Stop, get some help. Design your implants properly

SteelCon 2026

TRUSTEDSEC

whoami

- Brandon McGrath
- Red Team @ TrustedSec
- Fourth time speaker at SteelCon
- Blog at mez0.cc and pre.empt.blog

Talk Objective

- Look at the state of malware and telemetry detections
- Plan out how we can work within the boundaries
- Discuss detection points and weaponise gaps
- Generally, take the guess work out of payloads

Talk Objective

Core Techniques & Why They Were Chosen

1. Module Stomping (Sacrificial DLL Technique)

- **Why?** Most memory scanners focus on newly allocated private memory regions. By stomping a legitimate syster DLL, we operate inside an image-backed trusted memory space.
- **Strength:** Very strong against memory scanners and heuristic detection.

2. IAT Camouflage + Library Proxy Loading

- **Why?** Many EDRs monitor suspicious Import Address Tables and API resolutions.
- **Strength:** Provides excellent protection against static and behavioral analysis.

3. Proxy Execute API

- **Why?** Direct API calls are heavily monitored. Using proxy functions helps evade behavioral detection.
- **Strength:** Reduces suspicious API call patterns.

4. Advanced Anti-Analysis

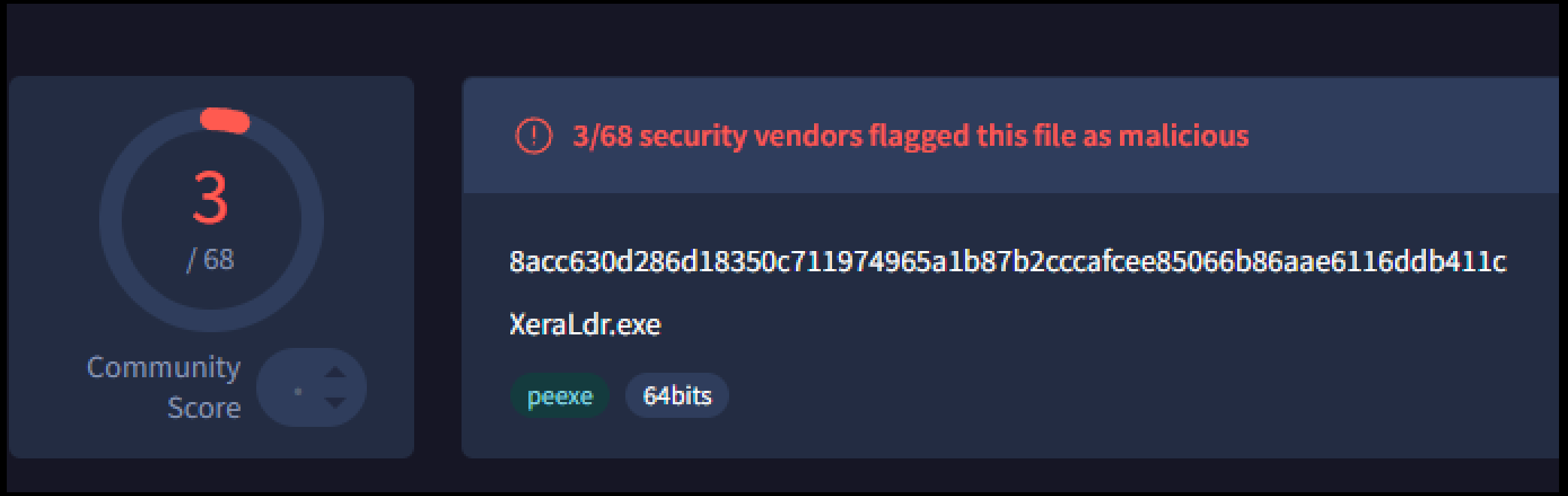
- **Why?** Prevents execution in virtual machines, debuggers, and sandboxes.
- **Strength:** Protects the loader during initial execution phase.
- **Reference:** Sandbox-Detection-Techniques

5. ChaCha20 Encryption + BaseN Encoding

- **Why?** Strong encryption makes static signature detection significantly harder.
- **Strength:** Payload remains encrypted until runtime.
- **Reference:** LibTomCrypt

6. Resource Section Payload Delivery

Talk Objective



State of Payload OpSec 2026

State of Payload OpSec 2026

- The first look at ETW breakage
- SysCalls: SysWhispers 1, 2, 3, and 4 – then HellsGate variants
- AMSI and CLM were extremely brittle
- **HEAVY** obfuscation worked well

https://www.mdsec.co.uk/2020/06/detecting-and-advancing-in-memory-net-tradecraft/

State of Payload OpSec 2026

- ETW and AMSI patching is an IOC
- HWBP is also an IOC
- SysCalls is now an IOC
- ML is WAY better
- Quieter open-source research
- Callstacks…

https://mez0.cc/posts/evaluating-implants-with-ember/

State of Payload OpSec 2026

- https://www.cobaltstrike.com/blog/behind-the-mask-spoofing-call-stacks-dynamically-with-timers
- https://0xdarkvortex.dev/hiding-in-plainsight/
- https://www.elastic.co/security-labs/call-stacks-no-more-free-passes-for-malware
- https://klezvirus.github.io/posts/Moonwalk-plus-plus/
- https://klezvirus.github.io/posts/Callback-Hell/
- https://klezvirus.github.io/posts/Byoud/

A word on static

Static Detections

ML

YARA

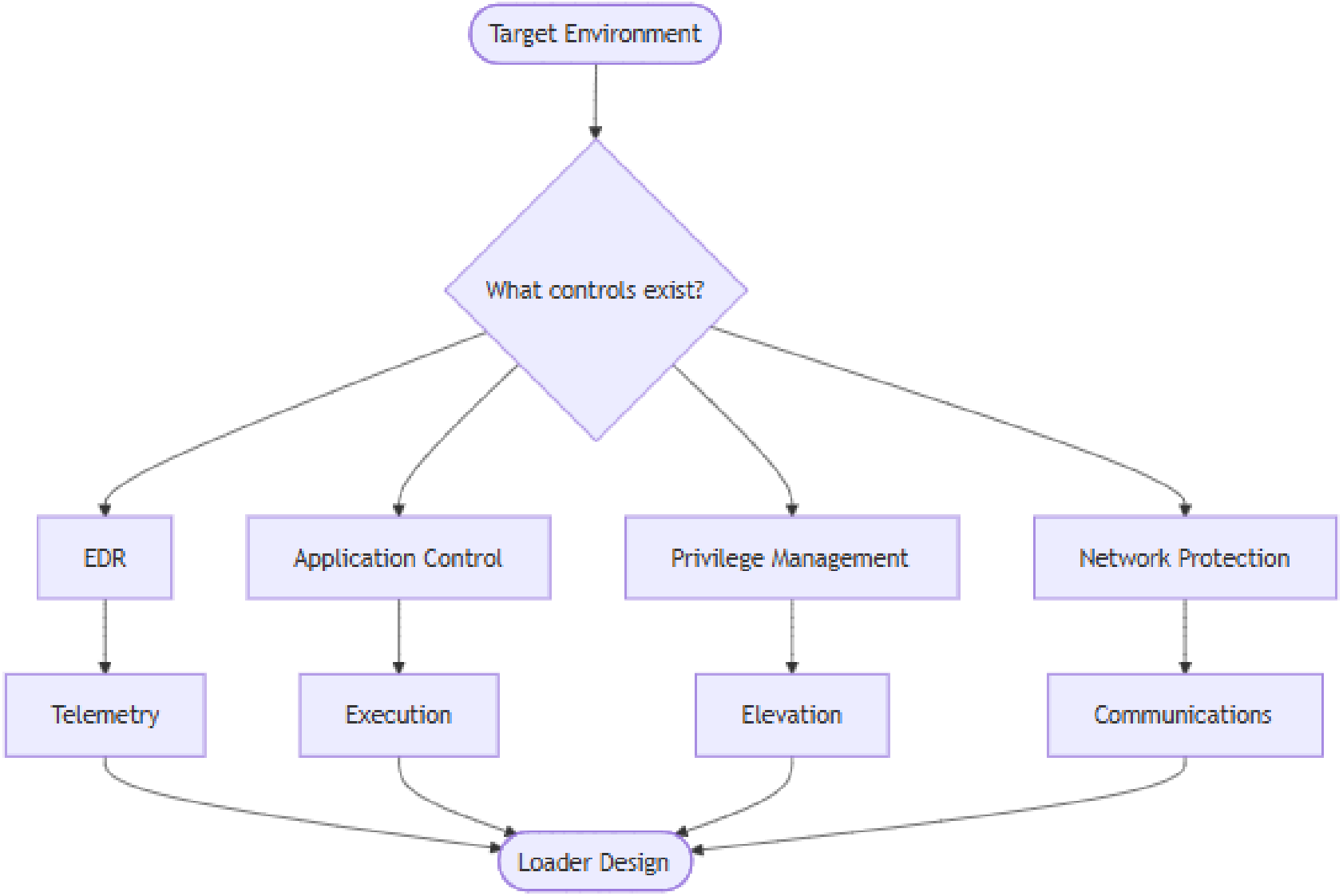
CAPA

Security Products that *matter*

Security Products

Control	Purpose	Example
EDR	**Detect and Respond to malicious behaviour**	**MDE & CrowdStrike**
Application Control	**Allow only approved software**	**ThreatLocker**
Privilege Management	Control who has elevated access	Beyond Trust
DNS / Web Protection	Prevent access to bad stuff	Cisco Umbrella

Security Products



Security Products

None of this has anything to do with endpoint hardening

EDR Mechanisms that *matter*

EDR

"A system of telemetry sources on a host with the ability to detect and respond to malicious activity"


Kernel Callbacks
WPF
MOTW
DLL
DLL Hooks
Notifications
AMSI
Callstacks
ETW
Minifilters
ML


Process, Thread, Registry, Network, and Memory Telemetry
Event Logs
Script/Interpreter Telemetry
Application Logs
Threat Intel
Sandbox

EDR

- INSANE amount of telemetry sources
- Finite of high-fidelity, rest can be considered enrichment
- Being aware of them is important to understand triage

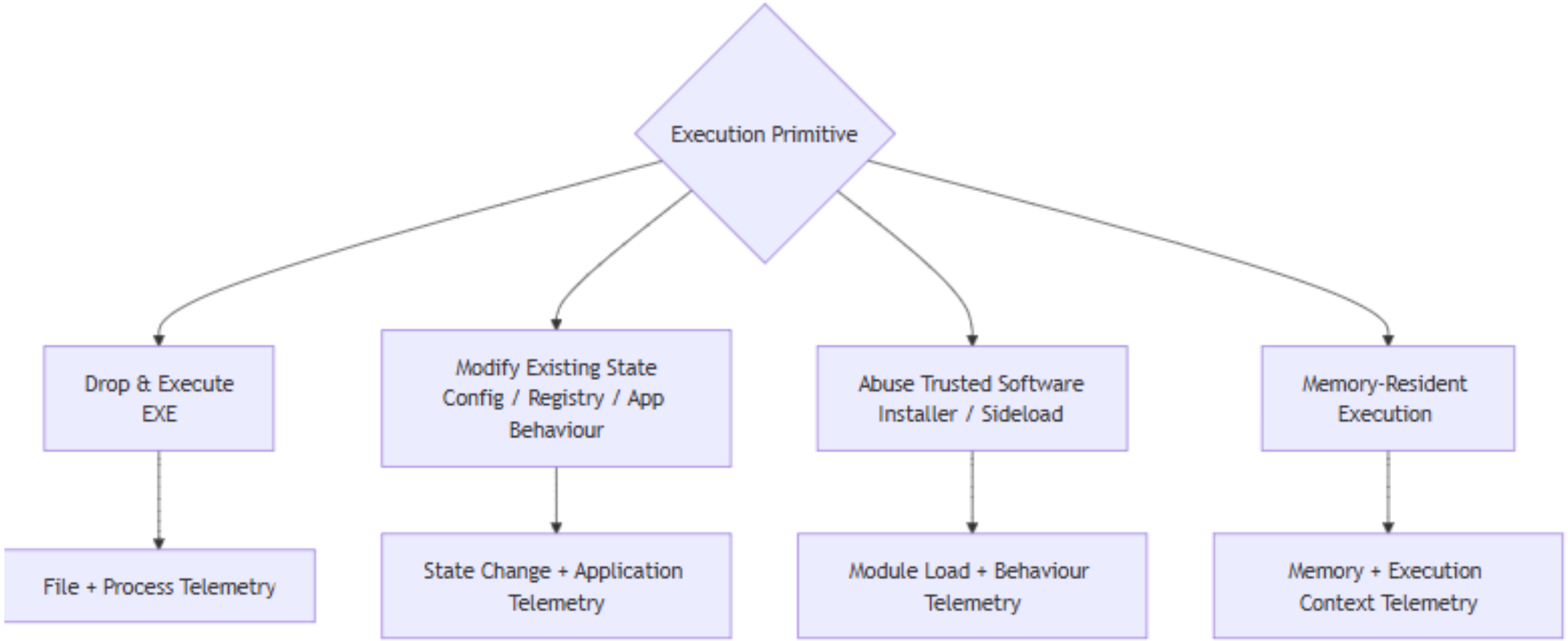
- https://pre.empt.blog/posts/maelstrom-6/
- https://pre.empt.blog/posts/maelstrom-5/

Designing a Loader: Primitives

The Scenario

We are on an assumed breach and have access over RDP to a VDI. To bring our tooling into the environment, we want to deploy our C2.

Payload Type



NOTE

- Isn’t in public rulesets like Elastic
- Has no prior research/blogs
- The primitive goes through benign channels
  - Signed binary
  - Doesn’t hit the typical loading logic
  - The execution tree is janky

The Usual Suspect

The Usual Suspect

```
int main() {
    # 1. shellcode either remote pulled, loaded from disk, or directly from payload
    LPVOID addr = VirtualAlloc(...);

    # 2. copy next stage to buffer
    memcpy(...)

    # 3. Make the memory location executable
    VirtualProtect(...);

    # 4. Create the thread pointing to the buffer
    HANDLE thread = CreateThread(...);

    # 5. Wait for the thread forever so it can run
    WaitForSingleObject(...);
}
```


The Usual Suspect

```
int main()
    # 1. shellcode either remote pulled, loaded from disk, or directly from payload
    LPVOID addr = VirtualAlloc(...);

    # 2. copy next stage to buffer
    memcpy(...)

    # 3. Make the memory location executable
    VirtualProtect(...);

    # 4. Create the thread pointing to the buffer
    HANDLE thread = CreateThread(...);

    # 5. Wait for the thread forever so it can run
    WaitForSingleObject(...);
}
```

[1]

[2]

[3]

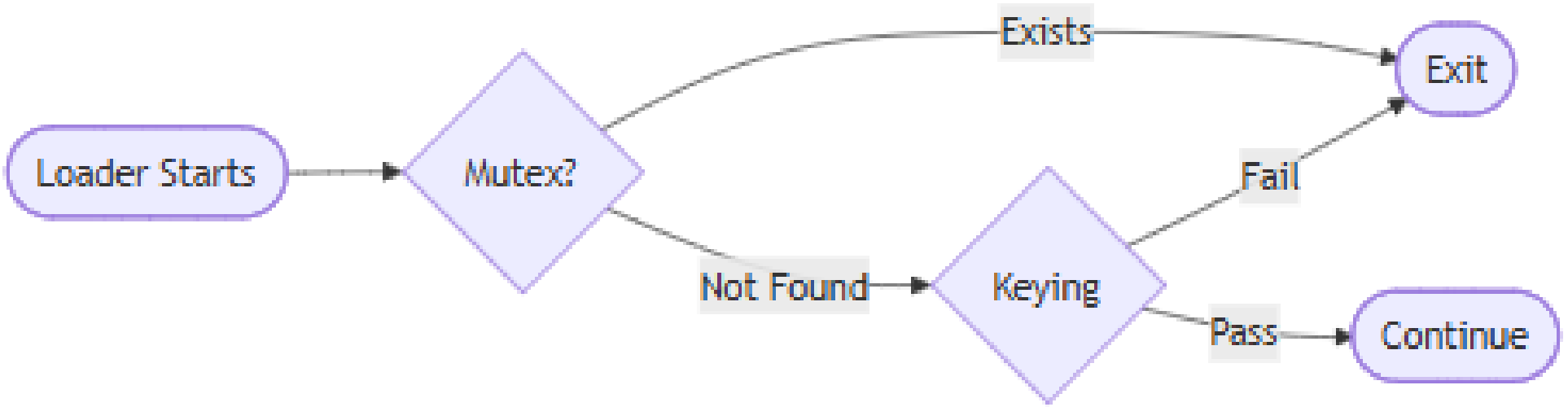
[4]

The Usual Suspect

- https://trustedsec.com/blog/windows-processes-nefarious-anomalies-and-you-memory-regions
- https://trustedsec.com/blog/windows-processes-nefarious-anomalies-and-you-threads

Prepping the Loader

Target Validation



Prepping the Runtime

Component	Options	Why it matters
API Resolution	Imports / GetProcAddress / Export Parsing	Static footprint & runtime visibility
Module Loading	LoadLibrary / Proxy / Existing Modules	DLL notifications & call stack quality
Imports	Static / Dynamic	Detection surface vs complexity

Prepping the Runtime

```
int __clrcall WinMain(HINSTANCE hInstance,
                      HINSTANCE hPrevInstance,
                      LPSTR lpCmdLine,
                      int nShowCmd
                      )
{
    if (check_keying() == FALSE) {
        return EXIT_FAILURE;
    }

    if (is_already_running() == TRUE) {
        return EXIT_FAILURE;
    }

    // 1. shellcode either remote pulled, loaded from disk, or directly from payload
    LPVOID addr = VirtualAlloc(...);
```

Hustling the Telemetry

Hustling the Telemetry: Userland

Telemetry	EDR Visibility	Consideration
Userland Hooks	API behaviour, function usage	Instrumentation changes what the agent can observe
DLL Notifications	Module loads and process state	Loading strategy affects visibility
AMSI	Script/content inspection	Execution path determines whether content is inspected
ETW	User/application telemetry	Actions generate observable events

Hustling the Telemetry: Runtime / Memory

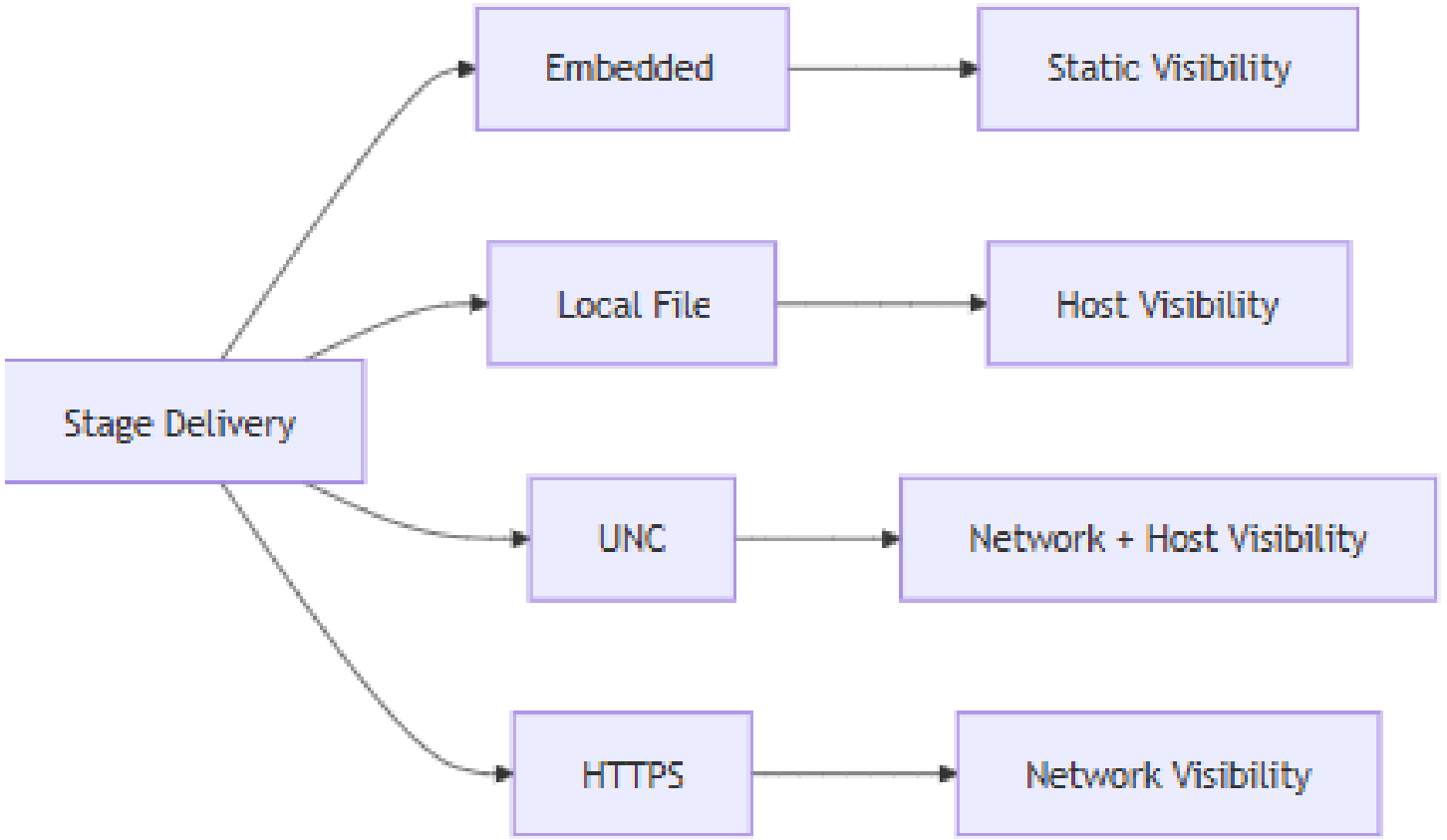
Telemetry	EDR Visibility	Consideration
Memory Scanners	Executable regions, injected content, artifacts	Lifetime and representation of payload matters
Sleep Masking	Dormant payload visibility	Payload state changes over time
Call Stack Analysis	Execution provenance	Where execution originates matters

Hustling the Telemetry: Kernel / Platform

Telemetry	EDR Visibility	Consideration
Kernel Callbacks	Process, thread, image, object activity	OS-level ground truth
Minifilters	File system activity	File operations remain observable
ETW-TI / Kernel ETW	Security-focused behavioural telemetry	Higher fidelity event sources
WFP	Network activity	Communications create external telemetry

Stage Delivery

Stage Delivery



Stage Delivery

```
int __clrcall WinMain(HINSTANCE hInstance,
                      HINSTANCE hPrevInstance,
                      LPSTR lpCmdLine,
                      int nShowCmd
                      )
{
    if (check_keying() == FALSE) {
        return EXIT_FAILURE;
    }

    if (is_already_running() == TRUE) {
        return EXIT_FAILURE;
    }

    LPVOID buffer = get_next_stage();

    if (buffer == nullptr) {
        return EXIT_FAILURE;
    }
```

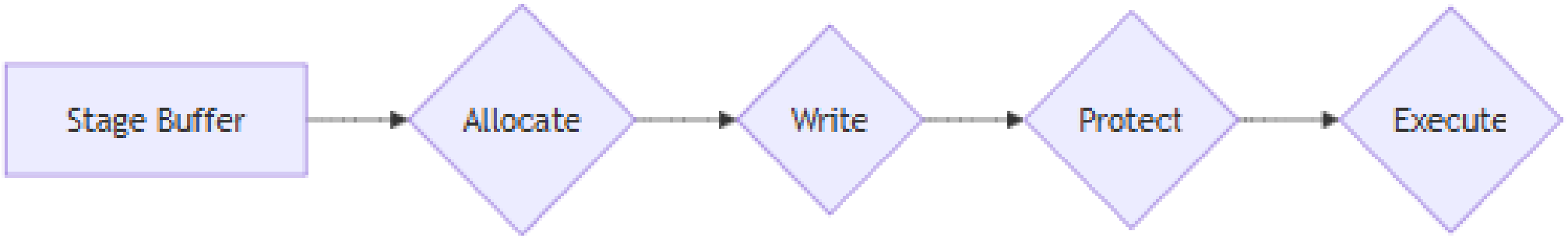
Stage Protection

Stage Protection

Technique	Benefit	Trade-off / Telemetry
Encryption	Protects the stage contents until runtime	High entropy, cryptographic routines, decryption logic
Encoding	Changes representation without cryptographic overhead	Easily reversible, may still produce identifiable patterns
Obfuscation	Makes analysis and static identification harder	Increased complexity, unusual code patterns
Masking / Hiding	Reduces obvious artifact visibility	Requires transformation and recovery logic

Stage Loading

Stage Loading

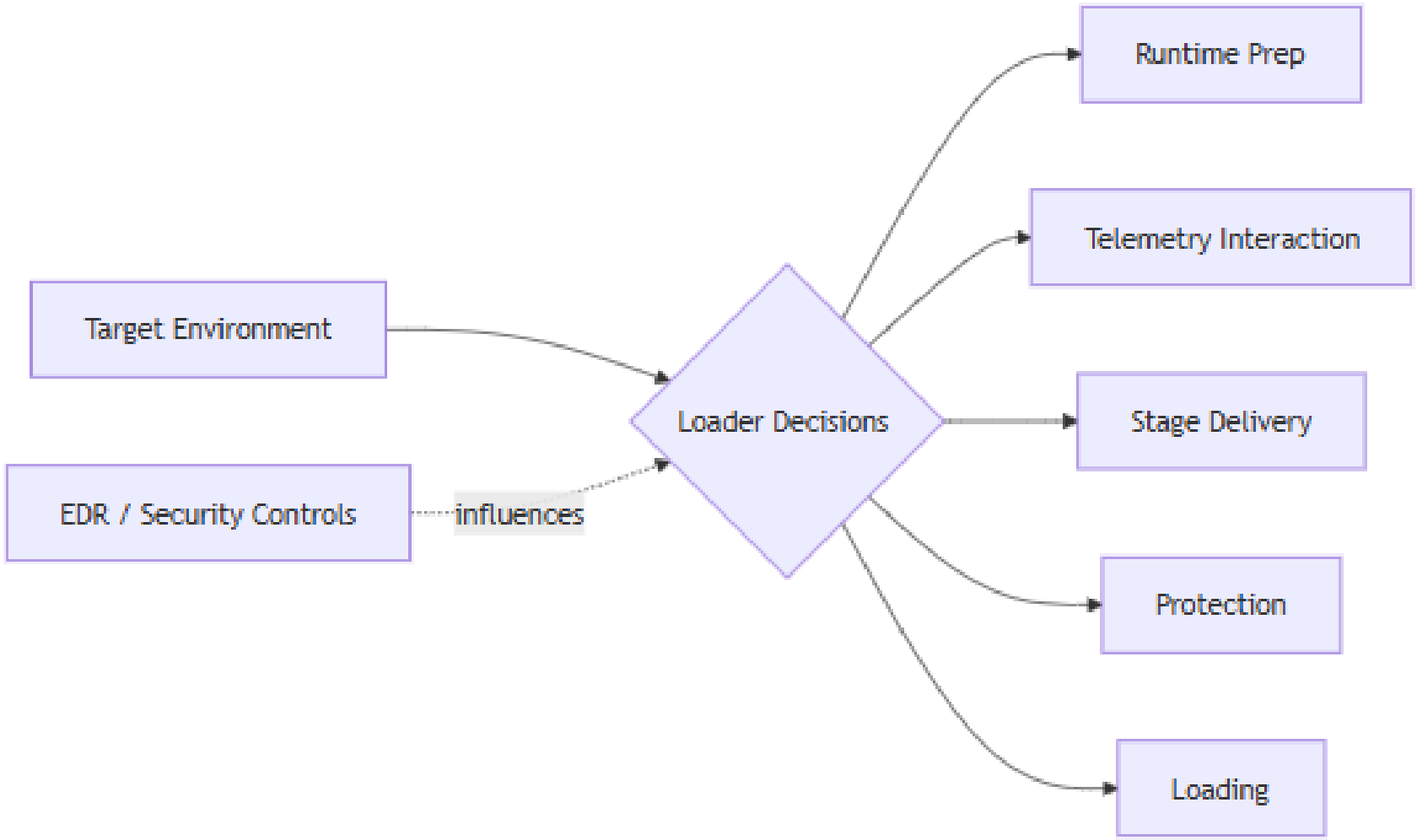


Stage Loading

Decision	Example primitives	Consideration
Allocate	VirtualAlloc, section mapping, existing region	Allocation telemetry, memory layout
Write	memcpy, mapped sections, shared views	Memory-write visibility
Protect	VirtualProtect, native APIs, pre-protected mappings	RW→RX transitions, page protections
Execute	CreateThread, callbacks, APCs, threadpool, DllMain, existing thread	Thread creation, execution context, call stacks

The Problem

The Problem


Runtime Prep
Telemetry Interaction
Target Environment
Loader Decisions
Stage Delivery
EDR / Security Controls
influences
Protection
Loading

The Problem

Monolithic Loader	Modular Framework
Fixed execution path	Replaceable components
One environment assumption	Adaptable to target telemetry
Rewrite after failure	Swap individual decisions
Hard to maintain	Evolves over time

Telemetry Budget

Detection Points

Telemetry Budget

- What are you willing to allow the EDR to see?

Reality	Implication
You cannot remove all telemetry	Choose where it is generated
Every design decision creates artifacts	Trade one visibility source for another
Reducing one signal may amplify another	Optimise for the environment, not invisibility
EDRs are built from accumulated observations	Assume detection logic evolves

Conclusion

Conclusion

- Avoid monolithic loaders
- Invest in versatile and modular frameworks
- Stay off Twitter payloads
- Track your generated payloads internally

REFERENCES AND ADDITIONAL READING

- https://0xdarkvortex.dev/hiding-in-plainsight/
- https://klezvirus.github.io/posts/Byoud/
- https://klezvirus.github.io/posts/Callback-Hell/
- https://klezvirus.github.io/posts/Moonwalk-plus-plus/
- https://mez0.cc/posts/evaluating-implants-with-ember/
- https://pre.empt.blog/posts/maelstrom-5/
- https://pre.empt.blog/posts/maelstrom-6/
- https://trustedsec.com/blog/execution-guardrails-no-one-likes-unintentional-exposure
- https://trustedsec.com/blog/windows-processes-nefarious-anomalies-and-you-memory-regions
- https://trustedsec.com/blog/windows-processes-nefarious-anomalies-and-you-threads
- https://www.cobaltstrike.com/blog/behind-the-mask-spoofing-call-stacks-dynamically-with-timers
- https://www.elastic.co/security-labs/call-stacks-no-more-free-passes-for-malware
- https://www.mdsec.co.uk/2020/06/detecting-and-advancing-in-memory-net-tradecraft/

TRUSTEDSEC

THANK YOU!