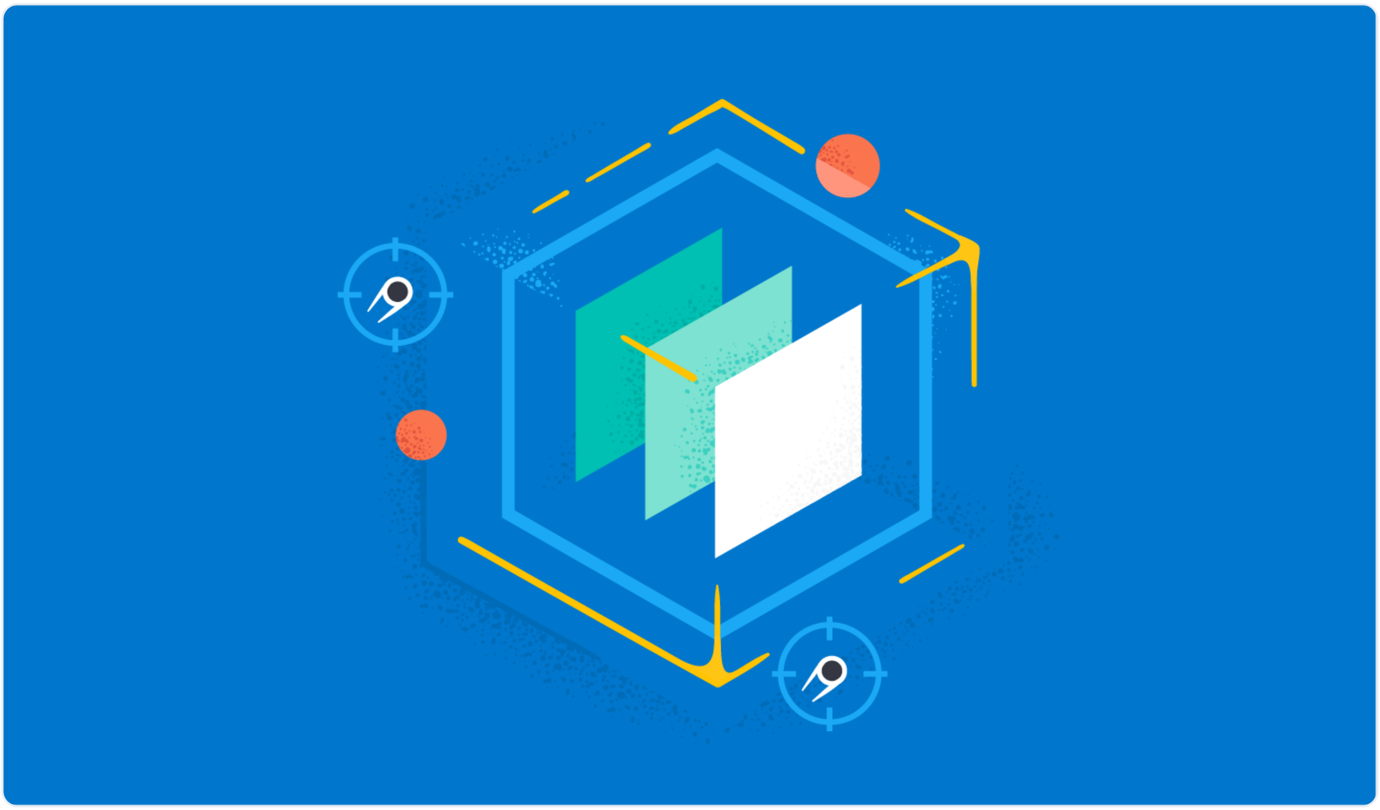
A taxonomy of endpoint security detection bypasses

By **John Uhlmann**

11 May 2023



I often see “EDR” used as a synonym for “industry-leading [endpoint security solutior](#)
There are times when this is accurate, but there are also times when I believe that this

generalization stymies discourse around current capability gaps in the endpoint security ecosystem.

In this blog post, I want to share my personal taxonomy for endpoint security products — albeit one that perhaps confusingly reuses existing terminology. The proposed taxonomy suggests using *complementary* feature set descriptions, each with strengths and weaknesses, rather than a generational definition with concentric features. This allows a more nuanced discussion about the specific defensive feature at play in a given scenario.

So yes, the “EDR” was bypassed at a point in time — but which set(s) of detection features were actually bypassed? Was it file content (AV), process behavior (EPP), or telemetry logging (EDR)?

Note: There is a significant Windows bias to this blog due to its market dominance in the endpoint operating system market.

The taxonomy

Evolution is messy. It is difficult to observe as it happens, and our understanding today may not be relevant tomorrow. This is also true if we could go back in time: we can't look at endpoint security through an outdated lens. So I’m going to define endpoint security by what I see now — which is not necessarily the same as what we understood at the time it started.

I’ve observed five distinct protection feature sets in contemporary endpoint security agents:

- **Endpoint Firewall** — Point in time network protection
- **Endpoint Antivirus (AV)** — Point in time executable content protection
- **Endpoint Protection Platform (EPP)** — Point in time runtime behavior (API) protection
- **Endpoint Detection and Response (EDR)** — retrospective network, content, and behavior detection
- **Data Loss Protection (DLP)** — sensitive data loss and leakage protection

	network	executable	behavior	behavior
point-in-time prevention	**Firewall**	**AV**	**EPP**	**DLP**
historical detection		EDR		

Currently (endpoint) **AV** is typically used to refer solely to the pre-execution **point-in-time scanning of binary or script content**. Modern AV typically includes a cloud platform component — to avail itself of greater computing resources to analyze first-seen samples and support slightly delayed point-in-time detection.

EPP was introduced as a market segment by Gartner to encompass the next evolution of endpoint security products. Indicative features included cloud assistance, behavioral detection, and remote response actions. Cloud assistance is now ubiquitous, and response is more commonly associated with EDR, so I've chosen to focus my EPP feature set definition on products that provide **point-in-time behavioral protection**. That is, products that will prevent certain API calls from executing based on the parameters specified and the system state.

A competing classification at the time was Next-Generation Antivirus (NGAV), though this term has since become increasingly more widely known for "signature-less" machine learning approaches to content classification.

A detection is the combination of some raw telemetry and some detection logic. The key differentiator for **EDR** over AV and EPP is that it sends **telemetry** — not just detections. The primary benefit is the ability for users to write (or tune) detection logic specific to the environment — though the trade-off is that these detections are inherently retrospective.

The secondary benefit, whether the vendor has realized it or not, is the ability to aspire toward **near-limitless historical detection**. This is distinct from the slightly delayed point-in-time detection that both AV and EPP can perform. An EDR arguably should be able to perform retrospective detection at times scaled in months and years, not just days and weeks. The exemplar here is that AV sends immediate alerts whereas EDR sends observed file hashes. If an AV updates its detection, it can only apply these from the current point in time onward — whereas an EDR can retrospectively indicate that malware was present in the past.

I have kept the focus on endpoint agent detection and prevention capabilities. There are also differences in the response capabilities of modern agents that I do not plan to address here. Related products in the security ecosystem have also been put to the side for now, particularly security orchestration, automation, and response (SOAR), managed detection and response (MDR), eXtended detection and response (XDR), and security information and event management (SIEM). I'm also ignoring DLP from here on.

Agent implementation details

So what does a x64 Windows Endpoint Security product look like under the hood in 2023?

The following courses provide a good overview of the landscape:

- Windows Filter Drivers by Winsider Seminars (Alex Ionsecu and Yardin Shafir)
- Windows Kernel Programming by Pavel Yosifovich

A simplified summary is as follows:

	Firewall	AV	EPP	EDR
Windows Filtering Platform callout driver **(or** Network filter driver**)**	✓			✓
File system filter driver and user-mode Antimalware Scan Interface		✓		✓
Kernel callbacks • Process • Thread • Image • Object • Registry			✓	✓

	Firewall	AV	EPP	EDR
Kernel ETW events, Security Event Log, and other Windows service ETW channels			✓[1]	✓
User mode hooks			⛔[2]	

[1]*EPP prevention must be synchronous, but EPP can still consume asynchronous signals such as ETW to inform future actions.*

[2]*User mode hooks are useful in limited scenarios such as sandboxing constrained runtimes to harden against exploitation and abuse. However, they cannot reasonably defend against malware that has achieved arbitrary user mode code execution. Similarly, endpoint security products implemented in the kernel cannot reasonably defend against kernel mode malware.*

One notable gap in the extension points that Microsoft provides is the lack of kernel callbacks when non-image executable user mode memory is allocated or modified.

Instead, Microsoft has only provided asynchronous ETW Threat Intelligence telemetry and a heavy-weight W^X mechanism called Arbitrary Code Guard. The former is excellent at uncovering long-term APT threats but asynchronous detection is arguably too slow to stop ransomware. The latter was designed primarily as an Edge exploitation mitigation and lacks adequate configuration flexibility to deploy in most environments.

Instead of waiting for Microsoft to improve its hardening, some leading-edge EPP vendors have chosen to implement their own exploitation protections using user mode hooks, though these approaches should eventually be superseded by Hardware-enforced Stack Protection. These hooks might also *opportunistically* provide some additional prevention, but they are not robust as a security boundary once code execution has been achieved. The "universal EDR bypass" claims are inaccurate, however.

This last resort hooking is also not to be confused with immature EPP products that have implemented user mode hooks where a kernel callback exists — likely in the pursuit of a faster time to market and now lingering as unpaid technical debt.

To be completely clear, we should not be deriding vendors for their (so far unsuccessful) attempts to use user mode hooks to defend a (likely) indefensible boundary. Instead, we should be imploring Microsoft to make the boundary defensible and to prioritize implementing the necessary security callbacks in the kernel.

So what's so great about EDR then?

The value of historical detection is best understood through the lens of 2013, when Anton Chuvakin **named this emerging category for Gartner**. Bitcoin started this year at just US$13, and cybercrime was mostly still a lot of banking trojans and credit card web-skimmers. Ransomware was still a cute footnote in threat reports, and it was the nation-state APT boogeyman that was headlining and keeping folks up at night. We all rallied around the cry to “assume breach” and the market delivered us EDR to create a hostile environment for these sophisticated APTs.

Case study: AV→EDR evolution

It is a well-established fact that file-based AV detections are trivial to bypass *at a point in time*. The actor simply needs to generate unique file content variations on a separate representative system until no detection occurs. They can then safely deploy on the target system. This is perhaps most famously expressed in David Bianco’s **Pyramid of Pain**. File content can be fingerprinted by its hash and (point-in-time) detections based on file hashes impose the least cost (pain) on adversaries.

In fact, if an actor continuously tests their deployed files for detection on their representative system, they can effectively bypass AV *for all time* by simply uploading new undetected variants to the target system before the endpoint AV is able to feasibly detect the previous variant. In most environments, it is untenable for AV to impose endpoint usability friction, so adversaries are almost guaranteed to win the race against the AV performing a definition update and a periodic or on-access scan.

This is where EDR comes in. The central premise of EDR is that, in the absence of an immediate detection, you still send sufficient telemetry to enable retrospective detection. For executable file content, file hashes with first-seen content are **perfect** content telemetry. Defenders can simply take a leaf out of the adversary’s playbook and continuously scan all seen binaries and note all detection verdict changes. This is infeasible to bypass *for all time*. In other words, file hashes are at the **base** of Pyramid of Pain for **point in time** detection but are at the **apex** of the Pyramid of Pain for **retrospective** detection. That hash is immutable proof of past compromise.

Fingerprints also simplify collaboration between defenders. This is particularly exemplified by VirusTotal, though other open platforms and **indicator of compromise (IoC) feeds** exist. This allows defenders to operate at machine-speed, sharing detection verdicts, requesting a second (or 60th) opinion, and even deduplicating storage. This **open approach** to security is an AV **force multiplier** which reduces mean time to detection (MTTD) from years and months to weeks and days. AV is stronger together.

The impact of this pain, however, depends on the adversary. Any nation-state programs directed to avoid attribution entirely and trying to dwell for years must absolutely factor this in. Ransomware criminals, however, are far less concerned by attribution by design and are typically not impacted by current AV MTTD. They operate at time scales of days and hours. In other words, it's useful to retrospectively detect nation-state actors and be able to estimate the impact based on how long they have dwelt on your network. It's less useful to detect ransomware when you've already received the ransom note.

Even with EDR in place, the only robust content protection is to simply block (or sandbox) all unknown file content from executing — either with strict application control (default deny) or a reputation-based approximation. However, for many organizations, the risk of business interruption because of such a control may be too high. That said, I would encourage all CISOs to periodically reassess whether current products are yet suitable for their environment. Much of the excessive user friction and maintenance overheads of the early application control implementations have been addressed by the **leading vendor**(s).

So what *did* I bypass?

AV, EPP, EDR, and firewall are all complementary layers of robust endpoint security. As mentioned at the outset, I believe that the overly generic use of the term "EDR bypass" has stunted our understanding of the purpose of each layer and potentially led us to incorrect conclusions. Or maybe the discourse just isn't occurring because too many vendors are relying on security through obscurity.

Either way, I suggest that we be more specific about the defensive layers involved when we discuss bypasses, as this will help us to understand the current gaps in our defenses.

Firewall bypasses

If a packet leaves the host for an unexpected destination, then it's a firewall bypass (or a known gap such as use of legitimate web services). This can typically be demonstrated by showing a successful command and control connection. However, if the DNS resolution or network connection is logged in your SIEM then it's not an EDR bypass.

We don't talk enough about these. Why is the default host firewall so permissive of outbound traffic from operating system processes? Why doesn't the operating system require software to provide a machine-readable manifest of the required network flows?

AV bypasses

If an unexpected binary or script is executed then it's an AV bypass. This is typically demonstrated by showing zero detections on VirusTotal though a screenshot is a more accurate demonstration as the AV engines used by vendors on VirusTotal do not always perfectly match the engine on the endpoint. Further, your AV may also block low reputation files at first sight.

However, if the file hash is logged in your SIEM (and if a copy of the content is available) then it's not an EDR bypass.

EPP bypasses

If an unexpected privileged operating system operation is performed, then it's an EPP bypass (or possibly a currently indefensible operating system component). This is typically demonstrated by showing a screenshot of the successful operation such as dumped credentials.

However, if any of the API calls made were logged in your SIEM then it's not an EDR bypass.

EDR bypasses

As indicated above, the first requirement for an EDR bypass is zero relevant logs in your SIEM. For example, I suspect that some of the previous research around insertion of firewall rules for EDR processes likely met this bar. However, in practice, zero logs alone may not be sufficient as the absence of logs from a live monitored endpoint could also be used as a detection.

Let's discuss some contemporary "EDR" bypasses and how I would classify them under my taxonomy.

Case study: User mode hook avoidance

The first class of "universal EDR bypass" is user mode hook avoidance — whether by unhooking, direct syscalls, single stepping, or otherwise.

However, if you manage to do this with a precondition of arbitrary user mode execution then you haven't bypassed a security boundary… you were already within the perimeter.

You could equivalently claim that you've bypassed ASLR, CFG, DEP, and MFA also.

In particular, you have not bypassed the kernel callbacks used by modern EPP and EDR nor have you bypassed Threat-Intelligence ETW telemetry — though these are not available for all syscalls. (Additionally, the latter only arrived in Windows 10 1703, so some vendors may not have prioritized implementation yet if they had previous and imperfect coverage via user-mode hooking).

In other words, if you reported this to an endpoint security product's bug bounty program, you'd be justifiably informed that it was "not a security boundary." That said, if the vendor makes functionality claims that you prove inaccurate then it is noteworthy research.

Case study: Bring your own vulnerable driver

The second class of "universal EDR bypasses" is the **Bring Your Own Vulnerable Driver** approach exemplified by **CheekyBlinder**, **EDRSandblast**, and their ilk. At its heart, this is actually a pair of AV bypasses — the tool and the known vulnerable driver.

The first point-in-time AV bypass is understandable for the reasons outlined earlier.

However, the second AV bypass is unacceptable in 2023. The adversary cannot modify the known vulnerable driver's code without invalidating its signature and causing Windows to refuse to load it. Sure, there will always be new vulnerable drivers found, but vendors should be quick to block and share details. Of course, many AV vendors just **trust Microsoft to block** these drivers even though **Microsoft is yet to add its own vulnerable drivers to its block list**.

There is also an EPP bypass here too. Code not from trustworthy software vendors should not have been able to load a driver and communicate with it.

Finally, there is an EDR bypass here **but for future actions only**. All of the activity leading up to the point should already have queued for logging. Hopefully the EDR aggressively flushes its queues before loading new drivers. It is possible in some other scenarios, however, that the relevant logs have not yet left the endpoint. Most endpoint security is designed (or configured by the customer) to fail open because security cannot unduly impede functionality.

Conclusion

This blog has helped explain the current endpoint security landscape, some of the contextual drivers for how the ecosystem evolved, and some of the current constraints imposed by the operating system that vendors must work within. Each functional layer of the endpoint security plays an important role in providing defense-in-depth. Understanding the purpose of each layer and its effectiveness against various classes of threats will help you choose appropriate products.

Learn more about endpoint security at Elastic.

FOLLOW US

ABOUT US

About Elastic
Leadership
DE&I
Blog
Newsroom

JOIN US

Careers
Career portal
How we hire

PARTNERS

Find a partner
Partner login
Request access
Become a partner

TRUST & SECURITY

Trust center
EthicsPoint portal
ECCN report
Ethics email

INVESTOR RELATIONS

Investor resources
Governance
Financials
Stock

EXCELLENCE AWARDS

Previous winners
ElasticON Tour
Become a sponsor
All events

Trademarks Terms of Use Privacy Sitemap