# 1. Workshop-Ergebnisse

|  |  |
| --- | --- |
| Themenbereich | Erkenntnisse |
| Allgemeine Herausforderungen | 1. Chat-GBT ist bei „grüner Wiese“/generischem Input nicht sonderlich hilfreich - es fehlen Kontext und klare Sicherheitsinformationen. 2. DFDs sind oft unzureichend, da sie keine implementierten Sicherheitsmaßnahmen enthalten. 3. Die Qualität der Analyse steht und fällt mit dem DFD: Magelnder Input, Mangelnder Output. 4. Ein reiner One-Shot-Ansatz ist unrealistisch -> Threat Modeling ist ein iterativer, dialogbasierter Prozess. 5. Nutzer sollten ermutigt werden, möglichst viele Informationen einzugeben - ChatGPT muss zur Nachfragelogik befähigt sein. 6. KI kann nicht vollständig autonom bewerten, es braucht Nutzerfeedback & manuelles Nachbessern. |
| Eingabeformate & Datenquellen | 1. Bildformate (z. B. DFD als Grafik) sind schlecht verarbeitbar, textuelle oder strukturierte Formate (z. B. JSON, YAML) sind besser. 2. Vorteil textbasierter Formate: günstigere und direktere Verwendung mit der API. 3. Möglichkeit, auch den **Output** wieder als JSON zu bekommen, um ihn weiterzuverarbeiten oder zu kürzen. 4. Nummerierung von DFD-Kanten erlaubt gezielte Rückmeldung zu einzelnen Datenflüssen. |
| Zielgruppen & Benutzerprofile | 1. Für Nicht-Security-Experten ist die Tiefe der Ausgabe oft zu viel, es braucht didaktische Aufbereitung. 2. Entwickler interessieren sich mehr für Frameworks, Bibliotheken und technische Umsetzungstipps. 3. Kleine Firmen/Freelancer profitieren besonders, da oft Know-how fehlt oder später nachgeholt wird. 4. Security-Profis würden das System eher als Zweitgutachter oder Challenge-Partner nutzen. 5. System sollte persönliche Expertenprofile berücksichtigen: je nach Vorwissen differenzierte Erklärungen anbieten. |
| Interaktion & Nutzerführung | 1. Das System sollte interaktiv Feedback einholen: **„Hier fehlen mir Infos, bitte nachspezifizieren“**. 2. Threat Modeling sollte eingebettet sein in agiles oder iteratives Vorgehen (Spiralmodell). 3. Das Tool sollte auch Systeme/ Ansätze von vor Wochen analysieren können, was hat sich seit letzter Iteration geändert? 4. System soll helfen, das DFD gemeinsam zu verbessern, nicht nur auswerten, sondern mitentwickeln. 5. Das Interface sollte den Nutzer stärker anleiten: „Was fehlt noch?“, „Was wurde schon implementiert?“. 6. Automatisches Parsen der Antwort nötig: nicht rohe ChatGPT-Ausgabe zeigen, sondern strukturieren. |
| Prompt-Design & Automation | 1. Prompt-Vorlagen könnten helfen, müssen aber sorgfältig genutzt werden (Gefahr: Anchoring-Effekt). 2. Vorlagen können helfen, Themen zu erschließen, in denen der Nutzer kein Experte ist (z. B. Krypto). 3. Gefahr der Überautomatisierung: Nutzer verfallen in „Autopilot“, vertrauen dem System zu sehr. 4. Hintergrund-Prompts sind sinnvoll, sollten aber die Eigenverantwortung des Nutzers nicht verdrängen. 5. Vorschlag: Prompts und Ergebnisse intern validieren oder mit bekannten Mustern/Datenbanken kombinieren. |
| Transparenz & Vertrauenswürdigkeit | 1. System soll seine **Unsicherheiten** klar kommunizieren („Ich vermute, weil ich es nicht genau sehe...“). 2. Vertrauen steigt mit menschlicher, transparenter Kommunikation: wo liegen Unsicherheiten? 3. Persönliche Erklärungstiefe: Experten brauchen keine Basisinfos, Personen ohne Fachwissen hingegen schon. 4. Wichtiger Hinweis: LLMs bleiben fehleranfällig -> Nutzer müssen darauf hingewiesen werden. |
| Zielbild für das Tool | 1. Ein **interaktives System**, das Nutzer durch die Modellierung begleitet und bei der Verbesserung unterstützt. 2. Fokus auf **Bedrohungen erkennen**, nicht unbedingt gleich Lösungen liefern. 3. Ideal für Architekten als Checkliste/Review-Tool zur Überprüfung, ob relevante Threats bedacht wurden. 4. Nicht Ziel: Komplettlösung für Security-Design, sondern **Assistenzsystem im Prozess** |