Deception Techniques In Computer Security: A Research Perspective

XIAO HAN, Orange Labs, France
NIZAR KHEIR, Thales, France
DAVIDE BALZAROTTI, Eurecom, France

A recent trend both in academia and industry is to explore the use of deception techniques to achieve proactive attack detection and defense — to the point of marketing intrusion deception solutions as zero-false-positive intrusion detection. However, there is still a general lack of understanding of deception techniques from a research perspective and it is not clear how the effectiveness of these solutions can be measured and compared with other security approaches. To shed light on this topic, we introduce a comprehensive classification of existing solutions and survey the current application of deception techniques in computer security. Furthermore, we discuss the limitations of existing solutions and we analyze several open research directions, including the design of strategies to help defenders to design and integrate deception within a target architecture, the study of automated ways to deploy deception in complex systems, the update and re-deployment of deception, and most importantly, the design of new techniques and experiments to evaluate the effectiveness of the existing deception techniques.

CCS Concepts: • **Security and privacy → Intrusion detection systems**; **Systems security**; **Network security**; **Software and application security**;

Additional Key Words and Phrases: Deception

ACM Reference Format:
Xiao Han, Nizar Kheir, and Davide Balzarotti. 2019. Deception Techniques In Computer Security: A Research Perspective. *ACM Comput. Surv.* 1, 1 (January 2019), 36 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In recent years, we have witnessed a surge in advanced cyber attacks, that are rapidly increasing both in number and in sophistication [Symantec 2016]. This threat affects both enterprises and end-users alike and inflicts severe reputation, social, and financial damage to its victims [Trend Micro 2015]. Unfortunately, despite the numerous efforts to raise awareness against cyber attacks, attackers are still able to infiltrate their target systems, leveraging multiple attack vectors such as zero-day vulnerabilities, flaws in software configuration, access control policies, or by social engineering their target into installing or running malicious software.

To thwart this vicious trend, the security community has proposed and developed numerous solutions to enhance the security of networks and information systems. Current solutions cover all traditional aspects of security, including intrusion prevention [Scarfone and Mell 2007], system hardening [Nguyen-Tuong et al. 2005], and advanced attack detection and mitigation [Ashfaq

Authors' addresses: Xiao Han, xiao.han@orange.com, Orange Labs, 44 Avenue de la République, Châtillon, 92320, France; Nizar Kheir, nizar.kheir@thalesgroup.com, Thales, 1 Avenue Augustin Fresnel, Palaiseau, 91767, France; Davide Balzarotti, davide.balzarotti@eurecom.fr, Eurecom, 450 Route des Chappes, Biot, 06410, France.

0360-0300/2019/1-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

et al. 2017]. These traditional measures, although essential in any modern security arsenal, cannot provide a comprehensive solution against Internet threats. Hence, complementary solutions have been recently investigated, with the aim to be more proactive in anticipating threats and possibly warn against attacks in their very early stages. In particular, deception techniques have attracted a considerable amount of interest in the research community [Almeshekah and Spafford 2014a; Anton 2016]. These techniques, initially inspired by the use of deception in the military domain, consists of orchestrating a *"planned set of actions taken to mislead attackers and to thereby cause them to take (or not to take) specific actions that aid computer-security defenses"* [Yuill 2006].

One of the first documents to mention the term *deception* as a way to enhance the security of computer systems is "The Cuckoo's Egg" book [1989], published by Cliff Stoll in 1989. The author describes how he set up a fictitious system environment, including a user account populated with a large number of fake documents to deliberately attract and delay an attacker while tracking his incoming connections to reveal his origin and identity. Such fictitious environment, designed to capture the interaction with an attacker, later became popular as a *honeypot* [Lance 2001]. In 1994, a system called Tripwire [Kim and Spafford 1994a] offered a new approach for file integrity checking by planting fictitious files and placing them under a file integrity monitor to detect intruders [Kim and Spafford 1994b]. The concept of planted fictitious files have then evolved into a diversified set of solutions including *honeyfiles* [Yuill et al. 2004] and *decoy* documents [Bowen et al. 2009].

In the following years, deception has been routinely used to protect computer systems and networks. Honeypots became a popular solution for the monitoring, analysis, understanding, and modeling of attackers' behavior [Cohen 1998; Nawrocki et al. 2016; Provos et al. 2004; Spitzner 2003a]. In 2003, soon after the first computer honeypot was released, the concept of *honeytoken* was proposed by Augusto Paes de Barros [2003]. Then Spitzner [2003c] broadened this concept to cover "*a digital or information system resource whose value lies in the unauthorized use of that resource*" and brought it to the attention of a larger public. Spitzner illustrated practical examples of honeytokens, including bogus social security numbers and fake credentials, which have later inspired many follow-up contributions including the use of honey passwords [Juels and Rivest 2013], honey URL parameters [Petrunić 2015], database honey tokens [Bercovitch et al. 2011; Čenys et al. 2005], and honey permissions [Kaghazgaran and Takabi 2015].

The use of deception techniques for computer security has been widely driven by the analogy with more conventional deceptive military tactics. In this scope, the concept has been broadened towards intrusion detection (IDS) and prevention (IPS) systems. Instead of indicating to the intruder a violation of a security policy, deception consists in responding to attackers with some pre-defined decoy actions, such as fake protocol messages [Goh 2007], response delays [Julian 2002], and crafted error messages [Michael et al. 2002]. For example, instead of blocking a detected intrusion on the web server, Katsinis and Kumar [2012] examined the use of a deceptive module that analyzes incoming HTTP requests and advises deceptive responses in case of an ongoing attack.

Most recently, during the French President election campaign, the digital team of Mr. Macron created fake accounts with a large number of fake documents [Nossiter et al. 2017]. By leaking these accounts to phishing attacks, they successfully delayed the advancement of the attackers and confused them with a large quantity of fake information. Moreover, the attackers rushed to modify some of the leaked documents, leaving some traces in their meta-data that provided information about their identity.

Scope of this Document

During our efforts to summarize existing work on deception techniques, we observed that many traditional security solutions already implement deception up to a certain level, as already observed by Cohen [1998]. For example, widely known techniques such as software diversity [Larsen et al. 2014], as well as anti-forensics techniques [Garfinkel 2007], are deceptive in nature, as they aim to hide and conceal system information against unauthorized access. Similarly, security scanners and crawlers mostly behave in a deceptive way in order to mimic normal users and hide their real purposes. Malware analysis systems also try to deceive malicious binaries by executing them in an instrumented sandbox as if they were running on a real victim terminal. To further complicate this already multifaceted domain, the existing literature often adopts a confusing terminology — interchangeably using terms such as concealment, fakes, baits, decoys, traps, and honey[pots|tokens|files|accounts|...].

This makes the concept of deception too broad to cover in a single document — as it manifests in different forms in almost all security domains. Therefore, in the rest of the document, we will only cover the techniques that are used as a **defense mechanism** to deceive an attacker while he is interacting with a target system. This definition (sometimes called *Intrusion Deception*) includes only the solutions that can be applied to secure real systems. It rules out other loose forms of deception adopted in the security field (such as faking the user agent of a web scanner) as well as standalone deceptive systems used to observe generic attacks and to monitor the overall threat landscape (such as conventional honeypots [Nazario 2009] or other tools developed by the Honeynet Project [1] with the goal of collecting information about attackers).

Moreover, we distinguish deception techniques from general moving-target defense techniques that mainly consist of adding dynamics (or movements) to a static system to augment the adversary's workload and reduce the chance of its successful attacks. Typical moving-target defense techniques include Address Space Layout Randomization (ASLR), software diversity, code obfuscation, and IP address shuffling [Okhravi et al. 2013]. Even though both deception and moving-target defense techniques may share common objectives, the way to achieve such objectives differ: one adds randomness and the other seeks direct engagement with the attacker. Note that moving-target defense techniques may also be applied for deception [Okhravi et al. 2014] and vice-versa [Han et al. 2017]. However, in this survey, we do not consider general moving-target defense techniques that merely increase the dynamics of a static system. This separation between moving-target defense and deception techniques has also been observed in the literature. For example, Crouse et al. [2015] compared the probabilistic performance of moving-target and deception techniques and Cohen [2010] showed that moving-target defense techniques may be enhanced by adding a deceptive cover to them. Readers interested in moving-target defense techniques may refer to related work, such as [Jajodia et al. 2012, 2011].

Finally, note that deception may also be used as an evasion technique, like in the case of decoy routing [Karlin et al. 2011; Nasr et al. 2017]. Decoy routing is designed to circumvent IP address-based network filtering by leveraging a decoy destination. A decoy router that supports a covert channel is implemented on the path between the user and the decoy destination. Therefore, the user is able to require and access filtered content via the covert channel. Nonetheless, for the sake of clarity and to narrow down the scope of this survey, we mainly focus on deception techniques in the literature that have been used for defensive purposes.

Survey Organization

In this paper, we survey the use of deception techniques in computer security as a defensive measure, while narrowing the scope of this survey to our definition of deception as introduced in Section 1. The first application of deception in the computer security field relates back to the early 1990s, largely inspired by its use in the military domain. Different forms of deception have been further explored in the cyber space, driving many efforts and leading to many technical

[1]http://honeynet.org

contributions and solutions over the last two decades [Jajodia et al. 2016; Rowe and Rrushi 2016]. Nonetheless, from a research perspective, we still lack a common understanding about deception techniques and their application in information and communication systems security. In particular, there is not yet a wide consensus among the research community about what the main goals are and technical challenges to achieve when using deception techniques as a defense mechanism. The real benefits for such techniques are also not clearly emphasized, nor the way deception techniques may complement other traditional security solutions. The modeling, deployment, update, and evaluation of deception techniques are also scarcely addressed in the literature — often resorting to qualitative statements and thus making it difficult to compare such techniques with other approaches.

Motivated by these key questions, this paper presents a systematic organization of previous work on deception techniques and their application in the cyber security domain. It discusses the main technical challenges that were addressed using these techniques and emphasizes the gaps and the pieces of the puzzle we still miss in this complex field of research.

This paper is organized as follows. Section 2 introduces our motivations, together with the proposed scheme and related work. Section 3 surveys the current and featured applications of deception techniques for cyber defense. Section 4 discusses the existing deception modelling approaches and the way they design and integrate deception techniques inside a target system. Section 5 discusses the different deployment scenarios, as well as the enforcement and monitoring of deception techniques. Section 6 overviews the different ways how deception techniques have been evaluated in the literature. Finally, Section 7 provides insights for future work and Section 8 concludes.

2 SURVEY METHODOLOGY

The goal of this survey is to investigate the role of deception techniques in computer security from a research perspective. To the best of our knowledge, there is no comprehensive survey that summarizes and classifies deception techniques in this context, which would considerably help to better structure and advance this field of research. Moreover, an increasing number of calls to conduct "scientific" security research have emerged in the last few years [Herley and van Oorschot 2017]. Nevertheless, the authors in [Herley and van Oorschot 2017] found that *"there is little clarity on what scientific means in the context of computer security research"*. Among other results, authors pointed out that unfalsifiable justification such as "deception improves security" are frequent and might result in pileup of (unnecessary) countermeasures. As some security scientific models failed to compare their outcomes with observable (real-world) measurements, the authors recommended that *"calls for more science should specify desired attributes, specific sources of dis-satisfaction with current research, and preferred types of research"*. We will try to address some of these points in this document.

2.1 Proposed Scheme

Motivated by the above factors, this paper discusses all these issues regarding deception techniques and provides an overview of deception-based defenses in computer security. The emphasizes is on the research gap, through focusing on open problems related to the modeling, deployment, and evaluation of deception techniques:

1. **Classification and Current Applications:** A first question to address when referring to deception techniques is why we should use such techniques and how they complement other traditional security mechanisms [Almeshekah and Spafford 2014a,b]. Only two decades ago, researchers were still wondering whether deception may be acceptable from a legal and ethical perspective [Cohen 1998]. Today, it is obvious that such questions are no longer an issue,

opening the door to a large number of proprietary (e.g., TRAPX[2], CANARY[3], ThreatMatrix[4]) and open source (e.g., DCEPT[5], Canarytokens[6]) solutions that explicitly mention and use deception for computer security defense. A main goal of this paper is thus to propose a comprehensive classification, survey the current applications of deception in computer security, and shed light on the key technical domains where deception has been implemented.

2. **Modeling:** Deception modeling refers to the set of rules and strategies that drive the defender into designing and integrating deception within the architecture and workflow of a target system. During our survey of scientific literature, we identified multiple deception strategies that leverage different properties of the target system, and different assumptions about the attacker behavior. For example, Cohen et al. [2002; 2001b] leveraged the use of attack graphs, which provide a compact representation of known attack scenarios and the way attackers can break into a target system. The authors integrated deception techniques in different positions of the graph, in order to lure attackers and to drive them towards fake targets. In [Carroll and Grosu 2011], authors proposed an alternative approach, leveraging models based on game theory, in order to describe different interaction sequences between attackers and the target system, and to apply deception while maximizing the defender's gain.
   In this paper, we survey existing deception modeling techniques and we discuss their main benefits and limitations. In particular, we observe that most deception modeling techniques either cover only specific attacks and use cases or they try to adapt and implement high-level strategies whose applicability remains questionable.
3. **Deployment:** Another key aspect of deception is the way it is deployed inside a target environment. The mode of deployment refers to the technical environment where the deception is being implemented inside a target system [Rowe and Rrushi 2016]. In this scope, deception may be either implemented as a standalone solution (e.g., honeypots) or fully integrated within the target system (e.g., fake documents [Voris et al. 2015]). Moreover, deployment also includes the placement strategies (either manual or automated) that place deception at specific locations within the architecture and workflow of a target system. To be effective, deception elements need to be generated to achieve perfect realism with regard to real data and system integration. However, it is still unclear how the deployment and placement strategy affects the effectiveness of the solution. Moreover, while the importance of re-training machine learning models or update attack signatures is well known in the intrusion detection domain, the importance of updating the deception element is still largely unexplored.
4. **Evaluation:** The last aspect that we cover in this survey is the experimental setup and the way previous work evaluate the proposed deception techniques. We discuss the main challenges researchers are faced with when evaluating deception and how these problems may drastically limit the soundness of the obtained results (both regarding the efficiency and the accuracy of the proposed solutions). Other solutions are typically evaluated by measuring their detection and false alarm rates — both of which are difficult to assess in the deception domain.
   We conclude by providing insights and discussing key features that should be taken into account in the future when evaluating deception techniques.

[2]https://trapx.com/
[3]https://canary.tools/
[4]https://attivonetworks.com/product/deception-technology/
[5]https://github.com/secureworks/dcept
[6]https://github.com/thinkst/canarytokens

2.2 Previous surveys

In [2003b], Spitzner discussed the use of honeypot and honeytoken technologies as a way to protect against the insider threat. In [2008], Rowe discussed different possibilities to integrate deception in honeypot systems — such as decoy information, delays, and fake error messages — and compared them to other opportunities for using deception in real computer systems. Voris et al. [2013] discussed the multiple use cases where decoys may be relevant for computer security. Juels and Tech [2014] discussed the use of honey objects (a generic term they used to refer to multiple types of deception) to improve the security of information systems. Jogdand and Padiya [2016] also analyzed IDS solutions and the way they may implement honey tokens, which are indeed a specific type of deception. In [2001a], Cohen et al. overviewed multiple issues that arise when applying deception in computer security, and further introduced their own framework for deception. The authors also discussed the major challenges to perform practical deceptions using this framework to defend information systems. Cohen also surveyed, in [2006], the historical and recent uses (until 2005) of honeypots and decoys for information protection, as well as the theory behind deceptions and their limitations. Our work is different as it surveys the recent contributions, focusing on the technical challenges and evaluations of deception, rather than studying the history of this concept and how it found its way into computer security.

More recently, multiple studies advocated the use of deception techniques to protect information systems and react against potential intruders [Almeshekah and Spafford 2014a; Jajodia et al. 2016; Rowe and Rrushi 2016; Virvilis et al. 2014]. These papers contributed to promote deception techniques and to provide interesting applications and use cases. However, they mostly introduced technical contributions and none of them aimed to classify and to provide a comprehensive survey on deception techniques and their use in computer security.

3 CLASSIFICATION

In this section, we introduce a classification of deception techniques along four orthogonal dimensions, including the unit of deception, the layer where deception is applied, the goal of the deception solution, and its mode of deployment.

3.1 Previous Classifications

Early deception classification schemes followed a traditional military deception classification scheme. For example, Cohen [1998] examined deception techniques based on the nature of these techniques, including the *"concealment, camouflage, false and planted information, ruses, displays, demonstrations, feints, lies and insight"*. This historical examination revealed that deception was far from being fully exploited in computer security. The same type of taxonomy has also been used in later works [Rowe and Rothstein 2004; Rowe and Rrushi 2016].

Rowe and Rothstein [2004] developed a theory of deception that is inspired from the computational linguistic theory of semantic cases. The semantic case roles include several dimensions such as participant (agent, beneficiary), space (direction, location), time, causality (cause, purpose), and quality (content, value). The authors explained a deception operation as an action that modifies the values of the associated case role. Cohen [2006] proposed a model of computer deceptions that groups deceptions by the hierarchical level of the computer at which they are applied, from the lowest hardware level to the highest application level. Information and signals between different levels can either be induced or inhibited in order to implement deception. Similarly, Almeshekah and Spafford [2014b] classified deceptions based on the system state and components where deception may be deployed. The top-level categories include the functionality of the system (system decisions, software and services, internal data) and state of the system (activities, configurations, and performance).

Gartner [2015] proposed a four layer deception stack including the network, endpoint, application, and data layers. It further analyzed the deception techniques implemented by current commercial products. Both Gartner [2015] and Almeshekah and Spafford [2014a] have examined the possibilities to deploy deception techniques during the different phases of a cyber attack, without any systematic classification.

3.2 Multi-Dimension Classification

Because of the wide spectrum of deception techniques and the different ways they may apply to computer security, it is very difficult to classify all of them along a single dimension that successfully incorporates the different research aspects. Traditional military deception classifications (such as [Cohen 1998]) and semantic classifications (such as [Rowe and Rothstein 2004]) allowed us to understand the use of deception techniques from different angles, which however are difficult to match with the needs of computer security. Moreover, most previous classifications considered only one dimension, such as the component, the granularity of each technique [Almeshekah and Spafford 2014b], or the layer where deception is applied [Pingree 2015]. Unfortunately, these mono-dimensional classifications miss other aspects of deception that are of equal importance, such as the threats covered by each technique, and the way they can be integrated inside a target system.

To address these limitations, we introduce in this section a new classification system based on four orthogonal dimensions. In our system, categories are mutually exclusive, which means that each existing deception technique may be classified within only one single four-dimensional vector. Our classification is also complete, so all existing deception techniques find a place in our schema. The four dimensions that we use for our classification are defined as follows.

Goal of deception captures the main purpose a specific deception technique is trying to achieve. This could be either to improve and complement attack detection, to enhance prevention, or to mitigate successful attacks. The first category includes those solutions designed to detect an attack, typically because it interacts with active traps or because it uses passive decoy information that are intentionally left accessible to be discovered by the attacker (e.g., decoy documents [Ben Salem and Stolfo 2011]). The second category covers instead those mechanisms that aim at confusing or distracting attackers from the real targets before an attack occurs (e.g., the deployment of deceptive network topology [Trassare 2013]). Finally, the last category targets on-going attacks and tries to reduce their damage, for instance by replying in a delayed manner [Julian 2002] or by redirecting attackers to a safe copy of the target system [Anagnostakis et al. 2005; Araujo et al. 2014].

Unit of deception refers to the granularity of the decoy asset that is used to implement deception. We use the same definition of granularity introduced by Almeshkah et al. [2014b], which includes the following units of deception: *decision* (e.g., by accepting a connection towards an unused IP address [Borders et al. 2007; Liston 2001]), *response* (e.g., a forged network response [Borders et al. 2007]), *service* (e.g., a decoy service [Cohen et al. 1998]), *activity* (e.g., a decoy computation activity [Kontaxis et al. 2014]), *weakness* (e.g., a simulated vulnerability [Araujo et al. 2014]), *performance* (e.g., a delayed response [Julian 2002]), *configuration* (e.g., a fake network topology [Trassare 2013]) and *data*.
We further refine the data unit into separate types, such as *parameter* (e.g., a honey URL parameter [Petrunić 2015] or a honey form field [Katsinis et al. 2013]), *file* (e.g., honeyfiles [Yuill et al. 2004] and decoy documents [Ben Salem and Stolfo 2011]), *account* (e.g., honey account [Bowen et al. 2010a; Chakravarty et al. 2011]), *user profile* (e.g., honey profiles [Webb et al. 2008]),

source code (e.g., decoy source code elements [Park and Stolfo 2012]) and *database record* (e.g., database honey token [Čenys et al. 2005]).

Layer of deception indicates the layer in which the deception element is applied. While there are different ways to organize the layers of an information system, we use a standard classification divided in network, system, application, and data layers. The network layer covers deception techniques that are accessible over the network and that are not bound to any specific host configuration. The system layer covers host-based deception techniques. The application layer covers deception techniques that are linked to specific classes of applications, such as web applications or databases. Finally, the data layer covers deception techniques that leverage user-specific data, such as fake accounts or fake documents.

Deployment of deception characterizes the way a deception technique may be integrated within a target system. Possible deployment modes include (1) *built-in* deception solutions that are integrated in the system at the design phase (e.g., in the source code [Julian 2002]), (2) deceptions that are *added-to* the system during operation (e.g., documents inserted in a file system [Voris et al. 2015]), (3) deceptions that are set *in-front of* a target system (such as a proxy or gateway [Brewer et al. 2010]), and finally (4) *isolated* solutions separated from the target system (e.g., fake accounts of a decoy server [Bowen et al. 2010a; Chakravarty et al. 2011]).

Table 1 presents a survey of previous work on deception and classifies them over the four aforementioned dimensions. Note that few studies that are referenced in Table 1 do not explicitly mention deception in their core contributions. Moreover, since deception is a generic concept that may apply to different aspects of security, certain studies introduce deception as an *add-on* functionality within a more comprehensive security framework. Table 1 lists existing work as long as their contribution fits with our definition of deception introduced in Section 1.

3.3 Overview of Intrusion Deception Techniques

During our survey, sometimes we found it particularly challenging to associate a given publication to a specific deception category, since many techniques may have been presented in a single work. Therefore, to keep a consistent classification of existing work on deception, we associate each publication to its main deception technique or to the one that has not been introduced before in case of multiple contributions.

For sake of clarity, since our classification leverages four distinct dimensions, we organize this section based on the layer of deception. But we discuss all four dimensions as part of our classification.

Layer I: Network

The network-based deception techniques we observed in the literature were mostly designed to address three categories of threats: *network fingerprinting*, *eavesdropping*, and *infiltration and attack propagation.*

Scanning & Fingerprinting. These typically occur during the early stage of an attack, in particular the reconnaissance phase, and enable an attacker to acquire information about the network topology and available assets by fingerprinting and scanning the network.

One of the first deception techniques that offered to interfere with the reconnaissance phase in order to obfuscate its results relied on sinkholing attack traffic, by redirecting malicious traffic to a set of fake machines that mimic the behavior of real terminals on the network [Borders et al. 2007]. Such decoy machines are commonly known as network tarpits [Liston 2001]. They create sticky connections with the aim to slow or stall automated scanning and to confuse human adversaries.

Reference	Technique	Unit	Layer	Goal	Deployment
[Liston 2001]	Network Tarpit	Decision	Network	Mitigation	Added-to
[Borders et al. 2007]	Network Tarpit	Decision	Network	Mitigation	In-front of
[Shing 2016]	Network Tarpit	Decision	Network	Mitigation	Added-to
[Le Malécot 2009]	Traffic forging	Response	Network	Mitigation	In-front of
[Trassare 2013]	Deceptive topology	Configuration	Network	Prevention	In-front of
[Smart et al. 2000]	OS obfuscation	Response	Network	Mitigation	In-front of
[Bowen et al. 2010a]	Honey accounts	Configuration	Network	Detection	Isolated
[Chakravarty et al. 2011]	Honey accounts	Configuration	Network	Detection	Isolated
[Cohen et al. 2001b]	Deceptive attack graph	Configuration	Network	Prevention	Added-to
[Cohen and Koike 2002]	Deceptive attack graph	Configuration	Network	Prevention	Added-to
[Cohen et al. 1998]	Decoy services	Service	Network	Prevention	Added-to
[Provos et al. 2004]	Decoy services	Service	Network	Detection	Added-to
[Rrushi 2011]	Deceptive simulation	Activity	Network	Prevention	Added-to
[Kontaxis et al. 2014]	Decoy computation	Activity	System	Prevention	In-front of
[Urias et al. 2016]	Deceptive simulation	Service	System	Mitigation	Added-to
[Wang et al. 2013]	Multi-layer deception	Multiple	Multiple	Detection	Added-to
[Murphy et al. 2010]	OS obfuscation	Parameter	System	Mitigation	Added-to
[Rowe et al. 2006]	Fake honeypots	Configuration	System	Mitigation	Added-to
[Kaghazgaran and Takabi 2015]	Honey permissions	Configuration	System	Detection	Built-in
[Rrushi 2016]	Decoy network device	Service	System	Detection	Added-to
[Michael et al. 2002]	Software Decoys	Response	Application	Detection	In-front of
[Araujo et al. 2014]	Honey patches	Weakness	Application	Mitigation	In-front of
[Crane et al. 2013]	Software trap	Weakness	Application	Detection	Added-to
[Julian 2002]	Delayed response	Performance	Application	Mitigation	Built-in
[Anagnostakis et al. 2005]	Shadow honeypots	Service	Application	Mitigation	In-front-of
[Brewer et al. 2010]	Decoy hyperlinks	Parameter	Application	Detection	In-front of
[Gavrilis et al. 2007]	Decoy hyperlinks	Parameter	Application	Detection	Added-to
[McRae and Vaughn 2007]	Honey URL	Parameter	Application	Detection	Added-to
[Petrunić 2015]	Honey URL parameters	Parameter	Application	Detection	Built-in
[Katsinis et al. 2013]	Decoy form field	Parameter	Application	Detection	In-front of
[Virvilis et al. 2014]	Honey accounts	Configuration	Application	Detection	Added-to
[Bojinov et al. 2010]	Honey password	Account	Data	Prevention	Added-to
[Juels and Rivest 2013]	Honey password	Account	Data	Detection	Added-to
[Juels and Ristenpart 2014]	Honey encryption	Account	Data	Prevention	Built-in
[Bowen et al. 2010b]	Honey accounts	Account	Data	Detection	Isolated
[Akiyama et al. 2013]	Honey accounts	Account	Data	Detection	Isolated
[Onaolapo et al. 2016]	Honey accounts	Account	Data	Detection	Isolated
[Yuill et al. 2004]	Honeyfiles	File	Data	Detection	Added-to
[Lazarov et al. 2016]	Honeyfiles	File	Data	Detection	Added-to
[Čenys et al. 2005]	Database honey tokens	Record	Data	Detection	In-front of
[Bercovitch et al. 2011]	Database honey tokens	Record	Data	Detection	Added-to
[Padayachee 2014]	Database honey tokens	Record	Data	Detection	Built-in
[Park and Stolfo 2012]	Decoy source code	Code	Data	Detection	Added-to
[Bowen et al. 2009]	Honeyfiles	File	Data	Detection	Added-to
[Ben Salem and Stolfo 2011]	Honeyfiles	File	Data	Detection	Added-to
[Voris et al. 2015]	Honeyfiles	File	Data	Detection	Added-to
[Nikiforakis et al. 2011]	Honeyfiles	File	Data	Detection	Added-to
[Liu et al. 2012]	Honeyfiles	File	Data	Detection	Added-to
[Kapravelos et al. 2014]	Honey web pages	File	Data	Detection	Added-to
[Webb et al. 2008]	Honey profiles	Profile	Data	Detection	Added-to
[Stringhini et al. 2010]	Honey profiles	Profile	Data	Detection	Added-to
[De Cristofaro et al. 2014]	Honey profiles	Profile	Data	Detection	Added-to

Table 1. Detailed overview of deception techniques

Shing [2016] further brought a few improvements to this idea by tuning the different options of the sticky connections to conceal network tarpits and prevent them from being easily recognized.

Another way to lure attackers and to randomize the outcome of their fingerprinting attempts was proposed by Le Malécot in [Le Malécot 2009]. They authors introduced a technique to skew the topology of the target network through random connection dropping and traffic forging. For example, Trassare [2013] defeated a particular type of scan, traceroute probes, by misleading an attacker through constantly revealing a false network topology.

Another type of threat, OS Fingerprinting, allows an attacker to gain valuable information about available Operating Systems (OS) and so to identify potential flaws and vulnerabilities. To conceal OS-related information and prevent it from being fingerprinted by an attacker (e.g., using the Nmap tool), multiple deception techniques have been proposed — that offer to mimic the network behavior of fake operating systems [Murphy et al. 2010; Smart et al. 2000] in an effort to mislead potential attackers. The use of deception techniques against this category of threat mainly aims at confusing attackers and delaying their advancement.

Eavesdropping. Network eavesdropping is a challenging attack to detect using traditional detection systems since the attacker usually remains silent on the network. Deception techniques have been introduced as an alternative way to protect against such attacks. A deception technique that is particularly relevant in this area is the use of honey accounts. These are usually dummy credentials that are shared over the network in order to be captured and used by elusive attackers, thus revealing their presence in specific networks such as wireless [Bowen et al. 2010a] and Tor [Chakravarty et al. 2011] networks.

Infiltration & Propagation. Cohen et al. in [2002; 2001b] conducted red teaming experiments on deception, through leveraging structured attack graph representations, in order to drive attackers into following fake attack paths that would distract them from their real targets. Similarly, HoneyD [Provos et al. 2004] and the Deception ToolKit [Cohen et al. 1998] offered to spawn multiple fictitious services and network IP addresses in order to fool the attackers and make them attack false targets. In addition to their application in traditional networks, deceptive simulation techniques [Rrushi 2011] have also been applied in the context of industrial control systems. These tools offer to monitor the network topology and to create fake but indistinguishable attack targets that are aimed to fool attackers and conceal potential victims.

Layer II: System

Deception techniques that have been implemented at the system layer were mainly aimed at addressing two categories of threats, namely *external attacks* and *insider threats.*

System Compromise. Wang et al. [2013] suggested the use of deception techniques in an effort to enhance the detection of system compromise attempts by introducing a multi-layer decoy framework that included decoys for user profiles, files, servers, and network / system activity. These decoys were supposed to conceal the real assets of an organization and protect them against targeted attacks. In a similar approach, Rrushi [2016] proposed the use of a decoy Network Interface Controller (NIC) for Windows operating systems. This decoy interface was intentionally set in order to lure and detect malicious software that may be running on the system. The idea is that benign software is not expected to use the decoy interface, which is how the authors were able to detect other malicious applications running on the system.

As opposed to the previous techniques that were mainly aimed at enhancing detection, Rowe et al. [2006] adopted a proactive deception approach that aims at preventing system compromise attempts. They suggested the use of fake honeypots that make ordinary but critical systems appear as real honeypots, which may confuse an attacker and turn him away from the compromised system. Similarly, Urias et al. [2016] offered to clone and migrate machines that are suspected of being compromised and to place them in a deceptive environment where network and system configurations are duplicated to mimic the real network environment. Lastly, Kontaxis et al. [2014] proposed to duplicate multiple times the entire application server to generate decoy computation activities.

Insiders. To detect and mitigate insider threats, Kaghazgaran and Takabi [2015] offered to extend role-based access control mechanisms with honey permissions. These are fake permissions in the sense that they assign unintended access permissions to only fake versions of sensitive system assets. By monitoring attempts to access or modify such fake assets, the authors are able to detect insiders who triggered these malicious attempts. Note that many other techniques applied at the data layer, such as decoy documents and fake source code, have also been proposed to deal with insider threats, and we discuss them in the last part of this section.

Layer III: Application

State of the art deception techniques applied at the application layer mainly address two threat categories, which are host-based software compromise and remote web-based attacks.

Software Compromise. Deception techniques have been extensively used in the literature as a way to protect software from commonly known vulnerabilities. They usually consist of deceiving attackers by either pretending fake (non-existent) vulnerabilities or by randomly responding to common vulnerability scan attempts. For example, a straightforward deceptive response would be to simulate system saturation by randomly adding delays in order to deceive potential adversaries [Julian 2002]. Michael et al. [2002] introduced the notion of intelligent software decoys that detect and respond to patterns of suspicious behavior (e.g., the interaction between a worm and the system component that it tries to infect), and maintain a repository of rules for behavior patterns and decoying actions. Araujo et al. [2014] converted software patches into fake but valid-looking vulnerabilities (also known as "honey-patches") that drastically limit the attackers capability to determine the successfulness of their attacks. In this scope, and upon detection of an attack exploiting the fake vulnerability, the system seamlessly forwards the attacker to a vulnerable decoy version of the same software. The authors further extended their honey-patch system with an instrumented compiler that automatically clean credentials in the un-patched version of the web server [Araujo and Hamlen 2015]. In [2005], Anagnostakis et al. introduced shadow honeypots that extend honeypots with anomaly-based buffer overflow detection. The shadow honeypot is an instance of the target application and shares its context and internal state. It is used to process anomalous traffic and to enhance the accuracy of anomaly detection.

Finally, Crane et al. introduced, in [2013], software traps that are dissimulated in the code as gadgets, and detect return-oriented programming attacks. These traps detect and notify an ongoing attack as soon as they are manipulated by the exploit.

Web Attacks. Brewer et al. [2010] proposed a web application that embeds decoy links. These links are invisible to normal users, but are expected to be triggered by crawlers and web bots that connect to the application. Similarly, Gavrilis et al. [2007] presented a deceptive method that detects denial of service attacks on web services by using decoy hyperlinks embedded in the web page.

Another approach to deceive web-based attacks also consists of using fake information disguised as web server configuration errors. Only malicious users are expected to manipulate or exploit these errors, which expose them to detection by the system. In this scope, Virvilis et al. [2014] introduced honey configuration files, such as `robots.txt`, including fake entries, invisible links, and HTML comments that indicate honey accounts, in order to detect potential attackers. Other studies proposed decoy forms [Katsinis et al. 2013] and honey URL parameters [Petrunić 2015] that display fake configuration errors in an effort to mislead attackers and to protect the target system.

Layer IV: Data

This section discusses state of the art techniques that use fake or decoy data to deceive attackers. They mainly protect against four categories of threats, namely *identity theft*, *data leakage*, *privacy violation*, and *impersonation*.

Identity Theft. Honey accounts have been used in the literature to track phishers [McRae and Vaughn 2007], detect malware [Akiyama et al. 2013; Bowen et al. 2010b], and also provide possibilities for researchers to study the malicious activities performed on stolen webmail accounts [Onaolapo et al. 2016]. In the same vein, Lazarov et al. [2016] created five fake Google spreadsheets containing decoy banking information and wire transfer details in order to shed light on the way that cyber-criminals use these fake spreadsheets. There are also other approaches that applied deception techniques to protect stolen user passwords. For example, to protect against situations where hashed user passwords have been leaked, Juels et al. introduced in [2013] honeywords (false passwords) in order to conceal true authentic passwords. Bojinov et al. [2010] introduced instead the concept of a theft-resistant password manager that executes on the client side and randomly generates new password instances. Finally, Juels et al. [2014] proposed "honey encryption", which creates a ciphertext that, when decrypted with an incorrect key or password, results in a valid-looking decoy message.

Data Leakage and Insiders. To mitigate or report a data leakage, multiple studies [Ben Salem and Stolfo 2011; Bowen et al. 2009; Voris et al. 2015] suggested the use of decoy documents that implement honeytokens, and beacon alerts that call home when they are opened. Alternatively, honey tokens that mimic sensitive information have also been integrated within databases (e.g., [Bercovitch et al. 2011; Čenys et al. 2005]) — mainly as a way to detect insiders who scan the database to obtain an unauthorized access to data. In the same vein, Park et Stolfo [2012] generated fake but believable Java source code to detect the exflitration of proprietary source code.

Privacy Violation. Honeyfiles that raise an alert when accessed or modified have been used in the literature in order to detect privacy violations for documents shared on web hosting providers [Nikiforakis et al. 2011] and Peer-to-Peer networks [Liu et al. 2012].

More recently, Kapravelos et al. [2014] proposed honey web pages that adapt the page structure and content according to a browser extension's expectations. These honey web pages allowed the authors to identify malicious behavior in the form of privacy violation in browser extensions.

Impersonation. Impersonation attacks on social networks have been first discussed by Bilge et al. [2009]. Deception techniques have been used to detect similar cases of impersonation and fake accounts. For instance, De Cristofaro et al. [2014] created and promoted 13 honey Facebook pages to study the fake likes that they would receive. Honey profiles have also been deployed on social networks in order to identify accounts that are operated by spammers [Stringhini et al. 2010; Webb et al. 2008].

4 MODELING

In this section, we look at theoretical models that have been proposed for deception techniques in computer security. In particular, we have identified two main axes of deception modeling in the literature. First, many contributions have proposed models to support planning and integration of deception in a target infrastructure or system. These models mainly propose some sort of methodology (that can either be a process, a probabilistic model, a practical model based on attack graphs, or a game theoretical model) to design where, when, and how deception may be integrated in computer security. In few cases, the authors have tried to understand the affects of deception in computer security by modeling the interaction between attackers and deception-enabled defenders, mostly based on game theory.

4.1 Deception Planning

We group the proposed models to plan deception into four categories by the modeling method that are 1) process model that defines the desired process and how it should be performed, 2) probabilistic model that uses probability to evaluate the benefits and cost of deception, 3) practical model that uses attack graphs to build deception, and 4) game theoretical model that computes an optimal defense strategy.

4.1.1 Process Model. Yuill [2006] presented a process model covering four major steps: 1) deception-operation development, 2) deployment, 3) target engaged, and 4) continuation decision and termination. The first step involves the planning of goals, the identification of possible targets, the creation of the deceptive element, and the preparation of the mechanism to engage the target. The next step consists of deploying the deception scenario at a location that is visible to the potential target. The target is engaged once he perceives and believes the deception scenario, and subsequently takes the planned action, which may be reported by properly designed feedback channels. Finally, a decision on whether to continue or terminate the deception is taken based on the results and the efficiency of the deception operation.

Similarly, Almeshekah and Spafford [2014b] proposed a model that includes three general phases: design, implementation and integration, and monitoring and evaluation. The first phase consists of specifying the strategic goals, defining the way that the target may interact with deception, creating the deception story, and identifying the feedback channels. Then defenders should integrate deception within a real environment, instead of being a separate disjoint component. Finally, the previously established feedback channels should be carefully monitored in order to enhance deception. Heckman et al. [2015] presented a similar but iterative deception life cycle management process, while De Faveri and Moreira [2016] proposed a similar but adaptive process model. Hassan and Guha [2016] proposed an abstract model based on a state machine with four states (default, ready, production, and determine) to provide a basic elementary view of the aforementioned models.

De Faveri et al. [2016] designed a goal-based approach to incorporate deception in the software development process in three separate phases. The first phase consists of modeling the system architecture and its goal in a specific domain of application. Then the security requirements are identified and a threat model is produced. The last step requires the application of the deceptive solution based on previously built models. The applied deception model is actually similar to the aforementioned process models.

Finally, Yuill et al. [2006] proposed a model that describes deceptive methods by the way that defenders may use to disable the attackers to discover a hidden asset. More precisely, the process model affects the attackers ability or behavior, by altering the results of their direct observation, the findings of their investigation, and the information they learned from other users.

4.1.2 Probabilistic Model. Rowe [2004] provided a probabilistic model of the attacker's belief in false excuses such as system crush, communication breakdown, and also the attacker's suspiciousness about whether he is being deceived. This model is helpful for defenders to plan when and how to deceive, while monitoring the attacker's belief in the proffered excuses. Rowe [2007] also proposed a cost-effective probabilistic model to assess the cost and benefits while planning deception. Wang et al. [2013] modeled the design of the multilayered deception system including decoys of user profile, files, servers, and network or system activity as a problem of optimization which aims to reduce the cost of deception deployment and the loss in case of a successful compromise. Using an urn model, Crouse et al. [2015] modeled the probability of an attacker successfully discovering a vulnerable computer in a network configured with honeypots under different scenarios.

4.1.3 Practical Model. Cohen et al. [2002; 2001b] modeled the process and path that attackers might use to compromise a computer using attack graphs. By introducing false targets in the attack graph, they modeled how to drive the attackers away from real targets.

4.1.4 Game Theoretic Model. Píbil et al. [2012] proposed a honeypot selection game that leveraged the network configuration for a target, the latter including multiple servers with different levels of criticality. The authors further accounted for the attackers' probes in their model in order to compute the optimal strategy for the defender. There also exists multiple contributions that offered to combine game theoretical approaches with attack graphs (or attack graph games) in order to enhance network security. Durkota et al. [2015b] first modeled an attacker who is perfectly informed about the number and type of honeypot that are being deployed in the network. Later on, the same authors proposed in [2015a] an approximate solution that accounts for attackers who only have a partial knowledge of the system in order to find a balanced defense strategy. Finally, the same authors illustrated in [2016] the attack graph game in a concrete case study.

In [2012], Clark et al. modeled the effects of deceptive routing in a wireless relay networks using a two-stage game in order to compute a solution for mitigating jamming attacks.

4.2 Interactions between Attackers and Deception Techniques

The last few years have witnessed an increasing effort towards applying game theory models in the computer security field, including also many contributions that were published in the GameSec conference [7]. In this survey, we mainly focus on studies that use game theory models to characterize the way deception may affect attackers.

First of all, few contributions in this area modeled honeypots as a defense mechanism using a game of incomplete information. Garg and Grosu [2007] characterized deception using a honeynet system where defenders may have the choice to conceal a regular host as a honeypot (or inversely) in response to the attackers' probe. Similarly, Carroll and Grosu [2011] modeled the way deception affects the attack–defense interactions based on a game theory where the players (defenders and attackers) have incomplete knowledge of each other. In this game, defender can deploy two deception defenses by either concealing a legitimate server as a honeypot or by making a honeypot look like a legitimate server. Çeker et al. [2016] modeled a similar approach, using game theory, that offers to disguise a real system as a honeypot (or vice-versa) in an attempt to mitigate Denial of Service (DoS) attacks.

To counter Operating System (OS) fingerprinting, Rahman et al. [2013] analyzed the interactions between a fingerprinter and the target system. The authors used a signaling-game which aims to find an efficient deceptive solution that disrupts the results of fingerprinting. Clark et al. [2015] further modeled the delay upon which a fingerprinter is able to identify real and decoy nodes. Using this delay, authors proposed another game model that characterized the IP address randomization in order to calculate an optimal strategy.

Gutierrez et al. [2015] presented a way of modeling the interaction between adversaries and system defenders using hypergames. Using this approach, the authors were able to model misperceptions resulting from the use of deception. Recently, Horák et al. [2017] presented a game

[7]http://www.gamesec-conf.org/

theoretical framework that modeled the attacker's belief while deception is deployed in a three-layer network.

5 DEPLOYMENT

Most deception techniques take advantage of their interactions with the remote attacker. Hence, the efficiency of such techniques hinges largely on their credibility, their coverage, and the location in which the deceptive elements are placed inside the target environment. In this section, we discuss these fundamental properties and we survey how these requirements were fulfilled in previous work.

In particular, the rest of the section focuses on the following four aspects: 1) how deception is deployed and integrated into a target environment, 2) how previous work approached the problem of optimal placement of deception elements, 3) how to achieve realistic and plausible deception, and finally 4) how to monitor, update, and replace deceptive elements already deployed on the field.

5.1 Mode of Deployment

We start our deployment study by looking at how deception techniques are positioned with respect to the target environment under protection. The techniques that we introduced in the previous section often adopt different deployment strategies, based on their granularity and on the objectives the defender wants to achieve when adopting a given deception technique. According to the taxonomy we presented in Section 3, we classify the different modes of deployment into four categories, depending on whether the deception is built inside the target system, placed in front of the system, later added to the system, or finally deployed as an isolated, standalone solution. Unfortunately, not all existing deception techniques come with a clear or unique deployment strategy. Therefore, this section covers only those techniques whose deployment may be clearly identified with respect to the target system.

Built-In. Only few studies in the literature suggest integrating deception already at the system design phase. In this scope, honey encryptions [Juels and Ristenpart 2014] offered to protect messages by creating dummy honey tokens and including them in the system by design. Similarly, Kaghazgaran [2015] proposed to embed, during the policy setup phase, honey permissions inside role-based access control models. In the same category, some other deception techniques have been integrated directly in the source code of an application. Examples include the injection of random delays in order to modify the normal behavior of a web-based search engine [Julian 2002], honeytokens added during the creation of a new database using aspect-oriented programming [Padayachee 2014], and setting honey HTTP request parameters in a web application in order to lure attackers by exposing dummy URL attributes [Petrunić 2015].

In-Front of. Deception techniques that are implemented in-front of a target system mainly consist of interposing the deception elements between the attacker and the system. This can be achieved by multiple ways, depending on whether the deception is integrated at the network, at the system, or at application layer. For instance, at the system layer, the deployment requires intercepting the execution flow of a process or an application, while at the network layer, it is necessary to intercept the traffic, often using reverse proxies and trusted certificates (in case of encrypted connections).

We identified two distinct methods in the literature that offer to modify the execution flow of an application in order to deploy deception. First, Michael et al. [2002] instrumented the software component to implement decoy actions. Other work leveraged specific application features. In particular, Čenys et al. [2005] used the Oracle database to intercept insert, select, and update operations and to integrate honeytokens. Similarly, the modular design of the Apache web server enables to register a hooking module that can implement deception techniques [Almeshekah 2015; Brewer et al. 2010; Katsinis et al. 2013].

At the network layer, deception is often used to mitigate network scanning and fingerprinting. These solutions are usually deployed on specific network devices, such as routers [Trassare 2013], gateways [Borders et al. 2007; Kontaxis et al. 2014; Smart et al. 2000], firewalls [Le Malécot 2009], or on dedicated monitoring modules (anomaly detection [Anagnostakis et al. 2005]). Note that it may also be possible to achieve similar results without intercepting the traffic. For example, the authors of [Liston 2001; Shing 2016] proposed techniques to monitor and reply to unanswered address resolution protocol (ARP) requests in order to produce false network topologies and to reduce the attacker's ability to scan the network.

Deception for web applications are usually implemented in a proxy [Han et al. 2017] or as a web server module [Brewer et al. 2010] in front of a web application, mostly for the purpose of detecting malicious users who trigger the injected decoys.

Added-To. This category includes techniques that are added or integrated into the system at runtime. Many existing approaches leverage this mode of deployment and use deception as a way to complement other traditional security mechanisms. In particular, decoy assets (IPs, services, applications, vulnerability, or data) are usually integrated in the infrastructure in order to mislead attackers and turn them away from other sensitive assets. To build such decoy assets, previous work adopted different approaches, such as the use of honey patches that simulate fake vulnerabilities [Araujo et al. 2014], simulation [Rrushi 2011], or by duplicating legitimate services [Urias et al. 2016]. Another interesting trend consists of adding artifacts to a real benign service in order to make it appear for attackers as a honeypot [Rowe et al. 2006].

Previous studies have also suggested to use honeyfiles to detect privacy violations, by placing them on online sharing platforms and cloud hosting services [Liu et al. 2012; Nikiforakis et al. 2011]. In the same vein, other studies suggested to use the same concept of honeyfiles in order to detect data leakage within corporate or private networks [Ben Salem and Stolfo 2011; Bowen et al. 2009; Voris et al. 2015; Wang et al. 2013].

The concept of (honey-)assets has also been extended to honey passwords [Bojinov et al. 2010; Juels and Rivest 2013], honey profiles [De Cristofaro et al. 2014; Stringhini et al. 2010; Webb et al. 2008], and even honey hyperlinks in the context of web applications [Gavrilis et al. 2007], in order to detect malicious usages and abuses of sensitive applications and data.

Standalone. This category includes deception techniques that are isolated from the target system. While honeypots are the most common type of isolated deception system, other methods have also been explored in the literature, such as the use of honey accounts to drive attackers into connecting to a decoy service that is isolated from the real authentic service [Bowen et al. 2010a; Chakravarty et al. 2011]. For example, Onaolapo et al. performed a real world experiment in which they (deliberately) leaked fake webmail accounts that are isolated from any other benign accounts on a webmail service [2016]. During their experiments, the authors had been able to monitor abuses of leaked webmail accounts and to learn more about the tactics used by attackers.

5.2 Placement

We now look at different strategies that have been proposed to support the placement of deceptive components. This covers the way existing deception techniques are deployed inside a target system (either manually or automatically) and the way these techniques are displayed to attract and deceive attackers. We organize this section into three parts, based on the ultimate objective of deception: attack prevention, detection, or mitigation.

5.2.1 Attack Prevention. The application of deception techniques for intrusion prevention currently takes two different aspects. On the one hand, it deters an attacker by displaying, for example, false targets among the path that attackers may go through [Cohen and Koike 2002]. In this case, the placement of false targets is guided by the knowledge of existing attack graphs. Multiple contributions leverage this knowledge to compute an optimal strategy to deploy honeypots in a specific network configuration based on a game-theoretical approach [Durkota et al. 2015a,b, 2016].

On the other hand, deceptive techniques may also be used as *chaff*, to confuse an attacker by hiding important information in a large amount of data. For example, to protect real user passwords Bojinov et al. [2010] offered to conceal them through dissimulating a large number of other decoy passwords. Similarly, Kontaxis et al. generated decoy computing activities [2014], and Rrushi et al. generated decoy network traffic [2011], in order to dissimulate sensitive applications and network connections respectively, so they may be less likely to be identified and compromised by an attacker. In this case, decoys are usually placed alongside the valuable assets.

5.2.2 Attack Detection. Many deception placement strategies have been previously explored in order to enhance attack detection. We classify these strategies into two different categories, depending on whether deception is being integrated inside the target system, or whether it is publicly exposed in order to enhance threat intelligence and to anticipate unknown attacks.

Integrated deception placement. In the first category, Gavrilis et al. [2007] used decoy hyperlinks in order to protect a website against flash crowds. The authors developed a placement strategy where they represent the target website as an undirected graph. The graph nodes represent the individual pages of the website, and the edges represent the hyper links between the pages. The attacker is modeled as a random walker, and the strategy consists of finding the optimal subset of decoy hyperlinks that minimizes the survival probability of the random walker in the graph.

Another example includes the placement of honeyfiles inside a target file system. First, Bowen et al. [2009] manually placed the honeyfiles at selected locations in the file system. Alternatively, Voris et al. [2013] developed an automated placement strategy that first searches the target file system to locate folders that have been recently used, as well as the folders of large number of files sharing similar features such as extensions. These folders are further selected as favorite locations to place honeyfiles. The same authors further distinguished two modes of automatic deployment: integrated and separated [Voris et al. 2015]. In an integrated deployment, decoy documents are co-located with real documents, while in a separated placement, decoy documents are deployed in isolated subfolders within the same root directory. Finally, Whitham [2013] suggested a more aggressive placement strategy, offering to place honeyfiles in all directories of the file system.

Finally, researchers have also studied the placement of decoy permissions within role-based access control policies [2015]. The placement strategy consists first in identifying high risk permissions. These are further duplicated as honey permissions, and their corresponding objects are duplicated as decoy objects. The authors select the sensitive roles in the policy whose risk exceeds a predefined threshold and assign the honey permissions to these roles for monitoring and detection.

Apart from honeyfiles, the placement strategy has been however under-studied for the majority of deception techniques. For instance, previous work has discussed how to implement honey tokens in a database [Čenys et al. 2005], but it is still unclear how these tokens should be placed — e.g., in a separated fake table or alongside legitimate data. A similar problem exists for honey parameters placed in a web application to enhance attack detection. Honey parameters can be added in many HTTP header fields and in the HTML source code, which results in a large number of placements. Moreover, a web application usually consists of multiple and various services. Currently, there is no systematic approach to place honey parameters to achieve optimal attack detection. The fundamental difficulty in achieving this is the lack of methods to describe the web application logic. As Li and Xue [2014] have pointed out, there is actually *"an absence of a general and automatic mechanism for characterizing the web application logic"*, which is however a prerequisite to achieve an effective deception placement.

Isolated deception placement. Isolated deception techniques follow different placement strategies, which may also depend on the defender's objectives and the monitored attacks. In particular, honey profiles are being commonly created on social networks and used in order to monitor and detect spam and infection campaigns [Stringhini et al. 2010; Webb et al. 2008]. Different strategies can be used to advertise and share these decoy profiles, including the use of dedicated advertisement services, and other underground services [De Cristofaro et al. 2014].

On the other hand, honeyfiles have been manually leaked on public hosting services [Nikiforakis et al. 2011] and file sharing networks [Liu et al. 2012]. This approach is commonly used to attract potential attackers and as early warning system in case of unauthorized access to personal user data. Moreover, honey accounts are actively provided to be collected and exfiltrated by malware, to study the goals of the attackers [Akiyama et al. 2013; Bowen et al. 2010b]. Similar information has been also deliberately distributed on malicious hacking forums to attract attackers and infiltrate criminal groups [Onaolapo et al. 2016]. Finally, honey URLs are used when testing and connecting to phishing sites in order to track the activities of the attacker [McRae and Vaughn 2007].

5.2.3 Attack Mitigation. When deception techniques are used for the purpose of attack mitigation, the placement strategy is generally in line with the type of attack whose damage needs to be reduced. For instance, upon the detection of a network scan, mitigation techniques such as network tarpits [Liston 2001; Shing 2016] and traffic forging [Le Malécot 2009] can be put in place. To mitigate software exploits, the safe duplication of target application enables to redirect attackers [Anagnostakis et al. 2005; Araujo et al. 2014] and even contain them [Urias et al. 2016]. Note that some techniques, such as chaff, can be deployed both for attack prevention and mitigation. For instance, if an attacker is able to steal the password file, populating it with a large number of fake entries [Bojinov et al. 2010] could somehow mitigate the password theft attack. Clark et al. [2012] modeled indirectly the placement of deceptive traffic in a wireless relay networks based on game theory in order to reduce the impact of jamming attacks.

5.2.4 Summary. During our survey of literature, we came to the conclusion that previous work on deception have mostly focused on the introduction of new deceptive techniques and elements, but it rarely discussed where (and how) such elements should be placed in the system to protect. Recent efforts that model deception using game theory [Durkota et al. 2015a,b, 2016] propose to find optimal defense strategies under certain assumptions about the network configuration and the attackers. Game-theoretical approaches may prove that such an optimal strategy exists under these assumptions. However, the gap between the optimal strategy and the real-world deployment is still unclear. In fact, without a clear methodology to test and measure the accuracy of deception techniques, a problem we discuss in Section 6.1, it is difficult to compare different placement strategies and decide which is better among several options.

5.3 Realistic Generation

A key property of any deception element is that it should be able to deceive an attacker into believing that it is real. Therefore, we now look at different approaches that have been proposed in the literature to create plausible and realistic deception elements. We divide again this section into four parts, based on the layer where deception is applied.

5.3.1 Network Layer. A common approach to generate fake network activity is to load and replay real network traffic, after replacing confidential information with decoy data. For example, Bowen et al. [2010a] suggested to simulate decoy network communication by recording real network traffic and inject into it decoy information such as fake IP addresses and payloads. Kontaxis et al. [2014] also discussed the generation of decoy HTTP traffic. In this case, the authors introduced templates to describe protocol messages and their parameters and then generated random permutations and modification of these parameters in order to create decoy HTTP connections. Shing [2016] tuned the parameters of sticky connections to produce more realistic traffic in order to evade the detection proposed by Alt et al. [2014].

5.3.2 System Layer. In general, there are two approaches to reproduce realistic system behaviors, either by *duplication* or by *simulation*. The first approach consists of duplicating a legitimate server [Kontaxis et al. 2014; Urias et al. 2016] or a target application [Anagnostakis et al. 2005; Araujo and Hamlen 2015] while removing possible sensitive information. The second solution relies instead in simulating the appearance, the exact behaviors, and the overall activity of an entire process and equipment [Rrushi 2011].

5.3.3 Application Layer. Different mechanisms have been proposed in the literature to generate realistic deception at the application layer. We classify these mechanisms into three categories, based on the goal of deception.

First, deception may consist of purposely modifying the behavior of an application in response to specific requests or attacks. In this scope, Julian [2002] proposed to alter the response time of an application to confuse an attacker by adding artificial delays proportionate to the average time the same application would take to process that specific user request.

The second category consists of altering the structure of an application in order to showcase a fake attack surface. For example, to render a decoy server less suspicious, Rowe [2006] generated fake, yet realistic file systems, using pieces extracted from a real file system. The author added random modifications, such as changes to the folder tree, creation of fake directories, or generation of fake files by combing pieces of real files. In the same category, Whitham [2013] described the generation of honey files by combining statistics (such as file names, sizes, access control attributes, and modification dates) from public files accessible on the Internet with statistics of usage behavior for the target system.

The third category includes techniques that consist of creating fake content in order to attract and detect attackers. To generate such realistic honeyfile content, Bowen et al. [2009] instrumented genuine and common financial documents such as invoices and tax forms by adding honey accounts, realistic (but fake) names, addresses, and other user familiar information. Whitham et al. [2017] proposed a more elaborated approach that consists of using natural language processing to generate realistic honey-text content. The authors first selected a median size file from the target directory to generate the template using language processing tags. They then collected characteristics from the target file system that are used to redistribute the original words. Finally, they placed the honeyfile into selected places inside the target directory.

5.3.4 Data Layer. The generation of realistic fake data mostly overlaps with the generation and simulation of artificial data (see for instance [Houkjær et al. 2006] for an overview of this area). Given the large scope of these techniques, we limit ourselves in this survey to those methods that have been previously used to generate honey tokens for deception.

In [White 2010], the authors introduced a method to create honey tokens that represent fake US citizens. They first observed the statistic features, values, frequencies, and dependencies of representative samples. Then they generated fake personal information according to previously obtained characteristics. Moreover, to generate decoy passwords, authors in [Bojinov et al. 2010] converted real user passwords into a set of semantic rules. For example, the following rule "$l_3d_1s_2$" refers to a password that includes a word of three letters, one digits, and two special characters. The authors further used a dictionary to generate similar passwords that match the previously obtained rules. Another approach by Juels and Rivest [2013] offered either to randomly change the last few characters, digits, or to use a probabilistic model that characterize real passwords.

Bercovitch et al. [2011] presented instead a method to automatically generate honey tokens in a database. First, the authors built various rules to characterize the database structure and relationships. Then, they generated fake, but realistic tokens that satisfied the previously-generated rules. Finally, they assigned a confidence score and ranked the honey-tokens based on their similarity with the real database content. Alternatively, Alese et al. [2014] offered to simply shuffle the records of a real database in order to generate fake honey tokens.

5.3.5 Summary. As discussed above, the realistic generation of honey-tokens is relatively well studied for few traditional domains — such as network traffic, files and file systems, passwords, and database environments. However, the application of deception techniques has recently broadened in scope to cover many other areas, such as web applications and cloud images, where a proper generation strategy has not yet been discussed.

5.4 Monitoring

A successful deployment of deception techniques drives attackers to interact with them, thus revealing their malicious activities. To collect and analyze such activities, a monitoring mechanism is indispensable. We discuss in this section whether and how the deception is monitored and whether the deception can be maintained and updated over the time.

Monitoring is more relevant to deception techniques that are used to enhance attack detection rather than for those aiming at prevention and mitigation. We group existing work that have clearly defined their monitoring approach into two categories, based on whether the system that raises the alerts is integrated into the technique itself or is instead implemented as a separate component. Also note that there are two distinguishable phases of attack detection, namely the triggering and monitoring of the alerts.

5.4.1 Integrated Alert. In this category, Bowen et al. [2009] proposed to insert a uniquely identifiable token hidden in the decoy document, which will be sent silently toward a remote server to trigger an alert. Two examples have been presented: a remote image embedded in MS Word document and a snippet of JavaScript code included in a PDF file [Nikiforakis et al. 2011].

Another example is the use of decoy URLs or hyperlinks that, when they are included in a web page, automatically fetch some content from a backend server. The use of such links always require a web server that these links point to, which can be the same as the legitimate server if decoy URLs are integrated within the target web application [Brewer et al. 2010; Gavrilis et al. 2007]. Otherwise, a separate web server is necessary [McRae and Vaughn 2007] to handle the deception elements. In any case, the monitoring is usually performed off-line by parsing the access log of the web server.

Finally, a software trap [Crane et al. 2013] triggers an alert when is exploited by an attacker. It also needs an extra handler to detect and respond to the attackers. However, no details of the way that the handler monitors the software trap were provided by the authors. Likewise, deception techniques such as honey URL parameters can be implemented in the source code together with the monitoring mechanism [Petrunić 2015]. Authors proposed to use centralized log management systems to identify and prevent attackers.

5.4.2 Separated Alert. The first version of honeyfiles proposed by Yuill et al. [2004] relied on a centralized monitoring approach. Honeyfiles were deployed on a file server where all file access operations werer monitored. In case of an access to a honeyfile, an alert is sent from the file server. In the same vein, Wang et al. [2013] implemented a system service on Microsoft Windows operating system that registers and monitors honey files and triggers an alert when an access to a honeyfile is detected.

Bowen et al. [2009] proposed another type of honeyfiles that contain honey accounts from prevalent web services, in particular Gmail. The authors then adopted a set of custom scripts to monitor and gather information about the account activity to detect attackers. A similar solution of including honey accounts in honeyfiles also been reused in other works [Lazarov et al. 2016; Liu et al. 2012]. Honey accounts usually require a dedicated module to monitor and detect attackers that use them. Chakravarty et al. [2011] manually created honey accounts on their decoy servers where they monitored the unauthorized accesses. Honey encryption [Juels and Ristenpart 2014] and honey passwords [Juels and Rivest 2013] enforced the generation of honey passwords at the moment of user account creation. In order to detect the misuse of generated fake passwords, they require online monitoring at the server side. Finally, honey permissions [Kaghazgaran and Takabi 2015] also require an extra module to monitor their use.

Lastly, to detect the selection of a particular honey token in a database, Cenys et al. [2005] also suggested the use of a specific handler module.

Summary. Independently from the way the actual alert is triggered, most deception techniques require an extra module or even a dedicated server to monitor and detect attacks. However, there is not yet a comprehensive monitoring system that incorporates all the different deception techniques to build a multi-layer proactive deception-based threat detection system. Moreover, it's still unclear for the users how they should integrate such system with respect to traditional defense mechanisms, and in particular with other existing monitoring systems.

Finally, a very important aspect to consider is the update and re-deployment of deception elements. After a deception technique has been deployed, if its not constantly modified, attackers can learn its nature and location and simply avoid it in their future attempts to break into a system. This sets intrusion deception apart from other detection mechanisms, where the knowledge of the defense solution does not necessarily undermine its effectiveness. Despite its fundamental importance and the fact that the re-deployment problem has already been mentioned by other studies [Katsinis et al. 2013; Wang et al. 2013], this aspect has been mostly ignored by the research community. In fact, if we know well how to update signature-based solutions or re-train model-based approaches, we know almost nothing on when, how, and how often a set of deceptive elements should be replaced with new ones. We do not even know what is the actual impact, in terms of reduced effectiveness, of not updating a deception deployment for long periods of time.

To the best of our knowledge, only one work exists in this space, in which Whitham [2013] implemented a form of continuous management in the case of honeyfiles. The system was designed to randomly retire honeyfiles, update their timestamp attributes, and create new honey files to maintain a desired ratio in the system.

6 MEASUREMENT & EVALUATION

This section covers the different techniques and existing experiments that have been used to evaluate the efficiency and coverage of intrusion deception. In 2006, Cohen discussed the existing experiments that attempted to evaluate deception, and came to the conclusion that most of them did not cover all the relevant aspects [2006]. In particular, the author stressed the fact that more elaborated experiments were required to better understand the issues behind the application of deception for cyber-defense. In this section, we review the previous experiments that offer to measure and evaluate deception techniques, with an emphasis on recent contributions that were not covered by Cohen's book [2006].

Note that during our effort to summarize existing experiments, we observed different objectives for the evaluation process itself, depending on the design criteria and the main properties the authors wanted to achieve by using deception. For example, Yuill et al. discussed properties such as the *"plausibility, receivability, verifiability, efficiency, and implementation"* [2006]. Three years later, Bowen et al. introduced a new set of properties for deception, including *"believability, enticement, conspicuousness, detectability, variability, non-interference and differentiability"* [2009]. In 2014, Juels et al. [2014] added two additional properties, namely the *"indistinguishability"* and *"secrecy"* of deception techniques, and suggested to use them to tune the decisions of an access control system against potential attackers.

While all these properties play an important role to achieve an efficient deception, most of them are difficult to formalize and measure, which makes their evaluation very challenging. We discuss, in this section, how previous work managed to evaluate these properties and the main lessons that we learned from those experiments.

The rest of the section is organized into four distinct categories, that respectively cover the following aspects: 1) the way previous works evaluated deception placement strategies, 2) the way they evaluated the plausibility and realism of deception, 3) the way they measured the efficiency and effectiveness of deception, and finally 4) the way they evaluated the accuracy of deception and its false positive rate.

6.1 Evaluation of Deception Placement

Only few contributions have been dedicated to evaluate and measure different deception placement strategies. In particular, we are aware of only three related studies that we discuss in this section, as summarized in Table 2. Note that many other studies evaluated implicitly this property as they were evaluating the effectiveness of deception. We discuss these specific studies later in Section 6.3.

To the best of our knowledge, there is not yet a structured approach in the literature to evaluate a deception placement strategy. Previous work mainly evaluated the additional effort introduced by the presence of deception, and the attacker's extra effort to compromise the target system [Bojinov et al. 2010]. Nonetheless, we observed two alternative approaches in the literature to evaluate the placement of deception techniques and their impact.

On the one hand, Garvilis et al. [2007] introduced a theoretical approach to embed decoy hyperlinks in a web site in order to detect Denial of Service (DoS) attacks. To evaluate the placement strategy for those hyperlinks, the authors introduced a probabilistic method that leverages a graph-based representation of the target web site. The nodes in the graph capture the individual web pages, and the edges capture the hyperlinks between the pages. Attackers are treated as random walkers in the graph, and the evaluation model computes the probability to detect an attacker with respect to the number and placement of decoy hyperlinks in the graph. Using their theoretical evaluation model, the authors were also able to compare their approach to other decoy placement strategies, such as the one based on genetic algorithms. Note that other theoretical approaches including game theory may evaluate implicitly the placement problem while searching for an optimal defense strategy. Such contributions [Clark et al. 2012; Durkota et al. 2016] usually evaluate the global deception-based defense strategy using a model specific utility function, which implies the deployment of deception under a particular configuration.

As opposed to this theoretical approach, other contributions in the literature experimentally evaluated the effect of deception placement strategies, mostly using groups of students or volunteers. For instance, Ben Salem and Stolfo [2011] set up an experiment that involved four groups of 13

Reference	Experiment subject	Metric
[Gavrilis et al. 2007]	[none]	detection probability
[Ben Salem and Stolfo 2011]	52 students	number of alerts
[Voris et al. 2015]	Students	comparison with manual placement

Table 2. Evaluation of deception placement

Reference	Layer	Experiment subject	Metric
[Bowen et al. 2010a]	Network	15 students	accuracy
[Shing 2016]	Network	tarpit detector	rate of detection
[Trassare 2013]	Network	traceroute	response of scan
[Rrushi 2011]	Network	students	signal detection theory
[Kontaxis et al. 2014]	System	[none]	control flow graph similarity
[Rowe 2006]	System	[none]	statistic similarity
[Julian 2002]	Application	4 students, 4 colleagues	evaluator reaction
[Whitham 2017]	Application	[none]	file content similarity
[White 2010]	Data	100 testers	number of detection
[Bercovitch et al. 2011]	Data	109 students and researchers	honeytokens quality

Table 3. Evaluation of deception generation

computer science students. The authors deployed a system that includes honeyfiles, and ran the system during one week to observe how students interacted with the system and whether they were able to unveil the decoy files. They observed that the location of a honeyfile in the system drastically impacts the number of users who will be tricked into accessing and opening the decoy document. Similarly, Voris et al. [2015] evaluated their deception tool by operating an experiment where decoy files were automatically placed on a benign user system, and a few volunteer users were asked to attack the system. They found that their automated decoy placement achieved similar results compared to a manually designed placement.

6.2 Evaluation of Deception Generation

This part relates to the human perception of the generated deception techniques and whether they are discernible from other genuine assets of the target system. Table 3 lists the existing approaches in the literature that performed an evaluation of the deception generation approach, including the metric used for the evaluation and the individual subjects involved in the experiment. Note that most experiments have been performed in controlled and instrumented environments, involving mostly volunteer human subjects. This section discusses the different evaluation strategies and the obtained results.

6.2.1 Network Layer. We observed, in this layer, two experimental approaches to evaluate the deception generation mechanism, including an automated, tool-based approach and a manual, human-based approach.

In the first category, Trassare [2013] proposed a network-based approach that deceives malicious traceroute attempts in order to conceal the topology and assets of a physical data communications network. To evaluate the generated deceptive topology, the author conducted experiments using the traceroute tool against different network topologies. The results of the experiments showed that the generated deceptive topology drastically limits the ability of traceroute probes to scan the network. Similarly, Shing [2016] measured the efficiency of deceptive network tarpits against an automated detector [Alt et al. 2014]. The author found that only 2.8% of the tarpit hosts had been identified as decoys by the detector.

The second category covers human-based evaluation attempts. In this area, Bowen et al. [2010a] evaluated the believability of the generated decoy traffic by using 15 human subjects, including students in network security, researchers, and security professionals. The participants were asked to manually discern the decoy traffic from the real one. During the experiments, even the most skilled participants successfully discerned in average 49.9% of the decoy traffic, which is equivalent to the probability of guessing randomly. Alternatively, Rrushi introduced in [2011] a network-based deceptive approach that is aimed at protecting industrial control systems. By generating decoy network traffic, the proposed solution enables a defender to intervene in the process of an attacker to select a target and thus steer it towards simulated fake devices. To evaluate the solution, the author asked students to try to discern genuine assets from other decoy assets in the target deceptive system using signal detection theory. The results of those experiments showed that decoy traffic drastically increased the uncertainty of students when picking and attacking their targets.

6.2.2 System Layer. To measure the similarity between a decoy and a genuine file system, Rowe [2006] used 36 statistical metrics such as the mean, standard deviation, minimum, and maximum values in order to characterize the structure and type of filenames, file types, size, the distribution of date, and the directory shape. The author found that the generated fake files are statistically discernible from other real files in the file system, but that the difference is small enough that it remains difficult for individual users to discern decoy and benign files. Kontaxis et al. introduced in [2014] a deceptive system that uses fake computation activities to prevent attackers from accessing unauthorized confidential information in the cloud. To evaluate the efficiency of the generated scheme, the authors proposed a binary instrumentation tool that compared the similarity between the control flow graphs for both replicated decoy servers and benign servers. In their evaluation, the authors have shown that it is difficult to notice the difference between legitimate and generated activities. Adversaries would have either to take the risk to trigger an alert or investigate significantly to identify data that they are interested in, which may reveal themselves.

6.2.3 Application Layer. In [2017], Whitham et al. proposed an automated tool to process the content of benign documents in order to generate fake documents with the same structure and semantic. To evaluate their approach, the authors introduced a similarity score that leverages the percentage of common words between a decoy and genuine documents. The similarity is expected to be the highest between the decoy document and its originating genuine document, and the lowest between the decoy document and other documents in the file system.

In a different use case, Julian [2002] conducted an experiment where users were asked to perform normal and harmful queries against a web application that adds deceptive delays. The author measured the reaction of test subjects to determine whether the subjects were able to discern deceptive from other regular processing delays for the application. During the experiment, all users had been misled by the delaying tactic.

6.2.4 Data Layer. White conducted in [2010] two sets of experiments that involved 100 participants. These participants were asked to discern real data from other automatically generated decoy content. In the first experiment, participants were unaware of the existence of honey tokens. When they were given a list of 20 genuine data, they believed that, in average, 3.67 of them were indeed decoys. Afterwards, participants correctly found, in average, almost the same number (3.36) of fake data within 3 lists of data in which generated decoys had been inserted. A second experiment revealed that the participants failed to correctly select the honey tokens, even when they were

Type	Reference	Experiment subject	Metric
Controlled	[Cohen et al. 2001b]	27 students	duration of attack
	[Cohen and Koike 2002]	7 students	ability to control attack path
	[Durkota et al. 2015b]	[none]	defender's loss
	[Yuill et al. 2004]	3 students	number of detection
	[Bowen et al. 2009]	20 users	ability to detect attackers
	[Brewer et al. 2010]	3 web bots	bot detection accuracy
	[Ben Salem and Stolfo 2011]	40 students	number of detection
	[Shabtai et al. 2016]	173 students	detection rate
	[Heckman et al. 2013]	4 red/blue teams	user reaction
	[Han et al. 2017]	150 CTF participants	number of detection
	[Araujo et al. 2014]	[none]	practicality
Real-world	[Webb et al. 2008]	spammer	spammer detection
	[Stringhini et al. 2010]	spam bot	spammer detection
	[Liu et al. 2012]	potential attackers	privacy violation detection
	[Bowen et al. 2010a]	snoopers at Defcon'09	snooper detection
	[Chakravarty et al. 2011]	malicious Tor exit nodes	eavesdropping detection
	[Borders et al. 2007]	potential attackers	attack reduction
	[Nikiforakis et al. 2011]	potential attackers	privacy violation detection
	[Lazarov et al. 2016]	potential attackers	detection of malicious activities
	[Onaolapo et al. 2016]	potential attackers	detection of malicious behaviors
	[Bowen et al. 2010b]	malware	malware detection
	[Akiyama et al. 2013]	malware	attack detection
	[McRae and Vaughn 2007]	phishing site	phisher detection

Table 4. Evaluation of deception effectiveness

aware that the samples did include decoy elements. The correctly selected number of honey tokens was similar to what would have been obtained if the participants had selected randomly.

Similarly, Bercovitch et al. [2011] evaluated the quality of generated honey tokens by showing both genuine data and generated examples to 105 individual subjects. The authors used the results of this study to tune their likelihood-based rating and to select undetectable honey tokens.

6.2.5 Summary. The most commonly used type of evaluation to measure the realism of deception consists of using a human evaluator to judge the generated deception, which is similar to a Turing test where a human is used to judge the behavior of a machine. Generally, previous work managed to evaluate and produce realistic deception, but in specific application domain such as network, files, and database. Equivalent experiments are still missing for other domains, including web applications.

6.3 Evaluation of Deception Effectiveness

The effectiveness of a given deception technique refers to its ability to achieve the desired functionality. Different methodologies have been used in the literature in order to evaluate the effectiveness of deception. We classify them into two main categories, as illustrated in Table 4. The first category includes evaluations that were conducted in a controlled environment, typically by involving few participants. The second category instead includes evaluations where deception techniques have been publicly exposed to the Internet and evaluated against real users and attackers.

6.3.1 Evaluation in Confined Environments. Different experimental setups have been introduced in the literature to evaluate the effectiveness of deception in instrumented and confined environments. These experiments have also adopted different strategies for evaluation, depending on whether deception applies at the network, system, application, or data layers.

Network Layer. Cohen et al. [2001b] measured the effects of network-based deceptive defenses by conducting experiments over simulated attack graphs. Participants to these experiments included students and few security experts (both security professionals and researchers). The participants were divided into two groups, including those who were not aware of existing deception techniques, and those who had been informed about the existence of deception. The obtained results have shown that network-based deception techniques were indeed effective as attackers spent more time trying to go through deceptive paths rather than through the real attack paths. The authors were also able to drive some conclusions on the cognitive confusing factors related to deception, through analyzing the forms filled by the participants during the experiment.

In [2002], Cohen et al. extended the previous experiments by introducing a more generic attack graph model designed to drive attackers towards fake targets. The experiments involved seven students who were asked to attack and try to compromise the system. The results were promising, with students constantly misguided and driven through fake attack paths that were introduced for this purpose. In [2015b], Durkota et al. modeled a particular attack graph using game theoretical approach. They evaluated the loss of defender with respect to the number of honeypots deployed in a simulated network.

System Layer. Heckman et al. [2013] organized a cyber-wargame within an instrumented environment. The game involved two distinct teams. A so-called blue team was tasked to set-up a command and control system and try to protect the system against attacks from a second red team. The blue team experimented multiple deception techniques to mislead the adversaries. The log analysis following the experiments resulted in very promising observations. In particular, the adopted deception techniques had a significant impact on the red team operation, as attackers spent a long time trying to compromise fake targets.

Application Layer. Araujo et al. [2014] offered to mitigate known vulnerabilities by implementing the concept of honey patches. In this scope, the effectiveness of their solution largely hinges to its applicability to a large number of known existing vulnerabilities. The authors evaluated their approach using an experimental environment that included an Apache HTTP server and a simulated web application. They evaluated the effectiveness of their approach according to the number of real application vulnerabilities that they were able to transform into decoy vulnerability through the concept of honey patch. In particular, the authors collected a total number of 75 vulnerabilities that affect their configuration and that were reported between the year 2005 and 2013. Overall, they found that 49 out of the 75 analyzed vulnerabilities (almost 65%) were indeed convertible into honey patches. In [2017], Han et al. evaluated existing deception techniques that may be used to detect web attacks, including honey parameters, honey account, and honey trap resource, etc. The authors leveraged a Capture The Flag (CTF) exercise where 150 participants were asked to find vulnerabilities within a specially designed e-commerce application. During the CTF exercise, they detected 64% of the participants that successfully found at least one vulnerability.

Data Layer. As described in Section 3, state of the art data layer deception techniques mostly consisted of generating and placing decoy user accounts or content. The evaluation of these techniques in instrumented environments mainly involved human participants who were asked to analyze and tell apart decoy from other real authentic accounts and data. In this scope, Yuill et al. [2004] tested their honeyfile system by deploying it on a honeynet and then asking a group of students to test and try to compromise the system. During the experiment, a honeyfile was considered effective when it contributed to detecting at least one attacker and that the attacker did not realize the fake nature of the honeyfile before an alert had been triggered. Overall, the authors found that the most effective honey files were those placed closer to the root directory of the file system.

In [2009], Bowen et al. evaluated their deception system by integrating it into a honeynet of several virtual machines. The authors posted multiple message invitations with dedicated accounts to their platform in order to encourage volunteer participants to connect and test their deception system. After one week of observation, the authors were able to collect 20 unique users. Five out of these participants triggered at least one alert associated with a decoy document.

In [2011], Ben Salem and Stolfo designed an experiment to evaluate the enticing and conspicuous nature of decoy documents. The authors asked a group of students to access an unlocked computer system, looking for financial documents. Several decoy documents were also placed in the system in order to evaluate the reaction and attitude of users when they discovered such documents. The results of the study showed that the use of decoys was very efficient in detecting malicious accesses to the system. In particular, all attackers were detected in the first ten minutes after they had connected to the target computers.

Finally, Shabtai et al. [2016] organized an experiment that involved 173 distinct participants. The experiment consisted of generating 50 loan requests, some including deceptive honey tokens. Participants, acting as individual bank employees, could choose to behave in a honest way by validating the loan request and obtaining a 10% commission. Alternatively, they could have acted in a malicious way by suggesting a private funding program from a competitor, and thus obtaining a 20% illegal commission. The participants were divided into two distinct groups, one informed about existing honey tokens in the loan requests and the other unaware of this fact. The results of the experiment showed that all malicious participants could be detected by planting honey tokens in 20% of the loan requests.

6.3.2 Real-World Evaluations. In open evaluations conducted on the Internet the administrator does not fully control the users who interact with the deception system. Therefore, these experimental setups usually enable to collect and analyze a wider set of interactions with the system. Nonetheless, they often lack the appropriate ground truth about the number and nature of each attack, as well as information about user incentives and real intentions when connecting to the system.

In this category, Borders et al. [2007] evaluated the effectiveness of decoy IP addresses in misleading external attackers. To do so, the authors set up an experimental testbed including an OpenFire gateway that controlled the traffic from the Internet towards three distinct workstations. Three experiments were conducted over a period of 21 days, involving both a normal and a deceptive OpenFire configurations. The normal configuration included default firewall rules that drop accesses to unauthorized ports and protocols. The deceptive configuration also included 36 unallocated IP addresses towards which the gateway would accept connections in an effort to mislead the attackers. An intrusion detection system was also configured to notify the administrator in case of successful attacks towards the three workstations. After the experiment, the number of successful attacks when using the deceptive configuration was reduced by almost 65% compared with the number of attacks that were reported when using the normal configuration.

To evaluate the effectiveness of honeyfiles, Nikiforakis et al. [2011] uploaded such decoy documents to 100 public file hosting services. These files were designed to call back a dedicated server so that the authors were informed when someone downloaded the decoy file. Over a period of one month, 80 unique IP addresses accessed the honeyfiles and triggered the notification to the

Reference	Experiment subject	Metric
[Gavrilis et al. 2007]	students and faculty members	number of false alerts
[Brewer et al. 2010]	human subjects	number of false alerts
[Ben Salem and Stolfo 2011]	57 students	number of false alerts
[Voris et al. 2015]	27 users	interference with normal activities
[Han et al. 2017]	258 users	number of false alerts

Table 5. False positive evaluation of deception techniques

remote server. More interestingly, the decoy files also included decoy credentials that enabled the remote users to connect to a fake web application that was implemented for the purpose of the experiment. The authors were able to observe that miscreants from 43 distinct IP addresses had successfully logged in 93 times with the fake account information leaked in the decoy document. Similarly, Liu et al [2012] posted five honeyfiles containing decoy accounts on a public file sharing network. Over a period of one month, 192 distinct users downloaded the honeyfiles, including 45 users who used the decoy accounts in a deliberate attempt to conduct identity theft attacks. Finally, Lazarov et al. [2016] constructed five fake Google spreadsheets with decoy banking information and hidden links. Over a period of 72 days, they found that the decoy document have been accessed 165 times and modified 28 times. Moreover, there were 174 clicks on the hidden links inside the same document.

In a rather different attempt to detect spammers, Webb et al. [2008] created 51 honey profiles on MySpace, and used these profiles to monitor and detect scam accounts. The authors were then able to collect 1,570 distinct friend requests over a period of four months. Similarly, Stringhini et al. [2010] produced 300 honey profiles on three distinct social networks. During their one year long experiment, the authors were able to detect 173 distinct spammers on Facebook, 8 spammers on MySpace, and 361 spammers on Twitter.

Finally, the evaluation of honey accounts has been mostly performed through open deployment and advertisement on popular and publicly accessible Internet services. For example, during one week, McRae and Vaughn [2007] submitted 11 honey accounts which contained decoy URLs to phishing sites to track down the phishers while they viewed the honey accounts. However, only two out of 11 worked successfully. Similar experiments with honey accounts have been performed on the wireless network at the Defcon 09 hacking conference [Bowen et al. 2010a], on malware executables [Akiyama et al. 2013; Bowen et al. 2010b], on the Tor network [Chakravarty et al. 2011], and on underground forum and online paste tool [Onaolapo et al. 2016]. All of them have been able to detect a variety of attackers, which empirically shows the effectiveness of honey account at detecting attackers.

6.3.3 Summary. Considering the current application domain, deception is found to be effective to delay and detect attackers both in controlled and real-world environment. Future deployment of deception in other domain still requires similar evaluation.

More importantly, only three studies has evaluated the false negative rate [Ben Salem and Stolfo 2011; Han et al. 2017; Shabtai et al. 2016] (testing honey documents, web deception techniques, and honey tokens respectively), which is fundamental to compare deception solutions with more traditional intrusion detection approaches. In real-world evaluations, the false negative rate is rarely measurable due to the lack of ground truth information about the number of attackers. In contrast, such evaluation is feasible in a controlled environment. Therefore, we believe future effort in this space should shift their focus on assessing the false negative rate.

6.4 False Alarms Evaluations

Compared to the previous metrics discussed in this section, the evaluation of the false positives rate when using deception in cyber defense has attracted much less interest among the security research community. In particular, we are unaware of any structured approach and methodology to evaluate the false positive rate when deception is being used for intrusion prevention and mitigation.

False positive measurements have been conducted in only a handful of publications that proposed deceptive techniques for intrusion detection, as shown in Table 5.

For example, Garvilis et al. [2007] conducted a real-world experiment where they exposed decoy links on the public web site of their university. The authors manually examined the web server logs to determine whether interactions with the decoy links were benign requests that would have resulted in false alarms. The experiment ran over a period of one month, during which the authors collected a total of 45,121 distinct requests — only 19 of which were marked as benign (corresponding to a 0.04% false positive rate). The remaining hits originated from various bots. Nonetheless, a main limitation of this approach is the lack of ground truth about the origin and real nature of false positives. In fact, the authors were unable to verify that the triggered links were indeed benign and not the result of malicious users who accidentally interacted with the server.

Following a similar approach, Brewer et al. [2010] introduced an experimental testbed that simulated a real web site. Multiple decoy links were also introduced to the web site in order to detect potential attackers. The authors further asked multiple participants to navigate through the web site in order to mimic benign user interactions with the server. All requests towards decoy links during this experiment were then considered false positives since the organizers were not expecting the participants to attack the system. In this case, no user has triggered the decoy links during the experiment, which led to a 0% false positives rate.

Similarly, Ben Salem and Stolfo [2011] evaluated the false positive generated while using decoy documents to detect attackers. Authors grouped 52 student into 4 groups installing respectively 10, 20, 30, or 40 decoy documents on their file system. Whenever a benign user opened the decoy document, an alert was generated. The number of alerts detected starting from one hour after the students had placed the decoy were respectively 2, 6, 9, and 24. Therefore, the authors concluded that the false positive rate increases with the number of decoy files. In [2015], Voris et al. also measured the false positive rate of decoy documents. In their experiment, 27 normal users were asked to install 40 generated decoy documents inside their file system. The authors then collected more than 318 hours of file access logs across all participants, finding that legitimate users accidentally touched decoys less than 7 times over a 8 hour workday. Thus, they suggested a simple threshold should be used to differentiate between attackers and legitimate users.

Finally, Han et al. [2017] performed an evaluation of the false positive rate of web deception techniques under a production environment. The authors integrated honey parameters, honey trap resource, etc., in a Content Management System (CMS) where authenticated users may have access to their private space. Over the period of seven months, no alert had been triggered by the deployed deception elements.

6.5 Summary

This section has reviewed different aspects of how to conduct experiments to rigorously evaluate deception techniques. Previous research have analyzed some of these aspects, such as the assessment of the quality of newly generated deception elements, better than others. Overall, the results seem to suggest that intrusion deception is an effective solution that can complement other defense approaches. However, we believe five points in particular still need more measurement experiments: 1) the assessment of optimal way to manually or automatically place deception elements in a target system, 2) the evaluation of the false negative rate to understand how many attack are performed without triggering any deception element, 3) the evaluation of the false positive rate, in particular in relationship with different placement approaches, 4) the evaluation of how detection degrades over time if deception elements are not constantly replaced with new ones, and 5) the evaluation of how deception can be integrated with other security solutions.

7 DISCUSSION

The research efforts we have surveyed in Sections 3, 4, and 5 describe multiple applications for deception in the computer security domain. In particular, Section 3 introduced a four-dimensions classification for deceptive solutions which aims to organize, from a scientific standpoint, the many existing efforts in this fast evolving field of research. Moreover, Section 4, 5, and 6 discussed the multiple challenges one is faced with when using deception as a defensive security measure and surveyed the way the security research community offered to handle these challenges from three different perspectives, including the modeling, deployment, and evaluation of deception. In this scope, the proposed scheme covers the common security properties, such as the usability (e.g., the generation, placement, and monitoring of deception in Section 5) and also the efficacy (e.g., the effectiveness of detecting attacks and the false positive rate in Section 6). While the large amount of research that we surveyed in this paper contributed to the development and testing of many deceptive solutions, this section highlights some gaps that we consider of particular importance in order to release the full potential of this technology and to achieve a wider consensus among the security research community.

7.1 Lack of Reproducible Experiments

In light of the significant progress in deception-based defenses that was made over the last few years, including the proposal of many new possible applications areas, there is an increasing need towards better evaluating and comparing all these proposed solutions. As discussed in Sections 6.3 and 6.4, the evaluation of deception is particularly challenging since this technology frequently requires a direct interaction with the attacker. In other words, it is often impossible to test deception techniques offline on previously collected datasets, forcing researchers to perform live experiments that are complex to setup and take a considerable amount of time to be executed. Moreover, the fact that humans may need to be included in the evaluation complicates the organization of reproducible and sound experiments.

7.2 Updating Procedures and Re-deployment of Deception

Deceptive solutions remain effective as long as the attacker may not be aware of their existence, including also the possibility for the attacker to properly distinguish a solution from the other genuine assets to be protected. To better understand the way the security research community addressed this property, we reviewed in Section 6.1 and 6.2 the existing state of the art on the generation of realistic and plausible deception, as well as the design of appropriate placement strategies that offer to place deception within a given target environment. Nonetheless, during our survey of the literature, we noticed that much less effort has been dedicated to the update of deception elements and their placement strategy, during a system operation, in order to cope with dynamic changes in the attacker behavior and the target system. It is important to note that the update of deceptive elements is not a straightforward task. Traps may become ineffective once they are too old, but replacing them with new ones may tip the attacker on which are the “fake” elements in the target system.

It is also interesting to note that most previous work focused on creating a “false reality”, that the defenders attempt to keep consistent [2006]. Neagoe et al. [2006] instead proposed to create inconsistent deception systems to mislead adversary without affording the overhead of maintaining the consistency of a complex system. The authors claimed that "inconsistency is an accepted part of life" [Neagoe and Bishop 2006] so that when an adversary discovers inconsistencies in a system, he may attribute his findings to systems faults or his misunderstanding. This setting may ease the problem of updating deceptive elements but unfortunately inconsistent deception has received little attention [Zhang 2012] until now. Moreover, to the best our knowledge, there is not yet any public evaluation of inconsistent deception against consistent deception, and it's still unclear which of them may stand longer with respect to the evolutionary attack landscape.

7.3 Complementary vs. Standalone Solution

Both the industrial and the research community have recently promoted the use of deception in the computer security field. However, a fundamental question is how deception may extend, complement, or replace other existing security solutions. Most of the existing research currently suggests that deception may be an effective solution to complement other defense approaches, while to the best of our knowledge, no previous work claims that deception is suitable for a standalone solution. In cases where deception is used as a complementary solution, it is necessary to understand the benefit and extra efforts of deception deployment in a real-world environment where classic defenses need to co-exist. A few studies have evaluated the financial cost of deploying deception under theoretic models [Rowe 2007; Wang et al. 2013]. In Section 5.4, we reviewed the monitoring of deception and found that there is not yet a comprehensive monitoring system that may incorporate deception into existing security solutions. Therefore, more research is needed to understand the appropriate way deception may extend other solutions.

8 CONCLUSIONS

In this paper, we presented a four dimensional classification of existing deception techniques. Note that our focus was not to discuss all deception techniques presented to date, but mainly to identify the different approaches that can be used to reinforce or substitute current intrusion detection and protection solutions. Our work presents a comprehensive analysis of previous studies, addressing a number of key aspects including the theoretical models that had been proposed for deception techniques in computer security as well as the generation, placement, deployment, and monitoring of deception elements. Finally, we examined previous measurements and evaluations of the effectiveness of deception techniques.

During our study, we found that the use of deception and the type of elements that can be used to deceive an attacker are well covered in dozens of publications. However, we identified several shortcomings such as a lack of a clear methodology to test and measure the placement, the accuracy, the false negative rate, and the false positive rate of such techniques. Therefore, we believe future research should focus on designing and conducting real-world experiments to measure and compare the effectiveness of intrusion deception solutions.

REFERENCES

Mitsuaki Akiyama, Takeshi Yagi, Kazufumi Aoki, Takeo Hariu, and Youki Kadobayashi. 2013. Active credential leakage for observing web-based attack cycle. In *International Workshop on Recent Advances in Intrusion Detection.*

Boniface Kayode Alese, FM Dahunsi, RA Akingbola, OS Adewale, and TJ Ogundele. 2014. Improving deception in honeynet: Through data manipulation. In *IEEE International Conference for Internet Technology and Secured Transactions (ICITST).*

Mohammed Almeshekah and Eugene Spafford. 2014a. The case of using negative (deceiving) information in data protection. In *International Conference on Cyber Warfare and Security.*

Mohammed H Almeshekah. 2015. *Using deception to enhance security: A Taxonomy, Model, and Novel Uses.* Ph.D. Dissertation.

Mohammed H Almeshekah and Eugene H Spafford. 2014b. Planning and integrating deception into computer security defenses. In *ACM Workshop on New Security Paradigms Workshop (NSPW).*

Lance Alt, Robert Beverly, and Alberto Dainotti. 2014. Uncovering network tarpits with degreaser. In *ACM Annual Computer Security Applications Conference (ACSAC)*.

Kostas G Anagnostakis, Stelios Sidiroglou, Periklis Akritidis, Konstantinos Xinidis, Evangelos P Markatos, and Angelos D Keromytis. 2005. Detecting Targeted Attacks Using Shadow Honeypots. In *Usenix Security*.

Chuvakin Anton. 2016. "Deception as Detection" or Give Deception a Chance? http://blogs.gartner.com/anton-chuvakin/2016/01/08/deception-as-detection-or-give-deception-a-chance.

Frederico Araujo and Kevin W Hamlen. 2015. Compiler-instrumented, Dynamic Secret-Redaction of Legacy Processes for Attacker Deception. In *USENIX Security*.

Frederico Araujo, Kevin W Hamlen, Sebastian Biedermann, and Stefan Katzenbeisser. 2014. From patches to honey-patches: Lightweight attacker misdirection, deception, and disinformation. In *ACM SIGSAC conference on computer and communications security (CCS)*.

Rana Aamir Raza Ashfaq, Xi-Zhao Wang, Joshua Zhexue Huang, Haider Abbas, and Yu-Lin He. 2017. Fuzziness based semi-supervised learning approach for intrusion detection system. *Information Sciences* (2017).

Malek Ben Salem and Salvatore J. Stolfo. 2011. Decoy document deployment for effective masquerade attack detection. In *Detection of Intrusions and Malware, and Vulnerability Assessment*.

Maya Bercovitch, Meir Renford, Lior Hasson, Asaf Shabtai, Lior Rokach, and Yuval Elovici. 2011. HoneyGen: An automated honeytokens generator. *IEEE International Conference on Intelligence and Security Informatics (ISI)* (2011).

Leyla Bilge, Thorsten Strufe, Davide Balzarotti, and Engin Kirda. 2009. All Your Contacts Are Belong to Us: Automated Identity Theft Attacks on Social Networks. In *20th International World Wide Web Conference (WWW)*.

Hristo Bojinov, Elie Bursztein, Xavier Boyen, and Dan Boneh. 2010. Kamouflage: Loss-Resistant Password Management. In *European Symposium on Research in Computer Security*.

Kevin Borders, Laura Falk, and Atul Prakash. 2007. OpenFire: Using deception to reduce network attacks. In *Security and Privacy in Communications Networks and the Workshops*.

Brian M. Bowen, Shlomo Hershkop, Angelos D. Keromytis, and Salvatore J. Stolfo. 2009. Baiting inside attackers using decoy documents. *International Conference on Security and Privacy in Communication Systems* (2009).

Brian M Bowen, Vasileios P Kemerlis, Pratap Prabhu, Angelos D Keromytis, and Salvatore J Stolfo. 2010a. Automating the Injection of Believable Decoys to Detect Snooping. *Proceedings of the Third ACM Conference on Wireless Network Security* (2010).

Brian M Bowen, Pratap Prabhu, Vasileios P Kemerlis, Stelios Sidiroglou, Angelos D Keromytis, and Salvatore J Stolfo. 2010b. Botswindler: Tamper resistant injection of believable decoys in vm-based hosts for crimeware detection. In *International Workshop on Recent Advances in Intrusion Detection*.

Douglas Brewer, Kang Li, Laksmish Ramaswamy, and Calton Pu. 2010. A link obfuscation service to detect webbots. *International Conference on Services Computing (SCC)* (2010).

Thomas E Carroll and Daniel Grosu. 2011. A game theoretic investigation of deception in network security. *Security and Communication Networks* (2011).

Hayreddin Çeker, Jun Zhuang, Shambhu Upadhyaya, Quang Duy La, and Boon-Hee Soong. 2016. Deception-Based Game Theoretical Approach to Mitigate DoS Attacks. In *International Conference on Decision and Game Theory for Security*. Springer.

A Čenys, D Rainys, L Radvilavičius, and N Goranin. 2005. Implementation of Honeytoken Module In DBMS Oracle 9ir2 Enterprise Edition for Internal Malicious Activity Detection. In *IEEE Computer Society's TC on Security and Privacy*.

Sambuddho Chakravarty, Georgios Portokalidis, Michalis Polychronakis, and Angelos D. Keromytis. 2011. Detecting traffic snooping in tor using decoys. *Workshop on Recent Advances in Intrusion Detection* (2011).

Andrew Clark, Kun Sun, Linda Bushnell, and Radha Poovendran. 2015. A Game-Theoretic Approach to IP Address Randomization in Decoy-Based Cyber Defense. In *International Conference on Decision and Game Theory for Security*. Springer.

Andrew Clark, Quanyan Zhu, Radha Poovendran, and Tamer Başar. 2012. Deceptive routing in relay networks. In *International Conference on Decision and Game Theory for Security*. Springer.

Fred Cohen. 1998. A note on the role of deception in information protection. *Computers & Security* (1998).

Fred Cohen. 2006. *The Use of Deception Techniques: Honeypots and Decoys.*

Fred Cohen. 2010. Moving target defenses with and without cover deception. http://all.net/Analyst/2010-10.pdf.

Fred Cohen et al. 1998. The deception toolkit. *Risks Digest* (1998).

Fred Cohen and Deanna Koike. 2002. Leading Attackers Through Attack Graphs with Deceptions The Attack Graph. *Computers & Security* (2002).

Fred Cohen, Dave Lambert, Charles Preston, Nina Berry, Corbin Stewart, and Eric Thomas. 2001a. A framework for deception. *National Security Issues in Science, Law, and Technology* (2001).

Fred Cohen, Irwin Marin, Jeanne Sappington, Corbin Stewart, and Eric Thomas. 2001b. Red teaming experiments with deception technologies. http://all.net/journal/deception/RedTeamingExperiments.pdf.

Stephen Crane, Per Larsen, Stefan Brunthaler, and Michael Franz. 2013. Booby trapping software. In *Proceedings of the 2013 workshop on New security paradigms workshop*.

Michael Crouse, Bryan Prosser, and Errin W Fulp. 2015. Probabilistic performance analysis of moving target and deception reconnaissance defenses. In *Proceedings of the Second ACM Workshop on Moving Target Defense*.

Emiliano De Cristofaro, Arik Friedman, Guillaume Jourjon, Mohamed Ali Kaafar, and M Zubair Shafiq. 2014. Paying for likes?: Understanding Facebook like fraud using honeypots. In *Conference on Internet Measurement Conference*.

Cristiano De Faveri and Ana Moreira. 2016. Designing Adaptive Deception Strategies. In *IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*.

Cristiano De Faveri, Ana Moreira, and Vasco Amaral. 2016. Goal-driven deception tactics design. In *IEEE International Symposium on Software Reliability Engineering (ISSRE)*.

Karel Durkota, Viliam Lisỳ, Branislav Bošanskỳ, and Christopher Kiekintveld. 2015a. Approximate solutions for attack graph games with imperfect information. In *International Conference on Decision and Game Theory for Security*. Springer.

Karel Durkota, Viliam Lisỳ, Branislav Bosanskỳ, and Christopher Kiekintveld. 2015b. Optimal Network Security Hardening Using Attack Graph Games.. In *IJCAI*.

Karel Durkota, Viliam Lisỳ, Christopher Kiekintveld, Branislav Bošanskỳ, and Michal Pěchouček. 2016. Case Studies of Network Defense with Attack Graph Games. *IEEE Intelligent Systems* (2016).

Simson Garfinkel. 2007. Anti-forensics: Techniques, detection and countermeasures.

Nandan Garg and Daniel Grosu. 2007. Deception in honeynets: A game-theoretic analysis. In *IEEE Information Assurance and Security Workshop*.

Dimitris Gavrilis, Ioannis Chatzis, and Evangelos Dermatas. 2007. Flash crowd detection using decoy hyperlinks. *International Conference on Networking, Sensing and Control (ICNSC)* (2007).

Han C Goh. 2007. *Intrusion deception in defense of computer systems*. Technical Report.

Christopher N Gutierrez, Saurabh Bagchi, H Mohammed, and Jeff Avery. 2015. Modeling Deception In Information Security As A Hypergame–A Primer. In *Proceedings of the 16th Annual Information Security Symposium*. CERIAS-Purdue University.

Xiao Han, Nizar Kheir, and Davide Balzarotti. 2017. Evaluation of Deception-Based Web Attacks Detection. In *ACM Workshop on Moving Targets Defense (co-located with CCS)*.

Sharif Hassan and Ratan Guha. 2016. Modelling of the State of Systems with Defensive Deception. In *IEEE International Conference on Computational Science and Computational Intelligence (CSCI)*.

Kristin E Heckman, Frank J Stech, Ben S Schmoker, and Roshan K Thomas. 2015. Denial and Deception in Cyber Defense. *IEEE Computer* (2015).

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* (2013).

Cormac Herley and Paul C van Oorschot. 2017. Sok: Science, security and the elusive goal of security as a scientific pursuit. In *IEEE Symposium on Security and Privacy (SP)*.

Karel Horák, Quanyan Zhu, and Branislav Bošanskỳ. 2017. Manipulating Adversary's Belief: A Dynamic Game Approach to Deception by Design for Proactive Network Security. In *International Conference on Decision and Game Theory for Security*. Springer.

Kenneth Houkjær, Kristian Torp, and Rico Wind. 2006. Simple and realistic data generation. In *Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment.

Sushil Jajodia, Anup K Ghosh, VS Subrahmanian, Vipin Swarup, Cliff Wang, and X Sean Wang. 2012. *Moving Target Defense II: Application of Game Theory and Adversarial Modeling*. Springer.

Sushil Jajodia, Anup K Ghosh, Vipin Swarup, Cliff Wang, and X Sean Wang. 2011. *Moving target defense: creating asymmetric uncertainty for cyber threats*. Springer Science & Business Media.

Sushil Jajodia, VS Subrahmanian, Vipin Swarup, and Cliff Wang. 2016. *Cyber Deception: Building the Scientific Foundation.*

Priyanka Jogdand and Puja Padiya. 2016. Survey of different IDS using honeytoken based techniques to mitigate cyber threats. In *IEEE International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*.

Ari Juels and Thomas Ristenpart. 2014. Honey encryption: Security beyond the brute-force bound. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*.

Ari Juels and Ronald L Rivest. 2013. Honeywords: Making password-cracking detectable. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*.

Ari Juels and Cornell Tech. 2014. A Bodyguard of Lies : The Use of Honey Objects in Information Security. *ACM symposium on Access control models and technologies (SACMAT)* (2014).

Donald P Julian. 2002. *Delaying Type Response for Use By Software Decoys*. Ph.D. Dissertation.

Parisa Kaghazgaran and Hassan Takabi. 2015. Toward an Insider Threat Detection Framework Using Honey Permissions. *Journal of Internet Services and Information Security (JISIS)* (2015).

Alexandros Kapravelos, Chris Grier, Neha Chachra, Christopher Kruegel, Giovanni Vigna, and Vern Paxson. 2014. Hulk: Eliciting Malicious Behavior in Browser Extensions.. In *USENIX Security*.

Josh Karlin, Daniel Ellard, Alden W Jackson, Christine E Jones, Greg Lauer, David Mankins, and W Timothy Strayer. 2011. Decoy Routing: Toward Unblockable Internet Communication.. In *FOCI*.

Constantine Katsinis and Brijesh Kumar. 2012. A Security Mechanism for Web Servers Based on Deception. *Proceedings of the The 2012 International Conference on Internet Computing (ICOMP)* (2012).

Constantine Katsinis, Brijesh Kumar, Security Technology, and Rapidsoft Systems. 2013. A Framework for Intrusion Deception on Web Servers. (2013).

Gene H Kim and Eugene H Spafford. 1994a. The design and implementation of tripwire: A file system integrity checker. In *ACM Conference on Computer and Communications Security*.

Gene H Kim and Eugene H Spafford. 1994b. Experiences with tripwire: Using integrity checkers for intrusion detection. In *System Administration, Networking, and Security Conference*.

Georgios Kontaxis, Michalis Polychronakis, and Angelos D Keromytis. 2014. Computational Decoys for Cloud Security. In *Secure Cloud Computing*.

Spitzner Lance. 2001. The Value of Honeypots, Part One: Definitions and Values of Honeypots. https://www.symantec.com/connect/articles/value-honeypots-part-one-definitions-and-values-honeypots.

Per Larsen, Andrei Homescu, Stefan Brunthaler, and Michael Franz. 2014. SoK: Automated software diversity. In *IEEE Symposium on Security and Privacy (SP)*.

Martin Lazarov, Jeremiah Onaolapo, and Gianluca Stringhini. 2016. Honey Sheets: What Happens To Leaked Google Spreadsheets?. In *USENIX Workshop on Cyber Security Experimentation and Test (CSET)*.

Erwan Le Malécot. 2009. MitiBox: camouflage and deception for network scan mitigation. In *Proceedings of the 4th USENIX Workshop on Hot Topics in Security (HotSec)*.

Xiaowei Li and Yuan Xue. 2014. A survey on server-side approaches to securing web applications. *ACM Computing Surveys (CSUR)* (2014).

Tom Liston. 2001. LaBrea:"Sticky" Honeypot and IDS. http://labrea.sourceforge.net/labrea-info.html. (2001).

Bingshuang Liu, Zhaoyang Liu, Jianyu Zhang, Tao Wei, and Wei Zou. 2012. How many eyes are spying on your shared folders?. In *ACM workshop on Privacy in the electronic society*.

Craig M. McRae and Rayford B. Vaughn. 2007. Phighting the phisher: Using Web bugs and honeytokens to investigate the source of phishing attacks. *Proceedings of the Annual Hawaii International Conference on System Sciences* (2007).

B Michael, M Auguston, N Rowe, and R Riehle. 2002. Software Decoys: Intrusion Detection and Countermeasures. In *Proceedings of the IEEE Workshop on Information Assurance*.

Sherry Murphy, Todd McDonald, and Robert Mills. 2010. An Application of Deception in Cyberspace: Operating System Obfuscation. In *International Conference on Information Warfare and Security*.

Milad Nasr, Hadi Zolfaghari, and Amir Houmansadr. 2017. The Waterfall of Liberty: Decoy Routing Circumvention that Resists Routing Attacks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*.

Marcin Nawrocki, Matthias Wählisch, Thomas C Schmidt, Christian Keil, and Jochen Schönfelder. 2016. A Survey on Honeypot Software and Data Analysis. *arXiv preprint arXiv:1608.06249* (2016).

Jose Nazario. 2009. PhoneyC: A Virtual Client Honeypot. *LEET* (2009).

Vicentiu Neagoe and Matt Bishop. 2006. Inconsistency in deception for defense. In *Proceedings of the 2006 workshop on New security paradigms*. ACM.

Anh Nguyen-Tuong, Salvatore Guarnieri, Doug Greene, Jeff Shirley, and David Evans. 2005. Automatically hardening web applications using precise tainting. *Security and Privacy in the Age of Ubiquitous Computing* (2005).

Nick Nikiforakis, Marco Balduzzi, Steven Van Acker, Wouter Joosen, and Davide Balzarotti. 2011. Exposing the Lack of Privacy in File Hosting Services.. In *LEET*.

Adam Nossiter, David E. Sanger, and Nicole Perlroth. 2017. Hackers Came, but the French Were Prepared. https://www.nytimes.com/2017/05/09/world/europe/hackers-came-but-the-french-were-prepared.html.

Hamed Okhravi, Thomas Hobson, David Bigelow, and William Streilein. 2014. Finding focus in the blur of moving-target techniques. *IEEE Security & Privacy* (2014).

Hamed Okhravi, MA Rabe, TJ Mayberry, WG Leonard, TR Hobson, D Bigelow, and WW Streilein. 2013. *Survey of cyber moving target techniques.* Technical Report. MASSACHUSETTS INST OF TECH LEXINGTON LINCOLN LAB.

Jeremiah Onaolapo, Enrico Mariconti, and Gianluca Stringhini. 2016. What Happens After You Are Pwnd: Understanding the Use of Leaked Webmail Credentials in the Wild. In *ACM SIGCOMM Internet Measurement Conference*.

Keshnee Padayachee. 2014. Aspectising honeytokens to contain the insider threat. *IET Information Security* (2014).

Augusto Paes de Barros. 2003. RES: Protocol Anomaly Detection IDS - Honeypots. http://seclists.org/focus-ids/2003/Feb/95.

Younghee Park and Salvatore J Stolfo. 2012. Software decoys for insider threat. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*.

AB Robert Petrunić. 2015. Honeytokens as active defense. *International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)* (2015).

Radek Píbil, Viliam Lisỳ, Christopher Kiekintveld, Branislav Bošanskỳ, and Michal Pěchouček. 2012. Game theoretic model of strategic honeypot selection in computer networks. *Decision and Game Theory for Security* (2012).

Lawrence Pingree. 2015. Emerging Technology Analysis: Deception Techniques and Technologies Create Security Technology Business Opportunities. *Gartner, Inc* (2015).

Niels Provos et al. 2004. A Virtual Honeypot Framework.. In *USENIX Security Symposium.*

Mohammad Ashiqur Rahman, Mohammad Hossein Manshaei, and Ehab Al-Shaer. 2013. A game-theoretic approach for deceiving remote operating system fingerprinting. In *IEEE Conference on Communications and Network Security (CNS).*

Neil Rowe. 2007. Planning cost-effective deceptive resource denial in defense to cyber-attacks. In *Proceedings of the 2nd International Conference on Information Warfare & Security.*

Neil C Rowe. 2004. Designing good deceptions in defense of information systems. In *IEEE Computer Security Applications Conference.*

Neil C Rowe. 2006. Measuring the effectiveness of honeypot counter-counterdeception. In *IEEE Annual Hawaii International Conference on System Sciences (HICSS).*

Neil C Rowe. 2008. Deception in Defense of Computer Systems from Cyber Attack. *Cyber Warfare and Cyber Terrorism* (2008).

Neil C Rowe, Binh T Duong, and E Custy. 2006. Fake Honeypots: A Defensive Tactic for Cyberspace. In *IEEE Information Assurance Workshop.*

Neil C Rowe and Hy S Rothstein. 2004. Two taxonomies of deception for attacks on information systems. (2004).

Neil C Rowe and Julian Rrushi. 2016. *Introduction to Cyberdeception.* Springer.

Julian L. Rrushi. 2011. An exploration of defensive deception in industrial communication networks. *International Journal of Critical Infrastructure Protection* (2011).

Julian L Rrushi. 2016. NIC displays to thwart malware attacks mounted from within the OS. *Computers & Security* (2016).

Karen Scarfone and Peter Mell. 2007. Guide to intrusion detection and prevention systems (idps). *NIST special publication* (2007).

Asaf Shabtai, Maya Bercovitch, Lior Rokach, Ya'akov (Kobi) Gal, Yuval Elovici, and Erez Shmueli. 2016. Behavioral Study of Users When Interacting with Active Honeytokens. *ACM Trans. Inf. Syst. Secur.* (2016).

Leslie Shing. 2016. *An improved tarpit for network deception.* Master's thesis. Monterey, California: Naval Postgraduate School.

Matthew Smart, G Robert Malan, and Farnam Jahanian. 2000. Defeating TCP/IP Stack Fingerprinting. In *Usenix Security Symposium.*

Lance Spitzner. 2003a. The honeynet project: Trapping the hackers. *IEEE Security & Privacy* (2003).

Lance Spitzner. 2003b. Honeypots : Catching the Insider Threat. In *Annual Computer Security Applications Conference.*

Lance Spitzner. 2003c. Honeytokens: The other honeypot. http://www.securityfocus.com/infocus/1713.

Cliff Stoll. 1989. *The cuckoo's egg: tracking a spy through the maze of computer espionage.*

Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. 2010. Detecting spammers on social networks. In *Proceedings of the 26th annual computer security applications conference.*

Symantec. 2016. Internet Security Threat Report . https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf.

Samuel T Trassare. 2013. A technique for presenting a deceptive dynamic network topology. (2013).

Trend Micro. 2015. Understanding Targeted Attacks: The Impact of Targeted Attacks. https://www.trendmicro.com/vinfo/us/security/news/cyber-attacks/the-impact-of-targeted-attack.

Vincent E Urias, William MS Stout, and Han W Lin. 2016. Gathering threat intelligence through computer network deception. In *IEEE Symposium on Technologies for Homeland Security (HST).*

Nikos Virvilis, Bart Vanautgaerden, and Oscar Serrano Serrano. 2014. Changing the game: The art of deceiving sophisticated attackers. *International Conference on Cyber Conflict, CYCON* (2014).

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.* ACM.

Jonathan Voris, Jill Jermyn, Angelos D Keromytis, and Salvatore J Stolfo. 2013. Bait and snitch: Defending computer systems with decoys. In *Proceedings of the cyber infrastructure protection conference, Strategic Studies Institute, September.*

Wei Wang, Jeffrey Bickford, Ilona Murynets, Ramesh Subbaraman, Andrea G. Forte, and Gokul Singaraju. 2013. Detecting Targeted Attacks By Multilayer Deception. *Journal of Cyber Security and Mobility* (2013).

Steve Webb, James Caverlee, and Calton Pu. 2008. Social Honeypots: Making Friends With A Spammer Near You. In *CEAS.*

Jonathan White. 2010. Creating personally identifiable honeytokens. In *Innovations and Advances in Computer Sciences and Engineering.* Springer.

Ben Whitham. 2013. Automating the generation of fake documents to detect network intruders. *International Journal of Cyber-Security and Digital Forensics (IJCSDF)* (2013).

Ben Whitham. 2017. Automating the Generation of Enticing Text Content for High-Interaction Honeyfiles. In *Proceedings of the 50th Hawaii International Conference on System Sciences*.

Jim Yuill, Dorothy E Denning, and Fred Feer. 2006. *Using deception to hide things from hackers: Processes, principles, and techniques*. Technical Report. DTIC Document.

J. Yuill, M. Zappe, D. Denning, and F. Feer. 2004. Honeyfiles: deceptive files for intrusion detection. *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop* (2004).

James Joseph Yuill. 2006. *Defensive Computer-security Deception Operations: Processes, Principles and Techniques*. Ph.D. Dissertation. Advisor(s) Vouk, Mladen and Anton, Ana I.

Du Zhang. 2012. The utility of inconsistency in information security and digital forensics. In *Recent trends in information reuse and integration*. Springer.