Lessons Learned from SunDEW: A Self Defense Environment for Web Applications

Merve Sahin, Cédric Hebert and Anderson Santana de Oliveira
SAP Security Research
{merve.sahin, cedric.hebert, anderson.santana.de.oliveira}@sap.com

Abstract—**Securing web applications is a tedious task: Current best practices range from the secure development lifecycle to the use of a wide variety of detective and reactive measures after deployment. Yet, these measures are not always sufficient to secure the applications. A recent idea is to provide the application with self-defense capabilities, by enhancing it with deceptive components and adding application specific detection points that will be used in runtime. While the idea has been partially explored before, it is not widely adopted in the industry, because of the lack of an end-to-end comprehensive solution, among other reasons.**

In this paper we introduce SunDEW, a multi-layer deception framework to provide self-defense capabilities to web applications. We discuss the main technical challenges when prototyping this idea and we validate its design through a CTF based experiment. We also evaluate how the participants respond to this defense mechanism, together with a user study. We make a number of observations to develop more robust deception techniques even when the attackers are aware of deception. In particular, we find that deceptive elements should be well intertwined with the application and mimic real functionality to be more effective. Moreover, when the attackers are informed about deception, they are likely to deviate from their regular attack path, to not interact with the application elements they find suspicious. On the other hand, attackers' initial reaction is to avoid automated attacks and brute-forcing the application. Instead, they prefer to be cautious and take the time to understand the application flow first. Overall, we observe that even if deception awareness decreases the effectiveness of deceptive elements, it adds a deterrent factor by causing attackers to self-restrict their actions. While our study is a first step to evaluate the robustness of application layer deception against informed attackers, our results suggest that notifying the attackers may bring several advantages to the defenders in any case.

I. Introduction

Deception, one of the oldest concepts in military strategy [56], has recently been gaining popularity in information security field, as an extra layer of defense. Several commercial solutions proposed as part of Moving Target Defense (MTD) or Run-time Application Self Protection (RASP) technologies provide easy-to-deploy deception elements at the network, data, or application levels [20], [54], [24], [11], [34], [3]. These elements are then monitored for malicious or anomalous behavior. Such deceptive elements are expected to result in less false positives compared to traditional defense mechanisms (e.g., IPS/IDS), reducing the efficiency of attackers by wasting their time and increasing the difficulty of attack planning [58], [44]. At a first glance, the idea of deception may seem to contradict Kerckhoff's principle that a security mechanism should not rely on secrecy or obscurity [38]. However, as long as the security of the system is not dependent on obscurity, addition of deceptive elements and misdirection still provides many advantages [6].

Deception has also been studied by the academic community since more than 20 years, however, as the concept of deception can be applied to different system security areas (e.g., at the network [14], data [52], or application layers [30]), each of these remains under-explored in terms of deployment methods, effectiveness, or lifecycle of deceptive techniques [31]. Several survey papers on deception attempt to systematize the knowledge, and draw attention to the need for further research [31], [27], [48], [43].

In this paper we focus on the use of deception and self-defense techniques to help secure web applications. Web applications are often the public facing components of enterprise systems, and they are exposed to a wide range of attacks. Symantec recently reported a 56% increase of attacks on Web service endpoints in 2018 [53]. With the rise of social engineering, phishing attacks (spear phishing, email compromise, email impersonation [32], [17], [59]) and credential stuffing [55], attackers often start with valid credentials to access the web application [19]. Moreover, it often takes several months before a security breach is discovered [25]. The actions that the attacker will take during this time (exploring the system, looking for vulnerabilities) are the motivation for planting the deceptive elements for detection. For such cases, deception can provide an extra layer of defense in addition to the traditional security measures (such as web application firewalls) that often implement generic measures against known attack vectors. The advantage of deception is that it can be designed to be specific to the application, addressing all of its capabilities and features [61].

Existing studies on application layer deception mostly focus on adding deceptive elements via a proxy [30], [28], which is an approach that we also adopt in this work. We extend this concept by adding further deception layers to cover the post-detection phase, to provide a better response action once a malicious request is detected. A naive response action adopted in previous studies [30], [28] is to just log the attack while returning a valid looking response. Note that, certain actions like terminating the session, adding timing delays or temporarily blocking the application [47], [35] might tip off the attacker.

Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb) 2020
23 February 2020, San Diego, CA, USA
ISBN 1-891562-63-0
https://dx.doi.org/10.14722/madweb.2020.23005
www.ndss-symposium.org

In contrast to the previous work, we redirect the malicious requests to a clone of the application that serves synthetic data. This allows to monitor the attacker's behavior (without putting the application data at risk) in an effort to learn his real purpose, as well as to identify the vulnerabilities he might find in the application. The idea of using "clone" systems was previously explored in different domains [7], [9], and was briefly discussed in the web application context [42], [47]. Based on this idea, *our first contribution is to present a multi-layer deception framework for web applications, and to analyze the technical and research challenges related to this approach.* We named our framework SunDEW (Self Defending Environment for Web applications), inspired from the carnivorous sundew plant that attracts and traps insects with its sticky leaves[1].

In the second part of the study, we focus on the robustness of deception - that is, when the attacker is aware of this countermeasure - to see how to improve our framework. As deception technologies gain more popularity and commercial solutions become more widely adopted, the assumption that deceptive techniques are obscure/hidden will not remain true for long. For deception to be relevant in the long term, it should remain effective even if the attacker is aware of it. In fact, deception can be considered as a cryptographic algorithm, where the deceptive elements themselves are the secret keys [13], [36]. Previous work measures how attackers react to deception in different domains (such as data or system layer deception) [31]. On data layer, Shabtai et al. find that awareness does not have an impact on attacker behavior [52]. On system layer, studies find that awareness makes the defense mechanism even more powerful because it increases the cognitive load of the attacker (such as increasing stress and reducing the belief in success [23], [18], [24]). To the best of our knowledge, our study is the first to analyze the impact of deception awareness on web application layer. For this, *we first implement a proof-of-concept of the SunDEW environment, and employ it in a Capture The Flag (CTF) competition* on an enterprise CTF platform (that is used for internal security training). Our experiment aims to answer the following questions:

- How do the attackers perceive and react to the deception technology?
- Does deception technology remain effective even when the attacker is aware of its use?
- How can the proposed framework be improved?

We find that, among the participants who were able to solve the challenge, 85% have changed their attack strategy due to being aware of deception. While 60% of participants had difficulty to work around it, the most common reaction was to avoid scripted attacks as well as using known attack automation tools. We also find that the effectiveness of deceptive elements decreases when the participants are aware of it. One lesson learned is that, for a more robust defense system we need to design the deceptive elements well intertwined with the application, as well as to design response actions that are realistic and that makes the deceptive elements look functional.

[1]https://www.britannica.com/plant/sundew

II. Related Work

A. Deception in web application layer:

Deceptive elements for web applications have been proposed to be deployed via a proxy in front of the application, via modifying the web server, or built in the application source code [31]. For instance, Fraunholz et al. present a reverse proxy framework that implements various deceptive techniques and evaluate the performance overhead of the framework [28]. Han et al. also propose to implement deceptive elements via a reverse proxy [30]. Moreover they use a CTF exercise to evaluate the effectiveness of deceptive elements, and deploy these elements in a real-world application to measure the false positive rate. Among 150 CTF participants, 56% triggered at least one of the deceptive traps. In addition, over 7 months of period, there were no false positive alerts triggered in the real-world deployment. Another study [26] focuses on the reconnaissance phase of web attacks to identify deceptive countermeasures, such as delaying responses to scanning attempts. The countermeasures are implemented in a web server, and evaluated against several vulnerability scanners. In our work, we propose to use deception in multiple layers, and to focus on the attackers' perception and on the robustness of the approach.

B. Use of duplicate systems for deception

The idea of deceiving attackers with a fake environment that is a copy of the real environment has been explored in a number of studies. For instance, Anagnostakis et al. aim to reduce the false positive rate of anomaly detectors by routing the potentially dangerous requests to an instrumented clone of the application (called a *shadow honeypot*) [7]. The shadow application is instrumented to detect certain failures such as memory violations, and able to rollback to a known good state after an attack. Similarly, Urias et al. propose to duplicate possible compromised machines and place them in a deceptive environment to further observe attacker behavior [57]. Kontaxis et al. propose to duplicate the entire application server multiple times, so that the adversary cannot know if he compromised the real server [40]. Araujo et al. [9] propose to implement honey-patches for known vulnerabilities. A honey-patch can detect an exploitation attempt and redirect the attacker to a *decoy* environment (that is a copy of the original environment with redacted sensitive data), while the attacker thinks the exploit was successful. This approach can be complementary to other forms of deception, as it is only effective for known vulnerabilities. In a further study, authors experiment with honey-patches in an educative CTF environment [10]. The closest idea to our study in terms of the deception framework is presented in [42]. We enrich this idea with application layer deception, discuss the technical and research challenges it brings, provide a prototype implementation, and evaluate it with a CTF exercise.

C. Impact of deception awareness

Several studies in data and system layer deception analyze how deception awareness affects the attackers' decision making process and cognition. Cohen et al. conduct controlled red teaming experiments in a network of multiple machines, where they gradually inform the participants about deception [18].

They find out that deception awareness reduces the belief in success and makes participants more likely to get frustrated and to give up early. Although most of the research questions of the study yields in statistically insignificant results, the authors conclude that the quality of deception is very important for its effectiveness, and that creating *content-oriented deceptions* will be necessary to deceive skilled attackers in the long term. Shabtai et al. [52] analyze user behavior in an experiment where the participants act as bank employees who need to approve loan requests. They can approve the loans in a legitimate or illegitimate way from which they obtain 10% or 20% commission respectively. The purpose of participants is to maximize their profit, however they risk to be detected if they approve a honey-loan in an illegitimate way. Half of the participants know that honey-loans were planted in the data, while the other half only knows without details that they can be detected if they make an illegitimate approval. The study finds out that, awareness of the honeytokens had no significant impact on the ratio of illegal actions taken. In other words, deception awareness does not decrease the attack detection rate. While this result is counter-intuitive, the paper states two possible reasons: The honey-loans may be so realistic that participants could not differentiate, and there were no significant consequence for illegal behavior, except losing a bonus. This study shows the difficulty of experiment setup for evaluating deception. Yuill et al. [62] analyze the psychological vulnerabilities while facing deception, that can be used in the context of computer security. They note that deception awareness might cause the attackers to believe that real security vulnerabilities are deceptive methods. One study that confirms this theory is [23]. In this study the authors conduct network penetration testing experiments with red team members. The experiment has four groups where deceptive elements are present or not, and where the participants are informed about deception or not. After each experiment, the authors conduct a survey that aims at finding cognitive, emotional and physiological effects of deception. They find that being informed about the presence of deception can cause self-doubt and reduce self-confidence. Moreover, only telling attackers that there might be deception (even if the network does not have deceptive elements) can make them more suspicious and drive them to change their attack strategy. In our study, we aim at answering similar questions for the web application defense: We analyze how attackers perceive deception and if being informed about deception affects their attack strategy.

III. SunDEW: A Multi-layer Deception Framework

In this section we present SunDEW, a self defense environment for web applications. We propose to use deception in three layers (application, system and data [31]) so that the attacker's user experience remains consistent, while the attack is detected and contained. An overview of the framework is given in Figure 1. Next, we explain each architectural layer of SunDEW.

A. Application layer

The simplest type of deception applied to web applications is to embed deceptive elements in the application source code or via a reverse proxy in front of the application [30], [28].

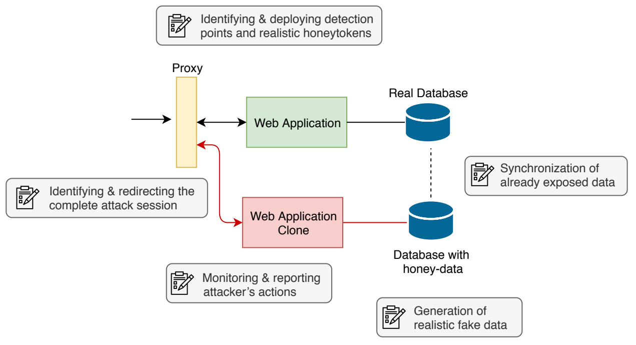

Identifying & deploying detection points and realistic honeytokens
Proxy
Web Application
Real Database
Synchronization of already exposed data
Identifying & redirecting the complete attack session
Web Application Clone
Database with honey-data
Monitoring & reporting attacker's actions
Generation of realistic fake data

Fig. 1. Overview of our deception framework and technical challenges.

These elements (such as a hidden HTTP POST parameter, or a URL in HTML comments) are only expected to be interacted with by attackers, not interrupting the normal usage. Furthermore, it is also possible to implement IDS-like, application-specific detection points to monitor any anomalies in the application behavior. To the best of our knowledge, the OWASP AppSensor project is the first to propose such detection points [46], [30]. Although detection points do not provide straightforward deception, they can be very useful in runtime application self defense. Application-specific deceptive elements and detection points can be determined via a threat modeling exercise [60], which will help to define believable decoys as well as relevant monitoring points. In this work we propose to combine deceptive elements with attack detection points to provide more extensive defense capabilities to the application. Combining the existing classifications and related work [31], [30], [28], [26], [46], [60], we list several examples of such elements in Table I.[2] For a complete picture, we also add the behavior based anomaly detection points. However, we believe that behavior based detection is more prone to false positives and should be treated more carefully.

A note on accuracy and false positives: One purpose of deceptive elements and application specific detection points is to reduce the false positive rates compared to more generic defense methods such as Web Application Firewalls (WAF). Indeed, the few studies that attempt to measure false positives report zero or very low false positive rates [15], [30], [51]. Note that detection points and deception do not provide 100% protection per se, but they can provide an improvement over only using a generic WAF or IDS. How they should be combined with generic defenses is also an open research topic [31].

B. System layer

On the system layer, we propose to deceive the attackers by redirecting them to an exact copy of the web application. Depending on the architecture, this application can run in a separate container or virtual machine that is well monitored. Once an attack is detected, we aim to keep the attacker in the clone application as long as possible by tainting the malicious session and by redirecting all the subsequent requests. The redirection can be implemented via a reverse proxy, as part

[2]The elements listed in bold have been implemented in our CTF challenge for experimentation (See Section IV).

	Examples		Goal
Deceptive Elements	Data	- Honey HTTP GET/POST parameters [30], [28] - **Honey cookies** [30] - **Honey HTML elements** (hidden form fields, commented out URLs / account credentials) [46], [30], [28] - Hyperlinks to track attacker [29]	Detection
	Configuration	- Honey disallow entries in /robots.txt [28], [26] - Honey permissions and accounts [37]	
	Weakness	- Honey vulnerability patches [30], [9], [8]	
	Response	- Web server version trickery [28], [26] - HTTP response status code tampering [28], [26] - Upload sinkholing (e.g., 200 response to PUT requests) [46]	Confusion
	Performance	- Latency adoption [28], [26]	
Detection points	Request Exception	- Unexpected HTTP method [46] - Unsupported HTTP Method [46] - Missing/duplicated request data [46] - Unexpected type/quantity of characters in the request [46]	Detection
	Authentication	- **Utilization of common passwords (e.g., ”123456”)** and usernames (e.g., ”admin”) [46]	
	Session	- Use of another user's session ID or cookie [46]	
	Input/Output	- Unexpected/deleted/modified cookie [46] - Violation of input data integrity [46] - **Violation of black lists** (e.g., SQLi or XSS patterns) [46] - Abnormal output data (length, format, structure) [46]	
	Access Control	- **Forced browsing attempts** for a non-existent / not authorized URL [46] - **Direct object access attempts with modified GET/POST parameters** [46]	
Behavior analysis	User Trend	- Deviation from normal GEO location [46] - Abnormal speed or frequency of use [46]	Detection
	Authentication	- High rate of logins/logouts to the application [46] - Multiple failed login attempts [46]	

TABLE I. LIST OF RUNTIME APPLICATION DEFENSE TECHNIQUES.

of the application, or via a dedicated micro service in a cloud environment.

The advantages of redirecting to a clone are multi-fold: it provides a seamless transition between the real application and the honey-environment, it enables the use of extensive monitoring tools (which may slow down the application in normal use), and most importantly, it allows the attacker to continue in his attack stages, which may reveal any unknown vulnerabilities and help us learn the ultimate objective of the attack. Moreover, this architecture allows to react to attacks immediately, rather than just logging (as proposed in some of the previous work [30], [28]) or blocking the requests. Note that the framework can also trap pentesters and legitimate vulnerability scanners in the application clone. However, the vulnerabilities they find will still be valid, as the application clone is no different than the real application.

C. Data layer

To protect the real application data, we propose to use a separate database instance in the application clone, with exactly the same schema, but containing synthetic data. Several previous works explore how to generate realistic fake data for deception or for application testing purposes. Most of the synthetic data can be automatically created using a deep learning generative model with differential privacy, as suggested in [4], and demonstrated in [21]. This approach is well adapted to produce most of the volume from the transaction data of the real application. In contrast to approaches using generative models, [33] can populate empty databases by taking user input or computing the data distribution from existing databases, to further generate test data. It would not be suited for deceptive purposes though, because it would leak sensitive data in the clone application. In [12], the goal is to understand data distribution by mining rules, then to sanitize sensitive data using a constraint solving anonymization method to generate honeydata. The issue with the latter is that it is not known to be resistant to re-identification and membership inference attacks, as is the case for differential privacy.

In this study we do not intend to provide an exhaustive tool with all data generation capabilities, but to provide some guidelines and draw attention to the need for more research in this area. In practice, to produce realistic data, several steps are required. For instance:

- Identifying publicly available information contained in database tables, such as organization names and addresses. Such elements can be copied as is to the clone database.
- Identifying the sensitive attributes, whose values shall never appear among the fake data items (e.g., values that depend on the user input).
- Identifying the objects with related and independent columns in order to maintain relationships in the generated data.
- Recreating all attributes of all tables considered sensitive in a completely synthetic manner.

Note that for direct identifiers (such as SSNs, passport numbers, license plate numbers, etc.), it is possible to use existing libraries for test data generation. For instance, *Faker* [22] provides a variety of pre-built data generation templates, enables localizing the data (e.g., selecting specific languages or countries for names and addresses), and it is easily extensible.

D. Technical Challenges and Research Questions

Our framework brings several challenges and open up new research areas.

1) Generating realistic deceptive elements: Deceptive elements added to the application should look and feel as part of the application, well integrated in the application context, which is not an easy task. Currently, there is no automated way of generating such elements specific to an application. For instance, while [30] automates the embedding of deceptive elements via a reverse proxy, authors still needed to go through the application to carefully select the names of the deceptive elements according to the content. The most relevant study in this context, [49], focuses on automatically creating honey HTML form fields for web applications. The authors collect the form field names from Alexa Top 10,000 websites and present an algorithm to select the most plausible field names for a given application. They then make a user study where they ask 75 students to look at 50 HTML forms and identify which of those have honey form fields. The results are significantly close to random selection, which means the participants were not able to identify the honey fields. While this study provides a good basis, more techniques need to be developed to automate the generation of different types of web honeytokens, which are content-specific, realistic, and blend in well with the application logic.

2) Fake data generation: As mentioned in Section III-C, automating the generation of synthetic data is another research challenge. For instance, finding out the data periodicity in the real application (to send "fresh" data to the clone database), as well as managing the data volume over time are some of the problems. We also need to have a good estimation of the longevity of attacks from the same individual or group as to present them with consistent data, when they return with the same leaked credentials. Another challenge is to faithfully reproduce the *unstructured* data (including the sensitive parts) to appear realistic.

3) Smart data exposure: In their book, Sushil et al. [36] state that the behavior of an *intelligently deceptive* system should be indistinguishable from the normal behavior, even if the user has interacted with the system before. While we propose to redirect a malicious session to the clone of the application on the fly, we need to ensure that the attacker's user experience will not be affected by this diversion and that there won't be discrepancies in the data visible to the attacker. Thus, we need a mechanism to remember which part of the application data was already visible to the attacker before he was detected and redirected. This can be implemented with a monitoring service running on the real system which transfers a copy of the data that was made visible to the user during the current session. Once the session expires, or after a certain amount of time, the data can be deleted.

4) Keeping the attacker trapped: To maintain the target application protected, the attacker should be kept trapped in the application clone during the whole attack session, and better, across multiple attack sessions. While it may not be possible to have a perfect solution, we believe that incorporating browser and device fingerprinting in the authentication process [41], [16], [5] can be useful, at least to distinguish legitimate users from attackers and to avoid sending a legitimate user to a clone. Note that, in any case, the reverse proxy would need to be well integrated with the authentication procedure of the application, to track the current active and blacklisted sessions, and to recognize the authentication failures and logout/login events.

5) Remediation: If a malicious activity is detected on one user's account, this account is likely to have been compromised, and any further connection with these credentials should be treated carefully. On the one hand, users should not be prevented from accessing the application and the system should avoid sending a legitimate user to the application clone. On the other hand, attackers may initiate parallel sessions from several browsers or scripts, leading them to being connected to both the application and its clone (serving different data) at the same time. One approach could be to immediately lock the victim's account [47] and to find a way to contact the real user as quickly as possible for a password change. Once the password is changed, the old password can be used as a detection point, ensuring all further initiated sessions via this password will be consistently diverted to a clone. *This approach also provides a remediation for the possible false positives, where a legitimate user has been redirected to the application clone.*

6) Deployment and scalability: Automated deployment of a self defense environment for a given web application would be the best way to reduce the overhead for application developers and to increase the usability of this solution. However, the large variety of Web technologies and frameworks makes this task very difficult. Moreover, on a cloud environment where each part of the application is served via a different micro service, spawning clones for all services and for each attack session may be impractical. Relying on a single clone for each application, where to send all attackers, may be a potential solution, at the cost of exposing to all caught attackers the real data seen by each of them before detection.

Another aspect to consider is the performance overhead of the framework. As the application or the reverse proxy will need to parse and analyze all requests and responses, the framework will increase the communication latency. Previous work [28] analyzes the performance overhead of the reverse proxy (without any performance optimization) and finds that the effect depends on the type of tampering performed by the proxy, while combining multiple deceptive elements does not necessarily increase the overhead additively. In a real world deployment, performance overhead of the framework should be well tested not to degrade the user experience. For high latency operations, it could be possible to delegate them to a separate component.

IV. Prototyping and Experiment Design

In this section we explain how we develop a prototype for the SunDEW framework and use it for our CTF exercise. We started by implementing a small web application with the Spring Boot framework [1], following the best practices for the available security features such as session management, access control and authentication [2]. Our application mimics a hospital management software where the patients and doctors

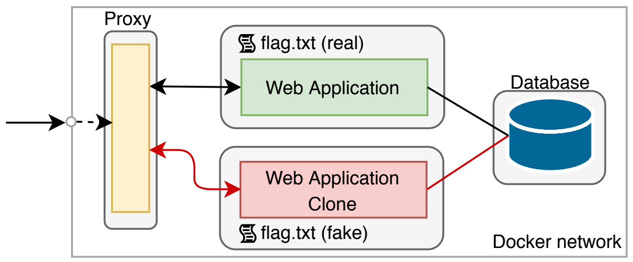
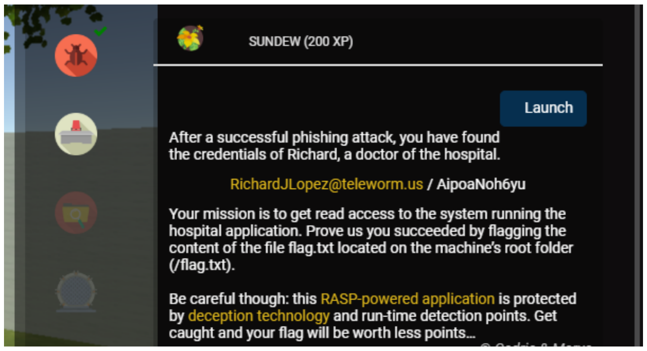


Fig. 2. Overview of our deception architecture and technical challenges.



After a successful phishing attack, you have found the credentials of Richard, a doctor of the hospital.

RichardJLopez@teleworm.us / AipoaNoh6yu

Your mission is to get read access to the system running the hospital application. Prove us you succeeded by flagging the content of the file flag.txt located on the machine's root folder (/flag.txt).

Be careful though: this RASP-powered application is protected by deception technology and run-time detection points. Get caught and your flag will be worth less points...

Fig. 3. Screenshot of the SunDEW challenge description.

can view and modify various data, protected by role based access control. We then made a small threat modeling exercise to decide on the deceptive elements and detection points. We have considered possible attacks on reconnaissance (e.g., directory bruteforcing), privilege escalation, insecure direct object reference, and weak account passwords. In contrast to the previous work [30], [28], we also consider the response actions in case a deception or detection element is tampered with. Table II lists all the elements, how they are monitored and the application's reaction upon a malicious action.

We implemented these elements partially in the application, and partially via a reverse proxy written in *Node.js*[3]. The proxy uses several packages that allows to manipulate the HTTP messages, such as *cookie-parser*[4] and *body-parser*[5]. Monitoring of the elements and the redirection procedure is also handled by the proxy. Moreover, the proxy keeps a database of session cookies together with the login-logout events for each user, besides the triggered deceptive elements and the sessions that must be redirected to the clone. All of the components (the reverse proxy, applications, database) are run in separate Docker containers in a Docker network, with a single interface for external communications.

In the next step, for the sake of the CTF challenge, we added an XXE (XML eXternal Entity [50]) vulnerability to the application, which will be triggered when a profile picture is uploaded with a specific payload. In a nutshell, the application uses a third party JAVA library that converts an uploaded SVG file (which is represented as XML) into a PNG file. However the parsing routine of the library is flawed, which makes it possible to read arbitrary files on the server via an XXE attack [39]. Note that we chose the vulnerability *after* deciding on the deception and detection elements. Moreover, we were careful that the exploitation of the vulnerability does not require interaction with these elements. Finally, we added a hint for the participants: The */notes/todo* URL commented out in HTML source points to a todo file, which mentions a hint about the implementation of profile picture upload. This URL is not a deceptive element.

Finally, in the challenge description we give participants a valid username/password combination. The scenario is that the attacker obtained valid credentials from a phishing attack and can reuse them to access the application like a legitimate user. We then warn the participants that the application is protected by deception technology and runtime detection points, and if they get caught, their flag will worth less points. A screenshot of the challenge description can be found in Figure 3.

Note that, we plant different flags in the real and clone applications. Once a participant triggers a deception/detection element, his session will be redirected to the clone and if then he exploits the vulnerability, he will access the flag in the clone application (i.e., the fake flag), which worths only half of the challenge points (100 points instead of 200). This provides the incentive to care about the defense mechanism employed.

To avoid the challenges related to the *smart data exposure*, we develop the CTF challenge using a simplified deception architecture: We use a single database instance for both the real and clone applications. This makes sure that the participants will not see any discrepancies in the data, when their session is redirected to the application clone. The challenge architecture can be found in Figure 2. Moreover, to be able to monitor the participants individually and to avoid the issues related to using a single database for both applications, we create a separate docker network instance for each player. Finally, for analysis, we collect *httpry*[6] logs from the applications and the proxy, for each user.

Evaluation strategy. Ideally, the best way to evaluate participants' reaction to deception would be to make a controlled experiment and inform only half of the participants about deception. However, we avoided this for several reasons. First, it would be unfair for the informed group as the challenge difficulty increases. Second, dividing participants would mean having less participants in each category, which could affect the significance of the results. Finally, the CTF continues for one month and participants have means to communicate about challenges. Thus, we instead decided to conduct a survey on the participants who are able to solve the challenge. In addition, we compare the detection rates with another, similar web challenge on the CTF platform to see if participants behaved differently when they are informed about deception.

Post-challenge survey. We designed this survey to analyze participants' experience with the challenge, how they perceive deception technology, and how they change their attack behavior. The survey is presented as another challenge on the CTF platform, and it is worth 50 points. However, this challenge is only available to the participants who were able to get one of the flags (real or fake) in the SunDEW challenge. As the SunDEW challenge is part of a large CTF competition that

[3]https://www.npmjs.com/package/http-proxy
[4]https://www.npmjs.com/package/cookie-parser
[5]https://www.npmjs.com/package/body-parser

[6]https://github.com/jbittel/httpry

Deception/Detection Element	Monitored Against	Reaction/Response	Detection rate
Honey “Username” cookie	Tampering	Reset to original value	1%
Honey “Role” cookie	Tampering	Reset to original value	4%
Honey hidden POST parameter	Tampering	No effect on the response (not vulnerable)	10%
GET parameter of /view_patient/id	IDOR attempts	HTTP 403 Unauthorized	50%
Password blacklist on authentication	Blacklist of weak passwords	HTTP 302 Authentication failed	8%
URL blacklist for GET requests	Blacklist from dirbuster	HTTP 404 Not Found	14%
SQLi blacklist for all input fields	Blacklist from sqlmap	No effect on the response (not vulnerable)	6%

TABLE II. List of the deception & detection elements used in the CTF experiment, and the individual detection rates.

continues for one month and open to all employees globally, we had to make sure that the participants who did not work on the challenge will not be answering the survey. Moreover, this allows for a more refined analysis: The participants should have spent enough time and efforts on the challenge, to be able to capture the flag. In addition, they are likely to be more experienced in information security, which allows us to evaluate our framework against stronger attackers. The questionnaire includes 17 questions, including single-answer, open-ended and multiple-answer ones.

V. Results

A. Overview

The CTF competition continued for 4 weeks in October, 2019. It included 50 challenges in various categories (e.g., web, binary analysis, forensics, cryptography). More than 400 participants was able to solve at least 1 challenge.

In total, 98 participants attempted to solve our SunDEW challenge. 51% of them have triggered at least one deception or detection element. Table II presents, for each element, the ratio of users who has triggered this element at least once. Note that the “id” GET parameter alone was able to trick 50% of participants to tamper with it. 18% of the participants have triggered more than one element.

Overall, 28 participants were able to exploit the vulnerability. 19 (68%) of them triggered a deceptive or detection element at least once, while 9 (32%) did not trigger any of the traps and accessed the real flag. These 28 participants were able to answer the survey later on. Thanks to the survey, we understand that the 9 participants who accessed the real flag have focused on the picture upload feature straightaway, by following the hints that we provided in the challenge: The */notes/todo* URL, and *SVG* listed as a supported file type (see Section IV).

In the next section, we will analyze the survey results to see how participants perceived deception.

B. Survey results

1) Participants’ profile: We start the survey with a few questions to learn about participants’ profile and experience. Most of the participants have developer or engineer roles in their daily job, except two MSc students and a pentester. Participants rate their information security experience as 3.7±1 on a scale from 1 to 5. Moreover, they rate their knowledge on deception technology before solving the challenge as 2.3±1. Overall, the participants seem to be quite experienced in information security field, and already familiar with the deception technology.

We then ask a single-answer question about how much time participants spent to research about deception technology before starting to solve the challenge. Figure 4 shows the results: only 25% of the participants researched about it for more than 15 minutes, while 28% did not do any prior investigation.

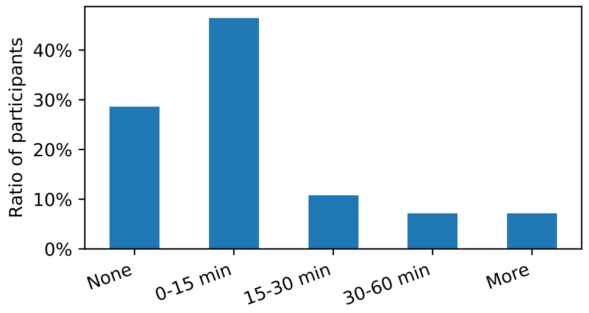


Fig. 4. How much time participants spent to research about the deception technology before starting the challenge.

2) Participants experience with the challenge: In an open-ended question, we ask participants to report any anomalous behavior they experienced during the challenge. Except few platform related issues, they did not report any anomalous behavior in the application. This shows that the redirection mechanism worked well in tricking the participants. When we ask participants whether they think they interacted with a honeypot server (instead of the real application server) at some point in the challenge, 89% of participants answered “No”, while in reality 68% did interact with a honeypot.

3) Participants’ perception of the deception technology: To evaluate how the participants perceive the deception technology, we first ask an open-ended question about whether knowing about deception and runtime defense had any impact on their attack strategy. 57% of participants answered yes and 43% answered no. Table III summarizes the additional explanations the participants report on this open-ended question. We can see that the participants whose strategy were affected by deception take different precautions depending on how they interpret the technology: Some of them avoid scanning the web server, while some avoid tampering with cookies or trying out XSS attacks.

We then ask a multiple-choice and multiple-answer question, where we list some of the possible strategies to avoid detection and ask the participants whether they adopted any of those behavior. Table IV lists all the answers and their popularity. Overall, 75% of participants reported to avoid brute forcing. Other popular answers was to avoid automation tools

Yes, deception had an impact on my strategy (57%)	No, deception did not have any impact (43%)
– I was very careful / cautious, - I avoid to use brute force attack. - especially I didnt try tampering with the cookies . - I investigated everything client side and interacted normally in the beginning. - I tried not to access .git and stuff, but finally still used dirbuster as I wasnt successful otherwise after some hours. – At the beginning, I tried to be quiet, without scanning the webserver and focused purely on the svg upload. But after a while, none of my payload worked out, so I started with the scanning, which might be loud on server side. – I avoided automated attacks/scanning (like port scan). – I tried not to access things that I was sure wasn't authorized, like an ID that didn't appear. Also, avoided XSS in the text fields. – I was focusing only on the target file, not other files in the system. – I used the URL of a colleague to try riskier stuff – It scared me.	– I did not search about deception. – My idea was to first solve the challenge without taking the honeypot into account. – I was fairly certain I will have another option to solve the challenge once again. (But in real life I would have been very scared not to be detected if I attacked the application in a way that could identify me.) – I did not find any honeypot so I am no affected. – TBH, I only read up a little and was not sure about the technique.

TABLE III. Answers to the open-ended question: "Did knowing about deception and RASP have any impact on your attack strategy? Please explain?"

and scripted attacks, as well as to find the vulnerability with the least amount of interaction. Indeed, when we search for the user-agent strings for known attack tools (e.g., Nikto, DirBuster, sqlmap, Postman) or scripting languages (e.g., python-requests, go-http, curl) in HTTP logs, we only found 13% of survey participants to use such tools (This ratio is 15% among the whole population of 98 participants). Thus, we observe that **deception technology is likely to push the attackers towards manual work (rather than using automation tools) and to be more careful in their interactions with the application**. Moreover, participants seem to perceive deception technology similar to the signature based attack detection methods. On the other hand, strategies that avoid the actual deception/detection points that we monitored (e.g., hidden HTML elements, forced browsing, GET parameters) were less popular.

While in the previous open-ended question 43% of participants reported that knowing about deception did not have an effect on their attack strategy, in this multiple-answer question, two thirds of those participants have selected at least one strategy they used to avoid detection. In fact, **overall, the behavior of 85% of survey participants were affected by the notion of deception.** An interesting comment was the following:

> Although I wanted to ignore the deception, I would say knowing about it still determined my attack path. I started to read all received files carefully (i.e. html for every page the normal user can use), and refrained mostly from wildly changing parameters. I also started by simulating a real doctor to see how the application is supposed to behave and to see the normal flow of the application.

However we also observe that, even if the participants take some precautions initially, **they are likely to fall back to regular attack strategies if they cannot find an attack vector.** In particular, one participant reports that he "gave up on most of these" precautions, and two other participant states that (Table III) after not being successful for a while, they started using scanning tools.

Finally, we observe that only a small number of participants have changed their browser user-agent and session cookies. This means that **browser fingerprinting can be a good way to track the attacker across different sessions.**

Later, we ask another multiple-choice, multiple-answer question to understand why the participants did not try avoid detection. Note that, we again allow all participants to answer this question, considering they may agree with some of the options even if they initially answered "Yes" for the strategy change. 43% of the participants reported they "did not know what to do to avoid detection", while 28% reported they "thought it was not possible to avoid detection". Combining these answers, **overall 60% of participants had trouble to identify a strategy against deception.** Moreover 28% stated they "did not understand what is deception technology", while only 7% stated they "did not care about earning less points from the challenge, if they are caught".

It is interesting to note that, even though the participants reported to be already familiar with the deception technology and they have spent time to research about it prior to the challenge, most of them still could not determine how to circumvent it. This can become an important advantage on the defense side, as the defense strategy becomes opaque and ambiguous to the attacker.

4) Participants' perception of deceptive elements : In the next part of the survey, we focus on how the participants perceive and interact with the deceptive elements. For each element, Table V summarizes the ratio of participants who considered it deceptive (observed from the survey), and who have interacted (e.g., modified) with this element (observed from the HTTP logs).

The first thing we notice is that the **participants interact less with the elements that they find more suspicious.** While most participants thought that the cookies were deceptive, they are more likely to tamper with the Role cookie, as its misconfiguration might lead to a privilege escalation attack. On the other hand, GET and POST parameters created less

Strategy	Ratio of participants that agree
I avoided brute-forcing the application.	75%
I tried to find the vulnerability with the least amount of interaction with the application.	60%
I did not try to login as admin/admin nor tried similar default passwords.	60%
I avoided using known automation tools (like sqlmap, dirbuster).	60%
I avoided launching scripted attacks.	53%
I searched online about deception technology to learn what kind of detection methods are used.	50%
I avoided fuzzing.	43%
I refrained from trying SQL injection.	39%
I avoided forced browsing.	32%
I avoided modifying hidden POST parameters.	28%
I avoided modifying GET parameters such as the patient ID in the URL.	25%
I used automated tools but modified them.	10%
I changed the browser user agent frequently to avoid detection.	7%
I changed the session cookie frequently to avoid detection.	3%

TABLE IV. ANSWERS TO THE MULTIPLE CHOICE QUESTION: "WHICH OF THE FOLLOWING STRATEGIES (IF ANY) YOU ADOPTED TO AVOID THE DETECTION METHODS THAT WE EMPLOYED (RASP AND DECEPTION TECHNOLOGY)?"

	Considered deceptive (survey)	Interacted with (HTTP logs)
Username cookie	53%	3%
Role cookie	61%	14%
Hidden POST parameter	28%	21%
GET parameter *id*	7%	61%

TABLE V. ANSWER TO THE QUESTION: "WHICH OF THE FOLLOWING APPLICATION ELEMENTS YOU CONSIDERED TO BE DECEPTIVE?"" VS. THE RATIO OF PARTICIPANTS WHO INTERACTED WITH THE ELEMENT DURING CTF.

suspicion among the participants. We then ask an open-ended question about the reasons why these elements were considered deceptive. 21% of participants state that the cookies were overwritten each time, and/or the changes to the hidden POST parameter had no impact. Thus, these participants have identified the deceptive elements only after interacting with them. While this is a good property for deceptive elements according to the previous work [30], we find that it may not be enough for preserving the deception, as it may tip off the attacker about being detected. Thus, we believe that **designing realistic responses is as important as designing the deceptive elements**. Moreover, our results show that, adding deceptive elements that "do not interfere with the normal behavior of application" [30] may not be useful anymore when the attacker is aware of deception. Instead, **deceptive elements should be designed to intertwine with the application logic and functionality to be resistant to attackers' deception awareness**. The fact that the GET and POST parameters were found less suspicious than the cookies supports this hypothesis.

On the other hand, if such simple-looking elements (e.g. Role cookie) were due to bad implementation practices and were actually vulnerable, deception awareness could deter attackers from tampering with them. Moreover, naive deceptive elements might obscure the more advanced ones by exploiting the attackers' expectations and cognitive biases [62]. Thus, deploying simple deceptive elements combined with more sophisticated ones can be a good defense strategy.

Furthermore, 28% of the participants just stated that the selected elements were looking "too suspicious" or "it is the feeling" they had, without giving specific reasons. 14% stated that manipulating these elements would be a "too easy" solution for this challenge.

5) Robustness of the redirection mechanism: In our proof-of-concept implementation, we redirect the attackers to the clone application by blacklisting their session cookie, once they trigger a deception/detection element. Our proxy tracks the subsequent changes to the session cookie (e.g., due to login/logout events) and keeps the attacker trapped in the clone application seamlessly. However, if the attacker modifies or deletes the session cookie, he will be able to escape the clone application.

As we mentioned in Section III-D4, one possible way to make the redirection mechanism more robust would be to use browser fingerprinting. However this is not a perfect solution: Indeed, in the survey, one participant reported to connect from a different browser and in incognito mode to verify his flag before submitting.

To learn more about participants' perception of deception, we ask two Yes/No questions about whether they think the use of a VPN and the modification of the browser user-agent would help them mitigate detection or would make them easier to get caught. Respectively, 75% and 64% of participants think that these actions would make them more suspicious. **Thus, being informed about deception can make the attacker less likely to deviate from normal usage, which could make fingerprinting more effective**.

Overall, we believe that more research is needed to track attackers across different attack sessions, by incorporating fingerprinting mechanisms or other means.

C. Effect of deception awareness in comparison to another challenge

In the last part, we compare the detection rates of our CTF challenge (Challenge #1) with another web challenge that has an SSRF (Server Side Request Forgery [45]) vulnerability (Challenge #2). As opposed to Challenge #1, Challenge #2 is not protected by the SunDEW environment, and there is no deception related information mentioned in the challenge description. However, we still added an additional, honey cookie to Challenge #2, and collected the *httpry* logs to see if any of the URL/Password/SQLi blacklists were violated. *Thus, in this section we only compare the detection rates of the common deceptive elements that were applicable to both of the challenges.*

	Challenge #1 (deception informed)	Challenge #2 (no deception)
URL blacklist	13.1%	25.0%
Password blacklist	7.8%	10.5%
SQL blacklist	6.5%	0%
Cookie tampering	3.9%	5.3%
Cumulative	*18.4%*	*36.8%*

TABLE VI. COMPARISON OF DECEPTION EFFECTIVENESS WHEN THE PARTICIPANTS ARE INFORMED ABOUT IT OR NOT.

In total, 76 CTF participants attempted to solve both of these challenges, with $\tilde{3}0\%$ of success rate (23 and 24 flaggers, respectively). Thus, we can say that the difficulty levels of both challenges were similar. Table VI summarizes the ratio of participants who triggered each deception/detection element, and the cumulative results. We find that the effectiveness of deception is lower when the participants are informed about it (18%) in comparison to not being aware of it (37%). We also apply a two proportion z-test to see if this difference is statistically significant. For the significance level of 0.05, p-value of the test is 0.0088, which means the detection ability of deceptive elements significantly differ when the participants are aware of the use of deception.

Note that this result contradicts the previous work on data layer deception: In their study, Shabtai et al. [52] finds that "the knowledge about the existence of honeytokens did not have a significant influence on the percentage of illegal actions performed using honeytokens". However, in this study, use of a honeytoken brings an immediate benefit to the participants (increasing profit) while in our study the participants may not gain immediate benefit (e.g., launching a certain attack may or may not help with reaching the flag).

D. Discussion

Our experiment is conducted as a CTF exercise, and with a web application that is quite small-scale, far from how a real world hospital management application would look like. This means the participants know that there must be a vulnerability and they receive hints about where it could be located, compared to a real attack where the attacker does not know whether there is a vulnerability that is exploitable. Moreover, the participants are only information security enthusiasts, not real attackers. Thus, the 51% detection rate we reported in this study may not be a good indication of the real-world effectiveness of deception. However, the CTF setup is one of the best available methods to evaluate deception [31] and the several observations we make helps to improve the quality and robustness of the existing deceptive techniques.

We find that, adding and removing the deceptive elements only at the proxy level (like proposed in [30], [28]) may not be adequate in the long term. For more realistic and robust deception, it is also necessary to develop reasonable response actions, and mimic functionality for the deceptive elements (for example, by adding a broken admin panel to the user interface when a honey role cookie is set to "admin"). In fact, combining naive elements with more sophisticated ones can be the best approach to increase the ambiguity of the defense mechanism. More experiments are needed on a larger application with more deceptive elements that are better integrated with the application.

As the deception technology becomes more popular, attackers may assume its existence by default, or may know about it. We find that, being informed about deception is likely to decrease the effectiveness of deceptive elements (at least the naive ones); however, it still adds a deterrent factor and pushes the attackers away from their regular attack path.

VI. Conclusions

In this paper we propose a self-defense mechanism for web applications, that relies on using deceptive techniques on several layers. We aim to detect an attacker via application layer deceptive elements, redirect him to an application clone seamlessly, and serve him fake application data while we monitor his actions. We develop a prototype of this framework and experiment with it during a Capture-The-Flag exercise. Our results show how the attackers perceive deception, how they would try to mitigate it, and how existing deceptive elements can be improved. Although implementing such a complete deception framework in real world would bring many challenges (that we also discuss in the paper), we believe that deception has the potential to become an effective defense layer even if it is not perfectly executed. In the future work, we aim to address the open challenges we list, evaluate the performance overhead of our framework, and conduct more experiments in a real-world deployment.

Acknowledgments

We thank Elton Mathias, Julian Schoemaker, and the rest of the SAP Security Education team for providing the support to deploy our CTF challenge on their platform. We also thank the CTF participants for enabling this research, and the anonymous reviewers for their valuable feedback.

References

[1] "Spring Boot," https://spring.io/projects/spring-boot, 2019.

[2] "Spring Security Architecture ," https://spring.io/guides/topicals/spring-security-architecture, 2019.

[3] Acalvio, "Shadowplex autonomous deception," https://www.acalvio.com/why-acalvio/, 2019.

[4] E. Al-Shaer, J. Wei, K. W. Hamlen, and C. Wang, "Using deep learning to generate relational honeydata," in *Autonomous Cyber Deception*. Springer, 2019, pp. 3–19.

[5] F. Alaca and P. C. van Oorschot, "Device fingerprinting for augmenting web authentication: Classification and analysis of methods," in *Proceedings of the 32Nd Annual Conference on Computer Security Applications*, ser. ACSAC '16. New York, NY, USA: ACM, 2016, pp. 289–301.

[6] M. Almeshekah and E. Spafford, *Cyber Security Deception*, 07 2016, pp. 25–52.

[7] K. G. Anagnostakis, S. Sidiroglou, P. Akritidis, K. Xinidis, E. Markatos, and A. D. Keromytis, "Detecting targeted attacks using shadow honeypots," in *Proceedings of the 14th Conference on USENIX Security Symposium - Volume 14*, ser. SSYM'05. Berkeley, CA, USA: USENIX Association, 2005, pp. 9–9.

[8] Andrew Useckas, "Why security teams need to virtual patch," https://blog.threatxlabs.com/why-security-teams-need-to-virtual-patch/, 2019.

[9] F. Araujo, K. W. Hamlen, S. Biedermann, and S. Katzenbeisser, "From patches to honey-patches: Lightweight attacker misdirection, deception, and disinformation," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS'14. New York, USA: ACM, 2014, pp. 942–953.

[10] F. Araujo, M. Shapouri, S. Pandey, and K. Hamlen, "Experiences with honey-patching in active cyber security education," in *8th Workshop on Cyber Security Experimentation and Test (CSET 15)*. Washington, D.C.: USENIX Association, aug 2015.

[11] Attivo Networks, "Threat detection," https://attivonetworks.com/solutions/threat-detection/, 2019.

[12] M. Bercovitch, M. Renford, L. Hasson, A. Shabtai, L. Rokach, and Y. Elovici, "Honeygen: An automated honeytokens generator," in *Proceedings of 2011 IEEE International Conference on Intelligence and Security Informatics*, July 2011, pp. 131–136.

[13] B. M. Bowen, S. Hershkop, A. D. Keromytis, and S. J. Stolfo, "Baiting inside attackers using decoy documents," in *Security and Privacy in Communication Networks*, Y. Chen, T. D. Dimitriou, and J. Zhou, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 51–70.

[14] B. M. Bowen, V. P. Kemerlis, P. Prabhu, A. D. Keromytis, and S. J. Stolfo, "Automating the injection of believable decoys to detect snooping," in *Proceedings of the Third ACM Conference on Wireless Network Security*, ser. WiSec '10. New York, NY, USA: ACM, 2010, pp. 81–86.

[15] D. Brewer, K. Li, L. Ramaswamy, and C. Pu, "A link obfuscation service to detect webbots," in *2010 IEEE International Conference on Services Computing*, July 2010, pp. 433–440.

[16] E. Bursztein, A. Malyshev, T. Pietraszek, and K. Thomas, "Picasso: Lightweight device class fingerprinting for web clients," in *Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices*, ser. SPSM '16. New York, NY, USA: ACM, 2016, pp. 93–102.

[17] A. Cidon, L. Gavish, I. Bleier, N. Korshun, M. Schweighauser, and A. Tsitkin, "High precision detection of business email compromise," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, 2019, pp. 1291–1307.

[18] F. Cohen, I. Marin, J. Sappington, C. Stewart, and E. Thomas, "Red teaming experiments with deception technologies," http://all.net/journal/deception/RedTeamingExperiments.pdf, 2001.

[19] CrowdStrike, "Global Threat Report: Adversary Tradecraft and the Importance of Speed," 2019.

[20] Cymmetria, "Deception services," https://cymmetria.com/products/deception-services/, 2019.

[21] L. Dymytrova, L. Frigerio, and A. S. de Oliveira, "Differentially private generative models," https://github.com/SAP-samples/security-research-differentially-private-generative-models, 2019.

[22] D. Faraglia, "Faker: a python package that generates fake data for you," https://github.com/joke2k/faker, 2019.

[23] K. Ferguson-Walter, T. Shade, A. Rogers, E. Niedbala, M. Trumbo, K. Nauer, K. Divis, A. P. Jones, A. Combs, and R. G. Abbott, "The tularosa study: An experimental design and implementation to quantify the effectiveness of cyber deception," in *HICSS*, 2019.

[24] Fidelis Cybersecurity, "Fidelis deception," https://www.fidelissecurity.com/wp-content/uploads/2019/04/Fidelis-Deception-1905.pdf, 2019.

[25] Fireeye, "Fireeye Mandiant Services Special Report," 2019.

[26] D. Fraunholz and H. D. Schotten, "Defending web servers with feints, distraction and obfuscation," in *2018 International Conference on Computing, Networking and Communications (ICNC)*, March 2018, pp. 21–25.

[27] D. Fraunholz, S. D. Antón, C. Lipps, D. Reti, D. Krohmer, F. Pohl, M. Tammen, and H. D. Schotten, "Demystifying deception technology: A survey," *CoRR*, vol. abs/1804.06196, 2018.

[28] D. Fraunholz, D. Reti, S. Duque Anton, and H. D. Schotten, "Cloxy: A context-aware deception-as-a-service reverse proxy for web services," in *Proceedings of the 5th ACM Workshop on Moving Target Defense*, ser. MTD '18. New York, NY, USA: ACM, 2018.

[29] D. Gavrilis, I. Chatzis, and E. Dermatas, "Flash crowd detection using decoy hyperlinks," in *2007 IEEE International Conference on Networking, Sensing and Control*, April 2007, pp. 466–470.

[30] X. Han, N. Kheir, and D. Balzarotti, "Evaluation of deception-based web attacks detection," in *Proceedings of the 2017 Workshop on Moving Target Defense*, ser. MTD '17. New York, NY, USA: ACM, 2017.

[31] ——, "Deception techniques in computer security: A research perspective," *ACM Comput. Surv.*, vol. 51, no. 4, pp. 80:1–80:36, Jul. 2018.

[32] G. Ho, A. Cidon, L. Gavish, M. Schweighauser, V. Paxson, S. Savage, G. M. Voelker, and D. Wagner, "Detecting and characterizing lateral phishing at scale," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, 2019, pp. 1273–1290.

[33] K. Houkjær, K. Torp, and R. Wind, "Simple and realistic data generation," in *Proceedings of the 32Nd International Conference on Very Large Data Bases*, ser. VLDB '06. VLDB Endowment, 2006, pp. 1243–1246.

[34] Illusive Networks, "Attack detection system," https://www.illusivenetworks.com/technology/platform/attack-detection-system, 2019.

[35] M. Izagirre, "Deception strategies for web application security: application-layer approaches and a testing platform," MSc Thesis at Lulea University of Technology, June 2017.

[36] S. Jajodia, V. Subrahmanian, V. Swarup, and C. Wang, *Cyber deception: Building the scientific foundation*, 01 2016.

[37] P. Kaghazgaran and H. Takabi, "Toward an insider threat detection framework using honey permissions," *J. Internet Serv. Inf. Secur.*, vol. 5, pp. 19–36, 2015.

[38] A. Kerckhoffs, "La cryptographie militaire," *Journal des Sciences Militaires*, pp. 161–191, 1883.

[39] Kevin Schaller, "XML External Entity (XXE) Injection in Apache Batik Library [CVE-2015-0250]," https://insinuator.net/2015/03/xxe-injection-in-apache-batik-library-cve-2015-0250/, March 2015.

[40] G. Kontaxis, M. Polychronakis, and A. Keromytis, "Computational decoys for cloud security," *Secure Cloud Computing*, pp. 261–270, 11 2013.

[41] P. Laperdrix, G. Avoine, B. Baudry, and N. Nikiforakis, *Morellian Analysis for Browsers: Making Web Authentication Stronger with Canvas Fingerprinting*, 06 2019, pp. 43–66.

[42] J. Lin, C. Liu, X. Cui, and Z. Jia, "Poster: A website protection framework against targeted attacks based on cyber deception," 2017.

[43] M. A. E. Mohd Efendi, Z. Ibrahim, M. N. Ahmad Zawawi, F. Abdul Rahim, N. A. Mohamad Pahri, and A. Ismail, "A survey on deception techniques for securing web application," in *2019 IEEE 5th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*, May 2019, pp. 328–331.

[44] NTT Security, "The rapid evolution of deception technologies," https://www.nttsecurity.com, 2018.

[45] OWASP, "Server Side Request Forgery," 2019.

[46] OWASP Foundation, "Appsensor detection points," https://www.owasp.org, 2015.

[47] ——, "Appsensor response actions," https://www.owasp.org, 2015.

[48] J. Pawlick, E. Colbert, and Q. Zhu, "A game-theoretic taxonomy and survey of defensive deception for cybersecurity and privacy," *CoRR*, vol. abs/1712.05441, 2017.

[49] C. Pohl, A. Zugenmaier, M. Meier, and H.-J. Hof, "B.hive: A zero configuration forms honeypot for productive web applications," in *30th IFIP International Information Security Conference (SEC)*, May 2015, pp. 267–280.

[50] PortSwigger Ltd., "XML external entity (XXE) injection," https://portswigger.net/web-security/xxe, 2019.

[51] M. B. Salem and S. J. Stolfo, "Decoy document deployment for effective masquerade attack detection," in *Proceedings of the 8th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, ser. DIMVA'11. Berlin, Heidelberg: Springer-Verlag, 2011.

[52] A. Shabtai, M. Bercovitch, L. Rokach, Y. K. Gal, Y. Elovici, and E. Shmueli, "Behavioral study of users when interacting with active honeytokens," *ACM Trans. Inf. Syst. Secur.*, vol. 18, no. 3, Feb. 2016.

[53] Symantec, "Internet Security Threat Report," https://www.symantec.com/content/dam/symantec/docs/reports/istr-24-2019-en.pdf, February 2019.

[54] ThinkstCanary, "Canarytokens," https://canarytokens.org, 2019.

[55] K. Thomas, J. Pullman, K. Yeo, A. Raghunathan, P. G. Kelley, L. Invernizzi, B. Benko, T. Pietraszek, S. Patel, D. Boneh, and E. Bursztein, "Protecting accounts from credential stuffing with password breach alerting," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, aug 2019, pp. 1556–1571.

[56] S. Tzu, *The Art of War*, ser. Dover Military History, Weapons, Armor. Dover Publications, 2002.

[57] V. E. Urias, W. M. S. Stout, and H. W. Lin, "Gathering threat intelligence through computer network deception," in *2016 IEEE Symposium on Technologies for Homeland Security (HST)*, May 2016, pp. 1–6.

[58] V. E. Urias, W. M. S. Stout, J. Luc-Watson, C. Grim, L. Liebrock, and M. Merza, "Technologies to enable cyber deception," in *2017 International Carnahan Conference on Security Technology (ICCST)*, Oct 2017, pp. 1–6.

[59] A. van der Heijden and L. Allodi, "Cognitive triaging of phishing attacks," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, 2019, pp. 1309–1326.

[60] C. Watson, M. Coates, J. Melton, and D. Groves, "Creating attackaware software applications with real-time defenses," September/October 2011.

[61] C. Watson, J. Melton, and D. Groves, "Appsensor application-specific real time attack detection & response," July 2015.

[62] J. Yuill, D. Denning, and F. Feer, "Psychological vulnerabilities to deception for use in computer security," in *DoD Cyber Crime Conference*, January 2007.