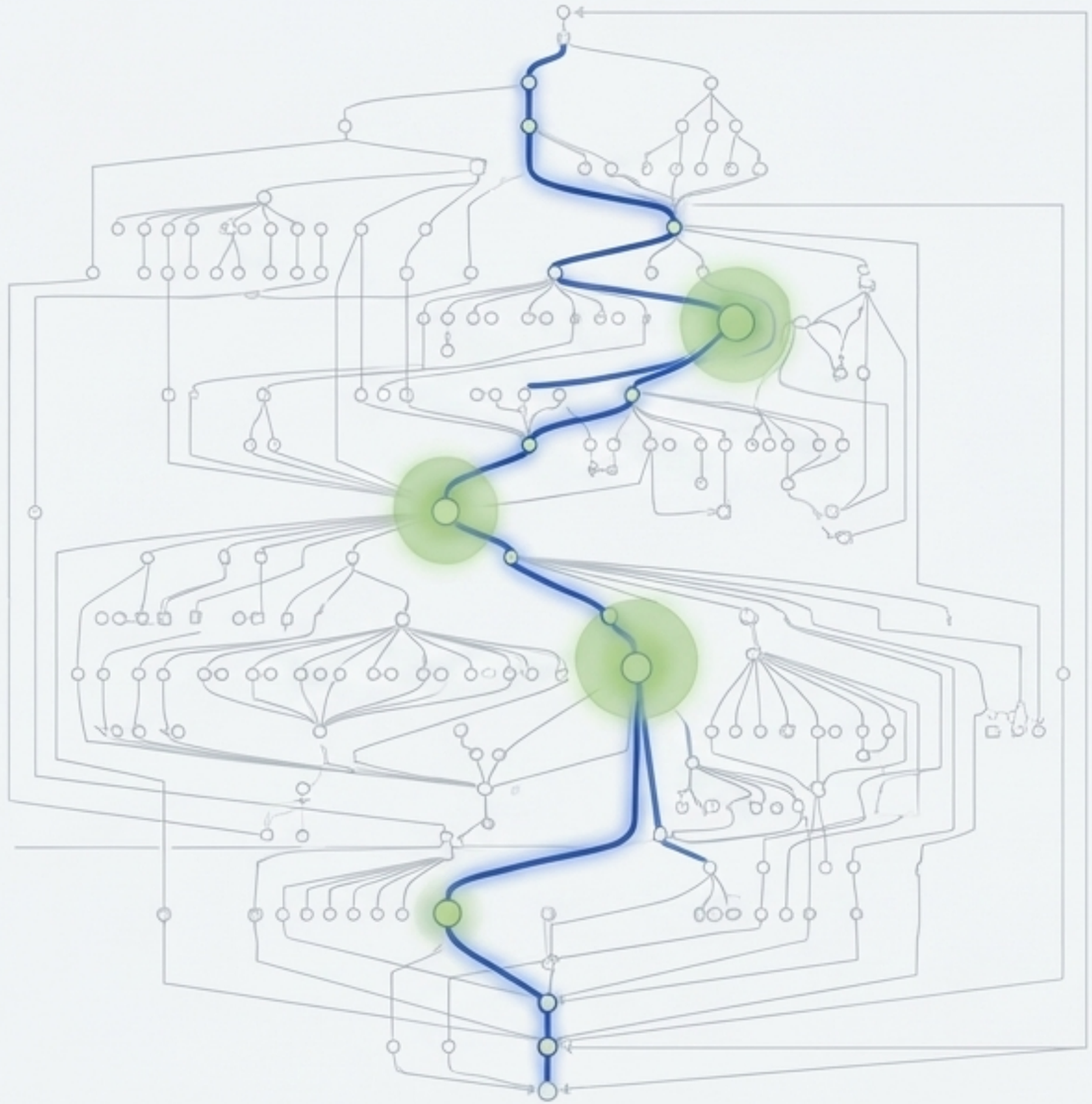
"We don't rely on assumptions or public APIs. Our engineering is built on a deep, verifiable understanding of Windows internals."

Process Description

- **Analysis:** We use disassemblers (IDA Pro, Ghidra) and debuggers (WinDbg) to analyze undocumented Windows components.
- **Mapping:** We map changes to LSASS, Kerberos, AD, and file-system internals after every update.
- **Adaptation:** This analysis enables us to build resilient components that adapt to changes, rather than break.

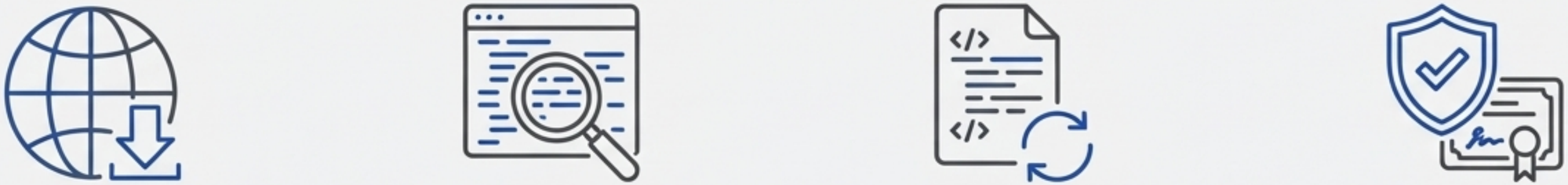
Why This Matters

- Enables us to operate in highly protected areas like LSASS.
- Guarantees resilience against undocumented changes from Microsoft updates.
- Unlocks the ability to create security controls that cannot be achieved through standard development practices.



The Simpity Patch Tuesday Response Cycle

From Global Update to Restored Stability in Under 48 Hours.



T+0 Hours

Microsoft releases patches. Automated systems begin binary analysis.

T+0-12 Hours

Reverse engineering team identifies critical changes in undocumented APIs, kernel structures, and LSASS memory layouts.

T+12-24 Hours

Development team adapts code to accommodate the discovered changes, recompiles, and runs initial validation.

T+24-48 Hours

The updated component is fully tested in our lab environment, signed, validated, and delivered to the client.

Outcome: Your product's deep integrations remain functional without extended downtime, long debugging cycles, or customer complaints.

Case Study: Ensuring Zero-Downtime LSASS Monitoring

Problem

An Identity Threat Detection & Response (ITDR) vendor's agent, which relied on LSASS instrumentation, would break for weeks after every major Windows update, damaging their reputation.

Root Cause

The agent used hardcoded memory offsets to read credential data. Microsoft's security updates consistently re-architected these layouts, breaking the agent.

Simpity Method

We re-engineered their module to stop relying on static offsets.

We built a dynamic analysis engine that programmatically identifies the location of critical data structures after each system boot and update, making the agent self-adapting.

Outcome

The time-to-fix for Patch Tuesday compatibility dropped from weeks to a consistent

24–48 hours.

Product downtime was eliminated, restoring customer confidence.

Case Study: Halting Ransomware on NAS Storage

Problem

A client's EDR solution was ineffective against ransomware targeting their NetApp NAS. Attacks encrypted critical file shares before the endpoint agent could react, leading to significant data loss.

Root Cause

The user-mode EDR agent had no direct, real-time visibility into file operations occurring over SMB/NFS at the driver level.

Simpity Method

We developed a file system minifilter driver that monitored I/O request packets (IRPs) for high-entropy write patterns characteristic of encryption.

Upon detecting the first few attempts, the driver immediately terminated the offending process at the kernel level.

Outcome

The solution stops ransomware attacks within the first

1-2 seconds

of activity. Malicious processes are terminated with near-zero data loss, before widespread file damage can occur.

Case Study: Blocking DCSync Attacks Without Disrupting Admins

Problem

An AD security product generated high false positives by blocking legitimate replication and backup tools that behaved similarly to Mimikatz DCSync.

Root Cause

Their detection logic was based on simple monitoring of the `DrsGetNCChanges` API call, which is used by both malicious and legitimate processes. It lacked the context to differentiate intent.

Simpity Method

We built a detection module that analyzed the full context of the API call: the parent process, the call stack, the user context, and whether the source was a known domain controller. This allowed us to distinguish a legitimate replication request from a malicious one.

Outcome

False positives were reduced by **over 95%**, allowing security teams to focus on real threats while administrators could perform their duties without interruption.

Our Engineering Methodology: From Architecture to Signed Driver

01

Deep Discovery & Architecture

We begin with analysis of requirements and reverse-engineering of the target environment to create a detailed technical design.

02

Secure Development (SDL)

C/C++/C# code is developed with continuous static analysis (SAST) and rigorous peer review.

03

Iterative PoC & Prototyping

We build functional modules to validate assumptions and demonstrate viability in the target environment early.

04

Rigorous QA & Performance Testing

We conduct multi-faceted testing for stability, resource consumption, and compatibility compatibility across Windows versions, including in HVCI/WDAC environments.

05

Deployment & Handoff

We deliver signed binaries, complete source code, and comprehensive documentation. No vendor lock-in.

Supported Technologies & Environments

Category	Technology / Environment	Level of Expertise
Windows Security	HVCI, WDAC, Credential Guard	**Expert** (Agent/Driver Development & Compatibility)
Active Directory	Large-Scale (500-1000+ DCs)	**Expert** (Performance optimization, zero-lag monitoring)
Authentication	Kerberos, NTLM, LSASS Internals	**Expert** (Packet-level analysis, safe instrumentation)
Windows Internals	Kernel Drivers (WDM/WDF), Callbacks	**Expert** (NTFS Minifilters, Process/Registry hooks)
File Systems	NTFS Driver Ecosystem, IRP Analysis	**Expert** (Minifilter development for Windows Servers)
NAS Storage	NetApp, EMC	**Advanced** (Driver-level monitoring via host)
Development	Microsoft Driver Signing Pipeline	**Core** Competency (EV Certs, HLK, WHQL)

Flexible Engagement Models for Security Engineering Teams

Dedicated Engineer

What: One or more Simpity engineers integrated directly into your team.

When: For long-term projects or augmenting your team with continuous Windows internals expertise.

Timeline: 6-24+ months.

Specialized Team

What: A self-contained Simpity team (devs, RE, QA) building a complex, standalone module.

When: To accelerate time-to-market for a new feature by outsourcing the most challenging component.

Timeline: 3-12 months.

Project Delivery

What: A fixed-scope engagement for a PoC, a specific module, or resolving a critical compatibility issue.

When: A low-risk way to validate a technical approach or solve an urgent, isolated problem.

Timeline: 1-4 months.

A Multi-Disciplinary Team Built for Deep Security

Our Mission

To provide security product companies with the specialized engineering required to build robust, reliable, and bypass-proof solutions. We are the engineering team for the engineers.

Our Specialist Roles

Kernel Engineers: C/C++ experts in WDM/WDF, focused on stability and performance under HVCI/WDAC.

Reverse Engineers: Specialists in IDA Pro & WinDbg who analyze undocumented Windows components.

Detection Engineers: Experts who translate attack behaviors into high-fidelity detection logic.

Active Directory Architects: Senior engineers experienced in monitoring global-scale AD environments.

Secure Coding Specialists: Leaders of our SDL, code reviews, and Microsoft signing pipeline.

Request a Technical Consultation or Explore Our Research

Request a Confidential Technical Consultation

Discuss your specific engineering challenge with our senior architects. All engagements are covered by a strict NDA.

info@simpity.eu

Schedule a Call

Access Our Engineering Briefs

- [PDF] Architectural Overview: Kernel-Level Interception
- [PDF] Methodology: Our Patch Tuesday Response Process
- [PDF] Technical Deep Dive: Instrumenting LSASS Safely
- [PDF] Simpity Deep Security Engineering Overview