ENDPOINT DETECTION & RESPONSE

ARCHITECTURAL FOUNDATIONS, ADVERSARIAL EVASION MECHANICS, AND THE AI-DRIVEN EVOLUTION

AUG 2026

FRONT-CODE.COM

Architectural Foundations, Adversarial Evasion Mechanics, and the AI-Driven Evolution of Endpoint Detection and Response

Keywords: *Endpoint Detection and Response (EDR), Extended Detection and Response (XDR), ETW-TI, Kernel Callbacks, eBPF, System Calls (Direct/Indirect), BYOVD, Provenance Graphs, GraphSAGE GNN, AI SOC Agents*

AUG 2026

Audience: *Cybersecurity leaders, enterprise identity administrators, threat intelligence analysts, and security engineering professionals*

Scope: *An exhaustive analysis of sub-kernel telemetry mechanisms, adversarial evasion techniques (syscalls, unhooking, BYOVD), causality-aware system provenance graph analytics, and AI-driven autonomous SOC triage architectures*

Front-Code.com

1. Historical Evolution and Conceptual Foundations

1.1 Paradigm Shift: From Static Antivirus to Continuous Telemetry

The operational architecture of enterprise endpoint security underwent a fundamental structural shift in July 2013 when Anton Chuvakin of Gartner introduced the taxonomy of Endpoint Threat Detection and Response (ETDR)[1]. This classification was formulated to address the systemic failures of conventional, signature-based Antivirus (AV) utilities and early Endpoint Protection Platforms (EPP)[5]. Legacy defensive mechanisms relied primarily on static indicators of compromise (IoCs), cryptographic file hashes, and rigid byte-sequence heuristics[5]. These methods proved structurally incapable of detecting polymorphic binaries, fileless memory-resident exploits, living-off-the-land (LotL) execution, and multi-stage Advanced Persistent Threat (APT) campaigns[3]. By 2015, industry nomenclature consolidated around the term Endpoint Detection and Response (EDR), establishing a defensive paradigm anchored in uninterrupted host-level telemetry ingestion, behavioral anomaly correlation, and post-breach response automation[1].

The empirical necessity for host-level behavioral visibility is demonstrated by modern threat metrics[3]. Historical security analyses reveal that up to 90% of successful enterprise intrusions and 70% of high-impact data breaches originate from compromised endpoint hosts[3]. Endpoints represent the most exposed and vulnerable interface within corporate infrastructure, serving as the primary access vector for privilege escalation, credential harvesting, and lateral movement[2]. Historically, organizations required an average of 287 days to identify and fully contain a complex data breach, accumulating immense financial and operational damages[5]. EDR architectures were engineered to eliminate this operational dwell time by shifting defensive strategies from static perimeter prevention to continuous runtime observability and automated host-level isolation[4].

1.2 Core Architectural Functional Pillars

Modern enterprise EDR platforms function as the core observational sensor network within contemporary Security Operations Centers (SOCs)[9]. The operational mandate of a modern EDR framework is structured across four primary technical pillars[1]:

Continuous Endpoint Telemetry Ingestion establishes uninterrupted recording of host-level system primitives[2]. This includes tracking process execution trees, parent-child process relationships, module loads, file system modifications, registry hive operations, active network socket bindings, handle creation events, and virtual memory allocations[2].

Behavioral Anomaly and Threat Detection applies real-time behavioral policy evaluation, dynamic Indicators of Attack (IoAs), static and runtime machine learning models, and complex heuristic analytics against incoming event streams to identify malicious execution sequences

as they unfold[4].

Automated Containment and Incident Response executes programmatically defined remediation playbooks upon threat detection[2]. Response mechanisms include isolating compromised hosts from adjacent network segments, terminating unauthorized process subtrees, revoking active access tokens, invalidating malicious handle references, quarantining compromised binaries, and reversing unauthorized registry modifications[2].

Forensic Investigation and Threat Hunting provides security analysts with interactive command interfaces, such as Live Discover and Live Response subsystems[3]. These interfaces enable rapid telemetry searching, root cause analysis (RCA), complete execution timeline reconstruction, and isolated payload detonation within dynamic sandbox environments[3].

1.3 Distributed Sensor Pipeline and Cloud Analytics Backends

An enterprise EDR platform is implemented as a distributed processing system composed of lightweight host-resident software agents, local real-time collection pipelines, and multi-tenant cloud analytics backends[2].

The host agent operates across both user-space and kernel-space environments to maintain uninterrupted surveillance over operating system primitives[11]. In user-space, the agent injects dynamic link libraries (DLLs) into running process contexts to establish API hooks that monitor runtime parameter passes[12]. In kernel-space, the agent registers executive notification callbacks and deploys hardware-assisted filter drivers to intercept low-level operations before user-mode code can tamper with the event stream[8].

Collected event telemetry is normalized on the host, passed through a circular ring buffer, compressed, encrypted, and streamed over secure TLS channels to a centralized cloud architecture[2]. The cloud backend aggregates high-velocity event streams from millions of distributed sensors, correlates multi-host event sequences against global threat intelligence repositories, applies computationally intensive graph analytics and machine learning inference, and projects actionable, context-enriched alerts to SOC analysts[4].

Technology Category	Primary Focus	Telemetry Source	Detection Engine	Response Capabilities	Structural Limitations
Antivirus (AV)	Known file payload blocking	Disk-based static file scans, static byte signatures	Hash matching, static byte sequences	File quarantine, host execution blocking	Ineffective against polymorphic binaries, fileless memory attacks,

					zero-days[3]
Endpoint Protection Platform (EPP)	Pre-execution threat prevention	File system, basic process memory execution state	Heuristics, static machine learning	Process termination, basic host network isolation	Blind to sophisticated post-exploitation, living-off-the-land techniques[5]
Next-Gen Antivirus (NGAV)	Context-aware execution blocking	Runtime process state, script execution context	Behavioral ML, script monitoring, IOA rules	Automated process killing, system state restore	Vulnerable to user-land hook stripping, kernel-level driver abuses[5]
Endpoint Detection & Response (EDR)	Post-execution detection, response, threat hunting	Comprehensive kernel & user-space event telemetry	Behavioral policy engines, provenance graphs, AI models	Network quarantine, handle killing, live response shell	Vulnerable to BYOVD, syscall unhooking, unmonitored OS subsystems[3]
Extended Detection & Response (XDR)	Cross-domain telemetry correlation	Endpoints, network traffic, cloud, identity (ITDR)	Cross-vector behavioral analytics, UEBA, AI agents	Automated multi-domain orchestration, SOAR playbooks	High deployment complexity, ingestion cost scalability bounds[1]

2. Low-Level Telemetry Generation and Sub-Kernel Observability

2.1 Event Tracing for Windows (ETW) and ETW Threat Intelligence (ETW-TI)

Event Tracing for Windows (ETW) serves as a core, high-throughput telemetry logging mechanism embedded within the Windows operating system architecture[8]. The ETW framework operates via a decoupled publish-subscribe model comprising Controllers, Providers, and Consumers[8]. While general ETW providers yield extensive diagnostic and operational performance data, Microsoft developed Event Tracing for Windows Threat Intelligence (ETW-TI) to supply non-bypassable security telemetry directly to security agents[8]. ETW-TI is implemented directly within the core kernel executable (ntoskrnl.exe), executing instrumentation routines at the native system invocation level[8]. Unlike standard user-mode ETW providers whose registration routines can be unhooked or blinded in user-space memory, ETW-TI generates kernel-signed telemetry events for sensitive operating system operations[8]. These operations include virtual memory management (VirtualAllocEx, WriteProcessMemory), remote thread creation (CreateRemoteThread), thread context manipulation (SetThreadContext), process hollowing, and driver loading sequences[8]. Because ETW-TI events are generated within kernel execution context during the execution of system call primitives, user-mode applications cannot disable this logging stream through standard memory modification techniques, making ETW-TI a primary telemetry source for detecting complex process memory injection[8].

2.2 Kernel Callbacks and Executive Interception

To achieve proactive execution blocking and real-time operational control, enterprise EDR solutions depend heavily on Windows Kernel Callback Mechanisms[8]. Kernel callback routines are registered by digitally signed, Kernel Mode Code Signing (KMCS) compliant filter drivers deployed by security vendors[12]. These callbacks instruct the Windows Executive subsystem to notify the security driver immediately prior to or following key kernel transactions[12]. Core kernel callback implementations include:
Object Callbacks (ObRegisterCallbacks) intercept process, thread, and desktop handle

creation or duplication requests[12]. This mechanism permits an EDR kernel driver to inspect the requesting process context and modify requested access masks in real time[12]. For instance, if an untrusted binary attempts to acquire a handle to a high-privilege system process with PROCESS_VM_WRITE or PROCESS_CREATE_THREAD rights, the EDR driver can strip these specific permission bits from the returned handle object, preventing downstream code injection while allowing benign process operations to continue[12].

Process and Thread Notification Routines (PspCreateProcessNotifyRoutine, PspCreateThreadNotifyRoutine) fire whenever a new process or thread is instantiated within the executive subsystem[2]. These routines allow the EDR driver to pause execution while inspecting parent process lineage, command-line execution parameters, environment blocks, process path locations, and cryptographic binary signatures before execution resumes[2].

Image Load Notification Routines (PspLoadImageNotifyRoutine) execute whenever an executable image, dynamic link library, or driver module is mapped into virtual memory space[8]. This callback facilitates immediate verification of code signing structures, detection of unauthorized DLL side-loading, and tracking of suspicious module mappings across host memory[8].

File System Minifilter Drivers attach to the I/O manager stack to monitor Input/Output Control (IOCTL) requests and file operations (file read, write, delete, rename, and extended attribute modification)[3]. Minifilter drivers provide the foundational telemetry required to identify rapid, unauthorized cryptographic operations characteristic of ransomware payloads[3].

2.3 Operating System Heterogeneity: macOS EndpointSecurity and Linux eBPF

Securing multi-platform enterprise infrastructures requires deep telemetry generation mechanisms tailored to non-Windows operating system architectures[11].

On macOS environments, contemporary EDR design has transitioned away from legacy Kernel Extensions (Kexts), which induced kernel panics and instability, in favor of Apple's EndpointSecurity API[11]. The EndpointSecurity framework exposes a C-based system extension subsystem communicating directly with the macOS kernel[11]. User-mode security agents—such as the open-source agent Sinter—subscribe to authorization (ES_EVENT_TYPE_AUTH_*) and notification (ES_EVENT_TYPE_NOTIFY_*) callback routines[11]. Authorization callbacks require the kernel to pause execution of a system event (such as process execution, file modification, or socket binding) until the security agent evaluates the context and explicitly returns an ES_AUTH_RESULT_ALLOW or ES_AUTH_RESULT_DENY verdict[11]. macOS telemetry collection focuses on tracking installer execution paths, including PKG installer pre-install and post-install scripts, DMG application bundle mounts, and third-party library installations (e.g., Python PIP modules)[11].

On Linux enterprise server infrastructures, traditional logging frameworks centered on the native auditd subsystem[16]. However, modern Linux EDR architectures rely on Extended Berkeley Packet Filters (eBPF)[11]. eBPF allows sandboxed, custom byte-code programs to execute directly within the Linux kernel context in response to specific system events without requiring custom

kernel module compilation or modifying kernel source code[11]. Security agents attach eBPF programs to kernel tracepoints, kprobes, uprobes, and socket filters, capturing process execution histories, file access patterns, and raw network packet attributes with minimal performance overhead[11]. Despite its efficiency, new Linux kernel interfaces—such as the io_uring asynchronous I/O framework—introduce unmonitored attack vectors if EDR agents are not explicitly configured to audit asynchronous ring buffer submissions[12].

3. Adversarial Evasion Methodologies and Subversion Mechanics

3.1 User-Mode API Hooking and Hook Bypass Techniques

To evaluate API parameter calls without incurring the performance costs of continuous Ring 3 to Ring 0 kernel context switches, EDR software injects vendor-specific dynamic libraries into user-mode processes[12]. These dynamic libraries establish API hooks within key export functions inside ntdll.dll (e.g., NtOpenProcess, NtAllocateVirtualMemory, NtWriteVirtualMemory, NtCreateThreadEx)[8]. Under standard operating conditions, an inline hook overwrites the initial assembly instructions of an export function with an unconditional jump (JMP) instruction[8]. This jump redirects execution flow into the EDR agent's inspection memory space, where behavioral policies analyze function parameters before transferring execution back to the native system call routine[8].

Adversaries routinely bypass user-mode inspection mechanisms through API unhooking and system call abuse[12]. API unhooking involves allocating memory to read a clean, unpatched copy of ntdll.dll directly from host storage, from \KnownDlls\, or from a clean, suspended child process[8]. The attacker then overwrites the hooked .text memory section of the in-memory ntdll.dll module with the pristine, unpatched assembly code from disk, stripping the EDR's JMP redirection instructions[8].

To bypass user-land API functions entirely without performing memory overwrites, offensive tools utilize Direct System Calls[12]. Instead of invoking NtWriteVirtualMemory through ntdll.dll, the malware embeds custom assembly code within its own executable binary[8]. This code loads the specific System Call Number (SSN) into the CPU register (EAX on x64) and issues the syscall opcode directly from its private memory allocation[8]. Frameworks such as Hell's Gate, Halo's Gate, and Tartarus' Gate dynamically parse the export address table of ntdll.dll at runtime or inspect adjacent unhooked neighbor functions to resolve correct SSNs across varying Windows build versions, bypassing user-mode hooked functions[8].

3.2 Advanced Call Stack Spoofing and Subsystem Evasion

To counter direct system calls, modern EDR detection engines analyze thread execution stacks

during system call processing using ETW-TI kernel trace events or sub-kernel stack unwinding[8]. When a syscall instruction transitions execution into kernel mode, the kernel inspects the execution context[12]. If the return instruction address on the call stack points to an unbacked, private executable memory space (RWX) rather than a valid instruction sequence within ntdll.dll or win32u.dll, the EDR flags a call stack anomaly[12].

To bypass stack unwinding validation, offensive toolsets employ Indirect System Calls[8]. In an indirect system call sequence, the malware sets up function arguments and loads the appropriate SSN into registers within its private memory, but deliberately avoids executing the syscall opcode locally[8]. Instead, the malware jumps to the location of a legitimate syscall; ret instruction sequence residing inside the clean, mapped address space of ntdll.dll[8]. When the kernel transitions to Ring 0, the return address recorded on the call stack points directly into ntdll.dll, satisfying call stack unwinding integrity checks[12].

Extending subsystem exploitation further, research presented at DEF CON 32 introduced HookChain[12]. HookChain bypasses user-mode inspection by targeting subsystem abstraction interfaces operating directly above ntdll.dll, manipulating execution flow before parameters enter hooked API functions[12]. In empirical testing across 26 commercial enterprise EDR agents, HookChain achieved an 88% overall detection bypass rate, highlighting ongoing vulnerabilities in user-space hook verification models[12].

3.3 Dynamic Telemetry Blinding: AMSI Patching and ETW Disabling

When threat actors execute malicious payloads within managed or interpreted runtime environments (such as PowerShell, CScript, or the .NET Common Language Runtime), execution transparency is provided by the Antimalware Scan Interface (AMSI)[8]. AMSI passes script buffers and dynamic memory allocations to the installed security engine prior to execution[8]. Threat actors suppress AMSI logging by executing user-mode memory patches targeting amsi.dll[8]. An exploit sequence resolves the memory location of AmsiScanBuffer or AmsiScanString, alters memory protection flags to PAGE_EXECUTE_READWRITE via VirtualProtect, and overwrites the initial assembly instructions with byte codes that return an AMSI_RESULT_CLEAN status or an execution error, disabling script scanning for the remainder of the process lifecycle[8].

Similarly, user-mode Event Tracing for Windows can be suppressed by patching the EtwEventWrite function within ntdll.dll[8]. Because EtwEventWrite handles serializing and transmitting user-space event logs to the ETW subsystem, overwriting the entry point of EtwEventWrite with an assembly return opcode (RET / 0xC3) immediately neutralizes user-mode ETW event generation without terminating the host process[8].

3.4 Kernel-Level Destruction: Bring Your Own Vulnerable Driver (BYOVD)

When user-mode manipulation techniques prove insufficient, advanced threat actors escalate execution privileges to Ring 0 via Bring Your Own Vulnerable Driver (BYOVD) attacks (MITRE ATT&CK T1068, T1014)[12]. While Windows Kernel Mode Code Signing (KMCS) prevents untrusted binaries from loading into kernel memory, BYOVD attacks exploit legitimately signed, trusted third-party drivers that contain arbitrary memory read/write vulnerabilities[12].

Once loaded with administrative privileges, the attacker uses custom Input/Output Control (IOCTL) requests to exploit the driver's memory access primitives[12]. The attacker uses these primitives to locate and modify kernel memory structures, zeroing out entries in kernel callback arrays (PspCreateProcessNotifyRoutine, ObRegisterCallbacks)[12]. Removing these pointers unregisters the EDR driver's visibility mechanisms at the kernel level[12]. Furthermore, BYOVD attacks can target protected user-mode processes (PPL - Protected Process Light) by stripping process protection flags or revoking security access tokens directly within the kernel EPROCESS structure, permitting process termination[12].

The operational weaponization of BYOVD driver exploitation has expanded significantly[12]. A single campaign abusing vulnerabilities in the TrueSight driver deployed over 2,500 distinct driver variants within a brief operational window[12]. In February 2026, the Reynolds ransomware strain integrated this process by embedding a vulnerable NsecSoft driver (CVE-2025-68947) directly within its primary executable binary[12]. This design eliminated the need for a secondary driver-staging phase, enabling the ransomware payload to unregister kernel callbacks and disable host security services within milliseconds of execution[12]. Threat analysis has documented similar driver abuse involving signed components, including the EnCase driver, deployed specifically for host security agent termination[12]. While Kernel Patch Protection (KPP / PatchGuard) periodically audits critical kernel tables, BYOVD modifications target dynamic data structures that execute between PatchGuard auditing cycles[8].

3.5 Process Injection, Living-off-the-Land, and Unmonitored Surface Commoditization

To avoid triggering static binary detections, modern threat actors minimize disk footprint by utilizing Living-off-the-Land Binaries and Scripts (LOLBAS) combined with memory injection[4]. Process injection methods continue to evolve to evade detection of memory allocation primitives[14]. Advanced frameworks abuse native OS interfaces, such as combining the Windows Fork API with trusted processes like OneDrive.exe, to execute shellcode without allocating new Read-Write-Execute (RWX) memory regions[15].

Furthermore, EDR bypass capabilities have commoditized rapidly[12]. EDR evasion utilities are traded widely across dark web markets, with standalone bypass tools starting at $300 and fully integrated cryptor-locker packages priced up to $10,000[12]. This accessibility enables a broad

spectrum of threat actors to deploy evasion capabilities historically restricted to advanced state-sponsored groups[12]. In red team testing conducted by the Cybersecurity and Infrastructure Security Agency (CISA), conventional EDR solutions detected only a small fraction of deployed payloads when evaluated against skilled adversaries[12]. Adversaries also bypass endpoint surveillance by pivoting to unmonitored devices—such as embedded Linux IoT hardware, perimeter network appliances, and smart webcams (as observed in Akira ransomware incidents)—where EDR agents cannot be deployed[12].

Evasion Category	Specific Technique	Low-Level Technical Mechanism	Adversarial Impact	Targeted Telemetry Vector	Primary Mitigation Strategy
User-Mode Hook Bypass	Direct System Calls (Hell's Gate / Halo's Gate)[14]	Invokes syscall instruction directly from private memory[12]	Bypasses all inline user-space API inspection hooks[12]	User-mode ntdll.dll inline API hooks[8]	ETW-TI stack unwinding, non-NTDLL syscall address checking[12]
Call Stack Evasion	Indirect System Calls[8]	Jumps to syscall; ret instructions inside mapped ntdll.dll [cite: 8, 12, 15]	Spoofs call stack return addresses, passing unwinding checks[12]	Kernel stack unwinding, thread context checks[12]	Hardware-assisted stack auditing, sub-kernel trace correlation
Subsystem Abstraction	HookChain Framework[12]	Manipulates subsystem execution interfaces operating above NTDLL[12]	Achieves 88% detection bypass across tested EDR engines[12]	Subsystem API wrappers, high-level hooks[12]	Comprehensive kernel trace correlation, hardware instruction tracing
Telemetry Suppression	AMSI Memory Patching[8]	Overwrites AmsiScanBuffer entry	Blinds script logging engines	User-mode AMSI script memory	Kernel-level byte-code auditing,

		point with immediate return bytes[8]	(PowerShell, .NET)[8]	buffers[8]	static script analysis
Telemetry Suppression	User-Mode ETW Patching[8]	Overwrites EtwEventWrite entry point with assembly opcode 0xC3 (RET)[8]	Neutralizes user-space ETW event serialization[8]	User-mode ETW telemetry providers[8]	ETW-TI kernel event generation, module integrity monitoring[8]
Kernel Subversion	BYOVD Driver Abuse (CVE-2025-68947)[12]	Exploits signed drivers to obtain Ring 0 arbitrary memory read/write[12]	Unregisters kernel callbacks, terminates PPL protected agents[12]	Kernel callback arrays, PPL process flags[12]	Hypervisor-Protected Code Integrity (HVCI), driver blocklisting
Context Execution	Windows Fork API Injection[15]	Abuses process cloning and trusted system binaries (OneDrive.exe)[15]	Conceals shellcode execution inside trusted contexts without RWX [cite: 15]	Handle creation events, memory region allocations[15]	Behavioral process graph tracking, non-RWX execution policies

4. System Provenance Graphs and Causality Analysis

4.1 Topology and Mathematics of System Provenance Graphs

To move beyond point-in-time detection and identify stealthy APT activity, modern EDR platforms employ System Provenance Graphs[16]. A system provenance graph is a directed acyclic graph $G = (V, E)$ constructed continuously from host kernel audit records[16]. Nodes (V) represent system entities, which are categorized into three core functional classes:

1. **Process Nodes ($V_{process}$)**: Active process instances executing within virtual memory spaces (e.g., cmd.exe, powershell.exe, slack.exe)[16].
2. **File Nodes (V_{file})**: Persistent storage objects, configuration files, binary modules, and dynamic libraries[16].
3. **Network Socket Nodes (V_{socket})**: Network abstractions representing active or historical socket connections[16].

Edges (E) represent directed, timestamped system interactions executed between entities[16]. Directed edges capture causal relationships, such as process creation ($V_{process} \rightarrow V_{process}$ via fork/exec), file modification ($V_{process} \rightarrow V_{file}$ via write), file read operations ($V_{file} \rightarrow V_{process}$ via read), and network communication ($V_{process} \rightarrow V_{socket}$ via connect)[16]. Provenance graphs provide a structured, causal record of operating system execution history[16].

4.2 Causal Traversal Vectors and the Dependency Explosion Phenomenon

When an EDR sensor detects a Point-of-Interest (POI) event—such as an unauthorized process memory write or credential extraction attempt—analysts execute causal graph traversal algorithms[16]. Causality analysis operates across two primary directional vectors[20]:
Backtracking Analysis traverses directed edges backward in time from the POI node to identify the initial entry vector[19]. For example, backtracking can trace a malicious reverse shell process back through an injected legitimate application (e.g., slack.exe), to a macro execution event, and ultimately to the initial phishing email file download[19].
Forward-Tracking Analysis traverses directed edges forward in time from the identified entry vector to determine total operational impact[16]. This process maps every file modified, registry key altered, downstream process spawned, and external IP address contacted during the attack lifecycle[16].
The core challenge facing provenance graph analysis is the Dependency Explosion Problem[16]. In long-running operating systems, utility processes (e.g., explorer.exe, svchost.exe, systemd) interact continuously with thousands of benign system files, sockets, and child processes[18]. Standard graph traversal algorithms treat every historical interaction as causally connected, causing the graph search space to expand exponentially[16]. Consequently, an investigation query centered on a single POI node can return a backtracking graph containing hundreds of thousands of irrelevant system edges, exhausting system memory, slowing query performance, and overwhelming analyst capacity[16].

4.3 Graph Neural Networks and Machine Learning Over Provenance Topologies

To automate anomaly detection across complex provenance topologies without relying on static rule generation, security researchers apply Graph Representation Learning[17].
UNICORN models continuous runtime provenance graphs using graph streaming and temporal histogram abstractions[22]. UNICORN generates a baseline profile of normal system behavior, enabling it to detect anomalous APT activity that deviates from standard topological execution patterns without requiring pre-defined attack signatures[22].
PROGRAPHER mitigates dependency explosion by converting kernel audit streams into temporal graph snapshots[21]. It maps nodes within each snapshot into low-dimensional vector representations using a Rooted Subgraph (RSG) embedding approach via graph2vec[21]. Sequence learning models evaluate snapshot transitions over time, identifying structural anomalies and presenting concise attack indicators to analysts[21].
ProvG-Searcher translates threat hunting into a vector-based approximate subgraph matching problem[17]. ProvG-Searcher converts external threat intelligence descriptions into query subgraphs, projecting graph structure into an order embedding space where causality relationships are evaluated via vector distance operations[17].
GraphSAGE Graph Neural Networks (GNNs) leverage neighborhood aggregation algorithms ($K =$ hop parameters with max-pooling aggregators) to generate structural node embeddings, which feed into Multi-Layer Perceptron (MLP) classification layers[19]. In evaluation against simulated APT scenarios involving the Empire framework—where attackers execute phishing attachment drops, fileless PowerShell payloads, C2 channel establishment, and code injection into legitimate applications like slack.exe—GraphSAGE GNN models isolated critical attack paths while suppressing background system noise, achieving a 97.67% reduction in false positive alerts and executing detection inference in 0.068 seconds[19].

4.4 Advanced Storage Architectures and Streaming Graph Pruning

Because kernel-level auditing generates gigabytes of raw provenance telemetry per host daily, maintaining long-term historical graph records creates significant data storage challenges[16]. Advanced storage and pruning architectures directly address these constraints:
DEHYDRATOR is an efficient provenance graph storage architecture designed to replace conventional relational (PostgreSQL) and graph (Neo4j) database systems[16]. DEHYDRATOR applies field mapping encoding to eliminate field-level log redundancy, followed by hierarchical structural encoding[16]. It trains a Deep Neural Network (DNN) to store graph topology patterns, maintaining concise error-correction tables to ensure fully lossless graph reconstruction[16]. In evaluations across datasets containing over one billion system audit records, DEHYDRATOR achieved an 84.55% reduction in total storage overhead while executing queries $7.16\times$ faster

than Neo4j and $7.36\times$ faster than PostgreSQL[16].
SParse optimizes real-time incident response by filtering background system noise from streaming audit logs[20]. SParse constructs a Suspicious Semantic Graph (SSG) from incoming telemetry streams using semantic transfer rules[20]. Upon encountering a POI event, SParse applies path-level influence metrics to prune non-essential causal edges[20]. In empirical tests on enterprise attack datasets, SParse compressed a backtracking graph containing 227,589 edges down to a critical component graph of 113 edges in 1.6 seconds, demonstrating a $25\times$ improvement in edge filtering efficiency over legacy methods[20].
TAPAS addresses graph expansion by implementing a process backbone transformation[18]. Recognizing that process nodes anchor system execution lineages, TAPAS strips away passive attachment nodes (such as short-lived temporary files) to preserve a streamlined process graph backbone[18]. Simultaneously, TAPAS applies temporal changed-localization algorithms to prune invariant, inactive graph regions, controlling long-term graph memory growth while retaining multi-month APT tracking capabilities[18].

Graph Framework / System	Core Computational Methodology	Storage & Graph Reduction Strategy	Target Adversarial / Operational Threat	Key Empirical Metric / Result
UNICORN [cite: 22]	Unsupervised graph streaming & temporal histogram modeling[22]	Continuous state abstraction and temporal clustering[22]	Long-term, stealthy APT campaign anomalies[22]	Signature-free detection of long-running anomalous behavior[22]
PROGRAPHER [cite: 21]	Snapshot extraction + Rooted Subgraph (graph2vec) embeddings[21]	Temporal snapshot segmentation, inactive edge pruning[21]	Dependency explosion in long-running audit logs[21]	Automated extraction of key attack indicators[21]
ProvG-Searcher [cite: 17]	Subgraph order embeddings & vector space mapping[17]	Node/edge aggregation graph simplification[17]	Pattern matching for known CTI threat subgraphs[17]	High-accuracy historical log searching with low false match rates[17]

GraphSAGE GNN [cite: 19]	Neighborhood aggregation ($K =$, max-pooling) + MLP classifier[19]	Structural node feature vector embedding[19]	Fileless execution, Empire framework injection[19]	**97.67% false alarm reduction**; 0.068s detection time[19]
DEHYDRATOR [cite: 16]	Field-mapping + hierarchical encoding + DNN representation[16]	Deep neural network memory with error-correction tables[16]	Provenance log storage scalability bounds[16]	**84.55% storage space reduction**; $7.16\times$ faster than Neo4j[16]
SParse [cite: 20]	Suspicious Semantic Graph (SSG) + path influence filtering[20]	Streaming path pruning based on semantic context[20]	Latency explosion during POI backtracking queries[20]	Compressed **227,589 edges to 113 edges in 1.6s** [cite: 20]
TAPAS [cite: 18]	Process backbone extraction + temporal changed-localization[18]	Process-only spatial backbone retention + temporal pruning[18]	Long-term provenance graph memory expansion[18]	Eliminates spatial and temporal graph redundancy[18]

5. Artificial Intelligence, Autonomous SOC Operations, and XDR Convergence

5.1 The Operational SOC Crisis: Alert Fatigue and Telemetry Saturation

Despite the observational capabilities of modern EDR agents, enterprise Security Operations Centers face a major operational challenge: alert fatigue[9]. High-velocity event collection across EDR sensors, SIEM engines, and network monitors creates a continuous stream of upstream alerts[9]. Security operational surveys indicate that mid-sized organizations manage an average of 960 security alerts daily, while large enterprise environments routinely receive over 3,000

daily alerts[9].
Because human analyst capacity is inherently limited, SOC teams are frequently forced to adopt risky operational compromises[9]. Up to 60% of security teams report suppressing strict detection rules or ignoring lower-priority alert queues entirely to prevent operational paralysis, creating windows of opportunity for threat actors operating below default alert thresholds[9].
The financial impact of this operational backlog is severe, as uninvestigated alerts allow attacker dwell times to extend across months[5]. Consequently, downstream AI-driven alert screening has become a core operational requirement for enterprise security architectures[9].

5.2 Paradigm Shift: Autonomous AI SOC Agents vs. Legacy Automation

The integration of Artificial Intelligence into security operations represents a major structural shift in threat investigation[9]. On the Gartner Hype Cycle for Security Operations, AI SOC agents represent an emerging Innovation Trigger[9]. Global enterprise investments in security AI and Generative AI (GenAI) reached $307 billion, accelerating a transition from manual dashboarding to autonomous SecOps workflows[9].
AI-driven triage architectures differ fundamentally from previous generations of operational tools[9]:
Security Information and Event Management (SIEM) platforms function primarily as centralized log repositories, alerting analysts that a security event occurred[9].
Security Orchestration, Automation, and Response (SOAR) technologies execute deterministic, pre-written, rule-based playbooks that lack adaptability when encountering novel attack patterns[9].
Interactive AI Copilots wait passively for human analyst prompts to summarize incident notes or format queries[9].
Autonomous AI SOC Agents operate as self-directed systems that autonomously initiate, investigate, correlate, and conclude alert investigations within established policy guardrails[9]. Rather than requiring analysts to review raw alert queues, AI agents evaluate every incoming event at machine speed and deliver complete, evidence-backed findings to analysts[9].

5.3 Architectural Paradigms of AI SOC Platforms

Enterprise adoption of AI SOC solutions is structured across three primary market architectural paradigms[9]:
AI-Native Investigation Platforms (e.g., Conifers CognitiveSOC, Intezer, Dropzone AI) are built from the ground up around autonomous investigation agents that manage end-to-end incident analysis[9].
SIEM-Embedded Copilots (e.g., Microsoft Security Copilot, CrowdStrike Falcon Platform, Palo Alto Networks Cortex XSIAM) integrate specialized AI assistants directly into established

enterprise security suits[9].
Hyper-Automation Frameworks (e.g., Torq HyperSOC) utilize high-speed execution engines to automate complex triage and decision-routing workflows[9].
Advanced solutions employ a Patent-Pending Mesh Agentic Architecture[9]. Rather than passing complex security incidents through a single Large Language Model (LLM)—which introduces execution latency and hallucination risks—Mesh architectures orchestrate specialized AI techniques[9]. These systems combine Domain-Specific Language Models (DSLMs), machine learning heuristics, statistical anomaly detectors, and graph representation models[9]. The architecture ingests institutional context—including asset criticality values, corporate risk policies, historical incident data, and baseline network configurations—enabling it to analyze EDR alerts in seconds and provide analysts with context-aware remediation recommendations[9].

5.4 Cross-Domain Correlation and Extended Detection and Response (XDR)

To eliminate host telemetry isolation, modern EDR architectures are rapidly expanding into Extended Detection and Response (XDR)[1]. While EDR provides granular observability over host OS kernels, XDR broadens this scope by unifying host telemetry with network traffic analysis, cloud workload monitoring, identity management interfaces, and Identity Threat Detection and Response (ITDR) modules[1].
Through open API integration frameworks, XDR platforms correlate host process execution anomalies with network telemetry and identity access events[1]. For example, if an EDR agent detects a process injection event, the XDR platform can automatically query User and Entity Behavior Analytics (UEBA) models to identify anomalous user access, review cloud infrastructure logs for concurrent unauthorized API calls, and execute automated orchestration playbooks to revoke compromised identity credentials across the enterprise[1]. This shift transforms EDR from an isolated host monitoring tool into an integrated component of an enterprise-wide security architecture[1].

AI / XDR Architectural Class	Core Operational Mechanism	Telemetry Integration Scope	Key Primary Strengths	Primary Operational Challenge
AI-Native Investigation Platforms	Autonomous multi-agent mesh orchestration[9]	Multi-vendor EDR, SIEM, identity logs[9]	End-to-end evidence-backed incident triage[9]	Integration overhead across legacy infrastructure[9]
SIEM-Embedd	Platform-nativ	Platform-nativ	Deep	Vendor lock-in,

ed Copilots	e LLMs & contextual assistants[9]	e telemetry sources[9]	integration within platform interfaces[9]	ingestion costs[9]
Hyper-Automation Frameworks	Event-driven agentic execution flows[9]	API-connected SecOps tools[9]	High-speed automated response execution[9]	Requires explicit guardrail configuration[9]
Extended Detection & Response (XDR)	Cross-domain multi-sensor correlation[1]	Endpoints, network, cloud, identity, email[1]	Comprehensive cross-vector threat visibility[1]	High deployment complexity, data normalization[9]

6. Strategic Engineering Conclusions and Operational Defense Roadmap

6.1 Synthesis of Findings

The technical evolution of Endpoint Detection and Response reflects a continuous competition between endpoint observability engines and low-level evasion techniques[1]. As threat actors commoditize advanced evasion primitives—ranging from dynamic user-mode hook removal and indirect syscalls to kernel-level BYOVD driver exploitation and living-off-the-land context execution—traditional point-in-time signature checks and static user-land hooks become increasingly insufficient[12].

To maintain effective endpoint defense, enterprise security architectures must advance across three primary engineering dimensions:

1. **Transition to Sub-Kernel and Hardware-Assisted Observability**: Organizations must reduce reliance on user-space dynamic library hooks (ntdll.dll) and prioritize non-patchable, kernel-native telemetry architectures (ETW-TI, eBPF) reinforced by hardware instruction auditing[8]. Driver integrity policies must enforce Hypervisor-Protected Code Integrity (HVCI) and continuous blocklisting of vulnerable signed drivers to neutralize BYOVD exploit vectors[12].
2. **Implementation of Causality-Aware Provenance Graph Analytics**: EDR platforms must incorporate dynamic graph representation learning mechanisms (such as GraphSAGE GNNs, SParse streaming path filtering, and DEHYDRATOR compressed graph storage)[16]. Evaluating system activity through historical process-file-network execution lineage rather than isolated alert triggers enables accurate identification of complex APT behavior while suppressing false positive alerts[16].

3. **Deployment of Autonomous AI SOC Triage Frameworks**: To resolve operational alert saturation, organizations must integrate specialized AI SOC agents into incident triage pipelines[9]. Utilizing mesh agentic architectures enables automated, machine-speed investigation of incoming EDR telemetry, allowing security analysts to shift from manual alert processing to proactive threat hunting and strategic security engineering[9].

6.2 Enterprise Endpoint Security Implementation Roadmap

The following structured roadmap outlines the phased technical requirements for deploying resilient, next-generation endpoint detection capabilities across enterprise operational environments:

Phase	Technical Pillar	Architectural Deployment Task	Target Technical Objective
Phase 1: Baseline Hardening	Sub-Kernel Observability	Enforce HVCI code integrity and deploy automated driver blocklisting policies[12].	Block BYOVD driver exploitation and unauthorized Ring 0 access[12].
Phase 1: Baseline Hardening	Telemetry Architecture	Enable ETW-TI providers and transition Linux auditing agents to eBPF tracing[8].	Secure non-bypassable, kernel-signed telemetry event generation[8].
Phase 2: Graph Analytics Integration	Causality Analysis	Deploy provenance graph engines using GraphSAGE GNN or SParse path filtering[19].	Automate root cause analysis and reduce false positive alert volume[19].
Phase 2: Graph Analytics Integration	Storage Optimization	Implement DEHYDRATOR hierarchical encoding and neural network compression[16].	Maintain multi-month provenance graph retention with low storage overhead[16].

Phase 3: Autonomous SecOps Shift	Agentic AI Triage	Deploy Mesh Agentic AI platforms for automated, context-aware alert investigation[9].	Execute machine-speed incident triage and eliminate manual queue backlogs[9].
Phase 3: Autonomous SecOps Shift	XDR Convergence	Integrate host telemetry with identity (ITDR), network, and cloud access gateways[1].	Enable automated, cross-domain containment playbooks across the enterprise[1].

Works cited

1. The Journey from Siloed Security to XSIAM - Palo Alto Networks, https://www.paloaltonetworks.com/resources/infographics/journey-to-xsiam
2. Who Invented EDR | History of EDR Security - Xcitium, https://www.xcitium.com/who-invented-edr/
3. Endpoint Detection and Response (EDR) Security Solutions - zenarmor.com, https://www.zenarmor.com/docs/network-security-tutorials/what-is-endpoint-detection-and-response-edr
4. What Is Endpoint Detection And Response (EDR)? | 2026 Guide - SelectHub, https://www.selecthub.com/endpoint-security/endpoint-detection-and-response/
5. edr - LMNTRIX, https://lmntrix.com/res/EDR_PUTTING-THE-X-FACTOR-IN-XDR.pdf
6. Cybersecurity Solutions Handbook - H-X Technologies, https://www.h-x.technology/security-solutions-full-review
7. Bakalářská práce - Theses, https://theses.cz/id/nqurjc/zaverecna_prace_Archive.pdf
8. Antivirus and EDR Bypass Techniques Explained - Vaadata, https://www.vaadata.com/en/blog/antivirus-and-edr-bypass-techniques/
9. Top 10 AI SOC Agents, Platforms and Solutions in 2026, https://www.conifers.ai/blog/top-ai-soc-agents/
10. AI-Driven Security Alert Screening and Alert Fatigue Mitigation in Security Operations Centers: A Survey - arXiv, https://arxiv.org/pdf/2605.08316
11. Infosec_Reference/Draft/L-SM-TH.md at master - GitHub, https://github.com/rmusser01/Infosec_Reference/blob/master/Draft/L-SM-TH.md?plain=1
12. EDR evasion: techniques, real-world breaches, and defenses - Vectra AI, https://www.vectra.ai/topics/edr-evasion
13. Best Extended Detection and Response Reviews 2026 | Gartner Peer Insights, https://www.gartner.com/reviews/market/extended-detection-and-response
14. direct-syscalls · GitHub Topics, https://github.com/topics/direct-syscalls
15. edr-bypass · GitHub Topics, https://github.com/topics/edr-bypass
16. arXiv:2501.00446v1 [cs.CR] 31 Dec 2024, https://arxiv.org/pdf/2501.00446
17. ProvG-Searcher: A Graph Representation Learning Approach for Efficient Provenance Graph Search - arXiv, https://arxiv.org/html/2309.03647v2
18. TAPAS: An Efficient Online APT Detection with Task-guided Process Provenance Graph Segmentation and Analysis - USENIX, https://www.usenix.org/system/files/usenixsecurity25-zhang-bo-tapas.pdf
19. Graph-Based Anomaly APT Attack Detection via Threat Intelligence, https://www.computer.org/csdl/journal/ec/2026/01/11410032/2en8nrat67u
20. SPARSE: Semantic Tracking and Path Analysis for Attack Investigation in Real-time - arXiv, https://arxiv.org/html/2405.02629v1
21. ProGraPher: An Anomaly Detection System based on Provenance Graph Embedding - USENIX,

https://www.usenix.org/system/files/usenixsecurity23-yang-fan.pdf
22. UNICORN: Runtime Provenance-Based Detector for Advanced Persistent Threats - James Mickens - Harvard University, https://mickens.seas.harvard.edu/files/2026/01/unicorn.pdf
23. What Is Endpoint Protection for Enterprises? - Palo Alto Networks, https://www.paloaltonetworks.com/cyberpedia/what-is-endpoint-protection
24. XDR | Stellar Cyber, https://stellarcyber.ai/category/xdr/