Chapter 1
Computational Decoys for Cloud Security

Georgios Kontaxis, Michalis Polychronakis, and Angelos D. Keromytis

Abstract Cloud-based applications benefit from the scalability and efficiency offered by server consolidation and shared facilities. However, the shared nature of cloud infrastructures may introduce threats stemming from the co-location and combination of untrusted components, in addition to typical risks due to the inevitable presence of weaknesses in the infrastructure itself. As a result, adversaries may be able to place themselves in monitoring proximity to high-value targets and gain unauthorized access to sensitive data. In this paper we present DIGIT, a system that employs *decoy computation* to impede the ability of adversaries to take advantage of unauthorized access to sensitive information. DIGIT introduces uncertainly as to which data and computation is legitimate by generating a mix of real and decoy activity within a cloud application. Although DIGIT may not impede intruders indefinitely, it prevents them from determining whether a captured system is handling actual or bogus processing within a reasonable amount of time. As adversaries cannot easily distinguish between real and decoy activity, they have to either risk triggering beacon-bearing data that can be traced back to them, or expend significant effort to pinpoint any actual data of interest, forcing them to reveal their presence.

1.1 Introduction

The multifaceted benefits of cloud computing have led to its rapid adoption for the deployment of online services and applications. As businesses and individuals increasingly rely on the cloud, the threat of unauthorized data access or full compromise of cloud services becomes more pertinent. The recent spate of security breaches in major online services [1–3, 6, 13, 25, 29] is indicative, and shows that despite major advances in security research and engineering, vulnerabilities in software components, protocol design, system configuration, operational procedures,

Network Security Lab, Columbia University
e-mail: {kontaxis,mikepo,angelos}@cs.columbia.edu

and other aspects of complex systems will continue to put cloud-based applications at risk.

The increasing sophistication of attack methods and exploitation techniques has shown that even the latest protection techniques can be bypassed, and the most up-to-date detection systems can be evaded. The added need of defending against tenants who have legitimate access but behave maliciously against other users of the same cloud service increases the complexity of the problem [23]. This situation necessitates the implementation of domain-specific "defense in depth" strategies that combine multiple and diverse security measures. Prior research on cloud security has focused on various aspects of cloud infrastructures, including data and network isolation [20], software attestation [11], and data availability [10,16]. Although most research efforts have focused on systems and methods for hardening cloud-based systems and enabling the detection and prevention of security incidents, less attention has been given to "second-line" defenses for hindering attackers that have managed to gain access to parts of a system, or insider threats.

In this work we propose the concept of *computational decoys*, a novel approach that encompasses deceptive information and "throw-away" computation to impede the ability of an adversary to take advantage of any initial success they may have in compromising a system. The main goal of our approach is to introduce uncertainty as to the validity and authenticity of data captured by an adversary after gaining unauthorized access in one or more hosts, and in some scenarios, reveal the presence of the adversary.

We have applied this approach in *DIGIT*, a deceptive information generation, injection, and tracking system aimed at detecting and confusing adversaries in cloud settings. The system is based on a large number of application replicas, some of which (the "deception set") are provided with fake inputs. An adversary controlling a malicious or compromised replica will be uncertain as to the validity of any captured data. The whole process is orchestrated by an application-level proxy that mixes and dispatches real and fake requests to primary and decoy application replicas, respectively. Primary and decoy replicas can be swapped at any time simply by changing the source of inputs, increasing the confusion to potential adversaries.

A key aspect of the proposed mechanism is the believability of the generated decoy traffic, and consequently the fidelity of the evoked computation on a decoy replica. Enticing decoy traffic is generated automatically based on real client traffic using context-aware protocol field randomization. The generated inputs can contain specially crafted data whose misuse by an adversary can be subsequently detected. Examples of such enticing bait information include documents with built-in "beacons," URLs or credentials to honeypots or sites whose access can be directly or indirectly monitored, credit card and bank account numbers with triggers, and so on [8,9]. Other types of decoy information that we propose to use include deceptive documents in the file system and entries in database tables (or entire databases)—the exact type of bait used depends on the application. Besides application-level protocols, deceptive computation can be introduced at different levels, e.g., by simulating user activity at a higher level through the generation of key strokes and mouse input operations.

1.2 Threat Model

Our threat model revolves around a cloud computing environment where partitioned applications handle data-oriented user requests. We consider adversaries that have infiltrated one or more but not all server instances or modules of a cloud application, and have the ability to monitor the execution of the server program as well as its data flows, including user requests and program responses. We assume that, in case more than one cloud instances have been breached, adversaries may correlate information received from different back-end points. However, adversaries do not have the ability to simultaneously monitor network traffic both inside and at the edges of the cloud. This means that adversaries cannot determine whether a specific connection comes directly from the outside, or originates from a cloud-local proxy.

Finally, we consider that the amount of data collected in a production environment prohibits efficient analysis by humans within a realistic time frame. Therefore, we assume that an adversary's efforts to verify the quality of captured information are automated and rely on behavioral heuristics as well as grammatical and statistical analysis of the data rather than human interpretation of the content and context of the collected information. Note that server instances are oblivious to the use of computational decoys, and thus cannot hint an attacker as to their existence. Any access of decoy information by an attacker will lead to an alert signaling the breach, as well as to the immediate identification of the breached instance, as decoys are unique to the environment in which they have been deployed and their recipient.

1.3 Design

In this section we present the design of our application-level system that uses computational decoys to deceive an infiltrating adversary stealing information. Our design can be overlaid on top of existing infrastructures without rearranging the production environment. Figure 1.1 illustrates the modular structure of our proposed system and presents the data relationships within the system itself, as well as the exchange of flows within a cloud-computing setup.

A set of application replicas with identical functionality as the original application servers and components receives decoy requests. All requests to application replicas are handled normally, as if they were real, resulting in decoy computation and system activities indistinguishable from those of real application instances. The main component of DIGIT is placed at the edge of the cloud where a typical SSL terminator or load balancer typically terminates user connections. This choice is made for two main reasons: first, the system must be interposed in the communication of clients with the cloud, and second, this placement enables the system to acquire a high-level view of the application instances operating at any given time. The overall operation of the system consists of four stages: incoming traffic interception and classification, decoy generation, decoy dissemination, and outgoing traffic reconciliation.

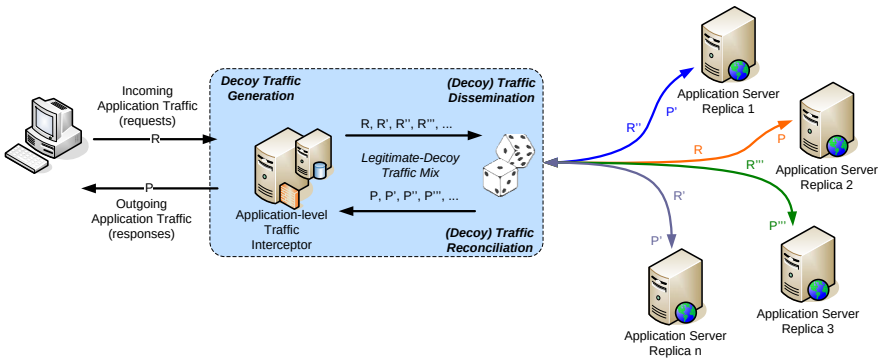

Fig. 1.1 High-level overview of DIGIT's architecture. DIGIT is designed as an overlay on top of existing cloud infrastructures, and consists of an application-level traffic interception and decoy generation system, and application replicas with identical functionality as the original application servers and components. The modular design of the system allows it to be easily extended with support for more applications (application-level traffic interception) and evolve the quality of the decoys (decoy generation).

Initially, the system intercepts and identifies the type of incoming application traffic. The type is defined as the combination of application protocol (e.g., HTTP) and target application (e.g., messages targeting the end points of an e-commerce site). Recognizing the type of incoming traffic is necessary for proceeding with traffic analysis and the generation of decoys tailored to the particular application. Traffic interception, shown in the center part of Figure 1.1, is organized around a series of application-specific modules that register traffic filters with the core interceptor module. Upon a match with one of these filters, the appropriate module is called and the system forwards any related incoming traffic to the analysis module.

That analysis module is aware of the application protocol and identifies specific messages to provide the necessary context to the decoy generation modules. The analysis phase takes into account context-specific information such as client sessions. The output of the analysis phase consists of an application and message specific template which will be used for decoy generation. Although we aim at producing such templates in real time, an administrator can provide them based on protocol specifications of a specific application. In that case, the output of the analysis phase is a reference containing a specific template for use. towards the decoy generation component.

The decoy generation component receives message templates (and optionally context-related information) and produces decoy messages that are required to be indistinguishable from the original message when they reach an application server instance. To do so it follows a generation-evaluation cycle that can be extended to be reactive to feedback from the evaluation phase.

The decoy dissemination component follows decoy generation and utilizes knowledge about the setup of a cloud environment to distribute decoys and legitimate messages among the instances of an application server. It makes no assumptions about

the status of an application, e.g., whether it is compromised or not. It may take into account information from the load balancer so as to distribute decoys and legitimate messages in a manner similar to the load balancer's intended behavior.

Finally, the decoy reconciliation component receives responses to the decoy and legitimate requests from the application instances, discards the decoy responses, and forwards the legitimate responses to the client. Optionally, if copies of the legitimate message were given to more than one instances, it attempts to synthesize a consistent view of the appropriate response through the formation of a response consensus.

Security Model.

Although DIGIT does not assume anything about the security of the cloud in which it operates, there should be security guarantees about its own components. The DIGIT proxy and its associated components should be protected against remote attacks from an adversary. This is analogous to the protection offered by a trusted platform module (TPM) [30] in single-system computing. As long as the TPM itself remains secure it can be effective in its role as a crypto-system.

Target integration.

Considering DIGIT as a gateway component in a cloud setup is only our initial approach. Effectiveness and scalability factors drive the question whether DIGIT could be realized closer to the actual system it protects. It would be interesting to investigate whether hardware, hypervisor or even application-driven approaches could introduce deceptive computation.

1.4 Decoy Generation

The key challenge in generating decoy traffic is that it should appear realistic and indistinguishable from actual user-generated traffic. We aim to satisfy this requirement based on the assumption that an attacker has a limited view of the cloud infrastructure and controls and monitors the behavior of a subset of the application server instances. Our goal is for decoy requests to carry the same properties as the actual user input and make the server to behave in the same way. We assume that actual and decoy computation on an application replica is the same as long as, given a real and a corresponding decoy request, similar or identical code execution paths are followed. Currently, we do not place any context-related constraints (e.g., a series of valid protocol messages or application requests that a user would be unlikely to perform in a specific order) as we assume that an attacker would not attempt to distinguish decoys in such manner. However, our approach can be extended to include more decoy evaluation heuristics.

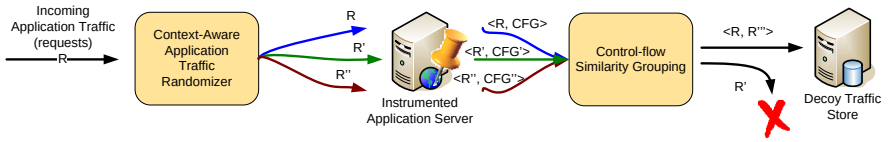

Fig. 1.2 Decoy traffic generation based on an (optionally feedback-assisted) generation-evaluation approach. As more evaluation heuristics become available, the quality of generated decoys can adapt to actual user traffic. Decoy messages that achieve high fidelity are stored in a database for future use.

Figure 1.2 presents the process of generating realistic computational decoys. Decoy generation begins with a set of templates for protocol messages generated by popular client applications (e.g., web browsers). Templates are used to generate random permutations in the acceptable value space for the parameters or content of a given message type. For instance, if the message is an HTTP GET request carrying the "PIN" parameter with a space of four numeric characters ([0-9]{4}) the system would generate all permutations. Alternatively, for a "search_word" parameter with a space defined by a dictionary of the English language we would generate an appropriate number of decoys or enough realistic decoys to satisfy a given quota.

The system then evaluates all generated decoys against the actual user input from a training set using the heuristics mentioned above. Decoy messages that exhibit similar or identical application server behavior are kept, while the rest are discarded. As dynamic binary instrumentation is computationally expensive, we make a time-space trade-off and store the produced decoys for future use rather than carry out real-time generation and evaluation.

Early Prototype.

To assess our decoy generation approach we have implemented as an early prototype the common scenario of web applications in a cloud environment. The focus of our prototype is the automated generation of HTTP message decoys, an effort which is expected to act as a guideline on practical requirements in system components and procedures so that we may adjust our design if necessary.

Our HTTP server is Lighttpd, which uses a single-threaded queue-based workflow for processing user requests. We chose this server due to its simpler control flow compared to multi-threaded event-based server implementations, which enables us to create more easily compare the control flow graphs of real and decoy executions. In a production environment, an attacker might have to deal with the complexity introduced by multi-threaded event-based servers when trying to identify abnormal execution behavior.

We implemented a tool for the Pin dynamic binary instrumentation framework [19] to output the control flow graph of Lighttpd, initially at the function level, and

later at the basic block level. The tool runs twice for the same legitimate user input, in our case an HTTP GET request, to identify the invariant parts of the control flow graph. We then run the tool for each generated decoy and compare the output graph to the invariant graph of the legitimate input to decide which variations of the original input qualify as realistic decoys.

Our example scenario consists of a simple service that handles single-keyword dictionary queries. The application returns an HTTP 404 "Not Found" status code and no content for queries with a keyword not contained in a pre-defined dictionary, and an HTTP 200 status and the relevant content for matching keywords. Our goal was to generate and evaluate decoys based only on the knowledge that the application expected HTTP GET requests that carried a parameter named "query," and that the parameter accepted input of arbitrary length in the character space a–z. Overall, assuming zero knowledge about the internals of the web application, we were able to produce decoy inputs that returned valid HTTP status codes and content given user inputs with the same behavior.

1.5 Related Work

The concept of deception in the context of computer systems and networks, with the aim to mislead intruders and reveal their presence and actions, has been applied in many variations and at multiple levels [24, 34].

The use of diversionary mechanisms for inducing intruders to spend precious time on non-essential part of a system, and eventually reveal their presence has been considered since the days of mainframe computers. Early proposals included the insertion of pseudo-flaws in existing system components and the installation of entrapment modules [17].

Fully-blown computer traps purposely set up and heavily monitored by security administrators to lure prospective intruders are widely known as honeypots [21, 22, 26, 28]. Honeypots do not have any legitimate users and do not provide any regular production service. Therefore, under normal conditions they should remain idle, neither receiving nor generating any traffic, or generating any other activity. Shadow honeypots [4] combine honeypots with network-level anomaly detection mechanisms to enable their integration with production systems.

Besides decoy systems or system components, the use of decoy information can also confuse intruders and unveil their actions. Decoys may consist of bogus medical records, credit card numbers, credentials, and other bait data relevant to each case, also known as honeytokens [27] or honeyfiles [33]. When a bait file is stolen and later accessed, it can transparently send an alert that reveals the location of the action. Bowen et al. have proposed techniques for generating believable decoys indistinguishable from actual data at the network and host level [8, 9]. They do so by capturing real-user actions in a production environment, such as opening documents and browsing the web, and then replaying within virtual machines to simulate the presence of a human operator, thus making them more believable to an infiltrating

adversary. Bojinov et al. [7] propose a methodology for generating password decoys that closely resemble the ones of a particular user. To do so they analyze the grammatical properties of each password, output corresponding templates and use them to generate similar, thus realistic, passwords.

Currently, our system uses protocol message templates to generate realistically looking application traffic and activity. However, several techniques for automatically extracting protocol specifications from their corresponding implementations [5, 18, 32], or protocol messages [12, 14, 15] can also be employed. Wang et al. [31] have employed differential black-box protocol analysis to uncover the syntax and semantics of application-level single-sign-on protocols. They are thus able to automatically identify message attributes that are unique to the session, user or device as well as integrity-verification fields, parameter propagation chains and authentication-enabling secrets.

1.6 Conclusion

In this paper we have presented our work on DIGIT, a system which employs computational decoys to introduce uncertainty and deceive an adversary who has infiltrated a cloud setup for stealing user information. Our design can be overlaid on top of an existing infrastructure which remains agnostic to the use of decoys. We generate realistic application-specific decoys by requiring that they carry the same properties and result in the same behavior by the server instance located at the backend of the cloud.

References

1. Hacker Posts 6.4 Million LinkedIn Passwords. `http://www.technewsdaily.com/7839-linked-passwords-hack.html`. December 2012.
2. Sony Hacked Again, 1 Million Passwords Exposed. `http://www.informationweek.com/security/attacks/sony-hacked-again-1-million-passwords-ex/229900111`.
3. Twitter detects and shuts down password data hack in progress. `http://arstechnica.com/security/2013/02/twitter-detects-and-shuts-down-password-data-hack-in-progress/`. February 2013.
4. Kostas G. Anagnostakis, Stelios Sidiroglou, Periklis Akritidis, Kostas Xinidis, Evangelos P. Markatos, and Angelos D. Keromytis. Detecting Targeted Attacks Using Shadow Honeypots. In *Proceedings of the* 14^{th} *USENIX Security Symposium*, pages 129–144, August 2005.
5. Guangdong Bai, Jike Lei, Guozhu Meng, Sai Sathyanarayan Venkatraman, Prateek Saxena, Jun Sun, Yang Liu, and Jin Song Dong. AUTHSCAN: Automatic extraction of web authentication protocols from implementations. In *Proceedings of the 20th Network and Distributed Systems Security Symposium (NDSS)*, 2013.
6. H. Berghel. Identity theft and financial fraud: Some strangeness in the proportions. *Computer*, 45(1):86 –89, jan. 2012.
7. Hristo Bojinov, Elie Bursztein, Xavier Boyen, and Dan Boneh. Kamouflage: Loss-resistant password management. In *Proc. of ESORICS'10*, 2010.

8. Brian M. Bowen, Vasileios P. Kemerlis, Pratap V. Prabhu, Angelos D. Keromytis, and Salvatore J. Stolfo. A system for generating and injecting indistinguishable network decoys. *Journal of Computer Security*, 20(2-3):199–221, 2012.
9. Brian M. Bowen, Pratap Prabhu, Vasileios P. Kemerlis, Stelios Sidiroglou, Angelos D. Keromytis, and Salvatore J. Stolfo. Botswindler: tamper resistant injection of believable decoys in vm-based hosts for crimeware detection. In *Proceedings of the 13th international conference on Recent advances in intrusion detection*, RAID'10, pages 118–137, Berlin, Heidelberg, 2010. Springer-Verlag.
10. Kevin D. Bowers, Ari Juels, and Alina Oprea. HAIL: a High-Availability and Integrity Layer for Cloud Storage. In *Proc. of CCS*, pages 187–198, 2009.
11. Andrew Brown and Jeff Chase. Trusted Platform-as-a-Service: A Foundation for Trustworthy Cloud-Hosted Applications. In *Proc. of CCSW*, 2011.
12. Paolo Milani Comparetti, Gilbert Wondracek, Christopher Kruegel, and Engin Kirda. Prospex: Protocol specification extraction. In *Proceedings of the 30th IEEE Symposium on Security and Privacy*, pages 110–125, 2009.
13. Computerworld. Microsoft BPOS cloud service hit with data breach, Dec 2010. `http://www.computerworld.com/s/article/9202078/Microsoft_BPOS_cloud_service_hit_with_data_breach`.
14. Weidong Cui, Vern Paxson, Nicholas C. Weaver, and Y H. Katz. Protocol-independent adaptive replay of application dialog. In *Proceedings of the 13th Network and Distributed System Security Symposium (NDSS)*, 2006.
15. Holger Dreger, Anja Feldmann, Michael Mai, Vern Paxson, and Robin Sommer. Dynamic application-layer protocol analysis for network intrusion detection. In *Proceedings of the 15th USENIX Security Symposium*, 2006.
16. Chris Erway, Alptekin Küpçü, Charalampos Papamanthou, and Roberto Tamassia. Dynamic provable data possession. In *Proceedings of the 16th ACM conference on Computer and Communications Security (CCS)*, pages 213–222, 2009.
17. Dennis Hollingsworth. Enhancing computer system security. Technical Report P-5064, RAND Corporation, Aug 1973.
18. Zhiqiang Lin, Xuxian Jiang, Dongyan Xu, and Xiangyu Zhang. Automatic protocol format reverse engineering through context-aware monitored execution. In *Proceedings of the 15th Network and Distributed System Security Symposium (NDSS)*, 2008.
19. Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. Pin: building customized program analysis tools with dynamic instrumentation. In *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*, PLDI '05, pages 190–200, New York, NY, USA, 2005. ACM.
20. Yogesh Mundada, Anirudh Ramachandran, and Nick Feamster. SilverLine: Data and Network Isolation for Cloud Services. In *Proc. of HotCloud*, 2011.
21. Niels Provos. A virtual honeypot framework. In *Proceedings of the* 13^{th} *USENIX Security Symposium*, pages 1–14, August 2004.
22. Niels Provos and Thorsten Holz. *Virtual honeypots: from botnet tracking to intrusion detection*. Addison-Wesley Professional, 2007.
23. Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proc. of CCS*, pages 199–212, 2009.
24. Neil C. Rowe and Hy S. Rothstein. Two taxonomies of deception for attacks on information systems. *Journal of Information Warfare*, 3(2):27–39, 2004.
25. Sophos. Groupon subsidiary leaks 300k logins, fixes fail, fails again, 2011 Jun. `http://nakedsecurity.sophos.com/2011/06/30/groupon-subsidiary-leaks-300k-logins-fixes- fail-fails-again/`.
26. Lance Spitzner. *Honeypots: Tracking Hackers*. Addison-Wesley Longman Publishing Co., Inc., 2002.
27. Lance Spitzner. Honeytokens: The other honeypot, Jul 2003. `http://www.symantec.com/connect/articles/honeytokens-other-honeypot`.

28. Clifford Stoll. Stalking the wily hacker. *Communications of the ACM*, 31(5):484–497, 1988.
29. The Wall Street Journal. Google Discloses Privacy Glitch, 2009 Mar. `http://blogs.wsj.com/digits/2009/03/08/1214/`.
30. Trusted Computing Group. TPM Main Specification. `http://www.trustedcomputinggroup.org/resources/tpm_main_specification`.
31. Rui Wang, Shuo Chen, and XiaoFeng Wang. Signing me onto your accounts through facebook and google: A traffic-guided security study of commercially deployed single-sign-on web services. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 365–379, Washington, DC, USA, 2012. IEEE Computer Society.
32. Gilbert Wondracek, Paolo Milani Comparetti, Christopher Kruegel, and Engin Kirda. Automatic network protocol analysis. In *Proceedings of the 15th Network and Distributed System Security Symposium (NDSS)*, 2008.
33. J. Yuill, M. Zappe, D. Denning, and F. Feer. Honeyfiles: Deceptive files for intrusion detection. In *Proceedings of the 5th IEEE Workshop on Information Assurance*, pages 116–122, Jun 2004.
34. Jim Yuill, Dorothy Denning, and Fred Feer. Using deception to hide things from hackers: Processes, principles, and techniques. *Journal of Information Warfare*, 5(3):26–40, 2006.