1. Workshop Ergebnisse

### **Allgemeine Herausforderungen & Überlegungen**

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.

### **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.
   * **Erfüllt:** User-Level wird genutzt. Dadurch können auch Freelancer ohne viel Knowledge das Tool nutzten
4. Security-Profis würden das System eher als Zweitgutachter oder Challenge-Partner nutzen.
   * **Erfüllt:** User-Level wird genutzt. Tool soll algemein beim ThreatModeling assistieren
5. System sollte persönliche Expertenprofile berücksichtigen: je nach Vorwissen differenzierte Erklärungen anbieten.

### **Interaktion & Nutzerführung**

1. Nummerierung von DFD-Kanten erlaubt gezielte Rückmeldung zu einzelnen Datenflüssen
2. Das System sollte interaktiv Feedback einholen: **„Hier fehlen mir Infos, bitte nachspezifizieren“**.
3. Threat Modeling sollte eingebettet sein in agiles oder iteratives Vorgehen (Spiralmodell).
4. Das Tool sollte auch Systeme/ Ansätze von vor Wochen analysieren können, was hat sich seit letzter Iteration geändert?
5. System soll helfen, das DFD gemeinsam zu verbessern, nicht nur auswerten, sondern mitentwickeln.
6. Das Interface sollte den Nutzer stärker anleiten: „Was fehlt noch?“, „Was wurde schon implementiert?“.
7. 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**.

## Detaillierte Beschreibung der Abgehakten Punkte im Interface

1. Chat-GPT is not helpful without context – generic input lacks security focus

|  |
| --- |
|  |

|  |
| --- |
| * You implemented this through:  ➤ build\_user\_level\_instructions()  ➤ build\_threat\_analysis\_prompt() which **forces structured input** (DFD, threat analysis task) and adds guidance. |

2. DFDs often lack info

* Addressed in the prompt generated by build\_threat\_analysis\_prompt(). It explicitly asks for **STRIDE-based threat analysis** and recommendations for **security controls**.

3. Quality of analysis depends on DFD quality

* n build\_threat\_analysis\_prompt() the AI is told to base its work on the DFD and provide a **structured output**, which highlights weaknesses when the DFD is poor. You also support extending the DFD dynamically.

### 4. Iterativer, dialogbasierter Prozess

* **Wie eingebaut:**  
  Du hast im Interface einen iterativen Chat-Flow mit Follow-Up-Optionen implementiert. Nach jeder Antwort wird der Nutzer aktiv gefragt, was er als Nächstes tun möchte (z. B. DFD erweitern, Analyse wiederholen). So wird kein One-Shot, sondern ein Dialogprozess gefördert.
  + **Erfüllt:** Chat-Verlauf wird gespeichert, History genutzt, multiple Sessions möglich → iterative Nutzung technisch möglich
  + In der Chat-Oberfläche wird **nach jeder Antwort** gefragt:  
    **„What would you like to do next?“**
  + Der chat\_container bleibt geöffnet, das input\_query-Feld bleibt aktiv, und du speicherst chat\_history, damit Nutzer schrittweise weiterarbeiten können.
  + Nutzer können bestehende JSON-Dateien iterativ hochladen und verfeinern.

### 5. Nutzer zur Eingabe von mehr Informationen ermutigen

* **Wie eingebaut:**  
  Durch die aktive Rückfrage-UI („What would you like to do next?“) sowie Zwischenfragen kannst du Nutzer dazu bringen, gezielt weitere Details anzugeben, z. B. beim DFD-Update.
  + Nach jeder Antwort erzeugt das System aktiv einen Vorschlag:  
    z. B. *„What do you want to do next?“*
  + Durch den Prompt (user\_level\_instructions) und die follow-up-Logik wird der Nutzer **nicht nur passiv abgeholt**, sondern zum Weiterdenken animiert.
  + Auch bei fehlenden JSON-Daten wird explizit gesagt, dass die Informationen ergänzt werden sollen.

### 7-9. Textbasiertes DFD-Handling und JSON

* **Wie eingebaut:**  
  Du lädst JSON-Daten hoch und verarbeitest sie textbasiert, statt Grafiken. Das System gibt das DFD auch im JSON-Format aus und zeigt es im Interface formatiert an, was gezielte Nachfragen zu Datenflüssen erlaubt.

7.

* + **Erfüllt:** Du akzeptierst JSON-Dateien, keine Bilder → Fokus auf strukturiertes Format
  +  Du nutzt st.file\_uploader() zum Einlesen von **JSON-Dateien** mit DFDs.
  +  Im Prompt gibst du vor, dass **die Antwort ebenfalls ein JSON-DFD enthalten soll**, inklusive nodes und edges.
  +  Beispiel-Ausgabe wird dem LLM direkt mitgegeben.
  +  Du **parst das JSON mit re.search**, zeigst es **formatiert mit st.code(..., language="json")** an.
  +  Deine DFD-Knoten sind nummeriert ("id": 1) → gezielte Diskussion über Datenflüsse wird möglich.

8. Vorteil textbasierter Formate: günstigere und direktere Verwendung mit der API.

* + **Erfüllt:** JSON-Upload, Ausgabe ebenfalls als JSON → textbasiert
  +  Du nutzt st.file\_uploader() zum Einlesen von **JSON-Dateien** mit DFDs.
  +  Im Prompt gibst du vor, dass **die Antwort ebenfalls ein JSON-DFD enthalten soll**, inklusive nodes und edges.
  +  Beispiel-Ausgabe wird dem LLM direkt mitgegeben.
  +  Du **parst das JSON mit re.search**, zeigst es **formatiert mit st.code(..., language="json")** an.
  +  Deine DFD-Knoten sind nummeriert ("id": 1) → gezielte Diskussion über Datenflüsse wird möglich.

9. den Output wieder als JSON zu bekommen, um ihn weiterzuverarbeiten oder zu kürzen.

* + **Erfüllt:** Prompt fordert JSON-Output, Antwort enthält JSON am Ende
  +  Du nutzt st.file\_uploader() zum Einlesen von **JSON-Dateien** mit DFDs.
  +  Im Prompt gibst du vor, dass **die Antwort ebenfalls ein JSON-DFD enthalten soll**, inklusive nodes und edges.
  +  Beispiel-Ausgabe wird dem LLM direkt mitgegeben.
  +  Du **parst das JSON mit re.search**, zeigst es **formatiert mit st.code(..., language="json")** an.
  +  Deine DFD-Knoten sind nummeriert ("id": 1) → gezielte Diskussion über Datenflüsse wird möglich.

### 10, 11, 14. Benutzerprofile & Anpassung der Erklärungstiefe

* **Wie eingebaut:**  
  Du nutzt die User-Level-Eingabe (1-5) und baust in den Prompt dynamisch unterschiedliche Anweisungen ein, die Erklärungen und Fachbegriffe an das Nutzerlevel anpassen – z.B. didaktische Erklärungen für Anfänger, technische Details und Frameworks für Fortgeschrittene, knappe Hinweise für Experten.

10. Für Nicht-Security-Experten

* + **Erfüllt:** User-Level wird genutzt
  +  Nutzer geben ihr Sicherheitswissen zu Beginn an via:
  + python
  + KopierenBearbeiten
  + st.select\_slider("Rate your IT security knowledge from 1 to 5", ...)
  +  In build\_user\_level\_instructions() erzeugst du automatisch **zielgruppenspezifische Prompts**.

11. Entwickler interessieren sich mehr für Frameworks, Bibliotheken und technische Umsetzungstipps.

* + **Erfüllt:** User-Level wird genutzt
  +  Du hast Level 4 als "Softwareentwickler oder IT-Architekt" definiert.
  +  In deinem Prompt-Template (build\_user\_level\_instructions) steht:  
    *"Provide practical implementation advice and reference frameworks (e.g., STRIDE, OWASP ASVS) where applicable."*
  +  Für Level 1–2 gibst du ein didaktisch aufbereitetes Expertenprofil vor, das technische Begriffe erklärt, Zusammenhänge erläutert etc

12. Kleine Firmen/Freelancer profitieren besonders, da oft Know-how fehlt oder später nachgeholt wird.

* + **Erfüllt:** User-Level wird genutzt. Dadurch können auch Freelancer ohne viel Knowledge das Tool nutzten. Btw vllt die Informatiker die bereits im Team sind, kein EXTRA IT\_Security EXPERTE nötig?

13. Security-Profis würden das System eher als Zweitgutachter oder Challenge-Partner nutzen.

* + **Erfüllt:** User-Level wird genutzt. Tool soll algemein beim ThreatModeling assistieren. Btw gibt Level 5 = IT-Security extra für Ecperten nicht zu viel unnötige infos raus

14. Expertenprofile berücksichtigen

* + **Erfüllt:** User-Level wird genutzt
  +  Deine Funktion build\_user\_level\_instructions() definiert für jeden Level (1 bis 5) **eine andere Sprach- und Erklärungstiefe.**
  +  Level 5 z. B. bekommt kurze, prägnante Antworten, keine redundanten Erklärungen.
  +  Level 1 bekommt erläuternde Texte mit Definitionen und Beispielen.
  +  Diese Anweisungen fließen dynamisch in jeden Prompt.

### 16, 17, 19, 20, 21. Interaktive Nutzerführung und strukturierte Ausgabe

* **Wie eingebaut:**
  + Das System fragt aktiv nach dem nächsten Schritt (Rückfragen) und fördert so iterative Verbesserung.
  + Antworten werden automatisch geparst, JSON extrahiert und im Interface strukturiert angezeigt, nicht als Rohtext.
  + Durch die sidebar und die Chat-History bleibt der Kontext erhalten.
  + Das Interface gibt dem Nutzer Orientierung, was er als nächstes tun kann.

16. Das System sollte interaktiv Feedback einholen: **„Hier fehlen mir Infos, bitte nachspezifizieren“**.

* +  Dein System fragt aktiv nach, wenn keine oder unvollständige JSON-Daten vorliegen.
  +  Du baust Rückfragen in die Prompt-Logik ein: *„If something is unclear, ask follow-up questions.“*
  +  Der Chatverlauf bleibt offen und reagiert dynamisch auf neue User-Inputs.

17. Threat Modeling sollte eingebettet sein in agiles oder iteratives Vorgehen (Spiralmodell).

* + **technisch möglich:** Sessions + History + JSON-Update ermöglichen Iteration
  +  Der Upload von JSON-Daten erlaubt Änderungen über mehrere Schritte hinweg.
  +  Durch die Chat-Historie (StreamlitChatMessageHistory) ist der Zustand jederzeit rekonstruierbar.
  +  Das System bietet nach jeder Runde neue Fragen an, um den Kreislauf zu fördern.

19. System soll helfen, das DFD gemeinsam zu verbessern, nicht nur auswerten, sondern mitentwickeln.

* + **Teilweise:** JSON wird eingelesen und Antwort kann DFD updaten → gemeinsames Verbessern angedacht
  +  Das System generiert **einen aktualisierten DFD** am Ende jeder Antwort.
  +  Die Aufforderung *„generate an updated Data Flow Diagram“* ist fest im Prompt.
  +  Du gibst dem Nutzer die JSON-Ausgabe zurück und fragst:  
    *„Möchtest du den DFD erweitern oder ändern?“*

20. Das Interface sollte den Nutzer stärker anleiten: „Was fehlt noch?“, „Was wurde schon implementiert?“.

* +  Nach jeder Antwort erscheint die Follow-up-Frage: *„What would you like to do next?“*
  +  Du hast die Möglichkeit vorbereitet, im Prompt auf Unsicherheiten und fehlende Informationen hinzuweisen.
  +  Die Sidebar erlaubt den Wechsel von Sessions – dadurch ist Kontextmanagement möglich.

21. Automatisches Parsen der Antwort nötig: nicht rohe ChatGPT-Ausgabe zeigen, sondern strukturieren.

* + - Du verwendest re.search() um die JSON-DFD-Ausgabe aus dem Antworttext herauszuziehen.
    - Der Text und das JSON werden getrennt dargestellt:
  + Antworttext als Fließtext
  + DFD separat mit st.code(...)

18.

* geht darum wieder an chatverlaufen weiterarbeiten zu können, die man bsp. letzte woche als letztes ebarbeitet hatte und der session sidebar macht das möglich.
  + **Erfüllt:** Chat-Verlauf wird gespeichert, History genutzt, multiple Sessions möglich → wieder am Chat weiterarbeiten

### 24. Vermeidung von Überautomatisierung

* **Wie eingebaut:**  
  Die aktive Nutzerführung mit Rückfragen verhindert, dass Nutzer nur passiv konsumieren. Sie bleiben eingebunden und müssen eigene Entscheidungen treffen.
  +  Deine Follow-up-Frage zwingt den Nutzer zur aktiven Entscheidung.
  +  Du gibst keine Lösung vor, sondern fragst nach dem nächsten Schritt.
  +  Das Prompt-Design ist so formuliert, dass es „structured guidance“ statt Entscheidungen vorgibt.

### 25, 26. Hintergrund-Prompts und Validierung

* **Wie eingebaut:**  
  Ein systematischer Prompt mit Hinweis auf Eigenverantwortung wurde erstellt. Externe Datenbanken oder Validierungen fehlen aktuell noch, aber die Struktur ist vorbereitet.

25. Hintergrund-Prompts sind sinnvoll, sollten aber die Eigenverantwortung des Nutzers nicht verdrängen.

* +  Dein memory\_prompt\_template betont:  
    "You point out uncertainties and formulate assumptions transparently."
  +  Außerdem:  
    "Instead of providing ready-made solutions, you offer well-founded assessments, thought-provoking impulses..."
  +  Dein Prompt enthält Referenzen zu STRIDE, OWASP etc.
  +  Du hast aber **noch keine echte Validierung gegen Datenbanken wie CVE integriert.**

### 27, 28, 30. Transparente Kommunikation der Unsicherheiten

* **Wie eingebaut:**  
  Im Prompt und in der System-Message wird der LLM angewiesen, Unsicherheiten explizit zu benennen und Nutzer auf die Fehleranfälligkeit hinzuweisen.

27.System soll seine **Unsicherheiten** klar kommunizieren („Ich vermute, weil ich es nicht genau sehe...“).

* +  Prompt-Anweisung:  
    "You point out uncertainties and formulate assumptions transparently."
  +  Du ermutigst Nutzer, Entscheidungen selbst zu treffen, statt blinden Vertrauen zu fördern.

28. Vertrauen steigt mit menschlicher, transparenter Kommunikation: wo liegen Unsicherheiten?

* +  Prompt-Anweisung:  
    "You point out uncertainties and formulate assumptions transparently."
  +  Du ermutigst Nutzer, Entscheidungen selbst zu treffen, statt blinden Vertrauen zu fördern.

30. wichtiger Hinweis: LLMs bleiben fehleranfällig -> Nutzer müssen darauf hingewiesen werden.

* +  Prompt-Anweisung:  
    "You point out uncertainties and formulate assumptions transparently."
  +  Du ermutigst Nutzer, Entscheidungen selbst zu treffen, statt blinden Vertrauen zu fördern.

### 31, 32, 33. Zielbild: interaktives Assistenzsystem, kein Komplettlösung

* **Wie eingebaut:**  
  Das gesamte Interface ist als interaktiver Assistent aufgebaut, der den Nutzer durch Modellierung begleitet. Es werden keine fertigen Lösungen gegeben, sondern Impulse und Hinweise. Fokus liegt auf Erkennen von Bedrohungen und Unterstützung.

31.Ein interaktives System, das Nutzer durch die Modellierung begleitet und bei der Verbesserung unterstützt.

* +  Dein gesamtes Interface zielt auf einen interaktiven Prozess ab.
  +  Du bietest keine fertige „Bedrohungsliste“ oder Score, sondern Impulse zur selbständigen Entscheidung.
  +  Das Tool eignet sich damit ideal als Begleiter z. B. für Architekturreviews oder Design-Diskussionen.

32. Fokus auf Bedrohungen erkennen, nicht unbedingt gleich Lösungen liefern.

* + Dein gesamtes Interface zielt auf einen interaktiven Prozess ab.
  +  Du bietest keine fertige „Bedrohungsliste“ oder Score, sondern Impulse zur selbständigen Entscheidung.
  +  Das Tool eignet sich damit ideal als Begleiter z. B. für Architekturreviews oder Design-Diskussionen.

33. Ideal für Architekten als Checkliste/Review-Tool zur Überprüfung, ob relevante Threats bedacht wurden.

* + **Erfüllt:** Fokus auf Begleitung & Assistenz ist da
  + Dein gesamtes Interface zielt auf einen interaktiven Prozess ab.
  +  Du bietest keine fertige „Bedrohungsliste“ oder Score, sondern Impulse zur selbständigen Entscheidung.
  +  Das Tool eignet sich damit ideal als Begleiter z. B. für Architekturreviews oder Design-Diskussionen.

34.

* dadurch, dass die KI immer hintergrundpropmts hat, nachfragen stellt, fehler aufweist im DFD und iterativ mit dem Nutzer arbeitet ist die gegeben
* abgehakt, da die KI durch Design, Prompt und UI ein iteratives, begleitendes Assistenzsystem bildet – wie du korrekt argumentiert hast

**1.Schritt**

Zielgruppe: Anfänger, Junior und Jinor Leute

Interface soll: DFD verbessern/vertieften, auswertung, transparenz verbessern (welche infos fehlen noch) (Output soll Analyse, detailierte Infos über DFD (4-5 Kanten))

Was brauch man als Input und was als Output und was muss in der Mitte passieren?????????

**Interface:**

Input: (JSON)-file und text input

Chat-Fenster (mit iterativen Versuch)

Output: text + JSON-file (strukturiert und hinweise auf Grenzen von Chat-GBT)

Logic-Desicion Tree aufbauen (DFD Kanten hinzufügen)

Beginn: Nutzer sollen sich in Knowledge einranken.

JSON-Datei einlesen → analysieren → Feedback generieren → neue JSON-Version → wieder analysieren usw.

|  |  |  |  |
| --- | --- | --- | --- |
| Phase | Eingabe | Prozess | Ausgabe |
| 1. Start | Erste JSON-DFD | Grundanalyse + STRIDE + Kantenerkennung (DFD- Prüfung der Struktur: Anzahl Knoten/Kanten, Typen, Verbindungen) | Analyse + neue JSON v1 |
| 2. Dialog 1 | JSON v1 + User-Input (Bestätigter oder angepasster DFD | Prüfung offener Fragen, Rückfragen stellen (Interaktiver Dialog: gezielte Fragen stellen, KI-Grenzen kommunizieren, Nutzeranweisungen geben, Markierung fehlerhafter oder verdächtiger Knoten/Kanten) | JSON v2 + präzisere Analyse |
| 3. Dialog 2 | JSON v2 + User-Input | Weitere Detaillierung, Validierung | JSON v3 ... |
| ... | ... | ... | ... |
| Finale | JSON final + STRIDE (Alle gesammelten Daten & Nutzerfeedback) | Endauswertung | Endbericht + JSON final |
|  |  |  |  |

* **Diskussion Workshop ergebniss als stichpunkte 1.-…**
* **diese in Methodik aufgreifen: benutzen ergebnis von 1. workshop in unserem interface. Bsp. Punkt 1 ahebn wir so und so ungestzt….**