Poster: Angry birding: evaluating application exceptions as attack canaries

Tolga Ünlü
Lynsay A. Shepherd
Natalie Coull
Colin McLean

Ünlü, T., Shepherd, L. A., Coull, N., & McLean, C. (2021) 'Poster: Angry birding: evaluating application exceptions as attack canaries'. In: L. O'Conner (Ed.), *2021 IEEE European Symposium on Security and Privacy, EuroS&P 2021: virtual conference, 6-10 September 2021: proceedings,* pp.701-703. Institute of Electrical and Electronics Engineers Inc. DOI: https://doi.org/10.1109/EuroSP51992.2021.00052

Poster: Angry Birding: Evaluating Application Exceptions as Attack Canaries

Tolga Ünlü, Lynsay A. Shepherd, Natalie Coull, and Colin McLean
Division of Cyber Security
Abertay University
Dundee, United Kingdom
Email: {t.unlu1200,lynsay.shepherd,n.coull,c.mclean}@abertay.ac.uk

***Abstract*—Application exceptions are anomalous events occurring within the application. These can be caused by common issues such as simple programming errors; however, they can also originate from the side-effects of a trial-and-error process used in active attacks. Utilising attacker-induced exceptions as a canary for intrusion detection has been demonstrated as a feasible technique for SQL injection detection, but this has not been applied to other types of attacks. This paper proposes an approach to consider attacker-induced application exceptions as attack canaries. The work is part of an ongoing investigation on integrating detective defences into applications through established development practices.**

***Index Terms*—Honeytoken, Canary, Intrusion Detection, Exception Monitoring, Developer-Centred Security**

I. Introduction

Monitoring exceptions is an approach to detect whether the normal flow of an application has been disrupted by events such as database or business logic errors. While exceptions can be critical in determining the root cause of an error, they are also a valuable source of information to prepare attacks, if they are exposed to malicious actors. For example, the payload of an SQL injection (SQLi) attack is prepared and adapted by utilising database error messages. This trial-and-error process applied against the injection payload is also required for blind SQLi attacks, where a successful or failed execution of a payload is inferred from side-channels based on execution time or returned content. A trial-and-error process is an essential stage of executing a successful attack in general, as this provides attackers with the opportunity to observe potentially exploitable vulnerabilities in a target application.
Application developers can also take advantage of this exploratory behaviour by monitoring attacker-induced exceptions from within the application. These attacker-induced exceptions are similar in functionality to a canary[1] and can, combined with other high-fidelity attack indicators, can make applications attack-aware [12]. Previous research has demonstrated the feasibility and effectiveness of the canary concept with the use of honeytokens for web intrusion detection [10] [6] [5]. Honeytokens are a deception-based detection method for malicious activities by luring attackers into interacting with fictitious but attractive artefacts such as seemingly sensitive parameter names. In contrast, the research proposed in this paper seeks to complement existing deception-based canary research via an evaluation of attacker-induced exceptions generated by common data sinks in web applications. The concept of using attack canaries based on exceptions is a developer-centred attempt at making the integration of intrusion detectors usable. Additionally, the proposed evaluation aims to explore whether exception-based attack canaries can emerge as a generic defence mechanism for web applications.

[1]A concept originating from coal miners using canary birds, less tolerant to toxic gases than humans, as a means of monitoring the air quality.

II. Problem Statement

The evaluation is part of a larger research project on usable attack-awareness integration methods for web applications. Prior research on existing integration methods has revealed that current developer-driven and agent-driven methods often fail to augment techniques already used by developers but require additional manual effort [11]. Furthermore, a survey conducted by Braun et al. [2] has shown that the most prevalent web application frameworks at the time of the analysis do not provide defence mechanisms similar to Control-Flow Integrity (CFI) techniques[2] for native software applications. A preliminary review of PHP-based web application frameworks indicates that this has not changed as of the time of writing, including mechanisms that utilize exceptions. For web frameworks in general, the most similar mechanism to the proposed attacker-induced exceptions appears to be the `SuspiciousOperation` exception classes of the Django web application framework [4]. However, these represent a limited set of security issues rather than actual attacks and require manual assignment by a developer. The OWASP AppSensor project [12], while considering actual attacks and also providing a reference implementation, is primarily a prescriptive guide for developers on how to instrument web applications with detection points.

[2]CFI techniques prevent a program's control flow from being hijacked by monitoring deviations of the executed control flow from a pre-computed representation of a program's control flow.

III. Motivation

Attacker-induced exceptions could provide a similar benefit in the case of intrusion detection as stack canaries do for buffer overflow attacks against native software applications [3]. For example, OpenRASP, an open-source Runtime Application Self-Protection (RASP) solution, implements SQLi detection through exception monitoring and demonstrates the feasibility of this approach for one type of attack [1]. Other types of attacks could also be detected with this approach, in particular those which also utilise error states or exceptions of the target application, such as Server-Side Request Forgery (SSRF).
From an integration perspective, extension-level instrumentation as performed by OpenRASP is not necessary since existing exception handling mechanisms at the application-level can implement the same functionality in a more lightweight manner, without increasing the attack surface. Improving exception-based monitoring of application issues can also be more appealing to developers, as intrusion detection can be aligned with the quality assurance efforts of development teams. The augmentation of existing development practices and tooling is also an opportunity to improve the experience of developers, alongside the usability of the tools and practices [9].

IV. Proposed Approach

The following section outlines the proposed evaluation, which consists of a monitoring setup to record attacker-induced exceptions, and an analysis of the recorded exceptions. Section II discussed a preliminary review on PHP-based web application frameworks, which will also be the language used in the proposed evaluation due to its wide-spread adoption on the web - 79.2% of the websites recognized in the daily updated report on server-side programming languages by W^{3}Techs [13].

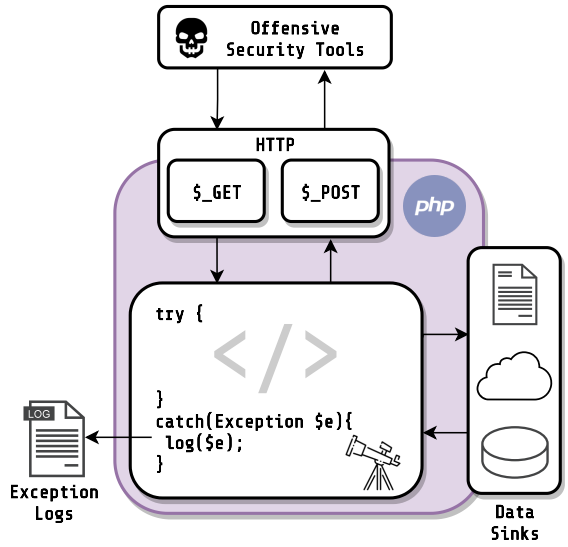

Fig. 1: Overview of the attacker-induced exception monitoring setup

A. Monitoring Setup

The setup, as seen in Figure 1, primarily consists of an instrumented PHP script with a selected set of framework components making use of data sinks from the latest PHP environment. These components will receive input from PHP's default API for HTTP requests, enabling the use of existing offensive web security tools for automated exception generation and the monitoring process. Exceptions generated and monitored during an ongoing attack will be logged for further analysis.

B. Detection Model

The type of attacks, or more specifically the attacker behaviour that the proposed approach aims to detect focuses on those that attempt to exploit taint-style vulnerabilities in the target application. A taint-style vulnerability emerges when a sensitive data sink, e.g. an API to insert data into a database, can receive tainted data from an untrustworthy data source such as an API to read incoming HTTP requests, where all parts of the request are in control of the user.

C. Component and Sink Selection

It is important that the selection criteria for the data sinks reflect the variety of possible destinations where data can flow within web applications in general. Regarding the instrumented PHP script in the monitoring setup, the sinks will cover the common built-in PHP APIs for file system, web and database interaction. The sinks determine the framework components which will make use of these. Figure 2 shows an example with cURL which is used in PHP web applications for HTTP communication. cURL is used by libraries like Guzzle which also provides the HTTP client functionality for web applications built with the Laravel framework [7]. The Guzzle HTTP client offers a set of exceptions which can be expected to be generated during an ongoing attack [8].

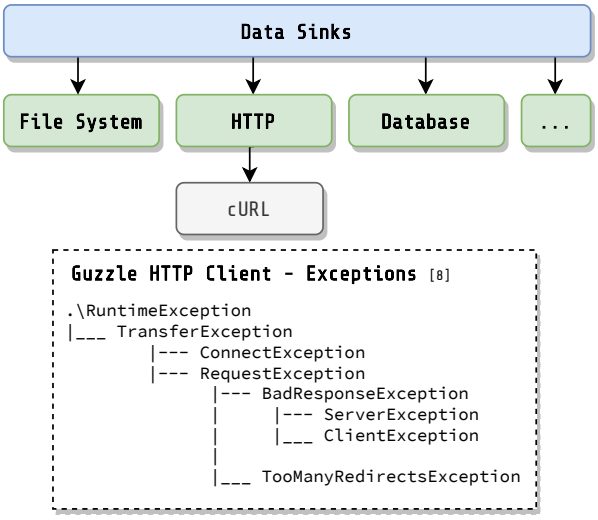

Fig. 2: Selection of cURL as a HTTP sink in PHP which is used by HTTP client libraries such as Guzzle.

D. Exception Analysis

The objective of the analysis is to develop a mapping methodology with which attacker-induced exceptions can be assigned to specific types of attacks and possibly the actual attack payload. An initial mapping methodology could be based on a combination of specific exceptions that occur in an active attack, considering other factors that are typical in a trial-and-error process such as the repeated generation of the same combination of exceptions.

E. Evaluation

The following criteria are selected to evaluate the proposed approach to address the objectives described in section II. This is a work in progress, thus the evaluation criteria may be subject to change:

- **Scalability:** While exceptions are an established concept in many programming languages, there are differences in implementation. This can impact the creation of attack canaries, requiring further analysis on the effort of adopting exception-based canaries for one programming language or framework component into another.

- **Accuracy:** An exception-based attack canary is, by design, not a preventive measure. The candidate exceptions or the combination of exceptions selected as canaries must be reliable indicators of attacks, such that the application can respond with defensive measures when these canaries are triggered.

- **Reproducibility:** In contrast to intrusion detection systems operating at the network-level or host-level, attack canaries embedded within an application can benefit from the surrounding application context. The context can become relevant to provide the appropriate information, e.g. the state that the application was in at the time of detection. This helps with attack reproducibility and supports developers in quickly remediating the associated vulnerabilities.

- **Effectivity:** The evaluation must also consider the limitations of the approach, and its effectiveness under certain conditions, with respect to the previously described evaluation criteria. This includes scenarios where an attacker might be able to skip the trial-and-error process against the actual target and craft an exploit in a lab environment instead. Such a scenario is not unlikely given that many web applications are based on web application frameworks and libraries that are open-source and thus available to attackers for security research.

V. Outlook

The proposed approach can provide insights on how established development practices may be underutilised for intrusion detection purposes. The motivation behind this approach and the attack-awareness research it is based on is to understand what attack indicators can be detected within a web application and whether it is feasible to infer generic detection patterns from these indicators. In turn, this could result in reusable detector components for web applications and frameworks. The proposed approach has been primarily presented as a defence mechanism applied at application runtime, however, it is not limited to this scope. Existing unit and integration testing modules in web application frameworks could also be augmented with a security testing module that can be used in automated tests to verify the functionality of detector components. Future research seeks to consider whether this form of tests might be a more usable and developer-centric alternative to security testing with offensive security tools.

References

[1] Baidu Security Lab. OpenRASP v1.0 official version released — Database exception monitoring and WebLogic support come as scheduled, 2019. https://mp.weixin.qq.com/s/JyR6LIhYau7aARH1v4Grbw.

[2] Bastian Braun, Christian v. Pollak, and Joachim Posegga. A Survey on Control-Flow Integrity Means in Web Application Frameworks. In *Nordic Conference on Secure IT Systems*, pages 231–246. Springer, 2013.

[3] Crispan Cowan, Calton Pu, Dave Maier, Jonathan Walpole, Peat Bakke, Steve Beattie, Aaron Grier, Perry Wagle, Qian Zhang, and Heather Hinton. StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks. In *USENIX security symposium*, volume 98, pages 63–78. San Antonio, TX, 1998.

[4] Django Software Foundation. Django Exceptions — Django documentation — Django, 2021. https://docs.djangoproject.com/en/3.2/ref/exceptions/#django.core.exceptions.SuspiciousOperation.

[5] Daniel Fraunholz, Daniel Reti, Simon Duque Anton, and Hans Dieter Schotten. Cloxy: A Context-aware Deception-as-a-Service Reverse Proxy for Web Services. In *Proceedings of the 5th ACM Workshop on Moving Target Defense*, pages 40–47, 2018.

[6] Xiao Han, Nizar Kheir, and Davide Balzarotti. Evaluation of Deception-Based Web Attacks Detection. In *Proceedings of the 2017 Workshop on Moving Target Defense*, pages 65–73, 2017.

[7] Laravel LLC. HTTP Client - Laravel - The PHP Framework For Web Artisans, 2021. https://laravel.com/docs/8.x/http-client.

[8] Michael Dowling. Quickstart — Guzzle Documentation, 2015. https://docs.guzzlephp.org/en/stable/quickstart.html#exceptions.

[9] Brad A. Myers, Andrew J. Ko, Thomas D. LaToza, and YoungSeok Yoon. Programmers Are Users Too: Human-Centered Methods for Improving Programming Tools. *Computer*, 49(7):44–52, 2016.

[10] Merve Sahin, Cedric Hebert, and Anderson Oliveira. Lessons Learned from SunDEW: A Self Defense Environment for Web Applications. In *Proceedings of the 2020 Measurements, Attacks, and Defenses for the Web (MADWeb) Workshop in the Network and Distributed System Security Symposium (NDSS)*, 2020.

[11] Tolga Ünlü, Lynsay A. Shepherd, Natalie Coull, and Colin McLean. A taxonomy of approaches for integrating attack awareness in applications. In *2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, pages 1–4. IEEE, 2020.

[12] Colin Watson, Michael Coates, John Melton, and Dennis Groves. Creating Attack-Aware Software Applications with Real-Time Defenses. *CrossTalk The Journal of Defense Software Engineering*, 24(5), 2011.

[13] W^{3}Techs. Usage Statistics and Market Share of Server-side Programming Languages for Websites, May 2021, 2021. https://w3techs.com/technologies/overview/programming_language.