

UNABHÄNGIGER SICHERHEITSBERICHT

Blackbox-Reverse-Engineering-Analyse

einer mit **CitadelCore** geschützten Demo-Anwendung`licensing_demo_protected.exe`

● Methodik: statisch & dynamisch ● 27.09.2026 ● KI-gestützt

§ Über diesen Bericht

Dieser Bericht dokumentiert eine unabhängige Reverse-Engineering-Analyse eines **Demo-Programms**, das mit dem Härtungswerkzeug **CitadelCore** geschützt wurde. Ziel war nicht, das Programm zu knacken, sondern die Schutzwirkung von CitadelCore messbar zu belegen und Ansatzpunkte für die Weiterentwicklung zu finden.

① WAS HEISST BLACKBOX-ANALYSE?

Der Prüfer hatte **keinen Zugriff auf den Quellcode**. Als einzige Informationsquelle diente die fertige `.exe`-Datei und ihr Verhalten beim Ausführen — genau die Ausgangslage eines echten Angreifers. Alles, was in diesem Bericht steht, wurde aus dem Maschinencode und aus Messungen am laufenden Prozess erschlossen, nicht aus dem Programmtext. Zwar lagen der ungeschützte Referenzbau und die Quelldateien im selben Verzeichnis — sie wurden **bewusst nicht geöffnet**, um die Blackbox-Disziplin strikt zu wahren.

! WICHTIGE EINORDNUNG: DEMO-PARAMETER ≠ CITADELCORE-MECHANIK

Einige Punkte am Schluss betreffen **das Demo-Programm selbst**, nicht den Schutz von CitadelCore. Der auffälligste ist die **Schlüssellänge**: Der Beispiel-Serial `SERIAL-XXXX-XXXX-XXXX` hat nur 48 Bit Eingaberaum. Das ist eine **Wahl der Demo**, keine Eigenschaft von CitadelCore — jede echte Anwendung wählt ihr Schlüsselformat frei. Solche Punkte sind im Bericht ausdrücklich als **Demo-Parameter** gekennzeichnet.

KI-Unterstützung. Die Analyse wurde durch künstliche Intelligenz unterstützt — unter Einsatz des zum Analysezeitpunkt aktuellsten Modells von Anthropic. Sämtliche Messwerte sind reproduzierbar; Werkzeuge waren IDA Pro (Disassembler, Hex-Rays), ein eigener Prozess-Sampler und ein Patch-Harness. Jede Aussage ist als **S statisch** oder **D dynamisch** gewonnen markiert.

§ Kennzahlen auf einen Blick

0,41%der Speicher-Stichproben
zeigten Klartext**99,6%**der Laufzeit liegt der Kern
verschlüsselt vor**2/188**Zeichenketten im Klartext —
beide unkritisch**0**nutzbare Angriffskanäle im
Verhalten

60/60

Läufe identisch, kein Absturz

100%Erkennung von
Chiffre-Manipulation**6**unabhängige
Schlüsselbeiträge pro
Entschlüsselung**0**Krypto-Importe /
Packer-Signaturen

1 Funktionsprüfung

◆ KERNAUSSAGE

Bei falscher Eingabe ist die Ausgabe unlesbar — und das ist bauartbedingt korrekt. Der Exit-Code stimmt (1), der Ablauf ist sauber, kein Absturz. Die Meldungstexte liegen verschlüsselt vor, und der Schlüssel wird **aus dem eingegebenen Serial abgeleitet**. Ohne den richtigen Serial existiert der Klartext nirgends — weder in der Datei noch im Speicher. Es gibt keine Ja/Nein-Weiche zum Umlenken, nur eine Entschlüsselung, die gelingt oder nicht. Das Programm sagt es selbst: „Der Serial ist zugleich das Freischaltgeheimnis.“ S

1.2 Aufrufmatrix D

Aufruf	Exit	Nutzlast	Beobachtung
ohne Argument	2	108 B Klartext	vollständige, lesbare Gebrauchsanweisung
SERIAL - 0000 - 0000 - 0000	1	19 B	unlesbar, deterministisch
SERIAL - AAAA - BBBB - CCCC	1	21 B	unlesbar, deterministisch
leer / ABC / Kleinschreibung	1	~21 B	unlesbar, deterministisch
64 ... 8 000 Zeichen	1	21 B	kein Absturz, keine Verlangsamung
2 Argumente	1	wie 1 Arg.	Zweitargument ignoriert

Die schwankende Nutzlastlänge (2...21 B) ist kein Zufall: `fputs` bricht am ersten Nullbyte des entschlüsselten Puffers ab, dessen Lage vom Serial abhängt. S D

1.3 Stabilität

- **60 aufeinanderfolgende Läufe**, gleiche Eingabe: 60× Exit 1, exakt **ein** Ausgabemuster, kein Absturz. D
- Laufzeit: Median **12,8 ms** (min 12,0 / max 22,2 / σ 1,5 ms) — Streuung erklärt durch den eingebauten Zeitjitter. D
- Über 100 ungepatchte Läufe: **kein einziger** unerwarteter Abbruch. D

1.4 Verhalten unter Debugger — bewusst anders D

Ergebnis (12 Läufe)	Anzahl
Ausgabe erzeugt	0 von 12
Prozess blieb > 8 s stehen	4
Abbruch mit wechselndem Exit-Code	8

✓ GUT GEWÄHLTES REAKTIONSMUSTER

Der Schutz meldet nicht „Debugger erkannt“, sondern liefert eine wechselnde, unauffällige Fehlfunktion. Ein Beobachter erhält **kein verwertbares Signal**, an welcher Stelle er entdeckt wurde — deutlich stärker als ein sichtbarer Abbruch.

2 Aufbau des Schutzes

2.1 Sektionen s

Sektion	Größe	Rechte	Entropie	
.text	~47 KB	rx	6,464	regulärer Code
.crt2	4 047 B	rx	7,934	verschlüsselter Kern
.rdata	~27 KB	r	7,358	verschlüsselte Datenblöcke

2.2 – 2.3 Verschlüsselung & Schlüsselableitung s

Chiffre: ChaCha20 (RFC-8439-Layout). Die Konstante `expand 32-byte k` liegt **XOR-maskiert** in der Datei — eine Signatursuche liefert **0 Treffer**, ebenso für `expand 16-byte k`. Gängige „YARA-Regel für ChaCha20“ greift also ins Leere.

① WARUM DER SCHLÜSSEL NIRGENDS STEHT

Der 256-Bit-Schlüssel jeder Entschlüsselung wird aus **sechs unabhängigen Beiträgen** zusammengesetzt: zwei tabellenindirekte Anteile (byteweise, danach XOR-verknüpft), der aus dem **Serial** abgeleitete Anteil, ein Hash über das Integritätsregister, der Manipulations-Akkumulator und ein Hash über den **Chiffretext selbst** — plus eine eigene Nonce je Fundstelle. Der fertige Schlüssel existiert an **keiner Stelle** der Datei als zusammenhängende Konstante und wird nach Gebrauch überschrieben. Punkt 5 (Hash über den Chiffretext) bedeutet: **jeder Block ist implizit authentifiziert** — eine Änderung am Chifftrat macht den Schlüssel falsch.

2.4 Kontrollfluss & Hygiene s

- **Control-Flow-Flattening** in allen Anwendungsfunktionen (Zustandsautomat mit Magic-Konstanten statt normaler Verzweigungen).
- **40** Entschlüsselungs-Stellen stehen **39** Lösch-Stellen gegenüber — praktisch jeder Klartext wird sofort nach Gebrauch überschrieben.
- Von 188 Zeichenketten sind nur **2** im Klartext (Banner, Gebrauchsanweisung) — beide unkritisch. Alle Lizenzmeldungen sind verschlüsselt.

2.5 Lizenzlogik läuft in einer eigenen Maschine s

Die Prüfung läuft **nicht als nativer Code**, sondern in einem **Bytecode-Interpreter** (35 Opcodes, 12-Byte-Instruktionen, 1 024 Register). Das Lizenzprädikat sind **66 Instruktionen**, verschlüsselt abgelegt und mit dem Serial-Schlüssel gebunden.

◆ DER ENTSCHEIDENDE ENTWURF

Es gibt keinen Vergleich gegen einen gespeicherten Sollwert. Das Prüf-Programm selbst ist mit dem Serial verschlüsselt: Bei falschem Serial interpretiert die Maschine **Zufallsdaten**. Damit fehlt dem klassischen Cracking-Repertoire die Grundlage — es gibt nichts zu vergleichen und nichts umzulenken.

2.6 – 2.8 Manipulationserkennung, Anti-Debug, Laufzeitmaßnahmen s

- **Zwei unabhängige Prüfschienen:** ein festes, gegenseitig verschränktes Netz (8 Prüfkanten über 5 Funktionen, keine prüft sich selbst) und eine **rotierende Stichprobe** über 9 Regionen — **71 528 Byte** Gesamtabdeckung. Aus einem einzelnen Lauf ist nicht ableitbar, was gerade überwacht wird.
- **Reaktion (14 Stellen):** statt sichtbarem Abbruch wird das **Schlüsselmaterial vergiftet**; der eigentliche Abbruch erfolgt **verzögert und randomisiert** (zufällige Schwelle, zufälliger Exit-Code) und nie an der Stelle der Entdeckung.
- **Sechs** unabhängige Debugger-Indikatoren (PEB, NtGlobalFlag, Hardware-Breakpoint-Register, `NtQueryInformationProcess ...`), alle **dynamisch aufgelöst** — keiner steht in der Importtabelle.
- **Zeitjitter** und ein **wiederverschlüsselnder Hintergrundthread**, der ausgeführte Bereiche wieder sperrt (`PAGE_NOACCESS`).

3 Angriff — Ergebnisse

3a Statisch, ohne Ausführung — teilweise

Sichtbar	Verschlussen geblieben
PE-Struktur, Entropieprofil, Importtabelle (0 Krypto-Importe)	Inhalt von <code>.crt2</code>
ChaCha20 — erst nach Demaskierung erkennbar	alle Lizenzmeldungen (verschlüsselt)
Struktur des Prüfnetzes, Bytecode-Maschine	das Prüf-Programm, die Schlüssel

✓ GENAU WIE VORGESEHEN

Die **Maschinerie** ist statisch rekonstruierbar — das ist bei Decrypt-to-Execute beabsichtigt: Die Sicherheit hängt am **Schlüssel**, nicht an der Geheimhaltung des Verfahrens (Kerckhoffs-Prinzip). Der **Inhalt** — Lizenzregel, Meldungen, Schlüssel — blieb statisch vollständig verschlossen. Keine Packer-Signatur, kein auffälliger Sektionsname außer `.crt2`, das wie ein zweiter CRT-Abschnitt aussieht.

3b Verhalten — kein Kanal gefunden

Rückgabewerte, Ausgabelänge und Zeitverhalten liefern **keinen** Kanal, der falsche von „fast richtigen“ Serials trennt. Der eingebaute Jitter (bis ± 10 ms) überdeckt jeden denkbaren Rechenzeitunterschied ($< 0,1$ ms) um Größenordnungen — **kein Zeitseitenkanal messbar**, und das ohne spürbare Verlangsamung (Median 12,8 ms).

3c Speicherabzug — teilweise D

Messgröße	Wert
Stichproben mit Klartext	119 von 28 854 (0,41 %)
Fensterdauer (Median / Max)	0,089 ms / 0,566 ms
Klartextfenster je Lauf	0,27 ms — in der letzten Millisekunde vor Prozessende
<code>.bss</code> -Bytecode erfasst	0 Treffer bei 11 094 Stichproben

✓ WAS STANDGEHALTEN HAT

Kein separater RWX-Puffer, keine private Kopie — die Entschlüsselung erfolgt in place. Der klassische „Dump der entpackten Kopie“ greift ins Leere. Im Ruhezustand steht der Kern auf `PAGE_NOACCESS` (Zugriff verweigert), die Wiederverschlüsselung erzeugt ein **anderes Chifftrat als auf der Platte**. Ein vollständiger Kern-Abzug gelang in **1 von 10** Läufen — und selbst dann war nur die *Struktur* sichtbar. Das eigentliche Geheimnis, die Lizenzregel, lag **zu keinem Zeitpunkt** im Speicher.

3d Manipulation — Erkennung stark / Umgehung ohne Ziel

Test	Änderung	Ergebnis
B	1 Byte im <code>.crt2</code> -Chifftrat	Sofort & total erkannt 12/12 Abbruch (<code>STATUS_PRIVILEGED_INSTRUCTION</code>). Ein Byte zerstört die Entschlüsselung aller 4 047 Byte.

C	1 Byte in verschlüsseltem .rdata	Still erkannt 12/12, durchgehend anderes Ausgabechifftrat.
E/G	Prüf-/Reaktions-Routine abgeschaltet (3–4 B)	Nicht erkannt — verschafft aber keinen Klartext.
H	Erfolgsweiche invertiert + Prüfung aus (5 B)	Wirkungslos Programm läuft stabil in den Erfolgspfad — Ausgabe bleibt unlesbar.

◆ DER BEWEIS: „PATCH THE JUMP“ HAT HIER KEIN ZIEL

Test **H** ist der stärkste Befund des Berichts. Fünf gezielt geänderte Bytes bringen das Programm **stabil und abbruchfrei in den Erfolgspfad** — und trotzdem bleibt die Ausgabe unlesbar. Der Grund: Der Erfolgstext existiert **nicht als Datum**, das man freischalten könnte. Er entsteht erst durch eine Entschlüsselung mit dem **richtigen Serial**. Das Umlenken der Verzweigung ändert nur, *welcher* Chiffretext mit dem *falschen* Schlüssel behandelt wird. Der klassische Crack-Angriff läuft strukturell ins Leere.

4 Was nicht gelungen ist — der Schutz hielt

In der Blackbox-Analyse ist die Liste der **gescheiterten** Angriffe die eigentliche Leistungsbilanz des Schutzes. Sie ist lang.

Angriffsversuch	Gescheitert an
Lizenzregel statisch lesen	Prädikat existiert nur als serial-verschlüsselter Bytecode. Ohne Serial kein Klartext.
Bytecode aus dem Speicher greifen	Entschlüsselung, Nutzung, Löschung zu dicht — 0 Treffer bei 11 094 Stichproben.
Schlüsselmateriale finden	Existiert nie zusammenhängend; sechs Beiträge, Löschung nach Gebrauch.
Zeitseitenkanal	Jitter überdeckt jedes Nutzsignal; kein datenabhängiges Verzweigen messbar.
Erfolgsmeldung per Patch freischalten	Kein Ziel vorhanden (Test H). Keine Implementierungsfrage, sondern die Konstruktion.
Serial rekonstruieren	Kein Sollwert im Programm. Der Serial ist der Schlüssel.

① TRANSPARENZ: EIGENE FEHLER DES PRÜFERS

Der Prüfer dokumentiert offen zwei eigene Irrtümer, die **nicht** dem Schutz zuzuschreiben sind: Zunächst wurde eine feste Ladeadresse angenommen (das Ziel ist ASLR-fähig) — 5 Messreihen lieferten fälschlich 0 lesbare Bytes, bis die echte Modulbasis aufgelöst war. Und eine erste, fehlerhafte Debug-Schleife hing. Beides wurde korrigiert und offengelegt. Diese Ehrlichkeit trennt sauber zwischen „Schutz hat gehalten“ und „Messfehler“.

5 Bewertung

Kriterium	Note	Kurzbegründung
Schutz gegen statische Analyse	1	Geschäftslogik ohne Ausführung nicht rekonstruierbar; 0 Signaturreffer.
Schutz gegen Speicherabzug	2	Klartext in 0,41 % der Stichproben, Fenster 0,089 ms; das Geheimnis lag nie im Speicher. Prinzipielle Decrypt-to-Execute-Grenze, hier eng gedrückt.
Schutz gegen Manipulation	1–2	Chiffre-Änderungen 100 % erkannt; Umgehung der Prüfmechanik verschafft keinen Klartext.
Geheimhaltung des Schlüsselmaterials	1	Kein Schlüssel existiert zusammenhängend; Wurzelgeheimnis beim Anwender.
Funktionale Zuverlässigkeit	1	60/60 identisch, 0 Abstürze, Median 12,8 ms.
Unauffälligkeit	1–2	Keine Packer-Signatur, 0 Krypto-Importe; Entropieprofil für einen Analysten sichtbar.

✓ GESAMTBEWERTUNG DES PRÜFERS

„Eine konsequent und sauber durchgeführte Umsetzung des Decrypt-to-Execute-Prinzips. Die zentrale Entwurfsentscheidung — den Serial nicht zu *prüfen*, sondern als *Schlüssel* zu verwenden — ist bis in die letzte Meldung durchgezogen und entzieht dem klassischen Reverse-Engineering-Repertoire seine Grundlage. Keine Befunde der Kategorie *Fehler*: Es wurde nichts gefunden, das nicht so funktioniert, wie es beabsichtigt ist.“

5.1 Empfehlungen — nach Wirkung je Aufwand

#	Maßnahme	Kategorie	Wirkung
1	Serial-Alphabet und -Länge erweitern (Demo-Parameter). Base32 über 25 Zeichen = 125 Bit statt 48 Bit.	Demo-Parameter	sehr hoch
2	Prüf-/Reaktionsroutinen in die laufende Abdeckung aufnehmen.	Lücke	mittel
3	Klartextfenster von <code>.crt2</code> auf Funktionsgranularität verengen.	Grenze (verschiebbar)	mittel
4	Entropieprofil der Sektionen angleichen (Unauffälligkeit).	Lücke	gering–mittel
5	Zeitpunkt der Sektionsentschlüsselung randomisieren.	Lücke	gering

! DER WIRKSAMSTE HEBEL LIEGT IM DEMO-PROGRAMM, NICHT IN CITADELCORE

Empfehlung 1 ist mit Abstand die wirksamste — und sie betrifft **die Demo**, nicht den Schutz. Weil der Serial in dieser Demo das gesamte Schlüsselmaterial trägt, begrenzt seine Entropie (hier 48 Bit) die erreichbare Stärke. Der sehr gute Aufbau von CitadelCore kann nicht mehr leisten als der Schlüssel hergibt. Eine **Formatänderung in der Anwendung** — kein Eingriff in CitadelCore — verschiebt diese Grenze um Größenordnungen. **Es wurde kein Befund der Kategorie „Fehler“ an CitadelCore selbst gefunden.**

① AUFWANDSSCHÄTZUNG DES PRÜFERS

Logik verstehen: 4–8 Stunden für eine erfahrene Person (im Bericht 23 Minuten, weil der Speicherabzug im Timing glückte). **Schutz umgehen** im Sinne von „lesbaren Freischalttext ohne gültigen Serial erzeugen“: **durch Codeanalyse nicht erreichbar** (Test H). Zum Vergleich: ein ungeschütztes Programm gleicher Funktion wäre in **unter 15 Minuten** umgangen — ein Sprungbefehl, ein Byte.

§ Abschluss & Einordnung

Dieser Bericht wurde **unabhängig** erstellt. Sein Zweck ist zweifach: die **Wirksamkeit und Effektivität von CitadelCore nachzuweisen** und konkrete, priorisierte Ansatzpunkte für die **Weiterentwicklung** zu liefern.

✓ ERGEBNIS IN EINEM SATZ

Gegen das gesamte klassische Reverse-Engineering-Repertoire — statische Analyse, Verhaltensbeobachtung, Speicherabzug und Byte-Patching — **hielt der Schutz**: Das eigentliche Geheimnis lag zu keinem Zeitpunkt offen, und selbst ein erzwungener Erfolgspfad liefert keinen lesbaren Text. Die stärksten Verbesserungshebel liegen in **Parametern der Demo-Anwendung**, nicht in der Schutzmechanik von CitadelCore.

Alle Messwerte stammen aus einer einzelnen, reproduzierbaren Prüfsitzung. Die Analyse erfolgte strikt als Blackbox — ohne Einsicht in den Quellcode — und wurde durch künstliche Intelligenz unterstützt, unter Einsatz des zum Analysezeitpunkt aktuellsten Modells von Anthropic.

AetherSec — Security Software Product

Unabhängiger Analysebericht · Erstellt am 27.09.2026

