Honeyquest: Rapidly Measuring the Enticingness of Cyber Deception Techniques with Code-based Questionnaires

Mario Kahlhofer
mario.kahlhofer@dynatrace.com
Dynatrace Research
Linz, Austria
Johannes Kepler University
Linz, Austria

Stefan Achleitner
stefan.achleitner@dynatrace.com
Dynatrace Research
Linz, Austria

Stefan Rass
René Mayrhofer
stefan.rass@jku.at
rene.mayrhofer@jku.at
Johannes Kepler University
Linz, Austria

Abstract

Fooling adversaries with traps such as honeytokens can slow down cyber attacks and create strong indicators of compromise. Unfortunately, cyber deception techniques are often poorly specified. Also, realistically measuring their effectiveness requires a well-exposed software system together with a production-ready implementation of these techniques. This makes rapid prototyping challenging. Our work translates 13 previously researched and 12 self-defined techniques into a high-level, machine-readable specification. Our open-source tool, Honeyquest, allows researchers to quickly evaluate the enticingness of deception techniques without implementing them. We test the enticingness of 25 cyber deception techniques and 19 true security risks in an experiment with 47 humans. We successfully replicate the goals of previous work with many consistent findings, but without a time-consuming implementation of these techniques on real computer systems. We provide valuable insights for the design of enticing deception and also show that the presence of cyber deception can significantly reduce the risk that adversaries will find a true security risk by about 22% on average.

CCS Concepts

• **Security and privacy** → **Web application security**; *Intrusion detection systems*; *Systems security*; *Network security*.

Keywords

cyber deception, effective deception, honeytokens, honeypots

1 Introduction

Cyber deception deceives adversaries about the true appearance of a software system, tricking them into taking (or not taking) actions that are not in their favor [108, 110, 102]. Imagine that an attacker has already broken into a container somewhere in your infrastructure, completely undetected by any security measures. At this stage, the goal of such an adversary could be to move laterally through your infrastructure and take over additional resources. We can defend against that by placing honeytokens [92, 93] in the container: Fake credentials or tokens that trigger an alarm when used. Such incidents may then be escalated to a human operator for further investigation. Benefits of honeytokens are: [40, 39]

(1) **Adversaries are slowed down** as they waste time with unsuccessful exploit attempts.
(2) **Defenders get strong indicators of compromise (IoCs)** from such alarms for incident resolution.
(3) **Reduces the risk of adversaries exploiting true weaknesses** because they are distracted by honeytokens.

Recent research has come up with great techniques to deceive attackers (§8.1). But are they effective? Will attackers fall for such traps, or will they see through them? After all, hackers are neither lazy nor stupid. Bowen et al. [24] introduced various properties that can guide us in designing effective decoys. Ben Salem and Stolfo [22] found six of them to be very important, the first three being detectability, conspicuousness, and enticingness. *Detectability* describes the necessary requirement to detect when a trap has been triggered. *Enticingness* describes how attractive a trap is for an adversary, how well it lures them and awakens desires and hopes to achieve their mission. *Conspicuousness* is similar to enticingness, but conspicuous traps are chosen by adversaries because they are easily found, clearly visible, or obvious, but not necessarily because they are attractive. To measure these properties with real humans, researchers typically use one of three methods (depicted in Figure 1): Capture The Flag (CTF) events, honeypots, or questionnaires.

CTF events, red team engagements, or cyber ranges [88, 41, 14, 11, 46, 40, 91, 56, 22, 101, 30, 29, 87, 21, 54, 7–9] are competitions where participants attack and defend software systems. Creating such environments for deception experiments is very labor-intensive because engineers have to setup the infrastructure, mimic a realistic app, and implement traps. The latter also presents various technical challenges [87, 55, 60].

Honeypots in the wild [24, 54, 83, 82, 25, 43] are software systems that want to be attacked. While deploying such honeypots brings the closest contact to real adversaries, it typically requires a well-exposed software system that is of interest to adversaries, along with a production-ready implementation of traps. In addition, it relies on waiting for attackers to come along and fall for the traps, resulting in slow feedback loops.
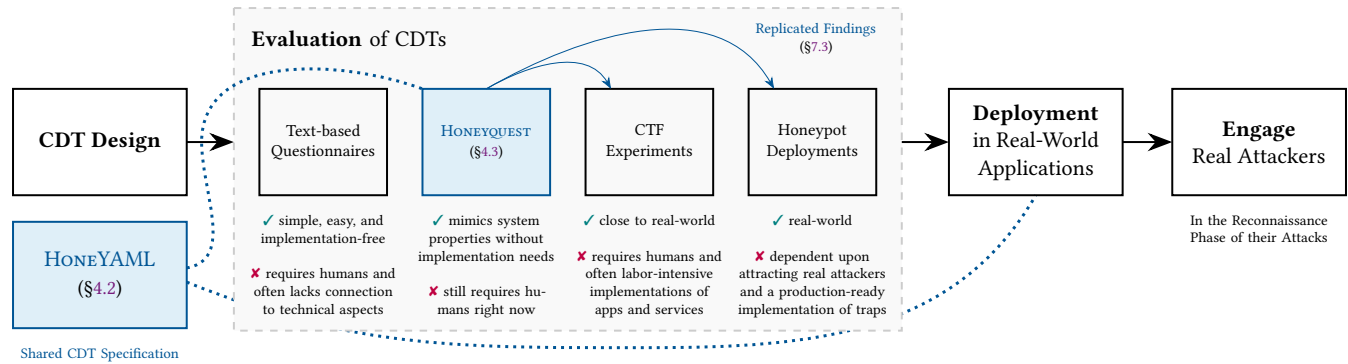
Figure 1: The lifecycle of designing, evaluating, and deploying CDTs, with the ultimate goal of engaging real adversaries.

Questionnaires [39, 88, 4, 37, 41, 82, 83, 11, 17, 46, 40, 23, 87] can rapidly test specific deception hypotheses. CTF events and honeypots can hardly measure psychological properties [38], which explains why they are often accompanied by questionnaires [87, 88, 41, 40, 46, 11, 83, 82]. However, we argue that text-based questionnaires quickly become detached from the technical "views" that adversaries typically gain from a system.

Our work introduces **Honeyquest** as a method that combines the benefits of questionnaires with the realism of CTF events and honeypots. Questions in Honeyquest – we call them *queries* – imitate the technical views that adversaries typically gain of a software system, e.g., by presenting a real file listing with honeytokens in it. We ask participants to mark what they would try to ⚓ exploit and where they spot potential ☻ traps. This allows us to measure the enticingness of various **Cyber Deception Techniques (CDTs)** in a fast and controlled manner. To bridge the gap to an actual technical implementation of CDTs within a software system, we introduce **HoneYAML**. We describe traps in our questionnaires with HoneYAML, but also use it to directly configure deception products. HoneYAML further allows us to clearly define traps and conduct easily reproducible experiments with them. We contribute:

(1) A method to test the enticingness of CDTs (§3).
(2) A translation of 13 previously researched [88, 87, 54, 77, 75, 83, 82] and 12 self-defined CDTs, into **HoneYAML**: A high-level, machine-readable specification of CDTs (§4.2).
(3) **Honeyquest**: A flexible open-source[1] tool for setting up studies that measure the enticingness of CDTs (§4.3).
(4) Results of a human subject study using Honeyquest: We show 47 humans ☻ 80 neutral, ⚡ 23 risky, and 🎣 71 deceptive components of a web application (§6). Our results validate many previous findings and also unveil new insights (§7). Raw data from that study is available in our repository.

2 Problem Statement

Ultimately, we want to use cyber deception to defend against adversaries. But first, we highlight the problem of designing reproducible experiments to measure the enticingness of CDTs. Then, as a case study, we consider CDTs that can secure web applications.

[1] https://github.com/dynatrace-oss/honeyquest

2.1 Lack of Reproducible Experiments

Experiments on deceiving humans necessarily involve real humans, which makes conducting and replicating such studies challenging. Han et al. [55] point out that "it is often impossible to test deception techniques offline" and that "[properties for achieving effective deception] are difficult to formalize and measure", which contributed to a widespread "lack of reproducible experiments" [55, Sec. 6-7].

To align with and replicate prior work (§7.3), we looked for works that provided at least three ingredients: (1) A detailed description of the tested CDTs beyond vague terms like "honeyfiles". (2) A quantitative evaluation of the effectiveness of these CDTs, beyond assumptions about attacker behavior. (3) A report on the results obtained, beyond aggregate statistics. We found most of these items in seven works [88, 87, 54, 77, 75, 83, 82], whose findings we could hence validate at least partially. Further work often lacked some details for confidentiality reasons. These items also inspired us to define CDTs with HoneYAML, have a theoretically-grounded approach to measure enticingness, and open-source raw results.

2.2 Defending Threats with Cyber Deception

We consider adversaries in cloud environments in the reconnaissance phase of an attack [66]. They may aim to establish a foothold on a system or are already inside it, trying to move laterally to complete their mission. Our work proposes a novel approach to evaluate what CDTs are most effective against adversaries at this stage of an attack, by measuring how well they entice attackers.

To demonstrate feasibility, we study four components of a web application, where CDTs can be applied. We chose these four because they are "mostly invisible to benign users" [54] and will not interfere [22] with legitimate activities:

- **File system.** Honeyfiles like "keys.txt" that appear sensitive, allow us to detect unauthorized access attempts.
- **.htaccess files** configure Apache servers. These should never be publicly accessible. We deliberately expose these files with sensitive paths in them (e.g., to a fake admin site) and detect attackers who access these paths.
- Attackers might observe **HTTP response** packets by probing endpoints. If we add HTTP headers that are indicative of known vulnerabilities, we aim to lure attackers into trying unsuccessful exploits for them.

- Attackers could monitor all **HTTP requests** of an application. By adding fake tokens to those requests, we aim to lure attackers into using them for subsequent attacks.

A deception systems can be structured into decoys and captors [35]. *Decoys* are the entities being attacked, e.g., a honeytoken, while *captors* perform the security-related functions, e.g., logging and alerting on access attempts. Our work focuses solely on the evaluation of decoys, which we call CDTs. Decoys and captors can be readily implemented: [60] Creating files is trivial in most operating systems. Monitoring access attempts to them can be achieved with architectures such as SELinux [69]. Intercepting, modifying, and monitoring HTTP packets is often achieved with a reverse proxy in front of applications [54, 16, 21, 44, 87, 78].

3 Measuring the Enticingness of Cyber Deception Techniques

This section presents an approach to quantify the enticingness of CDTs. This lays the groundwork for the design of Honeyquest (Figure 2) in §4, and its evaluation in §5.

3.1 Queries, Labels, Marks, and Annotations

In the reconnaissance phase, attackers explore their target. While probing our system, they might find certain properties depending on the technique used, i.e., they gain different "views" of our system.

Assume that an attacker has already managed to break into a container. They might perform the naive technique of "listing files" and observe Listing 1. In Honeyquest, we call this a query. A query is just plain text, i.e., a collection of lines.

```
drwxr-xr-x 25 elsa 4.0K Dec 30 08:36 .
drwxr-xr-x  4 root 4.0K Jun 21  2019 ..
-rw-------  1 elsa  57K Jan 13 14:48 .bash_history
-rw-r--r--  1 elsa 3.5K Sep 17  2017 .bashrc
drwx------  6 elsa 4.0K Sep 25 17:40 .config
drwxr-xr-x  6 elsa 4.0K Nov 14 09:08 .npm
drwx------  6 elsa 4.0K Nov 14 09:12 .yarn
-rwxr-xr-x  1 elsa 3.9K Dec  5 16:02 buildcsv.py
-rw-r--r--  1 elsa  12K Feb  6  2022 keys.json
```

Listing 1: A "file system" query with CDT DF3 injected in it.

We are now curious about the next move of an adversary. In a file system, possible actions may be reading a file, visiting a directory, or doing nothing at all. So we allow our adversary to place either **➤ exploit marks** or **🪤 trap marks** on each line in a query. Not marking anything is also a valid action – and the default, since no lines are marked initially. Placing an exploit mark means that an adversary sees a potential security weakness on that line. When presented with a file system, this signifies that the adversary would like to examine the file or directory on that line or attack it somehow. On the other hand, marking something as a trap means that an adversary definitely wants to avoid interacting with that line. In a file system, this would mean that these particular files must not be opened in order to avoid triggering an alarm.

Adversaries may want to try the most promising attack vector first. To let them express this, we number **answer marks** in the order in which they are placed on a line. These numbers are also visible to the user (Figure 6). This feature allows us to find out which parts of a query attract an adversary's attention first (§3.2.2).
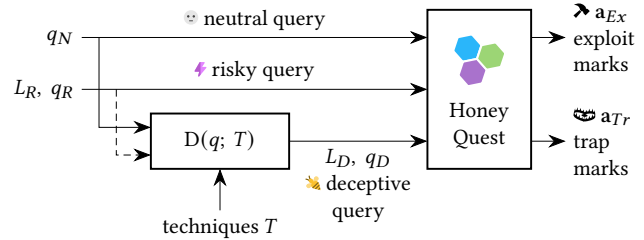

Figure 2: In Honeyquest, users are presented with ☻ neutral, ⚡ risky, and 🪤 deceptive queries. A line annotation set L indicates the risky or deceptive lines in the associated query q. An answer marks vector a holds placed marks in order. The probabilistic algorithm $D(q; T)$ makes queries deceptive.

To summarize, we imitate views on systems, frame them as queries, and let users mark what they want to exploit or avoid, in order. Recalling that one defined goal of cyber deception is "tricking [adversaries] into taking (or not taking) actions that are not in their favor" [108, 110, 102], and that "enticement depends upon the attacker's intent or preference" [24], *we are interested in queries, where adversaries mark the deceptive elements to be exploited, and not to be avoided, thus falling for a trap.* This expresses the "enticement" property from Bowen et al. [24] with exploit and trap marks.

Each query has a **label** that indicates which of these three strategies (§5.1) was followed in its design:

- ☻ **Neutral** queries may be harmless, secure, benign, well-protected, or of neutral appearance.[2]
- ⚡ **Risky** queries may be harmful, insecure, malicious, lack security measures, or have negative intent. Here, the system owner bears that risk, not the adversary.[3]
- 🪤 **Deceptive** queries want to grab the attention of an adversary, often by seeming risky. They contain CDTs.

Risky and deceptive queries have so-called **line annotations**, which store the exact line numbers that are risky or deceptive.

Put together, we get **Honeyquest** (Figure 2). Honeyquest shows users queries of different types and labels, where they can mark every line with ➤ exploit marks, 🪤 trap marks, or nothing at all. Deceptive queries are created by modifying a neutral or risky query (§4.2). For example, in a file system query, we might purposefully add a "keys.json" entry as a trap. Later, in a deployed deception system, we would monitor if a potential adversary accesses this file or tries to use one of the fake passwords that we deliberately placed inside of it for authentication.

Careful readers may find that it is often impossible to tell without context (e.g., implementation details) whether certain query elements are risky or deceptive. This is true, and is why we do not evaluate if users accurately identify a query's label, but what parts of a query users perceive as exploitable or deceptive.

---

[2]We intentionally chose the term "neutral" over "benign", because we do not want to imply the positive connotation that "benign" typically expresses.

[3]We have deliberately chosen not to use the common terms "malicious" or "vulnerable". Maliciousness expresses a harmful intent, which can rarely arise from a textual query alone. Vulnerabilities are commonly defined as weaknesses that might be exploitable. So while our risky queries could be interpreted as "weaknesses", we do not want to imply that they are exploitable.

3.1.1 Matching Answer Marks and Line Annotations. After users have placed their marks, we want to determine if they identified potential traps or risks. In other words, we just check if answer marks intersect with line annotations.

Table 1 introduces some terminology to express that clearly. The set L reflects the "ground truth" labels, and the set A holds user's answer marks. We later specialize the notation by talking about deceptive or risky line annotations, L_D and L_R, respectively, and about ➤ exploit and 😵 trap marks, A_{Ex} and A_{Tr}, respectively. When we need the order in which the marks have been placed, we will refer to the vectors $\mathbf{a}_{Ex}$ and $\mathbf{a}_{Tr}$ instead. Table 2 shows an example of a query that uses this terminology.

Consider a user answering a 🐱 deceptive or ⚡ risky query. A simple way to express that "answer marks A match line annotations L" is to check whether they intersect:

$$\checkmark \text{ match}: \quad L \cap A \neq \varnothing \qquad \textbf{✗} \text{ no match}: \quad L \cap A = \varnothing \qquad (1)$$

This matching criterion is sufficient for our experiment and is also well suited for expressing results with typical confusion matrices (Appendix A). It has the small drawback that answers that intersect the line annotations only partially are also "matching". Even worse, answers that place marks on every single line will always intersect – and therefore "match" – with every possible set of line annotations. However, in our experiment, only 0.19% of answers had marks on every single line. Further, only 0.82% of answers marked risky or deceptive lines only partially. This is not surprising, since only 6.82% of risky queries and only 4.55% of deceptive queries in our current dataset have more than one line annotated anyway. We therefore conclude that this simple matching criterion will not significantly bias our results. Appendix B discusses alternative matching criteria for different experimental conditions.

3.2 Research Questions

We differentiate between the **enticingness of deception** by itself, in the sense that humans fall for traps (Aspect A), and its **ability to be defensive**, by deliberately diverting attention (Aspect B). Thus, we ask the following research questions:

"To what degree are humans enticed by deceptive elements, true weaknesses and vulnerabilities, and will deceptive elements divert their attention away from true risks?"

This section formulates hypotheses on that question. We then select CDTs and risks for testing (§5), and then report (§6) and discuss (§7) results that answer this question.

*3.2.1 **Aspect A:** To what degree are humans enticed by deceptive and risky elements?* Consider that we show humans ☺ neutral (Q_N), 🐱 deceptive (Q_D), and ⚡ risky (Q_R) queries. We know which technique t was used to create deceptive queries and which risk is present in the risky ones (§5). To measure the enticingness of individual 🐱 CDTs, we group answers by CDT and count how often participants fell for traps, detected traps, or did not react to traps. Likewise, for ⚡ risks, we group by risks and count how often participants detected risks, have mistaken risks for traps, or did not react to risks. Appendix C lists the explicit formulation of those counts.

*3.2.2 **Aspect B1:** Do humans exploit deceptive elements before non-deceptive elements?* Let 🐱 $q_D \in Q_D$ be a deceptive query with deceptive lines L_D where more than one ➤ exploit mark was placed, i.e., $|A_{Ex}| > 1$. As before, we assume that a human "fell for a trap" when exploit marks intersect deceptive lines, i.e., $L \cap A_{Ex} \neq \varnothing$. Remember that participants were instructed to place marks in an order that indicates what they would like to exploit first. Let $a' \in A_{Ex} \cap L_D$ be the first exploit mark that marked a deceptive line and let $a'' \in A_{Ex} \setminus L_D$ be the first exploit mark that marked a non-deceptive line. a'' may be a risky or neutral line then. Let d'_B be the number of times that a' is ranked before a'' out of d_B samples where all of these aforementioned conditions hold.

We can phrase this null hypothesis H_0: When users place exploit marks A_{Ex} that intersect with deceptive lines L_D, whether a' or a'' is ranked first is up to chance. We chose a Binomial test that tries to reject $t = 1/2$ with the one-sided alternative that t is greater.[4]

$$t = \frac{d'_B}{d_B} \sim B(d_B, 1/2) \qquad (2)$$

---

[4]We used the `binomtest` function from SciPy [100] and computed the test's power with the `binom.power` function from the binom package [32].

Table 1: Terminology and common specializations.

Queries[a]	$q_N \in Q_N, \ q_R \in Q_R, \ q_D \in Q_D$	
Techniques[b]	$t \in T$	
Line Annotations	$L := \{\ell^1, \ \ell^2, \ ...\}$	$L_D, \ L_R$
Answer Marks[c]	$\mathbf{a} := (a^1, \ a^2, \ ...)$	$\mathbf{a}_{Ex}, \ \mathbf{a}_{Tr}$
	$A := \{a^1, \ a^2, \ ...\}$	$A_{Ex}, \ A_{Tr}$
Algorithm[d]	$q_D \leftarrow D(q; \ T) \in Q_D$ with $q \in Q_N \cup Q_R$	

Query and Annotation Types: ☺ N = Neutral. ⚡ R = Risky. 🐱 D = Deceptive. **Answer Types:** ➤ Ex = Exploit. 😵 Tr = Trap.

[a] All query sets are pairwise disjoint.

[b] We use the terms technique and CDT interchangeably throughout the paper.

[c] **a** is a vector since users place marks in order. For equations that do not need ordered marks, we define A as the set of unique elements from **a**.

[d] The probabilistic algorithm D makes a query q deceptive, by selecting a suitable but random element from T and applying it to q. Our experiments chose the particular technique t manually for consistency with q.

Table 2: Query with line annotations and answer marks.

Line Annot.	#	Query Line	Ans. Marks
⚡ $L_R = \{3\}$			➤ $\mathbf{a}_{Ex} = (4, 3)$
🐱 $L_D = \{4\}$			😵 $\mathbf{a}_{Tr} = (2)$
	1	`HTTP/1.1 200 OK`	
	2	`Server: Apache/2.4.1`	$a_{Tr}^1 = 2$
$\ell_R^1 = 3$	3	`X-Powered-By: PHP/5.1.6`	$a_{Ex}^2 = 3$
$\ell_D^1 = 4$	4	`X-Api-Server: /hko/api`	$a_{Ex}^1 = 4$

Description: A deceptive "HTTP response" query q_D with one ⚡ risky (RP3, true vulnerability, purple) and one 🐱 deceptive (DP3, injected weakness, orange) line annotation. The resulting query contains a true vulnerability as well as a trap. **Answer Marks:** A user placed three marks here. A 😵 trap mark in line 2 that was no trap, an ➤ exploit mark in line 3 on the true vulnerability, and another ➤ exploit mark in line 4 on the trap. The order indicates that our user would exploit line 4 first, and therefore fall for the trap first.

A greater ratio hints at a greater preference to mark traps first. This formulation is equivalent to the "believability" property for "a perfect decoy [...] that is completely indistinguishable from one that is not", as proposed by Bowen et al. [24].

3.2.3 Aspect B2: Are deceptive elements diverting an attacker's interest away from risky elements?

Instead of looking at what is marked first, we test if the presence of deception is so distracting that an attacker misses weaknesses and vulnerabilities entirely.

Consider that we have a set of risky queries Q_R. Let $\lightning\, q_R \in Q_R$ be a risky query with risky lines L_R. From that query, we derive a new deceptive query $q_D = D(q_R; T)$ with deceptive lines L_D and risky lines L'_R. Note that we only introduce L'_R because making a query deceptive means that we insert new lines, which could also change the line numbers of risky lines. We now present q_R and q_D to each participant (within-subject) and record the $\searrow$ exploit marks A_{Ex} that each of the two queries receives. As in all our experiments, we assume that a human has "interest in exploiting a line" when exploit marks intersect risky lines.

We phrase this null hypothesis H_0: When we show participants a risky query q_R and a derived deceptive query q_D, there is no difference in what they mark to exploit, i.e., the presence of deceptive lines does not distract them.

Table 3 formulates this with a 2×2 contingency table over two factors: "Did the participant mark the risky lines L_R to exploit in the risky query q_R" ("before" condition), and "did the (same) participant mark the risky lines L'_R to exploit in the derived deceptive (and still risky) query q_D" ("after" condition). To draw an analogy, think of deception as being the treatment for risky queries, where the disease "breaks out" when patients detect the risk. We test if the "deception treatment" effects the "disease break-out".

Table 3: Contingency table on attention diversion.

Match in q_D?	Match in q_R? ("before")	
("after")	✗ $L'_R \cap A_{Ex} = \varnothing$	✓ $L'_R \cap A_{Ex} \neq \varnothing$
✗ $L_R \cap A_{Ex} = \varnothing$	α	β
✓ $L_R \cap A_{Ex} \neq \varnothing$	γ	δ

Legend: A ✗ cross-mark indicates that answer marks A_{Ex} do not intersect with risky lines. A ✓ check mark indicates that they do intersect / match.

Our two factors and the single outcome are nominal, and our subjects are paired because every participant sees both q_R and q_D. This scenario is usually tested with a McNemar's test [70] and a two-sided alternative hypothesis. The one-sided alternative hypothesis that the presence of deception reduces the risk of marking risky lines is a Binomial test [36] with the following test statistic:

$$t = \frac{\gamma}{\beta + \gamma} \sim B(\beta + \gamma, 1/2) \tag{3}$$

To make this more intuitive, we compute the relative risk that describes how much the risk that humans mark risky lines is reduced (or increased), when deceptive lines are present:

$$RR = \frac{\Pr(\text{Match in } q_D)}{\Pr(\text{Match in } q_R)} = \frac{\Pr(L'_R \cap A_{Ex} \neq \varnothing)}{\Pr(L_R \cap A_{Ex} \neq \varnothing)} = \frac{\beta + \delta}{\gamma + \delta}$$

4 Prototype Design

Honeyquest, which is our tool to run interactive questionnaires, uses the three **query labels** ☺ neutral, $\lightning$ risky, and $\searrow$ deceptive and our four **query types**, as shown in Table 4. We generate deceptive queries with HoneYAML files, our description language for CDTs.

4.1 Risky Queries

There are three types of risky queries, inspired by MITRE's three knowledge bases (CVE for vulnerabilities [67], CWE for weaknesses [68], and CAPEC for attack patterns [20]):

- **Vulnerability queries** contain at least one indicator in the query that points to a known vulnerability.
- **Weakness queries** display an insecure pattern that might lead to a vulnerability.
- **Attack queries** showcase a deliberate attempt to do harm, often by exploiting a vulnerability or weakness.

The risky query in Table 4 is an example for a vulnerability query by indicating that the server is running Apache 1.0.3, which is vulnerable to CVE-1999-0067. Participants are not expected to know this, but to be suspicious of the version text.

Weaknesses on the other hand may lead to vulnerabilities. Listing 2 shows an example that is indicative for a potential path traversal weakness (CWE-22) in a web application. In this example, it appears that a user can potentially control the "file" parameter to request arbitrary files from the remote server file system.

```
GET /view?file=../Overview.php
Host: github.io
User-Agent: curl/7.68.0
Accept: */*
```

Listing 2: A risky "HTTP response" query of risk type "weakness" which inspired us to derive CDT DS4 from that risk.

To obtain an attack query, we can change the query parameter in Listing 2 to something like file=../../etc/passwd and make it look like a concrete path traversal attack (CAPEC-126).

4.2 Deceptive Queries and HoneYAML

While dry-running deception experiments is valuable, ultimately, we want to deploy them into real systems. To build a bridge to future work we designed a specification for CDTs that we use to make queries deceptive and which also serves as a configuration for tools that can deploy them [73, 60]. We envision HoneYAML to become an enumeration of CDTs some day, much like we have an enumeration of Common Vulnerabilities and Exposures (CVEs) [67].

Listing 3 shows how to define a CDT that adds a deceptive HTTP header. Within Honeyquest, the implementation of the "decoy-apiserver" CDT (DP3) that we show here is simply inserting a new line in the query payload. The resulting query will be labeled as deceptive, regardless of its original type. Our open-source repository contains all HoneYAML specifications that we created. Real-world systems that add deceptive elements to the HTTP protocol often use reverse proxies to do so [54, 16, 21, 44, 87, 78]. The same HoneYAML specification can be used to first evaluate CDTs with Honeyquest and later configure proxies.

Table 4: One representative example for each of the four query types in Honeyquest.

⚡ **Risky HTTP response** [P] query (with vulnerability RP2)

```
HTTP/1.1 200 OK
Date: Tue, 02 May 2018 04:32:14 GMT
Server: Apache/1.0.3 (Debian)
Vary: Accept-Encoding
Content-Type: text/html
```

🦡 **Deceptive file system** [F] query (with technique DF3 injected)

```
drwxr-xr-x 25 elsa 4.0K Dec 30 08:36 .
drwxr-xr-x  4 root 4.0K Jun 21  2019 ..
-rw-------  1 elsa  57K Jan 13 14:48 .bash_history
drwx------  6 elsa 4.0K Sep 25 17:40 .config
-rw-r--r--  1 elsa  12K Feb  6  2022 keys.json
```

🦡 **Deceptive .htaccess file** [H] query (with CDT DH1 injected)

```
<IfModule mod_rewrite.c>
  RewriteEngine on
  Redirect 301 "/admin" \
    "/plugins/kul/panel?role=view"
</IfModule>
```

😐 **Neutral HTTP requests** [S] query

```
0.120 POST https://shop.com/rest/user/export 200 OK (0.4 kB)
0.215 GET https://shop.com/rest/image-captcha/ 200 OK (4.1 kB)
0.381 GET https://shop.com/rest/user/whoami 200 OK (0.1 kB)
2.031 GET https://shop.com/rest/history 200 OK (30 bytes)
2.876 GET https://shop.com/api/Quantitys/ 200 OK (0.6 kB)
```

Legend: Purple shades indicate ⚡ risky lines and orange shades indicate 🦡 deceptive lines.

[P] HTTP response queries show HTTP response headers, but always without any payload.
[F] File system queries show the output of the UNIX command `ls -lah`, which lists all files in the current working directory and their metadata.
[H] .htaccess filequeries show the configuration directives in an .htaccess file, which is used to configure Apache web servers.
[S] HTTP request queries show requests made by a web application: Seconds since load, method, URL, response status code, and response size, unless empty.

```
kind: httpheader
name: decoy-apiserver
description: Header that points to an API endpoint
operations:
  - op: add
    key: X-Kube-ApiServer
    value: /hko/api
```

Listing 3: A HoneYAML specification for CDT DP3.

Because it is smart to imitate risks in deceptive queries, we might generate deceptive queries that look similar to risky ones. This is fine, as labels only say something about design strategies anyway.

4.3 Honeyquest

Honeyquest is a web-based application. Queries are read from a pre-computed query store that we have prepared from different sources (§5.1). New users experience the following:

(1) We ask for consent to collect anonymized data.
(2) We show them eight tutorial queries to teach them about queries, labels, and marks (Appendix E.7).
(3) We collect profile information (§5.2 and Appendix E.4).
(4) We sample one random query after another until we run out of queries. Queries are never shown twice to the same user. We also made sure that the first 100 queries included an equal number of neutral queries, deceptive queries with all possible CDTs, and risky queries with all possible risks.
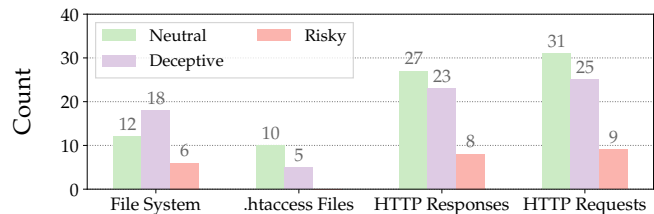
5 Experiment Design

This section summarizes the queries that we tested, the CDTs and risks we injected, and how we recruited participants.

We pre-tested the tutorial queries (Appendix E.7) with two colleagues who did not participate in the actual experiment. All of our 19 risks and 13 of our 25 CDTs were indirectly pre-tested because they had been used successfully in previous work. Nevertheless, we consulted domain experts who did not participate in the actual experiment to pre-test all the risks and CDTs that we developed.

5.1 Query Design

Our dataset consists of a total of 174 queries (Figure 3). We first collected 80 neutral queries. We then browsed through well-known vulnerabilities and weaknesses in web applications and manually derived 23 risky queries. Many risky queries were built by taking a neutral one and adding indicators of risks. A total of 71 deceptive queries were generated with the method explained in §4.2.

Figure 3: Distribution of neutral, deceptive, and risky labels.

Since we wanted to test many interesting risks and CDTs, we did not aim for a perfectly balanced dataset. However, this is not a problem because we do not evaluate metrics such as accuracy that would be sensitive to imbalanced datasets.

5.1.1 Design of Neutral Queries. Every single query in our open-source dataset carries a reference to its original source. Most were collected from the following real-world environments:

- **12 file system** payloads capture the output of the `ls -lah` command in the home directories of a few servers, containers, and personal computers in our lab. Sensitive content was manually anonymized or removed.
- **10 .htaccess files** were randomly picked by searching for ".htaccess" in open-source projects with Sourcegraph.[5]
- **27 HTTP responses** were randomly sampled from the "500K HTTP Headers" dataset [53] that crawled the HTTP

[5] https://sourcegraph.com/search

responses from the 500,000 most-visited websites in 2014, as ranked by the now discontinued company Alexa Internet.

- **31 sets of HTTP requests** were gathered by manually using the websites of popular web services and recording all HTTP requests that happened. We recorded traces for the websites of Amazon, Dropbox, Dynatrace, GitHub, Gmail, Google, Jira, the OWASP Juice Shop [94], TikTok, Wikipedia, and YouTube. Sensitive fields, names, and identifiers were manually anonymized or removed.

5.1.2 Design of Deceptive Queries. We manually picked and designed 25 CDTs: 12 were self-defined and 13 have been mentioned or evaluated in previous work. Table 9 lists all of them.

5.1.3 Design of Risky Queries. Most of our 19 risks are designed to resemble the categories of the OWASP Top 10 [96] and OWASP API Security Top 10 [95]. Some were inspired by OWASP ZAP security scanner rules [97]. Table 6 explains each risk with an example.

5.2 User Study Details and Ethics

We carefully reviewed our experiment to conform to ethical standards, protect the privacy of all participants, and follow best practices in user research [89]. Our institution has no IRB, so we instead conducted an ethics self-assessment [34] and obtained approval from our legal and privacy counsel. More details of the user study and ethical considerations are described in Appendix E.

Participants were recruited by posting messages to Slack channels of security professionals and to a Mattermost server of a local CTF team. 77 volunteers responded to that message and started the experiment. We had to discard all answers from 30 of them because they answered fewer than 8 warm-up queries (consisting of two pre-selected queries for each of the four query types; same for each participant), which left us with **47 participants**: 12 CTF players, where most of them are graduate students, and 35 security professionals, where most of them build enterprise security products. The skills of this target audience are very similar to those of real attackers. Unlike typical user studies on cyber security [55, 11], our study only had 11% students. Demographics, consent collection, timeline, and the preceding tutorial queries are described in Appendix E.

During the experiment, we collected self-reported profile information, and recorded how long it took users to answer queries. No personally-identifying information was collected. Participants were informed about the purpose of the experiment, about the presence of neutral, deceptive, and risky queries, and about their option to stop answering queries at any time, without negative consequences and without giving reasons. Participants were allowed to continue where they left off by visiting our web application again. In a second run of the experiment with 22 security professionals, participants could win a 50€ Amazon gift card in a lottery, if they answered at least 50% of the queries. All others received no incentives to participate. Security professionals were allowed to do this during their work time, CTF players participated in their free time without any compensation. Participants did not receive a performance report and where never informed if their answer marks were "correct".

Participants could comment on any query during the experiment and those that were freely disclosing their identity in the comments were invited to discuss them with us (§7.4).

6 Results

Our 47 participants answered 3,669 queries in total (Figure 4). All of them answered at least 8 queries. The median answer time per query was 19 seconds. Participants needed 45 - 60 minutes on average to answer all 174 queries.
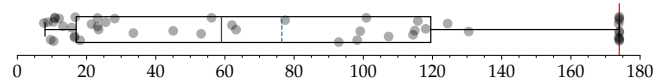

Figure 4: Boxplot on the number of answered queries per participant. Total Queries = 174. Mean = 76. Median = 59.

The subscript on the following percentage numbers is the 95% CI of the mean, calculated using Wilson's method [105].

- **Aspect A.** (Table 5, 9, 6)   Participants fell for 🪤 traps in $37_{\pm 2.4}\%$ of their answers. They recognized traps in $15_{\pm 1.8}\%$ of their answers. In $9_{\pm 1.4}\%$ of answers to 😐 neutral queries, they mistakenly classified something as a trap. Participants correctly identified ⚡ risks in $44_{\pm 4.5}\%$ of their answers and mistakenly classified something as a risk in $44_{\pm 2.5}\%$ of their answers to 😐 neutral queries.
- **Aspect B1.** (Table 9)   Deceptive lines were marked first in only 36% of answers. We cannot reject the null from §3.2.2, i.e., participants did not prefer CDTs over other elements.
- **Aspect B2.** (Table 8)   The presence of deception reduced the risk of marking a weakness or vulnerability by 22% on average. We tested this on a small set of 5 CDTs: The null hypothesis from §3.2.3 can be rejected ($\alpha = 0.05$) for all techniques combined ($p = 0.0013$), and for techniques DP3 and DP1 alone. We cannot reject the null for the other CDTs.

Table 5: Results on the enticingness of neutral queries.

	k	n	Mark Distribution
All Neutral Queries	80	1647	41% / 44%
File System	12	302	53% / 28%
.htaccess Files	10	195	31% / 63%
HTTP Responses	27	559	43% / 42%
HTTP Requests	31	591	37% / 47%

Description: Bars show the % of answers n to neutral queries with □ n_{Ex}/n ↗ exploit marks, □ n_{Tr}/n 🪤 trap marks, ■ $n_{\wedge}/n$ ↗ exploit and 🪤 trap marks, □ $n_{\varnothing}/n$ no marks at all. k are the number of neutral queries. n are the number of answers (not marks) to them. Bars without percentage numbers account for less than 15%. The tiny bars denote the 95% CI of that mean.

7 Discussion

This section discusses new insights about enticing and defensive deception, compares our results to previous work, and points out possible improvements for future experiments.

7.1 Aspect A: Enticing Deception

To discuss enticingness, we will begin to blur the distinction between traps and risks in this section. Ultimately, we do not want attackers to be able to distinguish between them, but rather want to learn how they react to certain query elements.

Table 6: Results on the enticingness of risks.

⚡ Risk	r	Mark Distribution
All Risky Queries	461	44% · 23% · 25%
File System	166	51% · 31%
RF1 `private-key.pem`	27	67% · 26%
RF2 `backup.tar.gz`	57	61% · 23%
RF3 `salphard.ovpn`	27	41% · 48%
RF4 `k8s-manifests` (directory)	28	39% · 43%
RF5 `ddns-update-key`	27	33% · 15% · 44%
HTTP Responses	155	54% · 15% · 24%
RP1 `Proxy-Auth.: Basic ...`	26	69% · 19%
RP2 `Server: Apache/1.0.3`	50	62% · 20% · 18%
RP3 `X-Powered-By: PHP/5.1.6`	39	59% · 21%
RP4 HTTP Request Smuggling[a]	13	54% · 15% · 23%
RP5 `Referer: https://...`	27	15% · 33% · 45%
HTTP Requests	140	26% · 20% · 49%
RS1 Brk. Fun.-Lvl. Auth.: Unauthenticated user requests privileged data	12	83%
RS2 Password Hash Parameter, e.g., `?user=maltier&hash=...`	12	75%
RS3 Mass Assignment (as in DS10)	11	45% · 46%
RS4 Brk. Obj.-Lvl. Auth.: User can request sensitive data from other user	12	25% · 17% · 58%
RS5 `/log?msg=...` Endpoint	27	19% · 26% · 51%
RS6 `/api.dev` Endpoint	12	17% · 17% · 50% · 16%
RS7 Huge Payload Sizes	12	92%
RS8 Mixing HTTP with HTTPS	30	33% · 57%
RS9 NoSQL Injection	12	17% · 83%

Aspect A: Bars show the % of answers r that ▥ r_{Ex}/r match ➤ exploit marks (= "risk detected"), ▥ r_{Tr}/r match 🪤 trap marks (= "risk mistaken for trap"), ▥ $r_\triangle/r$ placed marks elsewhere, ▢ $r_\emptyset/r$ had no marks at all. r are the number of answers (not marks) to queries with this risk. Bars without percentage numbers account for less than 15%. The tiny bars denote the 95% CI of that mean.
Risk Presence: In our dataset, every risk was present in exactly one query, with the exception of RF2 (2x), RP3 (2x), and RP2 (3x). This explains the relatively higher number of answers r on those three risks.

[a] HTTP request smuggling exploits how web servers handle HTTP requests, such that they initiate illegitimate requests.

Enticing deception should be neither too obvious nor too camouflaged. Our participants were most tempted to exploit authentication (passwords, tokens, hashes, cookies) and configuration elements. But, obvious elements like a "passwords.txt" file (DF5) or parameters with clear-text passwords (DS2), while still being comparably enticing, were often recognized as traps (all ≥ 36%). On the other end, harder-to-find traps or risks like a logging endpoint or mass assignment weaknesses (DS11, DS10) were rarely discovered, neither as something to exploit (all ≤ 14%) nor as a trap (all ≤ 3%). Sahin et al. also observed that more complex risks were tried less often in their CTF experiment [88].

In our file system queries, we saw that filenames containing the terms "backup", "config", "ovpn", or "k8s-manifests" received less 🪤 trap marks than more obvious terms like "key" or "password". This makes us believe that CDTs should be neither too obvious nor too camouflaged. Han et al. also speculated that the placement of CDTs should be neither too sparse nor too aggressive [54].

Imitating true risks is a promising method for designing deceptive elements. Our participants placed significantly more ➤ exploit marks on ⚡ true risks than on 🪤 traps ($44_{\pm4.5}$% vs. $37_{\pm2.4}$%; χ^2-test, $p = 0.0076$, $n = 2022$). This is reasonable, since true risks should be more enticing than traps, but, it shows that the best traps may need to only imitate true risks. This strengthens the idea proposed by Araujo et al. [16] on "honeypatching" true vulnerabilities such that they are technically fixed but still respond as if they were vulnerable when attacked.

Specifically, in our HTTP response and HTTP request queries, true risks like outdated Apache or PHP versions (RP2, RP3) and password hashes in a parameter (RS2) were more often exploited (all ≥ 59%) than deceptive tokens or cookies (DP1, DP2) in header fields (all ≥ 40%). Please note that findings for RP3 and RP2 might include a bias since we showcased similar risks in the tutorial, but without disclosing whether they are risky or deceptive. Also, the path traversal trap (DS4) was part of the tutorial but was surprisingly rarely identified as a trap ($14_{\pm6.1}$%) in the actual experiment.

Letting participants place marks on individual lines proves valuable for inventing new CDTs. Instead of only imitating known risks that received many ➤ exploit marks, we can also devise new traps by looking at what marks individual lines received (Table 10, 11). For example, filenames ".ssh", ".bash_history", and "data.csv" received 260, 151, and 32 exploit marks, respectively. All of them had ≤ 7 trap marks, the ".bash_history" even had zero. HTTP headers where version strings leaked Apache modules, e.g. "mod_ssl/2.2.17", received 32 exploit marks and only 6 trap marks.

Gaining such insights from otherwise neutral queries is possible because Honeyquest lets participants place marks on individual lines. We believe that this allows for more fine-grained analysis without increasing participants' cognitive load. Previous work often presented participants with pairs of questions (one knowingly genuine, one knowingly deceptive), and asked them to find the deceptive one [86, 78, 23, 83, 82]. Sahin et al. [86] let participants choose between placing genuine and deceptive marks on HTTP parameter names. It should be noted that Honeyquest's 🪤 trap marks are most similar to their deceptive marks, but Honeyquest distinguishes adversaries' intentions further by letting participants place either ➤ exploit marks or no marks at all. In Honeyquest, not placing any mark could be interpreted as placing a genuine mark.

Multiple iterations of Honeyquest can inform the design of more enticing CDTs. Our experiment is just the beginning of a feedback cycle that can inform the design of future, more enticing CDTs. Table 7 shows a possible ranking that would reward "enticing traps" and punish "ineffective traps". We see this as an early attempt to rank enticement, useful for continuous experiments with humans or for training autonomous agents.

Table 7: Reward matrix to maximize CDT enticement.

Qry.	No Marks	🪤 Trap Marks	➤ Exploit Marks
☺		◀	▶
⚡	◀	◀	▶
🪤	◀	◀	▶

Legend: Arrow direction and strength indicate our subjective judgment on how to rank enticement. ◀ = Less enticing. ▶ = More enticing.

7.2 Aspect B: Defensive Deception

Cyber deception is seen as an active form of cyber defense [56, 77, 111, 86]. Our results support and enrich this claim.

Some CDTs significantly reduce the risk of true weaknesses being exploited. Fascinatingly, we see that our participants were in fact distracted by the presence of deception (Table 8). Adding "X-ApiServer" and "X-DevToken" headers (DP3, DP1) reduced the risk that participants marked the true vulnerability by 32% and 27%, respectively. The true vulnerabilities that were missed due to the presence of the deceptive headers were vulnerable versions of Apache and PHP (RP2, RP3). In all cases where we have obtained enough statistical power, we can measure a significant reduction in risk. Our findings demonstrate that cyber deception can be an active form of cyber defense, reducing the risk of exploitation of true system weaknesses. We see more tests on hypothesis like this one as a promising direction for future work.

When participants fell for traps, the trap was not the first thing they marked — at least not in Honeyquest. Surprisingly, traps were clearly not what participants marked first for exploitation (Binomial test, $p \ll 0.001$, $n = 313$). We suspect that comparably enticing neutral query elements or the exact location of a CDT in the query (e.g., participants possibly read queries from top to bottom) might explain this finding. Thus, we do not recommend drawing conclusions from our participants' order of actions. Instead, we believe that CTF experiments are more appropriate for studying this aspect. However, to our knowledge, no CTF experiment has investigated this question yet. Also, little can be said if participants showed a preference to mark risks first, whenever they placed their ✗ exploit marks. Risks were marked first in 49% of cases ($n = 136$), which is not significantly different from random. Investigating which factors influence participants' mark preferences may be an interesting direction for future work.

7.3 Replication of Prior Findings

Compared to existing work, namely [83, 82, 75, 87], our results seem consistent. In all cases, we can enrich previous results. Appendix D details how we aligned our results to previous work.

What rarely received marks in Honeyquest, was also rarely exploited in a real-world CTF game. In most CTF games [54, 87, 88], all exploitable elements had to be discovered by CTF players, while ours were clearly presented to participants, which makes a comparison unfair. Han et al. [54] primarily evaluated placement strategies rather than specific techniques, thus, we refrain from making a direct comparison to their results.

However, we can still partially align our results with previous work. The SunDEW experiment [87] compared the elements that participants considered deceptive in a questionnaire with the elements with which the CTF players interacted ("considered deceptive" ratio in parentheses): "username" (53%) and "role" (61%) cookies, and deceptive GET parameters (7%). The ranking remained the same in our experiment. $15_{\pm 10.0}\%$ thought our cookie was deceptive (DP2) and $0_{\pm 12.1}\%$ thought the IDOR trap was deceptive (DS1). In a different CTF game by Sahin et al. [88] developers rarely tried to modify the "Content-Type" header field (between 5% and 13% of players). This is consistent with our results, where these headers were only marked 3 times (0.05% of all marks).

Table 8: Resulting contingency table on defensive deception.

| exploited before | ✗ | ✗ | ✓ | ✓ | | | |
| exploited after | ✗ | ✓ | ✗ | ✓ | | | |
🐦 CDT	α	β	γ	δ	▼ p-val.	▲ pwr.	▼ RR
All Techniques	**120**	**9**	**28**	**59**	**0.0013**	**0.9090**	**-22%**
DP3 API Server	19	1	9	16	0.0107	0.8415	-32%
DP1 Dev. Token	18	1	8	18	0.0195	0.7999	-27%
DP4 Proxy Referer	17	3	7	20	0.1719	0.3497	-15%
DS4 Path Traversal	43	2	3	4	~~0.5000~~	~~0.1038~~	-14%
DS3 SESSID Param.	23	2	1	1	~~0.8750~~	~~0.0598~~	+50%

Legend: Struck out tests violate the rule of thumb that all of the expected values are greater than 5, making the test less informative [107].

Our most enticing filenames were also most enticing in a real-world honeypot. Nikiforakis et al. [75] placed six files on public file hosting services and recorded how often attackers attempted to downloaded them. We showed three of them to our participants (reported download attempts in parentheses): "card3rz_reg_details.html" (22%), "customer_list_2010.html" (9%), and "SPAM_list.pdf" (5%). The ranking remained the same in our experiment, with $51_{\pm 14.3}\%$ (DF4), $43_{\pm 12.9}\%$ (DF6), and $17_{\pm 9.6}\%$ (DF8) of answers exploiting these names, respectively.

Our participants were also good at recognizing obvious traps. Rowe et al. [83, 82] showed participants a genuine and a deceptive file listing. Participants had to chose the one that "most likely came from a real system". 53.6% chose the real one. In our experiment, $26_{\pm 12.1}\%$ of answers marked traps in the deceptive query (DF9) and $11_{\pm 9.2}\%$ marked traps in the neutral one. This reinforces the hypothesis that humans show a tendency to detect traps for these specific queries.

Petrunić [77] proposed adding an "Admin=false" GET parameter to network requests (DS5), but did not report measures on attacker enticement. Our results showed that many answers wanted to exploit that ($37_{\pm 13.0}\%$). But, this trap might be too obvious with $22_{\pm 11.4}\%$ of answers also seeing a trap in it.

Our participants were also subjectively aggressive when placing marks. The work from Ferguson-Walter et al. provided experimental evidence that adversaries who know about the presence of deception tend to act more aggressively than unaware adversaries [40]. In the past, the opposite was believed to be true, i.e., that deception is only effective when it is well hidden and attackers are unaware of it [43, 83]. Results by Sahin et al. [87] also suggest that informing attackers about deceptive measures deters them, which ultimately benefits defenders. Our experiment did not intend to provide evidence for or against this conjecture. What can be said is that our participants placed an average of 1.8 marks per answer. But, only 3.86% of all query lines were risky and only 3.62% were deceptive. Some participants told us afterwards that they "thought [that] every single query has a lot of traps in it" and that they "better mark too much than too little". This fits in with a challenge mentioned later in §7.4, which is that some participants also felt the urge to answer queries correctly. In a similar survey [86], participants mislabeled at least 10% of genuine parameters as deceptive. This seems consistent with our results, where $9_{\pm 1.4}\%$ of answers to neutral queries saw traps in them.

Table 9: Results on the enticingness of Cyber Deception Techniques (CDTs).

🔍 CDT	Representative Description	k	d	Mark Distribution	d_B	d'_B/d_B
All Deceptive Queries		71	1561	37% 15% 23% 25%	313	36%
File System	*CDTs add files with deceptive names to the file system*	18	443	42% 21% 21% 16%	139	39%
DF1 private-key.pem		2	32	66% 22%	17	35%
DF2 backup.tar.gz		2	38	61% 24%	19	42%
DF3 keys.json		3	86	52% 26%	37	51%
DF4 card3rz_reg_details.html [75]		2	43	51% 19% 19%	15	47%
DF5 passwords.txt		2	41	46% 49%	12	25%
DF6 customer_list_2010.html [75]		2	53	43% 23% 25%	17	29%
DF7 config.ini		2	45	38% 40% 20%	15	33%
DF8 SPAM_list.pdf [75]		2	58	17% 21% 45% 17%	7	14%
DF9 Rowe et al. [83, 82]	e.g., examples, gif_files, idlold, wizard, ...	1	47	15% 19% 57%	0	–
.htaccess Files	*CDTs add directives that seem to leak sensitive paths*	5	128	54% 21%	14	36%
DH1 Admin Redirect	Redirect 301 "/admin" line leaks sensitive path	5	128	54% 21%	14	36%
HTTP Responses	*CDTs add headers with deceptive tokens, cookies, paths*	23	456	36% 30% 22%	103	24%
DP1 Developer Token	⚡ X-DevToken header has JWT token with a secret key	7	137	55% 20% 15%	46	33%
DP2 Cookie [54, 87, 88]	Set-Cookie header with admin=false in Base64	2	48	40% 15% 29% 16%	13	38%
DP3 API Server	⚡ X-ApiServer: /hko/api header leaks API path	7	134	30% 30% 30%	21	5%
DP4 Proxy Referer	⚡ X-Proxy-Referer header exposes a fake server's path	7	137	22% 47% 24%	23	17%
HTTP Requests	*CDTs add parameters or requests that imitate true risks*	25	534	30% 22% 37%	57	53%
DS1 IDOR Secrets [87]	Extra requests to a few /secrets/123 paths[a]	1	12	58% 42%	2	–
DS2 Clear-Text Pass. [88]	Add ?user=john&pass=carrot13 to /login request	1	28	54% 36%	12	67%
DS3 SESSID Param. [54]	⚡ Add ?SESSID=odq... query parameter	2	58	40% 16% 26% 18%	10	70%
DS4 Path Traversal [88]	⚡ Add ?file=../dist/Aq.svg query parameter	5	122	39% 20% 27%	10	40%
DS5 Admin Param. [54, 77]	Add ?admin=false query parameter	2	49	37% 22% 27%	6	50%
DS6 Unescaped JS [88]	Add GET parameter with raw JS	2	21	33% 57%	0	–
DS7 System Param. [54, 88]	Add ?system=prod query parameter	1	28	29% 25% 42%	4	–
DS8 Developer Endpoint	Add /api.dev requests	4	80	24% 26% 42%	6	33%
DS9 Unespaced JSON [88]	Add GET parameter with raw JSON	2	22	23% 32% 40%	3	–
DS10 Mass Assignment	Extra requests that set fields with GET parameters[b]	2	44	30% 54%	2	–
DS11 Log Endpoint	Add /log?msg=abc requests	3	70	31% 59%	2	–

Aspect A: (§3.2.1) The best CDTs are the ones that participants often fall for (higher ▦ d_{Ex}/d ratio is better) and rarely avoid (lower ▦ d_{Tr}/d ratio is better). Bars show the % of answers d that ▦ d_{Ex}/d match ➤ exploit marks (= "fallen for trap"), ▦ d_{Tr}/d match 🪤 trap marks (= "trap detected"), ▦ $d_\triangle/d$ placed marks elsewhere, ▢ $d_\varnothing/d$ had no marks at all. k are the number of queries in the dataset that had the associated CTD injected. d are the number of answers (not marks) that these queries received. Bars without percentage numbers account for less than 15%. The tiny bars denote the 95% CI of that mean.

Aspect B1: (§3.2.2) In the d_B answers where participants placed multiple ➤ exploit marks, d'_B is the number of times that a deceptive line was marked before a non-deceptive one. If there are at least $d_B \geq 5$ such cases, we perform a Binomial test on the null hypothesis that this ratio is random, i.e., $d'_B/d_B = 1/2$, with the one-sided alternative that deceptive lines were marked first more often than random. No test was significant (all $p \geq 0.1719$) with $\alpha = 0.05$.

Aspect B2: (§3.2.3) The ⚡ lightning bolt symbol indicates that this CDT was also evaluated on Aspect B2, but in Table 8.

[a] Insecure direct object references (IDOR) are vulnerabilities where potentially sensitive content can be retrieved by guessing (predictable) identifiers (IDs).

[b] Applications may automatically bind HTTP parameter names to fields in the underlying object, potentially enabling attackers to manipulate restricted fields.

Honeyquest is a useful tool for repeating cyber deception experiments in a sample-efficient and cost-effective way, while still yielding qualitatively similar results to CTF experiments. Compared to real-world deployments of CDTs, Honeyquest is more cost-effective since it does not require the time and effort to set up and maintain CTF experiments or honeypots. Experiments are also more sample-efficient than others because participants can answer a query within seconds instead of spending time on coding actual attacks. The decision-making processes of attackers and humans in general also include fast – but, not necessarily correct – heuristics [51, 61], which probably explains why participants can respond quickly to most queries.

7.4 Challenges and Limitations

While our results are insightful and seem consistent with prior research, we also faced several design challenges that we would like to share with future researchers.

Designing cyber deception experiments that imitate real-world scenarios presents many challenges. Some participants questioned whether our results from Honeyquest generalize to the real world since "participants will always behave differently in surveys". We argue that the tutorial (Appendix E.7), the four query types that accurately represent the real-world "views" of an attacker, and the participants' ability to place both ➤ exploit and 🪤 trap marks are a reasonable approximation of a real-world scenario.

Table 10: Top 40 individual query lines, ranked by the total number of ⤴ exploit marks that the line received.

ID	Type	Query Line	⤴ exploit n_{Ex}	🪤 trap n_{Tr}
🪝 DP1	P	X-DevToken: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbiI6IjMwZDk4NGI4Iiwicm9sZSI6ImNsdXN0ZXIifQ. J2c1XH2RsXDjaWJhCHSWR4bBNxEm7l2Q7WxrlG2vph4	76	27
	F	drwx------ 2 lpc lpc 4.0K Nov 2 09:21 .ssh	58	1
⚡ RP2	P	Server: Apache/1.0.3 (Debian)	55	1
	F	drwx------ 4 leonard leonard 4.0K Jan 14 2022 .gnupg	48	1
	F	drwx------. 2 root root 4.0K Sep 27 19:30 .ssh	46	2
⚡ RP3	P	X-Powered-By: PHP/5.1.6	41	9
	P	Set-Cookie: PHPSESSID=hLAGcA9qClz36kOr71sSgw; path=/	41	10
	F	drwx------ 2 furi0zuc furi0zuc 4.0K Jun 21 2019 .ssh	40	1
	F	-rw------- 1 lpc lpc 40K Dec 29 15:58 .bash_history	39	0
🪝 DP3	P	X-Kube-ApiServer: /hko/api	36	13
	P	X-AspNet-Version: 4.0.30319	34	4
	P	Server: Microsoft-IIS/7.5	33	0
	P	Server: Apache/2.2.17 (Unix) mod_ssl/2.2.17 OpenSSL/0.9.8e-fips-rhel5 mod_auth_passthrough/2.1 mod_bwlimited/1.4 FrontPage/5.0.2.2635	31	6
	F	-rw-------. 1 root root 21K Oct 25 19:26 .bash_history	30	0
	S	6.588 GET https://juice-shop.herokuapp.com/rest/admin/application-configuration 200 OK (7.2 kB)	29	7
	P	Transfer-Encoding: chunked	28	0
🪝 DF3	F	-rw-r--r-- 1 elsa elsa 12K Feb 6 2022 keys.json	28	13
	P	Server: nginx/1.2.4	27	0
🪝 DH1	H	Redirect 301 "/admin" "/plugins/kul/pages/admin/index.php?role=view"	26	11
	P	Server: Microsoft-IIS/6.0	25	0
	F	drwx------ 2 cathy cathy 4.0K Dec 7 14:04 .ssh	25	0
	F	-rw------- 1 IconThor IconThor 231K Dec 30 08:36 .zsh_history	25	1
	F	-rwxr--r-- 1 lpc lpc 919 Dec 28 14:31 query.sh	25	1
	P	X-Powered-By: PHP/5.3.18	25	2
	P	X-Powered-By: PHP/5.2.16	23	5
	F	drwx------ 2 donald donald 4.0K May 16 2022 .ssh	22	0
	P	Set-Cookie: ASP.NET_SessionId=fyi3sylqfunbwtdy03s4fdqv; path=/; HttpOnly	22	2
	P	X-Powered-By: PHP/5.4.5	22	2
	F	-rw-r--r-- 1 lpc lpc 8.1K Nov 24 10:18 data.csv	22	3
	S	9.912 GET https://juice-shop.herokuapp.com/rest/admin/application-configuration 200 OK (7.2 kB)	22	3
	H	RewriteCond %{THE_REQUEST} ^[A-Z]{3,}\s([^.]+)\.php [NC]	22	3
	F	-rw------- 1 elsa elsa 231K Dec 30 08:36 .zsh_history	21	0
	P	Server: nginx/1.6.0	21	0
	S	2.460 POST https://api.site.com/_private/browser/stats	21	3
	F	-rw-r--r-- 1 IconThor IconThor 88 Jul 11 2021 test.gpg	21	8
	F	-rw------- 1 furi0zuc furi0zuc 8.9K Jul 5 21:47 .bash_history	20	0
	F	-rw-r--r-- 1 leonard leonard 64 Jun 16 2019 .gitconfig	20	1
	P	Set-Cookie: bbsessionhash=f3cdd7987b326584c6ee6696f3033087; path=/; HttpOnly	20	5
	S	3.127 POST https://api.site.com/_private/browser/stats	19	2
	S	9.022 GET https://juice-shop.herokuapp.com/rest/products/search?q= 200 OK (3.5 kB)	19	3

Legend: If the line was annotated as 🪝 deceptive or ⚡ risky, we reference the associated CDT or risk identifier, respectively. The query type from which the line originated is abbreviated with **F** = File system, **H** = .htaccess file, **P** = HTTP response, and **S** = HTTP request. Full results can be found at https://github.com/dynatrace-oss/honeyquest.

Table 11: Top 15 individual query lines, ranked by the total number of 🪤 trap marks that the line received.

ID	Type	Query Line	⤴ exploit n_{Ex}	🪤 trap n_{Tr}
	P	X-Geek: What's black and white and red all over? Please don't kill our penguin-powered server.	6	27
🪝 DP1	P	X-DevToken: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbiI6IjMwZDk4NGI4Iiwicm9sZSI6ImNsdXN0ZXIifQ. J2c1XH2RsXDjaWJhCHSWR4bBNxEm7l2Q7WxrlG2vph4	76	27
	P	X-Bacon: I wonder what penguin bacon tastes like.	7	25
	F	-rw-r--r-- 1 lpc lpc 0 Oct 20 14:15 .sudo_as_admin_successful	14	22
🪝 DF5	F	-rw-r--r-- 1 leonard leonard 9.7K Jun 6 2022 passwords.txt	10	17
	P	X: 23	14	14
🪝 DF3	F	-rw-r--r-- 1 elsa elsa 12K Feb 6 2022 keys.json	28	13
🪝 DP3	P	X-Kube-ApiServer: /hko/api	36	13
🪝 DH1	H	Redirect 301 "/admin" "/plugins/kul/pages/admin/index.php?role=view"	26	11
🪝 DS5	S	7.640 GET https://juice-shop.herokuapp.com/rest/user/whoami?admin=false 200 OK (0.1 kB)	14	10
🪝 DS2	S	11.162 POST https://juice-shop.herokuapp.com/rest/user/login?user=johnson&password=carrot13 200 OK (0.8 kB)	15	10
	P	Set-Cookie: PHPSESSID=hLAGcA9qClz36kOr71sSgw; path=/	41	10
🪝 DF8	F	-rw-r--r-- 1 leonard leonard 43K Nov 20 2022 SPAM_list.pdf	5	9
🪝 DF6	F	-rw-r--r-- 1 leonard leonard 83K Nov 20 2022 customer_list_2010.html	12	9
⚡ RP3	P	X-Powered-By: PHP/5.1.6	41	9

Legend: If the line was annotated as 🪝 deceptive or ⚡ risky, we reference the associated CDT or risk identifier, respectively. The query type from which the line originated is abbreviated with **F** = File system, **H** = .htaccess file, **P** = HTTP response, and **S** = HTTP request. Full results can be found at https://github.com/dynatrace-oss/honeyquest.

Also, priming participants to imagine that they would encounter queries during reconnaissance activities provided sufficient context for most participants. Ultimately, it is impossible to tell whether attackers would behave in the same way in the real world. Although providing evidence on this point is beyond the scope of our work, §7.3 suggests that our results align with previous studies that have examined similar, real-world situations.

Future research on how best to represent context in cyber deception experiments would be valuable. One participant in our study wanted to know where in a software infrastructure (which server, container, etc.) they should imagine encountering certain file system entries. However, the same participant noted that specific files such as ".bash_history", ".ssh" or "config" are "always interesting" and hard to resist, regardless of the context. This begs the question of which CDTs have a similar (context-free) appeal?

The ultimate quality of a CDT is still influenced by attacker's and defender's ability to learn from each other. Regardless of whether defensive deception is modeled as a static or a dynamic game [81, 76], players can adapt their strategies over time and learn to improve. Honeyquest only provides snapshots of the enticingness of CDTs for one round of such a game. Repeated experiments, variations of CDTs, identification of contextual factors and skill levels are necessary to account for these dynamics.

The trade-off between template-like queries for controllable experiments and the need for diverse, neutral queries. Three participants noted that queries often looked similar, encouraging them to remember the differences between them, which is not what we want to measure. Ideally, template-like queries are preferred, providing control over specific elements to isolate the effect of a CDT. Nevertheless, a greater variety of pre-tested neutral queries are essential for future experiments. Our open-source dataset can serve as a starting point to build such a collection.

Participants enjoyed that Honeyquest felt like a game, but also felt an urge to answer queries correctly. Although participants knew that we did not score accuracy, four of them reported that they still felt the urge to "get it right" and "avoid making mistakes", which led them to spend a lot of time on each query. This phenomenon was reported by participants of all skill levels. A time limit within which a query must be answered might be beneficial in future experiments to prevent this behavior.

Overall, participants enjoyed the "different" and "game-like" experience of Honeyquest. We believe that gamification aspects, as also explored in studies on cyber security education [65, 64], are a promising avenue to explore further in studies on cyber deception.

The tutorial and our risky elements proved beneficial as an integrated skill check. Results of surveys like ours can easily be distorted by responses from incompetent participants; hence, we chose two populations with proven expertise to address this. However, this is more difficult to control in anonymous populations [11]. Honeyquest can mitigate such problems by removing answers from participants who barely recognized risks or who failed the tutorial.

Some CDTs and true risks can be difficult to distinguish from each other. The tutorial (Listing 11) taught participants that a final distinction may only be possible by knowing their actual implementation. We wonder what properties of CDTs can be represented by questionnaires, and what can only be represented by CTF experiments or honeypots.

8 Related Work

8.1 Honeypots and Honeytokens

Spitzner was among the first to introduce honeypots as a measure against insider threats [92, 93]. He describes honeypots as "an information system resource whose value lies in unauthorized or illicit use of that resource" [92]. Most of the honeypot software that has been researched in the last decades [73, 42] focuses on emulating protocols, processes, machines, or entire networks [79]. But, the term "information system resource" is broad enough to also cover honeytokens, which are no computers but rather digital entities. Their most common forms are: Honeytokens [93] and Canarytokens[6], honeyfiles, -pages, and -urls [109, 101, 22, 62, 77], honeypatches [16, 17, 15] (silently-patched vulnerabilities that still seem exploitable at the surface), honeywords [59] (can be decrypted with wrong keys and still yield plausible yet incorrect data), and honeypots, and -ports [79], e.g., classic SSH honeypots.

8.2 Taxonomies and Classifications

Numerous taxonomies and classifications of deception techniques have been introduced, adapted, and surveyed in the past decades [55, 111, 43]. Whaley [104] proposed one of the first military-focused theories on (non-cyber) deception back in 1982, which still influenced cyber deception taxonomies in 2004, as introduced by Rowe and Rothstein [84, 85]. Yuill et al. [108] described processes, principles, and techniques to hide things from adversaries. Later work by Mokube and Adams [72] in 2007 and Almeshekah and Spafford [12] in 2014 focused more closely on the technical aspects and human biases of cyber deception. Recent work like the one by Zhang and Thing [111] in 2021 aligned taxonomies closer to the cyber kill chain model and illustrated proposals on deception lifecycles.

Many surveys on cyber deception have been conducted [55, 43, 35, 63, 26, 71, 80, 58], specifically, ones with a focus on honeypot software [73], on securing web applications [33], on application layer deception [60], on IoT honeypots [42], or, on approaches using game theory and machine learning [112, 76].

Closely related, Han et al. [55] examined how the efficiency and effectiveness of a wide range of CDTs have been evaluated in the past (§8.3). Zhu et al. [112] summarized how approaches that use game theory and machine learning have been evaluated.

The OWASP AppSensor project [103], despite only partially addressing deception, serves as a hallmark for how to describe and taxonomize runtime application defense techniques and also inspired us to propose HoneYAML.

8.3 Evaluating Cyber Deception

Han et al. proposed four aspects for evaluating CDTs: [55]

(1) **Deception placement strategies** have been evaluated by simulation [48, 10], or by studies with students [101, 22].
(2) **Plausibility and realism of deception**, i.e., measuring how well deceptive assets are discernible from genuine assets. Zhu et al. [112] structured these evaluation testbeds into real testbeds, and ones based on probability models, simulation models [47, 99, 90, 106, 74], and emulation models [1–3].

---

[6]https://canarytokens.org

(3) **Effectiveness of deception**, i.e., measuring if it achieves its desired functionality. While "desired functionality" is open to interpretation, such experiments are generally conducted in either confined or natural environments.

(4) **Accuracy and false-positive rate (FPR) of deception.** While there are a few studies [22, 54] that tried to evaluate this, it is often impossible in most contexts because the base-rate of attacks [18] cannot be measured reliably.

Fraunholz et al. [43] summarized 14 studies that evaluated CDTs in natural environments with field studies. The rest of this section covers studies in confined environments. References in parentheses name the CDT that we replicated in our query design (Table 9).

The work from Sahin et al. [86] is closest to ours. They also evaluate the enticingness of CDTs (§7.1), but not their ability to be defensive (§7.2). They automatically generate realistic HTTP parameters for web application layer deception, and evaluated it with a survey where developers were given the link to an actual Swagger UI of a web application. The Swagger UI showed the available API endpoints, but without the possibility to interact with the application. Their "automatically generated parameters names were as realistic as manually selected ones". Similar work from the same authors provides an extensive list of CDTs for web applications, which they evaluated with a CTF challenge and questionnaires [87].

The HackIT tool [5, 6, 4] by Aggarwal et al. is conceptually similar to Honeyquest. It enables researchers to map real-world cyberattack scenarios into game-like environments. Unlike our queries, their scenarios require manual design. However, this makes experiments more flexible. Chadha et al. proposed the related Cyber-VAN tool [27, 7–9] which allows for a speedy and flexible setup of network-based deception experiments. Game-like environments are also used for cyber security education, e.g., with PenQuest [65, 64].

Rowe et al. [83, 82] (DF9) conducted an experiment where 14 humans were shown pairs of "real" and "fake" file listings.

Ferguson-Walter et al. [41] carried out a large study on network deception with 130 professional red teamers in a two-day exercise and Shade et al. [91] focused on host-based deception with 59 computer specialists. Unlike our work, they focused on honeypots. They are also one of the few authors who divided participants into four cohorts, based on knowledge about deception (informed, uninformed) and presence of deception (present, not present) [40].

Nikiforakis et al. [75] (DF4, DF6, DF8) demonstrated that attackers are actively searching for sensitive files on public file hosting services. They placed honeyfiles on them and recorded downloads from 80 unique IP addresses within one month.

Petrunić [77] (DS5) suggested adding a seemingly deceptive "Admin=false" GET parameter to URLs, which would presumably only ever be changed by an attacker.

Han et al. [54] (DP2, DS3, DS5, DS7) held a CTF game with 258 participants on a CMS system, where a transparent reverse proxy injected deceptive elements. They primarily evaluated placement strategies rather than specific techniques.

Sahin et al. [87] (DP2, DS1) used questionnaires and a CTF game (98 players) to evaluate their deception framework SunDEW. Other work by Sahin et al. [88] (DP2, DS2, DS4, DS6, DS7, DS9) used questionnaires (21 participants), and a CTF game (82 players) to evaluate developers' familiarity with web attack and defense mechanisms.

8.4 Effective Cyber Deception

In 1999, Tirenin and Faatz [98] were one of the first authors to suggest that deception must be dynamic in order to be effective, i.e., "it must present a continually-changing situational picture to the enemy" [98]. Cohen [28] motivated the need for a link between social sciences and technological development. In 2010, Bowen et al. [25] proposed a "Decoy Turing Test" that tasks humans to discern real from decoy network traffic.

Bercovitch et al. [23] developed HoneyGen in 2011 to generate honeytokens by mining characteristics from real data. Recent research has shifted towards creating dynamic and personalized CDTs [49], e.g., by profiling attacker behavior [74].

Many works that followed examined psychological aspects and decision-making processes of attackers [39, 41, 40, 37, 38, 57, 31, 30, 29, 52, 50, 46, 49, 22]. Ferguson-Walter [39, 40] showed that cyber deception affects an attacker's cognitive and emotional state, and that CDTs are effective even if attackers are aware of their use or merely believe it may be in use. Gonzalez et al. [49] found that attackers exhibit irrational behavior that leads to cognitive biases. Similarly, Gabrys et al. [46] observed a strong correlation between the emotional state of an attacker (confusion, self-doubt, confidence, frustration, and surprise) and the frequency of their reconnaissance activity.

9 Future Work

Future work may include enriching Honeyquest with more CTDs and a way to evaluate deception placement strategies [101, 22], teaching ML models to design CDTs [14, 19], mining our open-source dataset for interesting patterns [88, 74, 11], incorporating cognitive models into the experiment design [50], embedding educational aspects into Honeyquest [65, 64], implementing more query types, e.g., "robots.txt" files [45, 44, 54], evaluating CDTs that secure non-web applications, adapting Honeyquest to deceive vulnerability scanners [13], and, of course, replicating our results in more experiments and real-world deployments.

10 Conclusion

This work proposes a method to measure the enticingness of CDTs. We demonstrate its feasibility for four aspects of a web application, where we designed 25 CDTs and 19 risks, for an experiment with a high-quality sample of 47 humans (12 CTF players, 35 professionals). Our results provide a detailed overview of the enticingness of CDTs (Table 9) and show that deception can reduce the risk of finding a true risk by about 22% on average. Knowing such statistics, e.g., that humans fall for traps about 37% of the time, enables researchers to back up their theoretical models with our empirical numbers. Notably, we were able to replicate the goals of previous work with many consistent findings (§7.3), but without a time-consuming implementation on real computer systems. This strengthens the generalizability of our method to the real world.

Acknowledgments

References

[1] Stefan Achleitner, Thomas F. La Porta, Patrick McDaniel, Shridatt Sugrim, Srikanth V. Krishnamurthy, and Ritu Chadha. 2016. Cyber Deception: Virtual Networks to Defend Insider Reconnaissance. In *Proceedings of the 8th ACM CCS International Workshop on Managing Insider Security Threats (MIST '16)*. Association for Computing Machinery, Vienna, Austria, 57–68. https://doi.org/10.1145/2995959.2995962

[2] Stefan Achleitner, Thomas F. La Porta, Patrick McDaniel, Shridatt Sugrim, Srikanth V. Krishnamurthy, and Ritu Chadha. 2017. Deceiving Network Reconnaissance Using SDN-Based Virtual Topologies. *IEEE Transactions on Network and Service Management* 14, 4 (Dec. 2017), 1098–1112. https://doi.org/10.1109/tnsm.2017.2724239

[3] Jaime C. Acosta, Anjon Basak, Christopher Kiekintveld, Nandi Leslie, and Charles Kamhoua. 2020. Cybersecurity Deception Experimentation System. In *2020 IEEE Secure Development (SecDev '20)*. IEEE, Atlanta, GA, USA, 34–40. https://doi.org/10.1109/secdev45635.2020.00022

[4] Palvi Aggarwal, Yinuo Du, Kuldeep Singh, and Cleotilde Gonzalez. 2021. Decoys in Cybersecurity: An Exploratory Study to Test the Effectiveness of 2-Sided Deception. In *Proceedings of the 1st International Workshop on Adaptive Cyber Defense (IJCAI-ACD '21, arXiv:2108.11037)*. arXiv, Montreal, Canada, 1–11. https://doi.org/10.48550/arxiv.2108.11037 arXiv:2108.11037 [cs]

[5] Palvi Aggarwal, Aksh Gautam, Vaibhav Agarwal, Cleotilde Gonzalez, and Varun Dutt. 2019. HackIT: A Human-in-the-Loop Simulation Tool for Realistic Cyber Deception Experiments. In *Advances in Human Factors in Cybersecurity (AHFE '19)*. Springer International Publishing, Cham, 109–121. https://doi.org/10.1007/978-3-030-20488-4_11

[6] Palvi Aggarwal, Cleotilde Gonzalez, and Varun Dutt. 2020. HackIT: A Real-Time Simulation Tool for Studying Real-World Cyberattacks in the Laboratory. In *Handbook of Computer Networks and Cyber Security: Principles and Paradigms*. Springer International Publishing, Cham, 949–959. https://doi.org/10.1007/978-3-030-22277-2_39

[7] Palvi Aggarwal, Shahin Jabbari, Omkar Thakoor, Edward A. Cranford, Phebe Vayanos, Christian Lebiere, Milind Tambe, and Cleotilde Gonzalez. 2022. Human-Subject Experiments on Risk-Based Cyber Camouflage Games. In *Cyber Deception: Techniques, Strategies, and Human Aspects*. Springer International Publishing, Cham, 25–40. https://doi.org/10.1007/978-3-031-16613-6_2

[8] Palvi Aggarwal, Omkar Thakoor, Shahin Jabbari, Edward A. Cranford, Christian Lebiere, Milind Tambe, and Cleotilde Gonzalez. 2022. Designing Effective Masking Strategies for Cyberdefense Through Human Experimentation and Cognitive Models. *Computers & Security* 117 (June 2022), 102671. https://doi.org/10.1016/j.cose.2022.102671

[9] Palvi Aggarwal, Omkar Thakoor, Aditya Mate, Milind Tambe, Edward A. Cranford, Christian Lebiere, and Cleotilde Gonzalez. 2020. An Exploratory Study of a Masking Strategy of Cyberdeception Using CyberVAN. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 64, 1 (Dec. 2020), 446–450. https://doi.org/10.1177/1071181320641100

[10] Ron Alford and Andy Applebaum. 2021. Towards Causal Models for Adversary Distractions. In *Proceedings of the 2021 SIAM AI/ML for Cybersecurity Workshop (AI4CS-SDM '21, arXiv:2104.10575)*. arXiv, Online, 1–6. https://doi.org/10.48550/arxiv.2104.10575 arXiv:2104.10575 [cs]

[11] Asmaa Aljohani and James Jones. 2022. The Pitfalls of Evaluating Cyber Defense Techniques by an Anonymous Population. In *HCI for Cybersecurity, Privacy and Trust (HCII '22)*. Springer International Publishing, Virtual Event, 307–325. https://doi.org/10.1007/978-3-031-05563-8_20

[12] Mohammed H. Almeshekah and Eugene H. Spafford. 2014. Planning and Integrating Deception into Computer Security Defenses. In *Proceedings of the 2014 New Security Paradigms Workshop (NSPW '14)*. Association for Computing Machinery, Victoria, British Columbia, Canada, 127–138. https://doi.org/10.1145/2683467.2683482

[13] Tillmann Angeli, Daniel Reti, Daniel Schneider, and Hans D. Schotten. 2024. False Flavor Honeypot: Deceiving Vulnerability Scanning Tools. In *2024 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW '24)*. IEEE, Vienna, Austria, 399–406. https://doi.org/10.1109/EuroSPW61312.2024.00051

[14] Frederico Araujo, Gbadebo Ayoade, Khaled Al-Naami, Yang Gao, Kevin W. Hamlen, and Latifur Khan. 2019. Improving Intrusion Detectors by Crooksourcing. In *Proceedings of the 35th Annual Computer Security Applications Conference (ACSAC '19)*. Association for Computing Machinery, San Juan, Puerto Rico, USA, 245–256. https://doi.org/10.1145/3359789.3359822

[15] Frederico Araujo and Kevin W. Hamlen. 2016. Embedded Honeypotting. In *Cyber Deception: Building the Scientific Foundation*. Springer International Publishing, Cham, 201–231. https://doi.org/10.1007/978-3-319-32699-3_9

[16] Frederico Araujo, Kevin W. Hamlen, Sebastian Biedermann, and Stefan Katzenbeisser. 2014. From Patches to Honey-Patches: Lightweight Attacker Misdirection, Deception, and Disinformation. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS '14)*. Association for Computing Machinery, Scottsdale, Arizona, USA, 942–953. https://doi.org/10.1145/2660267.2660329

[17] Frederico Araujo, Mohammad Shapouri, Sonakshi Pandey, and Kevin Hamlen. 2015. Experiences with Honey-Patching in Active Cyber Security Education. In *Proceedings of the 8th USENIX Conference on Cyber Security Experimentation and Test (CSET '15)*. USENIX Association, Washington, DC, USA, 1–7. https://www.usenix.org/conference/cset15/workshop-program/presentation/araujo

[18] Stefan Axelsson. 2000. The Base-Rate Fallacy and the Difficulty of Intrusion Detection. *ACM Transactions on Information and System Security* 3, 3 (Aug. 2000), 186–205. https://doi.org/10.1145/357830.357849

[19] Gbadebo Ayoade, Frederico Araujo, Khaled Al-Naami, Ahmad M. Mustafa, Yang Gao, Kevin W. Hamlen, and Latifur Khan. 2020. Automating Cyberdeception Evaluation with Deep Learning. In *Proceedings of the 53rd Hawaii International Conference on System Sciences (HICSS '20)*. ScholarSpace, Maui, Hawaii, 1–10. https://doi.org/10.24251/hicss.2020.236

[20] Sean Barnum and Amit Sethi. 2007. *Attack Patterns as a Knowledge Resource for Building Secure Software*. Technical Report. Cigital, Inc., Washington DC, USA. 1–31 pages. https://api.semanticscholar.org/CorpusID:18455387

[21] Timothy Barron, Johnny So, and Nick Nikiforakis. 2021. Click This, Not That: Extending Web Authentication with Deception. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security (ASIA CCS '21)*. Association for Computing Machinery, Virtual Event, Hong Kong, 462–474. https://doi.org/10.1145/3433210.3453088

[22] Malek Ben Salem and Salvatore J. Stolfo. 2011. Decoy Document Deployment for Effective Masquerade Attack Detection. In *Proceedings of the 8th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA '11)*. Springer, Amsterdam, The Netherlands, 35–54. https://doi.org/10.1007/978-3-642-22424-9_3

[23] Maya Bercovitch, Meir Renford, Lior Hasson, Asaf Shabtai, Lior Rokach, and Yuval Elovici. 2011. HoneyGen: An Automated Honeytokens Generator. In *Proceedings of 2011 IEEE International Conference on Intelligence and Security Informatics (ISI '11)*. IEEE, Beijing, China, 131–136. https://doi.org/10.1109/isi.2011.5984063

[24] Brian M. Bowen, Shlomo Hershkop, Angelos D. Keromytis, and Salvatore J. Stolfo. 2009. Baiting Inside Attackers Using Decoy Documents. In *Security and Privacy in Communication Networks 2009 (SecureComm '09)*. Springer, Berlin, Heidelberg, 51–70. https://doi.org/10.1007/978-3-642-05284-2_4

[25] Brian M. Bowen, Vasileios P. Kemerlis, Pratap Prabhu, Angelos D. Keromytis, and Salvatore J. Stolfo. 2010. Automating the Injection of Believable Decoys to Detect Snooping. In *Proceedings of the Third ACM Conference on Wireless Network Security (WiSec '10)*. Association for Computing Machinery, Hoboken, New Jersey, USA, 81–86. https://doi.org/10.1145/1741866.1741880

[26] Matthew L. Bringer, Christopher A. Chelmecki, and Hiroshi Fujinoki. 2012. A Survey: Recent Advances and Future Trends in Honeypot Research. *International Journal of Computer Network and Information Security* 4, 10 (Sept. 2012), 63–75. https://doi.org/10.5815/ijcnis.2012.10.07

[27] Ritu Chadha, Thomas Bowen, Cho-Yu J. Chiang, Yitzchak M. Gottlieb, Alex Poylisher, Angello Sapello, Constantin Serban, Shridatt Sugrim, Gary Walther, Lisa M. Marvel, E. Allison Newcomb, and Jonathan Santos. 2016. CyberVAN: A Cyber Security Virtual Assured Network Testbed. In *2016 IEEE Military Communications Conference (MILCOM '16)*. IEEE, Baltimore, MD, USA, 1125–1130. https://doi.org/10.1109/milcom.2016.7795481

[28] Fred Cohen. 2006. The Use of Deception Techniques: Honeypots and Decoys. *Handbook of Information Security* 3, 1 (2006), 646–655. http://all.net/journal/deception/Deception_Techniques_.pdf

[29] Edward A. Cranford, Cleotilde Gonzalez, Palvi Aggarwal, Sarah Cooney, Milind Tambe, and Christian Lebiere. 2020. Adaptive Cyber Deception: Cognitively Informed Signaling for Cyber Defense. In *Proceedings of the 53rd Hawaii International Conference on System Sciences (HICSS '20)*. ScholarSpace, Maui, Hawaii, 1–10. https://doi.org/10.24251/hicss.2020.232

[30] Edward A. Cranford, Cleotilde Gonzalez, Palvi Aggarwal, Milind Tambe, Sarah Cooney, and Christian Lebiere. 2021. Towards a Cognitive Theory of Cyber Deception. *Cognitive Science* 45, 7 (2021), e13013. https://doi.org/10.1111/cogs.13013

[31] Edward A. Cranford, Christian Lebiere, Cleotilde Gonzalez, Sarah Cooney, Phebe Vayanos, and Milind Tambe. 2018-07-25/2018-07-28. Learning About Cyber Deception Through Simulations: Predictions of Human Decision Making With Deceptive Signals in Stackelberg Security Games. In *Proceedings of the 40th Annual Meeting of the Cognitive Science Society (CogSci '18)*. Curran Associates, Inc., Madison, WI, USA, 256–261. https://mindmodeling.org/cogsci2018/papers/0067/index.html

[32] Sundar Dorai-Raj. 2022. Binomial Confidence Intervals for Several Parameterizations. https://cran.r-project.org/web/packages/binom/binom.pdf

[33] A. I. Mohd Efendi, Z. Ibrahim, M. N. Ahmad Zawawi, F. Abdul Rahim, N. A. Mohamad Pahri, and Anuar Ismail. 2019. A Survey on Deception Techniques for Securing Web Application. In *2019 IEEE 5th International Conference on Big Data Security on Cloud, High Performance and Smart Computing and Intelligent Data and Security (BigDataSecurity & HPSC & IDS '19)*. IEEE, Washington, DC, USA, 328–331. https://doi.org/10.1109/bigdatasecurity-hpsc-ids.2019.00066

[34] European Commission. 2021. EU Grants: How To Complete Your Ethics Self-Assessment. https://ec.europa.eu/info/funding-tenders/opportunities/docs/2021-2027/common/guidance/how-to-complete-your-ethics-self-assessment_en.pdf

[35] Wenjun Fan, Zhihui Du, David Fernández, and Víctor A. Villagrá. 2018. Enabling an Anatomic View to Investigate Honeypot Systems: A Survey. *IEEE Systems Journal* 12, 4 (Dec. 2018), 3906–3919. https://doi.org/10.1109/jsyst.2017.2762161

[36] Michael P. Fay. 2014. *Exact McNemar's Test and Matching Confidence Intervals.* Technical Report. MRAN. https://mran.microsoft.com/snapshot/2015-02-24/web/packages/exact2x2/vignettes/exactMcNemar.pdf

[37] Kimberly J. Ferguson-Walter. 2020. *An Empirical Assessment of the Effectiveness of Deception for Cyber Defense.* Ph. D. Dissertation. University of Massachusetts Amherst, Amherst, Massachusetts. https://doi.org/10.7275/z0rb-ek46

[38] Kimberly J. Ferguson-Walter, Maxine Major, Dirk Van Bruggen, Sunny Fugate, and Robert Gutzwiller. 2019. The World (of CTF) Is Not Enough Data: Lessons Learned from a Cyber Deception Experiment. In *2019 IEEE 5th International Conference on Collaboration and Internet Computing (CIC '19).* IEEE, Los Angeles, CA, USA, 346–353. https://doi.org/10.1109/cic48465.2019.00048

[39] Kimberly J. Ferguson-Walter, Maxine M. Major, Chelsea K. Johnson, Craig J. Johnson, Dakota D. Scott, Robert S. Gutzwiller, and Temmie Shade. 2023. Cyber Expert Feedback: Experiences, Expectations, and Opinions about Cyber Deception. *Computers & Security* 130 (July 2023), 103268. https://doi.org/10.1016/j.cose.2023.103268

[40] Kimberly J. Ferguson-Walter, Maxine M. Major, Chelsea K. Johnson, and Daniel H. Muhleman. 2021. Examining the Efficacy of Decoy-based and Psychological Cyber Deception. In *Proceedings of the 30th USENIX Security Symposium (USENIX Security '21).* USENIX Association, Online, 1127–1144. https://www.usenix.org/conference/usenixsecurity21/presentation/ferguson-walter

[41] Kimberly J. Ferguson-Walter, Temmie Shade, Andrew Rogers, Elizabeth Niedbala, Michael Trumbo, Kevin Nauer, Kristin Divis, Aaron Jones, Angela Combs, and Robert Abbott. 2019. The Tularosa Study: An Experimental Design and Implementation to Quantify the Effectiveness of Cyber Deception. In *Proceedings of the 52nd Hawaii International Conference on System Sciences (HICSS '19).* ScholarSpace, Maui, Hawaii, 1–10. https://doi.org/10.24251/hicss.2019.874

[42] Javier Franco, Ahmet Aris, Berk Canberk, and A. Selcuk Uluagac. 2021. A Survey of Honeypots and Honeynets for Internet of Things, Industrial Internet of Things, and Cyber-Physical Systems. *IEEE Communications Surveys & Tutorials* 23, 4 (2021), 2351–2383. https://doi.org/10.1109/comst.2021.3106669

[43] Daniel Fraunholz, Simon Duque Anton, Christoph Lipps, Daniel Reti, Daniel Krohmer, Frederic Pohl, Matthias Tammen, and Hans Dieter Schotten. 2018. Demystifying Deception Technology: A Survey. https://doi.org/10.48550/arXiv.1804.06196 arXiv:1804.06196 [cs]

[44] Daniel Fraunholz, Daniel Reti, Simon Duque Anton, and Hans Dieter Schotten. 2018. Cloxy: A Context-aware Deception-as-a-Service Reverse Proxy for Web Services. In *Proceedings of the 5th ACM Workshop on Moving Target Defense (MTD '18).* Association for Computing Machinery, Toronto, Canada, 40–47. https://doi.org/10.1145/3268966.3268973

[45] Daniel Fraunholz and Hans D. Schotten. 2018. Defending Web Servers with Feints, Distraction and Obfuscation. In *2018 International Conference on Computing, Networking and Communications (ICNC '18).* IEEE, Maui, HI, USA, 21–25. https://doi.org/10.1109/iccnc.2018.8390365

[46] Ryan Gabrys, Anu Venkatesh, Daniel Silva, Mark Bilinski, Maxine Major, Justin Mauger, Daniel Muhleman, and Kimberly J. Ferguson-Walter. 2023. Emotional State Classification and Related Behaviors Among Cyber Attackers. In *Proceedings of the 56th Hawaii International Conference on System Sciences (HICSS '23).* ScholarSpace, Maui, Hawaii, 1–10. https://doi.org/10.24251/hicss.2023.106

[47] Nandan Garg and Daniel Grosu. 2007. Deception in Honeynets: A Game-Theoretic Analysis. In *2007 IEEE SMC Information Assurance and Security Workshop (IAW '07).* IEEE, West Point, NY, USA, 107–113. https://doi.org/10.1109/iaw.2007.381921

[48] Dimitris Gavrilis, Ioannis Chatzis, and Evangelos Dermatas. 2007. Flash Crowd Detection Using Decoy Hyperlinks. In *2007 IEEE International Conference on Networking, Sensing and Control (ICNSC '07).* IEEE, London, UK, 466–470. https://doi.org/10.1109/icnsc.2007.372823

[49] Cleotilde Gonzalez, Palvi Aggarwal, Christian Lebiere, and Edward A. Cranford. 2020. Design of Dynamic and Personalized Deception: A Research Framework and New Insights. In *Proceedings of the 53rd Hawaii International Conference on System Sciences (HICSS '20).* ScholarSpace, Maui, Hawaii, 1–10. https://doi.org/10.24251/hicss.2020.226

[50] Robert Gutzwiller, Kimberly J. Ferguson-Walter, Sunny Fugate, and Andrew Rogers. 2018. "Oh, Look, A Butterfly!" A Framework For Distracting Attackers To Improve Cyber Defense. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 62, 1 (Sept. 2018), 272–276. https://doi.org/10.1177/1541931218621063

[51] Robert S. Gutzwiller, Kimberly J. Ferguson-Walter, and Sunny J. Fugate. 2019. Are Cyber Attackers Thinking Fast and Slow? Exploratory Analysis Reveals Evidence of Decision-Making Biases in Red Teamers. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 63, 1 (Nov. 2019), 427–431. https://doi.org/10.1177/1071181319631096

[52] Robert S. Gutzwiller, Hansol Rheem, Kimberly J. Ferguson-Walter, Christina M. Lewis, Chelsea K. Johnson, and Maxine Major. 2024. Exploratory Analysis of Decision-Making Biases of Professional Red Teamers in a Cyber-Attack Dataset. *Journal of Cognitive Engineering and Decision Making* 18, 1 (March 2024), 37–51. https://doi.org/10.1177/15553434231217787

[53] Hacker Target Pty Ltd. 2014. 500K HTTP Headers. https://hackertarget.com/500k-http-headers/

[54] Xiao Han, Nizar Kheir, and Davide Balzarotti. 2017. Evaluation of Deception-Based Web Attacks Detection. In *Proceedings of the 2017 Workshop on Moving Target Defense (MTD '17).* Association for Computing Machinery, Dallas, Texas, USA, 65–73. https://doi.org/10.1145/3140549.3140555

[55] Xiao Han, Nizar Kheir, and Davide Balzarotti. 2018. Deception Techniques in Computer Security: A Research Perspective. *Comput. Surveys* 51, 4 (July 2018), 80:1–80:36. https://doi.org/10.1145/3214305

[56] Kristin E. Heckman, Michael J. Walsh, Frank J. Stech, Todd A. O'Boyle, Stephen R. DiCato, and Audra F. Herber. 2013. Active Cyber Defense with Denial and Deception: A Cyber-Wargame Experiment. *Computers & Security* 37 (Sept. 2013), 72–77. https://doi.org/10.1016/j.cose.2013.03.015

[57] Linan Huang, Shumeng Jia, Emily Balcetis, and Quanyan Zhu. 2022. ADVERT: An Adaptive and Data-Driven Attention Enhancement Mechanism for Phishing Prevention. *IEEE Transactions on Information Forensics and Security* 17 (2022), 2585–2597. https://doi.org/10.1109/tifs.2022.3189530

[58] Amir Javadpour, Forough Ja'fari, Tarik Taleb, Mohammad Shojafar, and Chafika Benzaïd. 2024. A Comprehensive Survey on Cyber Deception Techniques to Improve Honeypot Performance. *Computers & Security* 140 (March 2024), 103792. https://doi.org/10.1016/j.cose.2024.103792

[59] Ari Juels and Ronald L. Rivest. 2013. Honeywords: Making Password-Cracking Detectable. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security (CCS '13).* Association for Computing Machinery, Berlin, Germany, 145–160. https://doi.org/10.1145/2508859.2516671

[60] Mario Kahlhofer and Stefan Rass. 2024. Application Layer Cyber Deception without Developer Interaction. In *2024 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW '24).* IEEE, Vienna, Austria, 416–429. https://doi.org/10.1109/EuroSPW61312.2024.00053

[61] Daniel Kahneman. 2011. *Thinking, Fast and Slow.* Farrar, Straus and Giroux, New York.

[62] Martin Lazarov, Jeremiah Onaolapo, and Gianluca Stringhini. 2016. Honey Sheets: What Happens to Leaked Google Spreadsheets?. In *9th USENIX Workshop on Cyber Security Experimentation and Test (CSET '16).* University College London, Austin, TX, United States, 1–8. https://www.usenix.org/conference/cset16/workshop-program/presentation/lazarov

[63] Zhuo Lu, Cliff Wang, and Shangqing Zhao. 2020. Cyber Deception for Computer and Network Security: Survey and Challenges. https://doi.org/10.48550/arXiv.2007.14497 arXiv:2007.14497 [cs]

[64] Robert Luh, Sebastian Eresheim, Stefanie Größbacher, Thomas Petelin, Florian Mayr, Paul Tavolato, and Sebastian Schrittwieser. 2022. PenQuest Reloaded: A Digital Cyber Defense Game for Technical Education. In *2022 IEEE Global Engineering Education Conference (EDUCON '22).* IEEE, Tunis, Tunisia, 906–914. https://doi.org/10.1109/educon52537.2022.9766700

[65] Robert Luh, Marlies Temper, Simon Tjoa, Sebastian Schrittwieser, and Helge Janicke. 2020. PenQuest: A Gamified Attacker/Defender Meta Model for Cyber Security Assessment and Education. *Journal of Computer Virology and Hacking Techniques* 16, 1 (March 2020), 19–61. https://doi.org/10.1007/s11416-019-00342-x

[66] Mandiant. 2013. *APT1: Exposing One of China's Cyber Espionage Units.* Technical Report. Mandiant, Inc. https://www.fireeye.com/content/dam/fireeye-www/services/pdfs/mandiant-apt1-report.pdf

[67] David E. Mann and Steven M. Christey. 1999. Towards a Common Enumeration of Vulnerabilities. In *Final Report of the 2nd Workshop on Research with Security Vulnerability Databases (WVDB '99).* CERIAS, Purdue University, West Lafayette Indiana, 1–13. https://api.semanticscholar.org/CorpusID:250641205

[68] Robert A. Martin, Steven M. Christey, and J. Jarzombek. 2006. The Case for Common Flaw Enumeration. In *Proceedings of Workshop on Software Security Assurance Tools, Techniques, and Metrics (SSATTM '05).* National Institute of Standards and Technology, Gaithersburg, MD, United States, 1–7. https://api.semanticscholar.org/CorpusID:110186969

[69] Bill McCarty. 2004. *SELinux: NSA's Open Source Security Enhanced Linux.* O'Reilly Media, Inc., Sebastopol, CA. https://www.oreilly.com/library/view/selinux/0596007167/

[70] Quinn McNemar. 1947. Note on the Sampling Error of the Difference Between Correlated Proportions or Percentages. *Psychometrika* 12, 2 (June 1947), 153–157. https://doi.org/10.1007/bf02295996

[71] Pilla Vaishno Mohan, Shriniket Dixit, Amogh Gyaneshwar, Utkarsh Chadha, Kathiravan Srinivasan, and Jung Taek Seo. 2022. Leveraging Computational Intelligence Techniques for Defensive Deception: A Review, Recent Advances, Open Problems and Future Directions. *Sensors* 22, 6 (Jan. 2022), 2194. https://doi.org/10.3390/s22062194

[72] Iyatiti Mokube and Michele Adams. 2007. Honeypots: Concepts, Approaches, and Challenges. In *Proceedings of the 45th Annual Southeast Regional Conference (ACM-SE '07)*. Association for Computing Machinery, Winston-Salem, North Carolina, 321–326. https://doi.org/10.1145/1233341.1233399

[73] Marcin Nawrocki, Matthias Wählisch, Thomas C. Schmidt, Christian Keil, and Jochen Schönfelder. 2016. A Survey on Honeypot Software and Data Analysis. https://doi.org/10.48550/arXiv.1608.06249 arXiv:1608.06249 [cs]

[74] Amirreza Niakanlahiji, Jafar Haadi Jafarian, Bei-Tseng Chu, and Ehab Al-Shaer. 2020. HoneyBug: Personalized Cyber Deception for Web Applications. In *Proceedings of the 53rd Hawaii International Conference on System Sciences (HICSS '20)*. ScholarSpace, Maui, Hawaii, 1–10. https://doi.org/10.24251/hicss.2020.243

[75] Nick Nikiforakis, Marco Balduzzi, Steven Van Acker, Wouter Joosen, and Davide Balzarotti. 2011. Exposing the Lack of Privacy in File Hosting Services. In *Proceedings of the 4th USENIX Conference on Large-scale Exploits and Emergent Threats (LEET '11)*. USENIX Association, Boston, MA, United States, 1–8. https://www.usenix.org/conference/leet11/exposing-lack-privacy-file-hosting-services

[76] Jeffrey Pawlick, Edward Colbert, and Quanyan Zhu. 2019. A Game-theoretic Taxonomy and Survey of Defensive Deception for Cybersecurity and Privacy. *Comput. Surveys* 52, 4 (Aug. 2019), 82:1–82:28. https://doi.org/10.1145/3337772

[77] A.B. Robert Petrunić. 2015. Honeytokens as Active Defense. In *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO '15)*. IEEE, Opatija, Croatia, 1313–1317. https://doi.org/10.1109/mipro.2015.7160478

[78] Christoph Pohl, Alf Zugenmaier, Michael Meier, and Hans-Joachim Hof. 2015. B.Hive: A Zero Configuration Forms Honeypot for Productive Web Applications. In *ICT Systems Security and Privacy Protection (IFIP SEC '15)*. Springer International Publishing, Hamburg, Germany, 267–280. https://doi.org/10.1007/978-3-319-18467-8_18

[79] Niels Provos. 2004. A Virtual Honeypot Framework. In *Proceedings of the 13th USENIX Security Symposium (USENIX Security '04)*. USENIX Association, San Diego, CA, USA, 1–14. https://www.usenix.org/legacy/publications/library/proceedings/sec04/tech/provos.html

[80] Xingsheng Qin, Frank Jiang, Mingcan Cen, and Robin Doss. 2023. Hybrid Cyber Defense Strategies Using Honey-X: A Survey. *Computer Networks* 230 (July 2023), 109776. https://doi.org/10.1016/j.comnet.2023.109776

[81] Stefan Rass and Stefan Schauer. 2018. *Game Theory for Security and Risk Management: From Theory to Practice*. Springer International Publishing, Cham. https://doi.org/10.1007/978-3-319-75268-6

[82] Neil C. Rowe, E. John Custy, and Binh T. Duong. 2007. Defending Cyberspace with Fake Honeypots. *Journal of Computers* 2, 2 (April 2007), 25–36. https://doi.org/10.4304/jcp.2.2.25-36

[83] Neil C. Rowe, Binh T. Duong, and E. John Custy. 2006. Fake Honeypots: A Defensive Tactic for Cyberspace. In *Proceedings of the Annual 2006 IEEE SMC Information Assurance Workshop (IAW '06)*. IEEE, West Point, NY, USA, 223–230. https://doi.org/10.1109/iaw.2006.1652099

[84] Neil C. Rowe and Hy S. Rothstein. 2004. Two Taxonomies of Deception for Attacks on Information Systems. *Journal of Information Warfare* 3, 2 (2004), 27–39. jstor:26502783 https://www.jstor.org/stable/26502783

[85] Neil C. Rowe and Julian Rrushi. 2016. *Introduction to Cyberdeception*. Springer International Publishing, Cham. https://doi.org/10/d65q

[86] Merve Sahin, Cédric Hébert, and Rocio Cabrera Lozoya. 2022. An Approach to Generate Realistic HTTP Parameters for Application Layer Deception. In *Applied Cryptography and Network Security (ACNS '22)*. Springer International Publishing, Rome, Italy, 337–355. https://doi.org/10.1007/978-3-031-09234-3_17

[87] Merve Sahin, Cédric Hébert, and Anderson Santana De Oliveira. 2020. Lessons Learned from SunDEW: A Self Defense Environment for Web Applications. In *Proceedings 2020 Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb '20)*. Internet Society, San Diego, CA, USA, 1–12. https://doi.org/10.14722/madweb.2020.23005

[88] Merve Sahin, Tolga Ünlü, Cédric Hébert, Lynsay A. Shepherd, Natalie Coull, and Colin Mc Lean. 2022. Measuring Developers' Web Security Awareness from Attack and Defense Perspectives. In *2022 IEEE Security and Privacy Workshops (SPW '22)*. IEEE, San Francisco, CA, USA, 31–43. https://doi.org/10.1109/spw54247.2022.9833858

[89] Jeff Sauro and James R. Lewis. 2012. *Quantifying the User Experience: Practical Statistics for User Research*. Elsevier/Morgan Kaufmann, Amsterdam Waltham, MA. https://doi.org/10.1016/C2010-0-65192-3

[90] Aaron Schlenker, Omkar Thakoor, Haifeng Xu, Fei Fang, Milind Tambe, Long Tran-Thanh, Phebe Vayanos, and Yevgeniy Vorobeychik. 2018. Deceiving Cyber Adversaries: A Game Theoretic Approach. In *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '18)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 892–900. https://par.nsf.gov/biblio/10050303

[91] Temmie B. Shade, Andrew V. Rogers, Kimberly J. Ferguson-Walter, Sara Beth Elson, Daniel K. Fayette, and Kristin E. Heckman. 2020. The Moonraker Study: An Experimental Evaluation of Host-Based Deception. In *Proceedings of the 53rd Hawaii International Conference on System Sciences (HICSS '20)*. ScholarSpace, Maui, Hawaii, 1875–1884. https://doi.org/10.24251/hicss.2020.231

[92] Lance Spitzner. 2003. Honeypots: Catching the Insider Threat. In *Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC '03)*. IEEE, Las Vegas, NV, USA, 170–179. https://doi.org/10.1109/csac.2003.1254322

[93] Lance Spitzner. 2003. Honeytokens: The Other Honeypot. https://www.symantec.com/connect/articles/honeytokens-other-honeypot

[94] The OWASP Foundation Inc. 2014. OWASP Juice Shop. https://owasp.org/www-project-juice-shop/

[95] The OWASP Foundation Inc. 2019. OWASP API Top 10:2019. https://owasp.org/www-project-api-security/

[96] The OWASP Foundation Inc. 2021. OWASP Top 10:2021. https://owasp.org/Top10/

[97] The OWASP Foundation Inc. 2023. OWASP ZAP. https://www.zaproxy.org/

[98] Walt Tirenin and Don Faatz. 1999. A Concept for Strategic Cyber Defense. In *Proceedings of the 1999 IEEE Military Communications Conference Proceedings (MILCOM '99, Vol. 1)*. IEEE, Atlantic City, NJ, USA, 458–463. https://doi.org/10.1109/milcom.1999.822725

[99] A.J. Underbrink. 2016. Effective Cyber Deception. In *Cyber Deception: Building the Scientific Foundation*. Springer International Publishing, Cham, 115–147. https://doi.org/10.1007/978-3-319-32699-3_6

[100] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C. J. Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, and Paul van Mulbregt. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17, 3 (March 2020), 261–272. https://doi.org/10.1038/s41592-019-0686-2

[101] Jonathan Voris, Jill Jermyn, Nathaniel Boggs, and Salvatore Stolfo. 2015. Fox in the Trap: Thwarting Masqueraders via Automated Decoy Document Deployment. In *Proceedings of the Eighth European Workshop on System Security (EuroSec '15)*. Association for Computing Machinery, Bordeaux, France, 1–7. https://doi.org/10.1145/2751323.2751326

[102] Cliff Wang and Zhuo Lu. 2018. Cyber Deception: Overview and the Road Ahead. *IEEE Security & Privacy* 16, 2 (March 2018), 80–85. https://doi.org/10.1109/msp.2018.1870866

[103] Colin Watson, Dennis Groves, and John Melton. 2015. AppSensor Guide: Application-Specific Real Time Attack Detection & Response. https://web.archive.org/web/20240120084438/https://owasp.org/www-pdf-archive/Owasp-appsensor-guide-v2.pdf

[104] Barton Whaley. 1982. Toward a General Theory of Deception. *Journal of Strategic Studies* 5, 1 (March 1982), 178–192. https://doi.org/10.1080/01402398208437106

[105] Edwin B. Wilson. 1927. Probable Inference, the Law of Succession, and Statistical Inference. *J. Amer. Statist. Assoc.* 22, 158 (June 1927), 209–212. https://doi.org/10.1080/01621459.1927.10502953

[106] Hua Wu, Yu Gu, Guang Cheng, and Yuyang Zhou. 2020. Effectiveness Evaluation Method for Cyber Deception Based on Dynamic Bayesian Attack Graph. In *Proceedings of the 2020 3rd International Conference on Computer Science and Software Engineering (CSSE '20)*. Association for Computing Machinery, Beijing, China, 1–9. https://doi.org/10.1145/3403746.3403897

[107] Frank Yates. 1934. Contingency Tables Involving Small Numbers and the $X2$ Test. *Supplement to the Journal of the Royal Statistical Society* 1, 2 (1934), 217–235. https://doi.org/10.2307/2983604 jstor:2983604

[108] Jim Yuill, Dorothy Denning, and Fred Feer. 2006. Using Deception to Hide Things from Hackers: Processes, Principles, and Techniques. *Journal of Information Warfare* 5, 3 (2006), 26–40. jstor:26503456 https://www.jstor.org/stable/26503456

[109] Jim Yuill, Mike Zappe, Dorothy Denning, and Fred Feer. 2004. Honeyfiles: Deceptive Files for Intrusion Detection. In *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop (IAW '04)*. IEEE, West Point, NY, USA, 116–122. https://doi.org/10.1109/iaw.2004.1437806

[110] James Joseph Yuill. 2007. *Defensive Computer-Security Deception Operations: Processes, Principles and Techniques*. Ph. D. Dissertation. North Carolina State University, Raleigh, NC, USA. https://repository.lib.ncsu.edu/handle/1840.16/5648

[111] Li Zhang and Vrizlynn. L. L. Thing. 2021. Three Decades of Deception Techniques in Active Cyber Defense - Retrospect and Outlook. *Computers & Security* 106 (July 2021), 102288. https://doi.org/10.1016/j.cose.2021.102288

[112] Mu Zhu, Ahmed H. Anwar, Zelin Wan, Jin-Hee Cho, Charles A. Kamhoua, and Munindar P. Singh. 2021. A Survey of Defensive Deception: Approaches Using Game Theory and Machine Learning. *IEEE Communications Surveys & Tutorials* 23, 4 (2021), 2460–2493. https://doi.org/10.1109/comst.2021.3102874

A Expressing Results with Typical Confusion Matrices

☺ **Neutral queries.**   Consider that a user anwers a ☺ neutral query $q_N \in Q_N$. Neutral queries never have line annotations. If the user places no marks on q_N, we consider this as a "true negative" (TN) classification. If the user places some, we say this is a "false positive" (FP) classification.

🦆 **Deceptive and** ⚡ **risky queries.**   As defined in Equation (1), we say that "answer marks A match line annotations L" if they intersect each other. Given a deceptive query 🦆 $q_D \in Q_D$, where 🪱 trap marks A_{Tr} match deceptive lines L_D, we say this as a "true positive" (TP) classification. If they do not match, we count a "false negative" (FN) instead. Given a risky query ⚡ $q_R \in Q_R$, the same rule applies, but ➤ exploit marks A_{Ex} are matched against risky lines L_R instead. There are at least four ways to match one of the two kinds of answer marks against one of the two kinds of line annotations. If we want to assess if participants fell for a trap, we would adapt the previous formulation to match ➤ exploit marks A_{Ex} against deceptive line annotations L_D instead. In all cases, we can arrange the counts in a confusion matrix. Table 12 shows this for the aforementioned formulation. Then, we can derive metrics such as accuracy, precision, or recall. Table 13 shows the confusion matrices of our experiment, whose results we also presented in §6.

Table 12: Matching answer marks and line annotations.

$q \in Q_N$	TN	$A = \varnothing$	FP	$A \neq \varnothing$
$q \notin Q_N$	FN	$L \cap A = \varnothing$	TP	$L \cap A \neq \varnothing$

Table 13: Results on the enticingness of traps and risks.

Qry.	TN	FP	FN	TP	ACC	PPV	TPR	FPR
🦆	1403	136	1325	234	53%	63%	15%	9%
⚡	857	682	256	204	53%	23%	44%	44%

Description: Confusion matrix and metrics on how well users are enticed by traps and risks. The opposing class was always a neutral query.

B Alternative Matching of Answer Marks and Line Annotations

There are five mutually exclusive variations on how answer marks A can possibly intersect with (non-empty) line annotations $L \neq \varnothing$:

$$A = L \qquad \text{marked } L \text{ exactly} \qquad (A1)$$
$$A \neq \varnothing \wedge A \subset L \qquad \text{marked } \textit{only} \text{ some in } L \qquad (A2)$$
$$A \nsubseteq L \wedge L \cap A \neq \varnothing \qquad \text{marks } \textit{only} \text{ overlap with } L \qquad (A3)$$
$$A \neq \varnothing \wedge L \cap A = \varnothing \qquad \text{marked } \textit{only} \text{ lines not in } L \qquad (A4)$$
$$A = \varnothing \qquad \text{no marks, but non-empty } L \qquad (A5)$$

The criterion in §3.1.1 assumes that answer marks match line annotations when lines are marked exactly (A1) or partially (A2), while also allowing overlaps (A3) with other (not-annotated) lines. In all three cases, it is valid to imply that a user at least partially identified a risk or a trap.

C Details on Counting Answer Marks

Aspect A (§3.2.1) required a more concrete formalization of the matching criteria of §3.1.1. We grouped answers to our queries by query type, and computed the following counts:

- n_{Ex} *How often were neutral lines mistaken for traps?* Number of answers that *only* received ➤ exploit marks A_{Ex}.
- n_{Tr} *How often were neutral lines mistaken for risks?* Number of answers that *only* received 🪱 trap marks A_{Tr}.
- $n_\wedge$ *How often were neutral lines mistaken for risks and traps in the same answer?* Number of answers that received both ➤ exploit marks A_{Ex} and 🪱 trap marks A_{Tr}
- $n_\varnothing$ *How often have humans not reacted to neutral lines?* Number of answers to neutral queries without any marks.

- d_{Ex} *How often fell humans for traps?* Number of answers, where ➤ exploit marks A_{Ex} match deceptive lines L_D.
- d_{Tr} *How often were traps detected?* Number of answers, where 🪱 trap marks A_{Tr} match deceptive lines L_D.
- $d_\triangle$ *How often have humans reacted to other lines?* Number of answers with marks but no match on decept. lines L_D.
- $d_\varnothing$ *How often have humans not reacted to traps?* Number of answers to deceptive queries without any marks.

- r_{Ex} *How often were risks detected?* Number of answers, where ➤ exploit marks A_{Ex} match risky lines L_R.
- r_{Tr} *How often were risks mistaken for traps?* Number of answers, where 🪱 trap marks A_{Tr} match risky lines L_R.
- $r_\triangle$ *How often have humans reacted to other lines?* Number of answers with marks but no match on risky lines L_R.
- $r_\varnothing$ *How often have humans not reacted to risks?* Number of answers to risky queries without any marks.

D Aligning Prior Work to Our Cyber Deception Techniques

This section describes how we mapped our CDTs (Table 9) to techniques from prior work [88, 87, 54, 77, 75, 83, 82]. A comparison and discussion of the results can be found in §7.3.

Nikiforakis et al. [75] (DF4, DF6, DF8) Their experiment placed six files on public file hosting services. We randomly injected three of those, whose names seem most likely to represent a real weakness, in some of our file system queries: "SPAM_list.pdf" (DF8), "customer_list_2010.html" (DF6), "card3rz_reg_details.html" (DF4). We omitted the other names ("phished_paypal_details.html", "Paypal_account_gen.exe", "Sniffed_email1.doc") because these names sound more like they would only be found on an adversary's computer and not on a real server.

Petrunić [77] (DS5) We randomly appended the suggested "admin=false" parameter in our HTTP request queries.

Han et al. [54] (DP2, DS3, DS5, DS7) Their experiment primarily evaluated placement strategies rather than specific techniques. We mapped their mention of an "additional cookie" to our CDT that injects a deceptive cookie into HTTP headers (DP2). We assumed that their mention of "honey GET parameters" is similar to our three CDTs that inject parameters into URLs: "admin=false" (DS5), "SESSID=odq..." (DS3), and "system=prod" (DS7).

Rowe et al. [83, 82] (DF9) Their experiment showed humans pairs of "real" and "fake" file listings. We used the pair that they illustrated in the paper to create two file system queries: Listing 4 shows the ☺ "real" file listing and Listing 5 shows the 🐱 "fake" file listing (with every line except for "." and ".." annotated as deceptive).

```
drwxr-xr-x 2 gaitan gaitan 4.0K Nov 30 17:42 .
drwxr-xr-x 4 gaitan gaitan 4.0K Nov 30 17:42 ..
-rw-r--r-- 1 gaitan gaitan 1.4K Nov 30 17:42 specv
-rw-r--r-- 1 gaitan gaitan 3.2K Nov 30 17:42 other
-rw-r--r-- 1 gaitan gaitan 1.1K Nov 30 17:42 rp
-rw-r--r-- 1 gaitan gaitan 6.7K Nov 30 17:42 gilmore
-rw-r--r-- 1 gaitan gaitan 1.2K Nov 30 17:42 int
-rw-r--r-- 1 gaitan gaitan 5.2K Nov 30 17:42 trash
-rw-r--r-- 1 gaitan gaitan 2.0K Nov 30 17:42 old.imsl
-rw-r--r-- 1 gaitan gaitan 2.7K Nov 30 17:42 flynn
```

Listing 4: Representation of "real" file listing [83, 82].

```
drwxr-xr-x 2 cooper cooper 4.0K Nov 30 17:42 .
drwxr-xr-x 4 cooper cooper 4.0K Nov 30 17:42 ..
-rw-r--r-- 1 cooper cooper 1.2K Nov 30 17:42 examples
-rw-r--r-- 1 cooper cooper 2.0K Nov 30 17:42 gif_files
-rw-r--r-- 1 cooper cooper 3.2K Nov 30 17:42 idlold
-rw-r--r-- 1 cooper cooper 5.2K Nov 30 17:42 wizard
-rw-r--r-- 1 cooper cooper 2.7K Nov 30 17:42 114564-01
-rw-r--r-- 1 cooper cooper 1.4K Nov 30 17:42 template
-rw-r--r-- 1 cooper cooper 1.1K Nov 30 17:42 target_n_horiz
-rw-r--r-- 1 cooper cooper 6.7K Nov 30 17:42 ass2
```

Listing 5: Representation of "fake" file listing [83, 82].

Sahin et al. [87] (DP2, DS1) Their experiment tested seven CDTs in a CTF experiment. They had a "Username" and "Role" cookie with similar detection rates that we mapped to our CDT that injects a deceptive cookie into HTTP headers (DP2). They also had a deceptive GET parameter on a "/view_patient/$id" endpoint that we mapped to our CDT with a "/secrets/$id" endpoint (DS1).

Sahin et al. [88] (DP2, DS2, DS4, DS6, DS7, DS9) Their experiment recorded 17 attack vectors that participants tried in their CTF experiment. We designed CDTs for six of them: "Cross-site scripting" (found in payloads with "<script>" tags) as a CDT that adds unescaped JavaScript (DS6). "Credential guessing" (found in payloads with clear-text credentials) as a CDT that adds clear-text passwords (DS2). "SQL injection" (found in payloads with unescaped quotes) as a CDT that adds unescaped JSON (DS9). "Cookie tampering" as a CDT that adds a deceptive cookie into HTTP headers (DP2). "Client-side bypass" (found by tampering with a "system" parameter) as a CDT that adds a "system=prod" parameter into URLs (DS7). "Path traversal" (found in payloads with ".." strings) as a CDT that imitates a path traversal vulnerability (DS4). Lastly, their "Content-Type header attack" (found by header tampering) was not mapped to any of our CDTs, but we counted how many participants marked lines containing "Content-Type" in our queries.

E   User Study Details

E.1   Experiment Website and Tutorial

Participants who have agreed to share their data with us (Appendix E.3), were then directed to a tutorial (Appendix E.7) to familiarize them with the experiment. After answering the profiling questions (Appendix E.4), the actual experiment began. Figure 6 shows the user interface for all subsequent 174 queries.

A manual investigation of the answers to the tutorial questions revealed that all participants understood the interface and the experiment. This is not surprising, as the tutorial was also pre-tested to ensure that it is understandable. Two colleagues who did not participate in the actual experiment pre-tested the tutorial.

E.2   Recruitment Message

Would you lend me some of your valuable time to advance research on cyber deception and prove your secure coding skills? We created an interactive game, where you have to think and act like you were a hacker: **LINK**

*If you can participate, please do so by **DATE**. Answering all questions will take you between 30 - 60 minutes. But, you can stop any time. You can also continue later. Progress saves automatically. If you would like to discuss some queries with us afterwards, leave us a comment with your name.*

E.3   Data Privacy Consent and Intent Form

Honeyquest is a game where you have to identify security vulnerabilities in web applications. Be careful, some of the vulnerabilities are traps trying to trick you into thinking something is vulnerable. During the game, we will collect some data to help us advance research on cyber deception:

*We store a cookie on your computer to identify you.***Why?** *So that we know which answers belong to the same person, even when you continue the game later.*

We store your profile information, like your job, years of experience, and skill level. **Why?** *So that we can research, if there are differences among professions.*

We store your answers and the time of your answers. **Why?** *So that we can research what kinds of questions humans are good at and what kinds of questions are hard to get right.*

We do not store your IP address, location, name, email address, or any other PII. **Why?** *Because we don't need it.*

E.4   Participant Profiles

Figure 5 shows our participant's answers to these questions:

- What describes your current profession best?
  - **Development:** Developer, Engineer, Architect
  - **Operations:** System Administrator, SRE
  - **Security Operations:** Penetration Tester, Incident Detection and Response, Product Security
  - **Business:** Manager, Leader, Sales, Marketing
  - **Research:** Researcher, Scientist, Innovator
- How would you describe your secure coding skills?
  - **None:** What do you mean by secure coding?
  - **Little:** I only heard about a few concepts.
  - **Good:** I get the basics but still need guidance.
  - **Advanced:** I apply secure coding concepts regularly.
  - **Expert:** I educate others about secure coding.
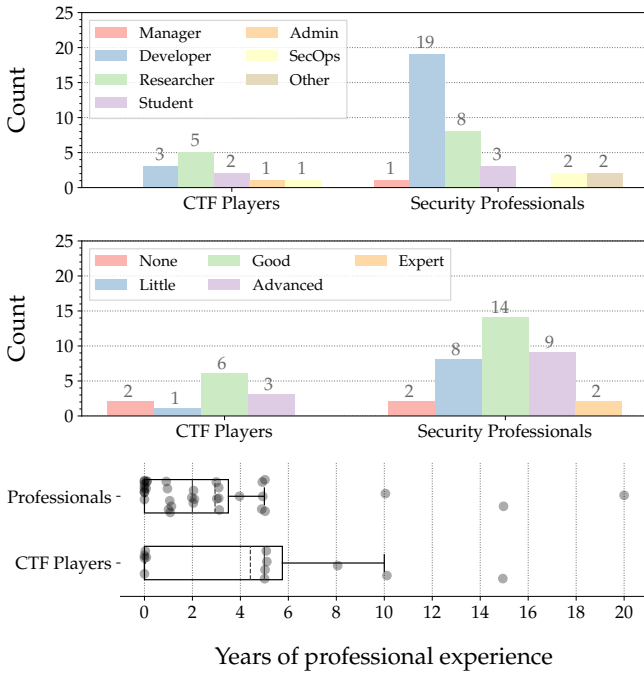- Roughly, how many years have you been professionally involved in the field of cyber security?

Figure 5: Our participants' self-reported profiles.

E.5 Participant Demographics

We asked participants about their professional role, secure coding skills, and years of professional experience in the field of cyber security (§E.4). We did not collect demographic information, but we can describe the target audience (all of which are located in Europe) to which we posted our recruitment message (§E.2):

- **Security Professionals** were predominantly male and between 20 and 45 years old. The majority of them had a university degree in Computer Science.
- **CTF Players** were predominantly male and between 18 and 35 years old. The majority of them were graduate Computer Science students.

E.6 Study Timeline and Incentives

The user study was conducted in two phases, each with exactly the same experimental setup. The first phase with 23 participants (13 professionals, 10 CTF players) was held in February 2023. The second phase with 24 new participants (22 professionals, 2 CTF players) was held in January 2024.

To increase the number of participants in the second phase, we introduced an incentive to win a 50€ Amazon gift card, if they answered at least 50% of queries. We promoted this incentive a few days after the second phase started. 18 of the 22 security professionals (in that phase) joined after that promotion. In the end, 8 of them answered enough queries to qualify for the incentive.

E.7 Tutorial Queries

Every participant had to answer these 8 tutorial queries prior to the actual experiment. Lines are only highlighted in the paper.

```
You are reading a tutorial QUERY.
A query is simply a text of a certain TYPE.
Honeyquest shows you queries of different types.

We want to understand how you would respond to them,
if you act like a hacker or cyber security researcher.
```

Listing 6: Tutorial query 1 / 8.

```
The following is a QUERY of type HTTPHEADERS, meaning,
you observe that an application is making this HTTP request:

> GET /wiki/Cat HTTP/1.1
> Host: en.wikipedia.org
> User-Agent: curl/7.68.0
> Accept: */*

Behind the scenes, Honeyquest classified this query as NEUTRAL.
This means, there is nothing RISKY or DECEPTIVE about it.

If you agree that this query is NEUTRAL, click the button above.
```

Listing 7: Tutorial query 2 / 8 with neutral HTTP headers.

```
Correct! This query was indeed NEUTRAL.

Let's look at another query of the same type:

> HTTP/1.1 200 OK
> Date: Wed, 04 Jan 2016 23:18:20 GMT
> Server: Apache/1.0.3 (Debian)
> Content-Type: text/html
> Transfer-Encoding: chunked

If you think this query is NEUTRAL, click the button, as before.
But, if you see a VULNERABILITY please mark the indicative line.
You can mark and unmark lines by clicking to the LEFT of a line.

Hint: There is exactly one vulnerability to be found here.
```

Listing 8: Tutorial query 3 / 8 with risky "Server" header.

```
Well done! You found the vulnerability:

> Server: Apache/1.0.3 (Debian)

When we say VULNERABILITY, we mean an indicator for it.
The vulnerability here is CVE-1999-0067.
The old version of Apache indicated that.

We don't expect you to look that up.
Think like an attacker looking for vulnerabilities to EXPLOIT.
```

Listing 9: Tutorial query 4 / 8.

```
Let's look at another query.

> GET /wiki/view?file=../articles/Cat.php 200 OK
> Host: en.wikipedia.org
> User-Agent: curl/7.68.0
> Accept: */*

This time, you may also mark lines as TRAPS instead.
A trap wants you to believe that there is something to exploit.
You want to avoid them so you don't waste time or set off alarms.
You can mark and unmark traps by clicking to the RIGHT of a line.

So, you now have three options:

- Mark lines that you would like to EXPLOIT
- Mark lines that you think are TRAPS (and therefore, will avoid)
- Continue without marking, if you think this query is NEUTRAL
```

Listing 10: Tutorial query 5 / 8 with deceptive path traversal.

8 / 182

Assume you are a hacker seeing the following .htaccess file   ⓘ

▶ Submit selection

⚡ An exploit mark means that you see a vulnerability / want to access this path / or want to use it in an attack
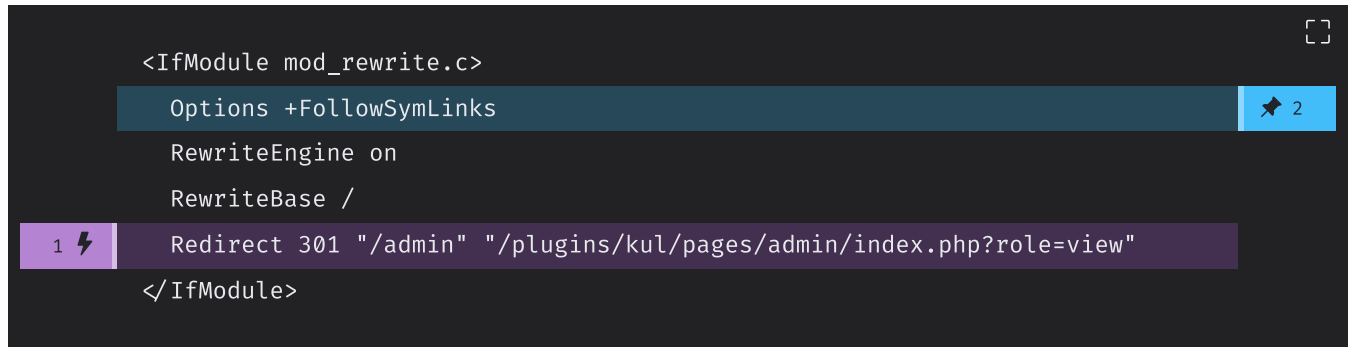✧ A trap mark means that you think this directive is a trap you must avoid

```
<IfModule mod_rewrite.c>
    Options +FollowSymLinks                                              📌 2
    RewriteEngine on
    RewriteBase /
1⚡  Redirect 301 "/admin" "/plugins/kul/pages/admin/index.php?role=view"
</IfModule>
```

Figure 6: A screenshot of the web-based user interface of Honeyquest that shows a query of the type ".htaccess file". In this example, the user placed an ↗ exploit mark on line 5 and a 🐻 trap mark on line 2. When hovering over the info icon right next to the query type, a tooltip with an extensive description of the syntax in the query is shown. The progress bar at the top shows how many of the queries have already been answered. The little pin on the progress bar indicates how many queries an average player has answered. Users can submit feedback or report mistakes by clicking on the speech bubble in the lower right corner.

```
This was a bit harder, wasn't it? You probably marked this line:

> GET /wiki/view?file=../articles/Cat.php 200 OK

This looks like a path traversal vulnerability, doesn't it?

Honeyquest classified this query as either RISKY or DECEPTIVE.

- RISKY queries contain vulnerabilities or weaknesses
- DECEPTIVE queries contain traps you must avoid

You may now wonder if this was truly a vulnerability or a trap.
We don't know either. That depends on the actual implementation.

What matters is only what you - the hacker - THINK it was.
Honeyquest wants to understand how you PERCEIVE such queries.
```

Listing 11: Tutorial query 6 / 8.

```
Let's look at one last query. This time, it is of type FILESYSTEM.

> -rw-r--r--  1 goofy goofy 3.5K Sep 17  2017 .bashrc
> drwx------  7 goofy goofy 4.0K Jan 12  2022 .cache
> drwx------  6 goofy goofy 4.0K Sep 25 17:40 .config
> -rw-r--r--  1 goofy goofy   64 Jun 16  2019 .gitconfig
> drwxr-xr-x  5 goofy goofy 4.0K Jul 10  2021 git-crypt
> drwx------  4 goofy goofy 4.0K Jan 14  2022 .gnupg
> lrwxrwxrwx  1 goofy goofy   19 Jan 23  2018 html -> /srv/html
> drwxr-xr-x  4 goofy goofy 4.0K Aug  1  2021 app-browser
> -rw-r--r--  1 goofy goofy 816M Nov 18  2020 nginx-proxy-logs.txt
> drwxr-xr-x  5 goofy goofy 4.0K Mar 21  2021 terraform-saas

Here, marking something to EXPLOIT means opening the file or folder.
Marking something as a TRAP means you want to AVOID opening it.

You might also have noticed that we put numbers next to your marks.
With them, indicate the ORDER in which you like to EXPLOIT things.
```

```
A few notes:

- Marking multiple lines is OPTIONAL. Marking one or none is fine.
- There might be queries where indicating an order makes no sense.
  Ignore the numbers then.
- We don't expect you to go over every single line and mark it.
  Remember, you are a hacker, rather tell us your next move,
  not an exhaustive list of all possible moves.
```

Listing 12: Tutorial query 7 / 8 with neutral file system.

```
You are all set, here is a brief summary.

Honeyquest shows you NEUTRAL or RISKY or DECEPTIVE queries.
You can answer as many questions as you like and come back later.
Honeyquest saves your progress automatically.

Think like a hacker and tell us your NEXT MOVE.

- You can CONTINUE without marking anything
- You can mark lines to EXPLOIT or mark them as a TRAP to avoid
- You can indicate the ORDER in which you would like to exploit

That's it. Enjoy the game!
```

Listing 13: Tutorial query 8 / 8.

E.8   Image Attribution

⚡ Lightning Thunder Icon by *svgrepo.com* licensed under CC0
🐝 Bee Icon by *bypeople.com* licensed under CC BY 4.0
☺ Neutral Face Icon by *joypixels.com* licensed under CC BY 4.0
↗ Hammer Icon by *Muh Zakaria* licensed under CC BY 3.0
🐻 Bear Trap Icon by *Daniela Howe* licensed under SIL OFL 1.1