# An API Honeypot for DDoS and XSS Analysis

G Leaden, Marcus Zimmermann, Casimer DeCusatis, *Fellow, IEEE*, and Alan G. Labouseur
Marist College
School of Computer Science and Mathematics
Poughkeepsie, NY 12601
{G.Leaden1, Marcus.Zimmermann1, Casimer.DeCusatis, Alan.Labouseur}@Marist.edu

***Abstract*—Honeypots are servers or systems built to mimic critical parts of a network, distracting attackers while logging their information to develop attack profiles. This paper discusses the design and implementation of a honeypot disguised as a REpresentational State Transfer (REST) Application Programming Interface (API). We discuss the motivation for this work, design features of the honeypot, and experimental performance results under various traffic conditions. We also present analyses of both a distributed denial of service (DDoS) attack and a cross-site scripting (XSS) malware insertion attempt against this honeypot.**

## I. Introduction

The number and severity of cyber attacks has grown significantly in recent years [1], [2]. Cyber attackers have managed to pull off virtual bank heists, distributed denial of service (DDoS) attacks powered by botnets and Internet of Things (IoT) devices, and power outages caused by malware [2]. Our National Science Foundation (NSF)–sponsored *SecureCloud* test environment aims to combat the growing number of cyber attacks against cloud networks using autonomic, zero trust techniques [3].

Our recent *SecureCloud* test environment implementation utilizes new software developed for testing cybersecurity techniques on cloud networks. Part of this system uses G-star, the Dynamic Graph Database [4] to organize, visualize, and analyze cyber attack data. G-star, interoperating with the PostgreSQL object relational database through G-star Studio [5], allows us to perform graph theoretical analysis and relational queries on cyber-attack data.

Key contributions of this work include the following:

- introduces the idea of an API honeypot
- describes our implementation of an API honeypot
- demonstrates DDoS and malware API attack analysis
- discusses performance characteristics of our honeypot
- shows how we enable security experts to analyze data and develop remedies for emerging API attacks

The remainder of this paper is organized as follows: Section II introduces the idea of an API honeypot. Section III describes the software design and features of our API honeypot, Pasithea. Section IV presents an analysis of data received from both the initial G-star REST API logs and current data collected by Pasithea. Section V explains results from Pasithea performance testing. Finally, Section VI ends this paper with a discussion of our initial conclusions and plans for future work.

## II. Background

Not long after making G-Star Studio available online, we observed a number of unauthorized connection attempts to its Application Programming Interface (API). These attacks specifically targeted G-star Studio's REpresentational State Transfer (REST) API.

A REST API is among the most commonly used API architectures today [6]. There are many examples of recent attacks against REST APIs, including well-publicized attacks on the Nissan Leaf smart car [7] and the U.S. Internal Revenue Service database. Existing APIs do not always follow security best practices, and developers might be lulled into a false sense of security by believing that their API will not be an attack target. In an effort to improve the security of our *SecureCloud* system and other REST APIs, we felt it would be advantageous to create a profile of these attacks, so we developed a low-interaction API honeypot.

Honeypots are servers or systems that mimic critical parts of a network, effectively distracting attackers and logging attack information in the process [8]. Disguising a honeypot as an API allows us to analyze and understand API attack patterns. Since our defensive responses improve as we collect more data, we can effectively use the attackers' strengths against them; the more our honeypot is attacked, the better our defense posture can become.

Because our original REST API was not designed to comprehensively log attack data, we only captured timestamp, command type, and command text from the initial attacks. Addressing these shortcomings, we created a new API honeypot, named Pasithea (the Greek goddess of rest), capable of creating an attack profile that includes the user agent, the IP address, and other information extracted from the G-star API. Data gathered by Pasithea helps us determine where attacks are coming from, how they can be classified, and what can be done to defend against such attacks. Pasithea has been deployed online and the data it collects (as well as data from other sources) have been integrated into our NSF *SecureCloud* test environment.[1]

[1]It is important to note a potential source of confusion regarding the terms "API" and " honeypot". Honeypot APIs, though similar in name, are not to be confused with Pasithea, which is an API honeypot. A honeypot API is an API made to interface with a specific honeypot (like an SSH honeypot or an SDN honeypot), returning useful data or executing certain methods specified by the request [9]. An API honeypot (such as Pasithea) on the other hand, is entirely different; it functions as a proper honeypot itself by gathering data on unauthorized API requests. This type of API honeypot represents a novel approach to attack analysis that will hopefully lead to the development of more secure and robust APIs for cloud-based applications.

Our honeypot enables security experts to analyze attacks and develop remedies for emerging threats, to avoid falling into framework-complacency. While various types of honeypots have existed for some time [10], [11], and many security projects use APIs to make their data more easily consumable, the use of a REST API *itself* to attract malicious traffic and collect attack data does not appear to have been previously studied.

## III. Construction Principles

We developed Pasithea using Java, a common server-side programming language, and NanoHTTPD [12], a lightweight HTTP library written in Java that receives HTTP requests and returns responses. Implementing this kind of functionality enables Pasithea to simulate a real application server in a lightweight and independent manner. It accepts any kind of request, regardless of the HTTP method, URI requested, or request body. Pasithea then logs the current time, the HTTP method, the path the client attempted to access (e.g. /index.html), the client's IP address, and the user agent data. Clients always receive a

```
<h1>404 Not Found</h1>
```

response, regardless of which resource they attempt to access.

In order to ensure attackers do not fingerprint Pasithea as a honeypot, we modeled our API honeypot after G-star Studio's "real" API.[2] But Pasithea always returns a 404 error, while G-star Studio, when prompted with a valid request, will return JSON-formatted data. The consistent 404 response is what makes Pasithea an unidentifiable, low-interaction honeypot. It is indistinguishable from a normal HTTP server whose valid URIs attackers do not know.

We are hosting our honeypot on an Amazon Web Services Elastic Compute Cloud (AWS EC2) instance using its "free micro" tier. We chose AWS both because of its appealing free tier model and because we were familiar with the security policies and standards that Amazon sets in place. We modified those default security policies within our AWS instance to enable access to the port hosting our API honeypot. Pasithea is currently indexed on Shodan [13], a web search engine that indexes internetconnected devices. Shodan is known for being frequented by the hacker community, making it likely that we will be able to collect additional attack data.

## IV. Attack Analysis

Pasithea is intended to help investigate multiple types of attacks. In particular, we observed and analyzed a distributed denial of service (DDoS) flood and the attempted use of cross-site scripting (XSS) commands. In a DDoS attack, an attacker floods a network or service with information or requests in an attempt to exhaust some finite system resource such as memory [14]. The goal of a DDoS attack varies, but it is most commonly intended to disrupt legitimate users from accessing information or services provided by the network.

The DDoS attack on G-star lasted from May 25, 2017 until June 1, 2017, creating over 275,000,000 log entries. During this time, the requests per second ($R/s$) steadily rose, starting at $500R/s$ and peaking at over $6000R/s$. As an unintended side effect, the log file grew to over 18 GB in size until G-star ran out of storage on its 20 GB cloud-hosted server. The requests received during the DDoS attack all contained the command type $HEAD$ and the command text "home".

We gathered a simple random sample of 150 requests collected by the G-star logs between February 6, 2017 and May 25, 2017 (a period covering normal operations and the attack). This data was then rendered into a Hive plot [15] to interpret the attack. Hive plots are a perceptually uniform, scalable visualization for network analytics. We used a four-axis graph to reflect the relationships among timestamp, data source, command type, and response message (Fig. 1). The timestamp axis is the most heavily populated, containing distinct plotted points for each second during which a log entry was created. The source and message axes are closely related because there is only one plotted point on each axis. Source is the source of the response message returned by G-Star, and message is the response itself. In the context of the log file we analyzed, the source is always "back-end" (meaning the web server in this case) and message is always "Unknown command:". Lastly, the plotted points on the command type axis delineate unique HTTP request methods such as GET, POST, HEAD, PUT, DELETE, etc.

Fig. 1 displays all 150 requests, while Fig. 2 highlights the outlying requests that did not use the command type GET. Rather, these requests used the command types POST and HEAD. The POST requests are shown as the plotted point second closest to the center of the axis, while the HEAD requests are the farthest plotted point from the center of the axis. Fig. 2 represents requests that used methods not commonly employed by web browsers or web crawlers, two of the most frequent sources of unwanted requests found in data gathered by Pasithea. This suggests these requests were deliberate and potentially malicious in nature.

Cross-site scripting attacks aim to inject malicious scripts

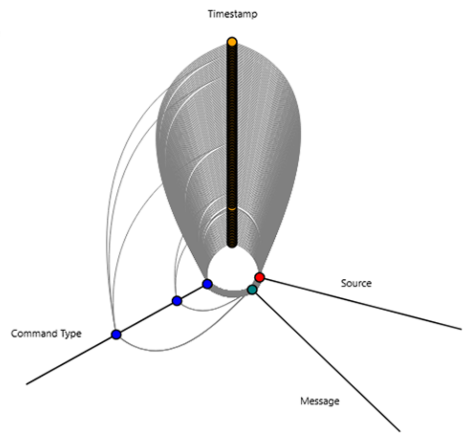

Fig. 1. Hive plot displaying a random sample of 150 points from G-star logs. Data sampled is from February 6, 2017 – May 25, 2017.

[2]Please refer to the *ACM Inroads* article on G-star Studio [5] for technical details about developing a Java-based REST API.

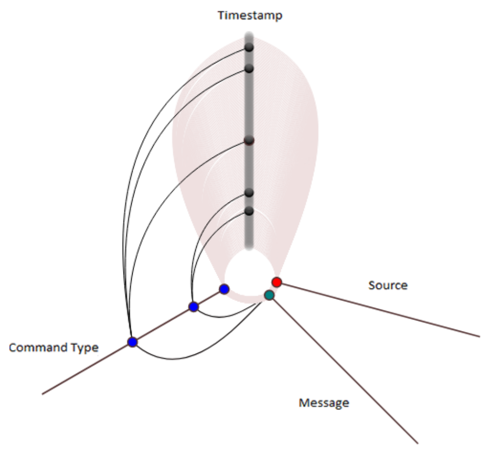

Fig. 2. Prominent nodes denote "injections" into G-star that differ from normal traffic. Data sampled is from February 6, 2017 – May 25, 2017

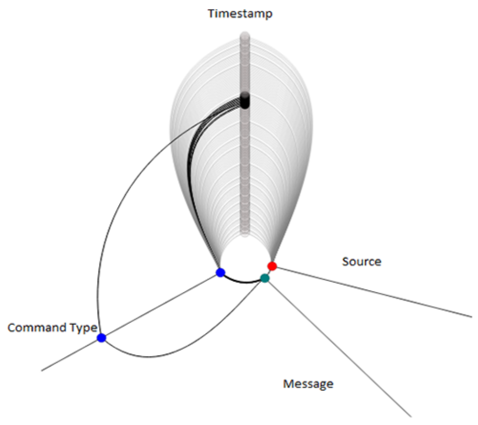

Fig. 3. Prominent nodes display the XSS attempt on G-star. Data sampled is from March 11, 2017 – March 16, 2017

into an otherwise benign or trusted website [16]. Further investigation of the specific commands being attempted as an XSS attack revealed the following:

```
GET cgi
POST command.php
GET ;rm$IFS-f$IFS'
GET ;wget$IFS-O$IFS'
GET ;chmod$IFS'777'$IFS'
GET ;sh$IFS-c$IFS'
```

Fig. 3 displays a selected sample of entries collected by the G-star logs between March 11, 2017 and March 16, 2017. The highlighted requests have a span of 25 seconds where the XSS commands were attempted. This shows both the short amount of time it took to run this malware insertion attempt and the use of GET and POST request methods for one attack.

We determined that a very similar attack sequence was also recently documented by a Finnish cyber-security company F-Secure [17]. The commands attempted with the XSS attack are intended to upload a PHP: Hypertext Preprocessor (PHP) file, then send a series of commands that, if executed, would remove a file using the $IFS variable found in PHP. Afterwards, it would attempt to download a file with the same variable name and change the permissions on said file so that it can be executed by any user. Then the attacker would execute the file. Researchers from F-Secure attribute this attack profile to a Peer to Peer (P2P) botnet named TheMoon [18]. This example illustrates how we can use data collected from attack attempts to isolate and attribute the attack, provided that we can determine what types of attacks are taking place.

Pasithea was subsequently deployed and is currently active on our AWS EC2 instance in Ashburn, Virginia. Our log files indicate cursory web crawls from Baidu, a Chinese search engine, and some attempts at exploiting a known vulnerability in Apache Tomcat web servers using "GET /manager/html" [19]. We continue to monitor this instance and additional results will be reported in future papers.

## V. Performance Tests

Performance testing is a critical part of development, so it is important to demonstrate Pasithea's performance under different loads and its ability to log many – potentially thousands – of incoming requests in a short amount of time. In other words, it must respond fast enough to keep malicious users interested while also being stable enough to receive high volumes of incoming requests. To test this, we ran a series of benchmarks using the Apache Bench (ab) tool [20]. This tool allows us to designate a number of completed requests to be sent to our API honeypot while varying the number of simulated concurrent users. The results from these tests are displayed in Fig. 4 and Fig. 5.

We researched a baseline response time for a RESTful API to give this data appropriate context. In doing so, we discovered two separate internal tests from software development and web monitoring companies, 3PillarGlobal [21] and Site24x7 [22]. Paired with some research on the human perception of performance [23], we concluded that a 300-ms response time is expected under normal traffic conditions in order for the API honeypot to appear realistic. Data collected on Pasithea indicates that we fall well within this range given a concurrency level of 500 simultaneous users. In addition, we continued tests at much higher concurrency levels to assess

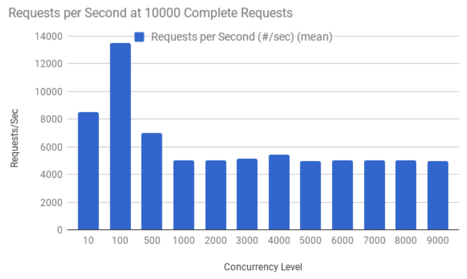

Fig. 4. Requests processed per second at varying concurrency levels

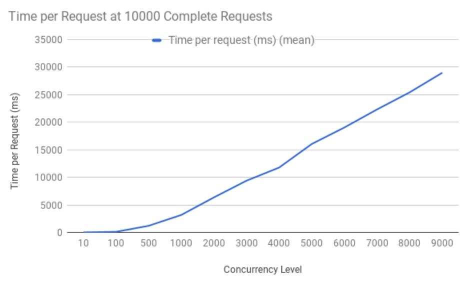

Fig. 5. Mean time to complete a single request at varying concurrency levels

how well Pasithea would perform under extreme stress, like the attempts we saw on the G-star API. Pasithea can, with time, handle a concurrency level of over 9000 simultaneous requests while still logging more than 90% of the requests received.

Since our implementation of request/response for Pasithea was deliberately kept very simple (only responding with 404 errors), we have thus far been unsuccessful in driving Pasithea hard enough during performance testing to reach a point where it is unable to handle a significant amount of requests. Pasithea hits a $R/s$ plateau at a concurrency level of 1000 (see Fig. 4), but continues to perform well at 9000. With enough storage space, we believe that Pasithea could withstand a substantial attack, such as the one seen on G-star, and be able to log information about the attack for analysis.

## VI. Conclusions and Future Work

Today's API security landscape is more like a "wild west" of conflicting standards than a safe, civilized, city of consistency. This has led to an influx of attacks directed at APIs on many fronts. Based on the attacks targeting G-star and our research into related attacks, we have constructed an API honeypot, Pasithea, with Java and NanoHTTPD to help combat and detail future attacks on the API landscape. Using hive plots and other graph and relational tools, we have analyzed a real-world attack on G-star, demonstrating how DDoS and XSS attacks can be uncovered and attributed so that an appropriate defense may be deployed. Our performance data suggests that Pasithea should be able to keep a malicious user interested with fast response times while also maintaining composure and stability under high traffic loads. This allows us to develop accurate API attack profiles that will help shape the future of API security.

Our next steps include extending Pasithea's REST interface to extract more data from attackers while still maintaining its cover as an unidentifiable API honeypot, reporting additional results as Pasithea spends more time in the "wild west" of the Internet collecting more data, and exploring higher interaction versions of an API honeypot where we would be able to respond with artifical data.

## Acknowledgments

The authors would like to thank the IBM/Marist Joint Study as well as Dayna Eidle, Thomas Famularo, Jonathan Heiles, Mary Ann Hoffmann, and Thomas Magnusson for their contributions to this paper. It is better because of them.

## References

[1] Symantec, "Internet Security Threat Report (ISTR)," Symantec Corporation, Tech. Rep. 22, 2017.

[2] IBM Security, "IBM X-Force Threat Intelligence Index 2017," International Buisness Machines, Tech. Rep., 2017.

[3] C. DeCusatis, P. Liengtiraphan, A. Sager, and M. Pinelli, "Implementing zero trust cloud networks with transport access control and first packet authentication," in *2016 IEEE International Conference on Smart Cloud (SmartCloud)*, Nov 2016, pp. 5–10.

[4] A. G. Labouseur, J. Birnbaum, P. W. Olsen, S. R. Spillane, J. Vijayan, J. Hwang, and W. Han, "The g* graph database: efficiently managing large distributed dynamic graphs," *Distributed and Parallel Databases*, vol. 33, no. 4, pp. 479–514, 2015.

[5] A. Labouseur, "G* studio: an adventure in graph databases, distributed systems, and software development," *Inroads*, vol. 7, no. 2, pp. 58–66, 2016. [Online]. Available: http://dl.acm.org/citation.cfm?id=2896823

[6] Daniel R. Cogan, ""REpresentational State Transfer in the Modern Internet"," http://scholarship.claremont.edu/cmc_theses/1387, 2016, cMC Senior Theses. 1387.

[7] Troy Hunt, "Controlling vehicle features of Nissan LEAFs across the globe via vulnerable APIs," https://www.troyhunt.com/controlling-vehicle-features-of-nissan/, Feb 2016, online, Accessed 7/15/2017.

[8] William W. Martin, "Honey Pots and Honey Nets - Security through Deception," May 2001.

[9] "Http:BL API Specification," online, Accessed 7/25/2017.

[10] C. Stoll, *The Cuckoo's Egg: Tracking a Spy Through the Maze of Computer Espionage*. New York, NY, USA: Doubleday, 1989.

[11] N. Provos, "A virtual honeypot framework," in *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, ser. SSYM'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 1–1. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251375.1251376

[12] "NanoHttpd," https://github.com/NanoHttpd/nanohttpd, 2017, online, Accessed 7/15/2017.

[13] Lawrence Fernandes, "Shodan: The Hackers Search Engine," https://www.cybrary.it/0p3n/intro-shodan-search-engine-hackers/, Mar 2016, online, Accessed 7/19/2017.

[14] "Security Tip (ST04-015) Understanding Denial-of-Service Attacks," https://www.us-cert.gov/ncas/tips/ST04-015, 2013, online, Accessed 7/15/2017.

[15] M. Krzywinski, I. Birol, S. J. JM, and M. A. Marra, "Hive plots, a rational approach to visualizing networks," *Briefings in Bioinformatics*, vol. 13, no. 5, pp. 627–644, 2012. [Online]. Available: +http://dx.doi.org/10.1093/bib/bbr069

[16] "Cross-site Scripting (XSS)," 2016, online, Accessed 7/15/2017.

[17] Mikko Hypponen. and Tomi Tuominen., "F-Secure State of Cyber Security," http://branden.biz/wp-content/uploads/2017/02/cyber-security-report-2017.pdf, Feb 2017, online, Accessed 7/15/2017.

[18] Bing Liu, "TheMoon - A P2P botnet targeting Home Routers," https://blog.fortinet.com/2016/10/20/themoon-a-p2p-botnet-targeting-home-routers, Oct 2016, online, Accessed 7/19/2017.

[19] Tony Lee, "Manually Exploiting Tomcat Manager," http://blog.opensecurityresearch.com/2012/09/manually-exploiting-tomcat-manager.html, Sep 2012, online, Accessed 7/19/2017.

[20] "ab - Apache HTTP server benchmarking tool," https://httpd.apache.org/docs/2.4/programs/ab.html, online, Accessed 7/15/2017.

[21] Singh Sukhwinder, "Performance Testing of a REstful API using JMeter," https://www.3pillarglobal.com/insights/performance-testing-of-a-restful-api-using-jmeter, online, Accessed 7/12/2017.

[22] "Performance Metrics of Rest API Monitor," https://www.site24x7.com/help/performance-metrics/rest-api.html, Jul 2015, online, Accessed 7/12/2017.

[23] Denys Mishunov, "Why Perceived Performance Matters, Part 1: The Perception Of Time," *Smashing Magazine*, Sep 2015, online, Accessed 7/12/2017.