

The Canary In The Coal Mine:

Implementing Web Based Canaries and other HoneyTokens

Pavel S Frolikov
Frolikov@ucdavis.edu
University of California Davis
Davis, California

Matthew Scates
msscates@ucdavis.edu
University of California Davis
Davis, California

Michael Yang
mzyang@ucdavis.edu
University of California Davis
Davis, California

ABSTRACT

The growing number and significance of web applications have led to the growth in attacks against networks, which imposes great vulnerability to information security. Therefore, more aggressive forms of defense techniques to supplement the existing security approaches have been brought to the table. One of these techniques, the Honey Token, involves the use of deception to collect information about attacks. A Honey Token is a security deception technique that leaves data intentionally to be probed, attacked, or compromised by attackers to collect information from the source of attackers.

KEYWORDS

HoneyPot, HoneyToken, Web-Canaries, Computer Security, Security Design

1 INTRODUCTION

The UC Davis website often has hackers trying to break into the site for various reasons, and often we don't know who these hackers are. As a result, we are going to help identify those hackers by implementing an array of different Honey Tokens. Honey Tokens are the fictitious records used to deceive attackers. In this project, we implement multiple different approaches to Honey Tokens designed to lure hackers. Table 1, contains the types of Honey Tokens implemented in this project, and a brief description.

1.1 Why Honey Tokens?

Honey Tokens are ideal for all because,

- (1) easy to implement
- (2) require little resources
- (3) require little maintenance
- (4) provide an accurate data set

Thanks to the simplicity, accuracy, and strength of Honey Tokens, they are especially useful for smaller organizations that do not have many resources to delegate to web security. As seen in Figures 1 and 2 we used two web pages from the UC Davis site, making the visual implementation simple, cheap (since it was already done), and quick. After we added a few Honey Tokens we are now able to get very accurate data points from people of interest (attackers trying to break into the system). Honey Tokens are also good at catching hackers without risk to the main system because they are separate from the main system, yet look like they are part of the main system. Thus any "damage" to the Honey Token will not affect the main system.

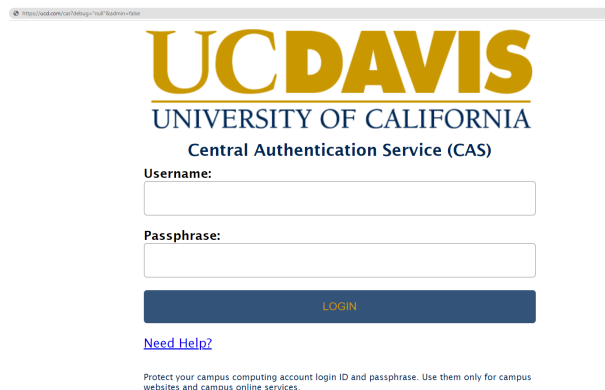


Figure 1: Deceptive CAS Login Screen

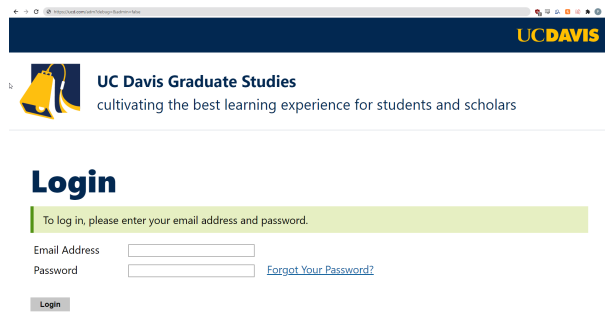


Figure 2: Deceptive Graduate Admissions Login Screen

2 BACKGROUND

Prior to this project, none of us have heard of Honey Tokens, so before we could begin our development we needed to understand why they're useful, how one might go about implementing them, and just answer general questions about the topic.

2.1 Prior Research

To start, we as a group read a handful of academic papers and professional blogs on the different implementations of Honey Tokens. There were a few lectures on Honey Tokens, the most useful lecture came from MMY a Russian university, during the lecture the speaker clearly explained the benefits of why Honey Tokens are a powerful tool that should not be overlooked. as shown in table 2.

"The hypothesis that honey technologies are not widely deployed in production systems turned out to be correct. Of those surveyed, only 25% had honeypots, 9% had honeynets, and 15% had

Table 1: Honey Tokens Implemented

	Description	
Canary-based Parameters	A canary-based approach to a deceptive web page designed to lure hackers into attacking what is known as a deceptive Honey Token	The objective of this Honey Token is to trick hackers into thinking UC Davis has a vulnerable site. by putting web attributes such as debug and admin in the URL publicly rather than secretly.
Robots.txt	Adding deceptive links to the Robots.txt file, to check for malicious web crawlers.	Our Honey Tokens redirected to a real web page, so the attacker would not know that they were being tracked.
Storing on the file system	Storing access tokens on files in the web server's file system	If the access token gets entered we record the incident.
SQL Injection	To detect hackers trying to exploit an SQL vulnerability, we log users who add in ' OR 1=1 into the password field.	This tricks vulnerable code that uses SQL into accepting 1=1 rather than the password.
Bogus Login Information	Added dummy admin account login information	If login with credentials attempted, the incident gets recorded.

What	How
Defense	Learn from the adversary and Adapt Lay Traps to catch subtle yet abnormal activities
Research	Understand how attackers think what works what doesn't, and what they're after

Table 2: Honey Token use cases[8]

honeytokens.”[10]

This is a really interesting finding from a Ukrainian Cybersecurity study. Despite the fact that Honey Tokens, or even honeypots, have many advantages, such as simplicity, little overhead, and accurate and precise data sets. Security teams seem to not use them at all. Luckily UC Davis already has a small honeypot set up, and it would be quite beneficial to add more simple security measures.

2.2 Zones

Honey Tokens can be implemented on many parts of the system. Each "Zone" represents a different part of the full Website. Placing Honey Tokens in the following Zones allows us to get a better understanding of how adversaries attempt to break into our system. [5]

2.2.1 Zone 1, The Web Server. Honey Tokens on the Web Server involve strategically placing sensitive information on the File System, letting us know where attackers are attempting to gain unauthorized access from. Some examples include

- text documents with access tokens
- Robots.txt

2.2.2 Zone 2, The Web Application. Canary based Honey Tokens (Section 3.2) are a perfect example of a Web Application defense.

2.2.3 Zone 3, The Database Server. Placing Honey Tokens in the Database Server allows us to defend and identify weaknesses in the database that adversaries will attempt to exploit.

2.3 web2py

Web2py is a full-stack python based web framework, where we just need the basic skills to do our development. Our web pages are ripped straight out of the UC Davis web-page: the CAS login screen, and graduate admissions login portals. We decided on these pages as they would most likely be places of interest for attackers.

2.4 DETERLab

DETERLab is a security and education-enhanced version of Emulab, a shared testbed providing a platform for research in cyber-security and serving a broad user community, including academia, industry, and government. [9] Before we hosted our project on PythonAnywhere and the Security office's servers, we used DETERLab to test our early prototypes and ideas for the project.

2.5 PythonAnywhere

After we did our initial testing on DETERLab, we moved onto PythonAnywhere for the remainder of our development and deployment, which is a web-host designed for web2py (and other server-based python interfaces such as bottle and py4web), this streamlined our development and allowed for rapid (re)deployment.

2.6 Campus Information Security Office

This project was actually suggested by the Campus Information Security office, and they helped us along the way. Jeff suggested that we use DETERLab for our initial testing, and he also spun up a Linux server for us to use. Thank you!

2.7 Struggles along the way

During the development of this project, we ran into a number of unexpected hurdles.

2.7.1 py4web. Initially, instead of using web2py, we wanted to use py4web for the web framework. This is a newer evolution of web2py, but it's more feature-rich than we needed and not as stable. When we first set up our environment in Deter, we quickly ran into many issues with py4web and collectively decided that it would be

best for us to pivot to web2py as it has all of the features that we need, and just works.

2.7.2 cURL. When looking into the different approaches that hackers use when looking for vulnerabilities, there was quickly one tool that stood out.

cURL

This is a very powerful command-line utility that allows for transferring data over URLs. When we first implemented our logging utility, we assumed that everyone would be accessing the web server via a browser with a GUI.

We quickly learned that this is not the case. The logging tool that we developed did not work whenever someone tried to access the website via cURL- it would actually crash the entire server. So, we had to account for cases when the user is accessing our website via cURL or any other similar command-line tools.

2.7.3 Apache. After the bulk of our development was complete, and we have already tested our initial builds on DETERLab and PythonAnywhere, the Campus Information Security Office spun up a Linux server for us. One major hurdle that we encountered during the setup of a web server was Apache. We didn't have to deal with it with PythonAnywhere because they have a streamlined, basically, one-click setup to publish a web application. Herewith a bare-bones Linux server we ran into tons of issues, all coming back to Apache not cooperating with us. Luckily, eventually after over 15 hours of playing around with configs, re-installations we got it working.

3 METHODOLOGY AND IMPLEMENTATION

Throughout the course of the project, we have been working in contact with the ISO to help us deploy our deceptive website. Our project contains two main web pages: the main UC Davis CAS login page, and the graduate admissions login page. We picked these login pages because they were simple to duplicate, many people in UC Davis go to these pages, and they would be a high priority for hackers.

3.1 Logging

The entire purpose of these Honey Tokens is so we can log these incidents. Whenever one of our Honey Tokens gets triggered, we record all of the following information into a database table as seen in figure 3

- Incident ID
- Page the incident occurred on, and updated value if applicable
- IP Address
- TimeStamp GMT
- Operating System
- Platform
- Scraper (boolean)
- Tablet (boolean)
- Mobile (boolean)
- Browser

logTable.id	logTable.logfield	logTable.ip	logTable.currTime	logTable.platform	logTable.os	logTable.bot	logTable.mobile	logTable.ts
124	cas, true	47.45.89.201	2021-11-27 01:21:54	Mac OSX 10.15.7	Macintosh	False	False	False
125	cas, tre	185.220.102.248	2021-11-27 01:32:11	Windows10	Windows	False	False	False
127	true	24.5.5.145	2021-11-27 01:52:01	None	None	None	False	False
129	robots.txt	23.236.146.162	2021-11-30 09:06:45	Windows10	Windows	False	False	False
130	CAS/DEBUG/ true	98.238.242.67	2021-11-30 23:06:11	Windows10	Windows	False	False	False
132	cas, true	67.166.147.73	2021-12-01 00:04:45	Mac OSX 10.15.7	Macintosh	False	False	False
134	None	107.20.33.217	2021-12-01 02:54:48	None	None	None	False	False
135	cas, false	67.166.147.73	2021-12-01 04:39:40	Mac OSX 10.15.7	Macintosh	False	False	False
136	robots.txt	98.238.242.67	2021-12-02 00:04:11	Windows10	Windows	False	False	False
139	cas, true	168.150.24.207	2021-12-02 00:15:12	Mac OSX 10.15.7	Macintosh	False	False	False

Figure 3: Example of log Database Table

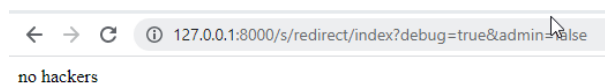


Figure 4: Visual example of the canary being changed, animated here: <https://imgur.com/JOxCOfD>

3.2 Canary-based Parameters

One of our first Honey Token implementations was canary-based. These are parameters that should not ever be changed by the client, so when the values are updated, there is a high likelihood that someone with malicious intent is in our system. If the correct conditions are met we will log the incident. We use enticing names such as debug and admin to lure in hackers. If the parameters were something along the lines of ?lang=en, there wouldn't be much of a reason for adversaries to play around with them. The canary in figure 4 is admin=false. This is a Zone 2 Honey Token.

This parameter based canary is used in coordination with other Honey Tokens we set up in our system (3.4 and 3.6)

3.3 Robots.txt

The Robots.txt file (Zone 1 Honey Token) at the root of websites, is commonly used for indexing websites using web spiders. Often, attackers will try to use the Robots.txt file to their advantage to find places that they should not have access to.

- Disallow: /private
- Disallow: /debug

Our Robots.txt entry has two traps in it. There is no legitimate way to access /private and /debug.

3.3.1 private. When an adversary would go to /private will log the incident, and redirect them to the CAS login screen, to make them believe that nothing happened.

3.3.2 debug. The /debug route will redirect you to an access string. See 3.4

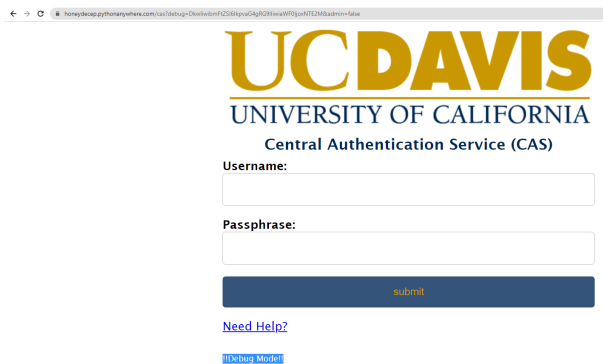


Figure 5: "Debug Mode enabled with Access token"

3.4 Storing on the file system

Earlier in 3.2 we discussed the different parameters that we implemented, ?debug= is one of them. When an adversary enters the access token (discussed in 3.3) into this parameter, they will be in "debug mode" as seen in figure 5. This of course is not actually a debug mode, simply a red herring. (Zone 1)

3.5 SQL Injection

One common approach for adversaries to gain access to systems is by attempting an SQL Injection on vulnerable parts of websites. This Honey Token (Zone 3) was more of a proof of concept, as we only are checking for some of the more common Injection techniques. Such as setting the password field to " OR 1=1". If we detect someone attempting an SQL Injection to gain unauthorized access, we log the incident.

In future implementations we would have liked to flesh out this Honey Token more, as SQL Injections are a staple.

3.6 Bogus Login Information

The last Honey Token (Zone 3) that we implemented in our system is storing dummy admin login information in the /adminmode route. The adversary could get to this route by setting the (Section 3.2) ?admin= parameter to true. This will display an "Admin Mode" button on the Web Page that would redirect the user to an administrative panel.

On that panel is a downloadable CSV file that contains the login information for an admin account. When an adversary downloads this CSV file, the incident would be reported. When an adversary would attempt to log in using the information on the CSV file, this would also be reported.

4 OUTCOMES

Our Honey Tokens are an effective approach to catching and deceiving potential attacks due to the specific criteria it is targeting during the attacking process. The main objective for this project was the implementation and deployment, which we did successfully accomplish.

4.1 Future Work

In the future, we will give our honeypot more deployment time to catch potential attacks and perform data analysis based on the collected attackers' information. For example, we want to compare the Honey Tokens in our honeypot with each other to see which ones get used by attackers more than others. In addition, we will also investigate and implement other Honey Token methods. As said in Section 3.5, we would also like to revisit our SQL Injection Honey Token and implement a more robust solution.

ACKNOWLEDGMENTS

Special thanks to the UC Davis Information Security Office for suggesting a testbed, and spinning up a server for us!

A INSTRUCTIONS FOR RUNNING (LOCALLY MACOS)

- (1) Unzip Web2py.zip
- (2) cd to the web2py root directory
- (3) sudo python web2py.py as seen in figure 6
- (4) Keep Server IP checked at Local IPv4 (127.0.0.1)
- (5) Pick an available Server Port
- (6) Pick an Admin Password, "1" is recommended for simplicity
- (7) You will be redirected to `http://127.0.0.1:PORT/cas`
- (8) To access the logs, go to (If asked to log in use the Admin Password that you selected) `http://127.0.0.1:PORT/s/appadmin/select/db?query=db.logTable.id>0`
- (9) THIS SHOULD ALSO WORK FOR LINUX SYSTEMS. BUT MAC RECOMMENDED

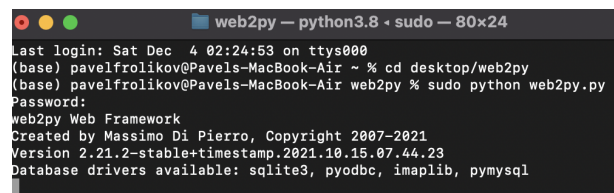


Figure 6: command line

- (1) Important Links
- (2) CAS Page `http://127.0.0.1:PORT/cas`
- (3) Graduate Admissions Login Page `http://127.0.0.1:PORT/adm`
- (4) robots.txt `http://127.0.0.1:PORT/robots.txt`
- (5) where to see logs `http://127.0.0.1:PORT/s/appadmin/select/db?query=db.logTable.id>0`

WINDOWS

- (1) CODE has been tested on MacOS.
- (2) for windows. download the binaries at `http://web2py.com/init/default/download`
- (3) from web2py.zip place the folder /s/ which is located in web2py/applications into the applications (web2py/applications/) folder of the windows binaries directory
- (4) AND place routes.py from web2py.zip into the main web2py folder (web2py/) in the windows binaries directory

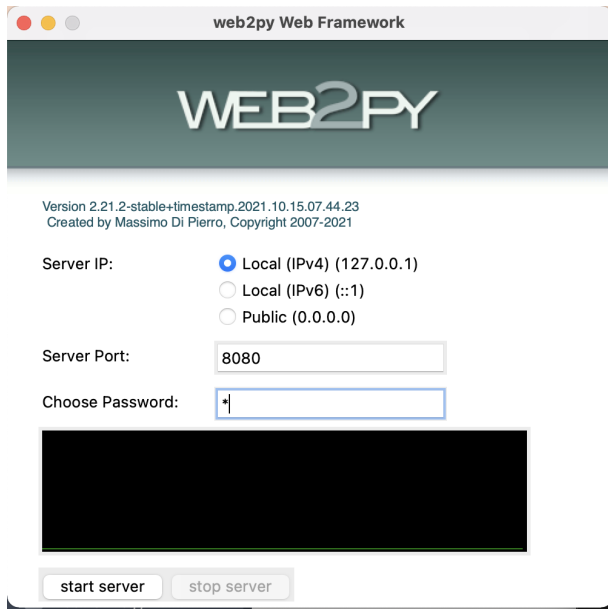


Figure 7: web2py gui

B PROJECT LINKS

<http://www.web2py.com/>

<https://www.isi.deterlab.net/index.php3>

<https://ucdavis.edu/>

<http://nob.cs.ucdavis.edu/classes/ecs235a-2021-04/>

<https://github.com/y76/ecs235a/>

<https://curl.se/>

REFERENCES

- [1] Overleaf Guide to \LaTeX
<https://www.overleaf.com/learn/latex/Tutorials>
- [2] Wikibooks \LaTeX Guide
<https://en.wikibooks.org/wiki/LaTeX>
- [3] Honey Tokens and honeypots for web ID and IH
Rich Graves
- [4] Honeypots: Concepts, Approaches, and Challenges
Iyatiti Mokube, Michele Adams
- [5] How to Mock a Bear: Honeygot, Honeygot, Honeygot Honeytoken: A Survey
Paul Lackner
- [6] Honeygot, Honeygot, Honeytoken: Terminological issues
Fabien Pouget, Marc Dacier Hervé Debar
- [7] Honeytokens As Active Defense
Robert Petrunić
- [8] - A Journey Into Deception Based Security - All about Honeygot Honeytokens
<https://www.youtube.com/watch?v=u6nXOGkRXOw>
- [9] DETERLab
<https://www.isi.deterlab.net/index.php3>
- [10] The State of Honeygot: Understanding the Use of Honey Technologies Today
Andrea Dominguez
- [11] cURL
<https://curl.se/>