WhisperKey: Creating a User-Friendly URL Shortener

Eunice Lin, Flora Tan, Linda Wang

May 12, 2016

Abstract

URL shorteners convert long, unwieldy URLs into shorter ones and redirect all requests to the shortened URL to the original website. In this project, we investigate the security flaws of ShoutKey, an easy-to-use temporary URL shortener that maps simple dictionary words to a given URL of interest.

We first conduct a security analysis of the vulnerabilities encompassed by Shoutkey. Then, we conduct a standard dictionary attack to access private URLs, storing the successful dictionary words in a separate database to conduct a second, more optimized dictionary attack, and analyzing the successful redirects to determine if there is any leakage of private user data. Finally, we propose a more secure implementation of ShoutKey called WhisperKey and empirically analyze how WhisperKey is designed to be less prone to security attacks than ShoutKey is.

1 Introduction

1.1 URL Shorteners

A URL shortener accepts a URL as input and generates a short URL. The service maintains an internal database mapping each short URL to its corresponding original URL so that any online access using a short URL can be resolved appropriately.

1.2 ShoutKey

ShoutKey is a temporary URL shortener that takes a long URL and maps it to a shorter url to aid sharing of links. Unlike other popular URL shorteners such as Tinyurl or Bitly that map the original URL to a random string of letters and numbers, ShoutKey was designed to be user-friendly by using keys that are standard English words that are easily pronounced or spelled. This makes it easy for users to share a link to those in close proximity, for example, by verbally telling the key to a friend across the room or writing it on a whiteboard at the front of class. Anyone who goes to the ShoutKey link can access the corresponding URL.

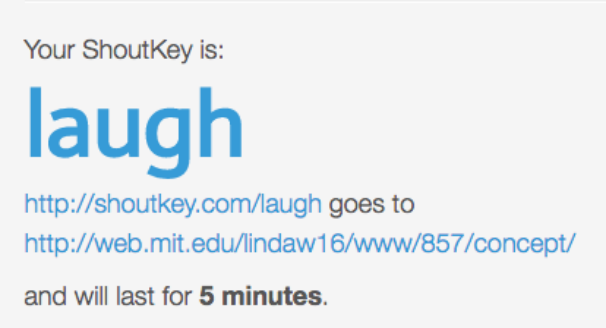

Figure 1: A ShoutKey with "laugh" as its key. Going to shoutkey.com/laugh will redirect the user to the given website for since its creation, after which the key may be reused to redirect to another site.

2 Background

A recently published paper by Georgieve & Shmatikov (2016) analyzed the security of URL shorteners such as Microsoft OneDrive and Google

Maps [1]. They demonstrated how short-URL enumeration could be used to discover and read shared content stored in the OneDrive cloud and find information shared using Google Maps. The authors found that 7% of the OneDrive accounts exposed in this fashion allow anyone to write into them. They also discovered that short-URL enumeration for Google Maps revealed directions that users shared with each other, enabling inference about residential addresses, true identities, and sensitive locations, such as abortion, mental-health, and addiction-treatment clinics medical facilities, as well as prisons and juvenile detention centers.

ShoutKey is a very familiar URL shortener in the classroom. It has garnered a significant audience of educators who can easily share links to class activities by telling the class the simple word to the ShoutKey link. At MIT, it has been used in courses including 6.813 (User Interface and Design) and 6.005 (Elements of Software Construction). Given the recently published papers regarding the security breaches of these URL shortening services, we were motivated to investigate any potential security flaws of ShoutKey since it is widely used by educators and students alike on campus.

3 Security / Attack Model

The use of the ShoutKey shortening service imposes risks on users submitting URLS. These threats are discussed in the following sections.

3.1 Privacy

In our project, we describe an experiment in which we enumerate words in the English dictionary to search for secret URLs that users have shortened using the ShoutKey website. By implementing a simple dictionary attack, a malicious third-party could easily view and make edits to private documents that were intended to be shared only with those who have knowledge of the corresponding ShoutKey. If the third party has malicious intent and had a target URL in mind, he could easily enumerate through the dictionary until he finds the right ShoutKey that redirects to the URL of interest. We found several URLs that led to private video conference calls, Google documents, Google forms, and even coding interview links that we could access and edit. More information regarding the results of our attack can be found in Section 4.1.2.

3.2 Sensitive Information

Often, users are not aware of the fact that once URLs are submitted to a URL shortening service, the URLs are no longer private. At the very least, the administrators of the service will have access to the URLs. Submitting secret URLs using ShoutKey therefore compromises the privacy of the data contained in the URLs. From the results of our dictionary attack as described in Section 4.1.2, we found several ShoutKeys that redirected to unlisted YouTube videos, or videos that are not searchable using YouTube's search interface. This therefore breaks YouTube's privacy models for users. We also found Google documents that appeared to be feature more private data, including unpublished research and dissertations, that should have been disclosed only to the authors but was accessible to anyone with the ShoutKey link.

ShoutKey currently has no published Privacy Policy on its website that assures the privacy of submitted URLs. Clearly, however, given the results of our attack, there exists a significant portion of users who did not heed the lack of privacy settings and went on to share very sensitive information using ShoutKey. This suggests a clear need for more secure measures to address the vulnerabilities of ShoutKey.

4 ShoutKey Security Analysis

4.1 Dictionary Attack

Since ShoutKey only uses dictionary words, we conducted a dictionary attack that tested http://shoutkey.com/[key] for every English word as the key to see if any of the URLs redirected, taking

special note of whether any of these redirected URLs led us to private data.

4.1.1 Enumeration

The dictionary we used was the built-in words standard file on all Unix operating systems, which contains 235,886 English words. We wrote a Python script that makes HTTP requests to http://shoutkey.com[key] using each word as the key, and then check the URL to see if it is different from the original URL. If it is different, that means the ShoutKey has successfully redirected.

Users can choose to set their ShoutKey links to expire after a period of time between 5 minutes and 12 hours. In order to save these redirect links to be visited later, we store data for each HTTP request into a MySQL database that includes the key, the redirected URL (or NULL if it did not redirect), and the time the request was made in a MySQL database. From this, we can then revisit these URLs even if the ShoutKey has already expired. We can also easily retrieve the number of redirected links and their URLs, or the total amount of time it took to make all the requests.

4.1.2 Results

In our security analysis, we made a total of 1,745,650 ShoutKey HTTP requests. This was approximately 6 full passes through the dictionary, plus some incomplete passes due to loss of internet connection, closing laptop, etc. Of these HTTP requests, 340 ShoutKey links successfully redirected. The most common types of websites that ShoutKey redirected to were Google Documents, download links, and education material.

We were interested in determining if any of the data that we found contained private information. We define private data as any data that is not intended to be shared publicly. Some examples include Google Documents with restricted sharing privileges, Google Forms or Doodles for private groups, and any personal information, such as names or photos of individuals.

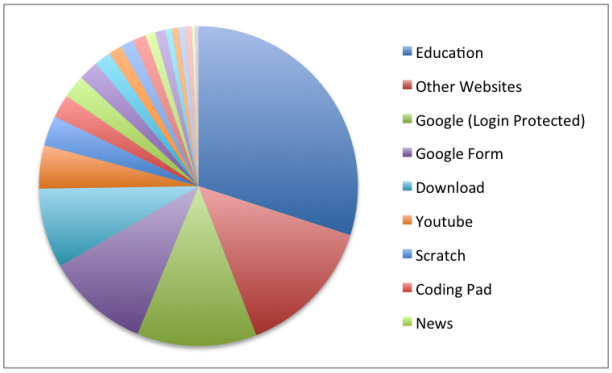

Figure 2: The types of URLs that redirects from ShoutKey.

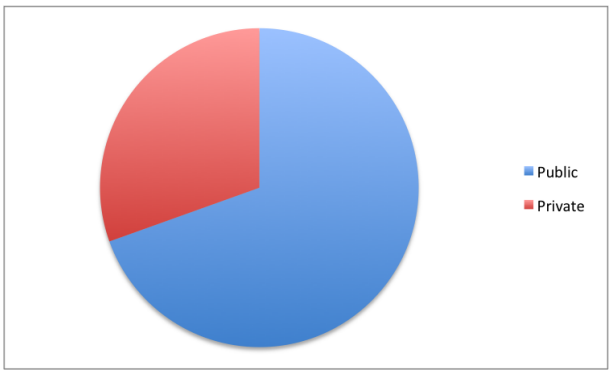

Figure 3: Approximately one third of the URLs we found redirected to private data.

We found several 36 ShoutKeys that redirected to Google Documents that allow any user with the link to view and edit the document. By finding the correct ShoutKey that redirects to the Google Document, we were able to potentially view and edit these documents, share the documents, and view the full names and default photos of all collaborators of the document. We were also able to view the revision history of the documents, which allowed us to see the full names and default photos of all users who have ever contributed to the document, including when and what they changed.

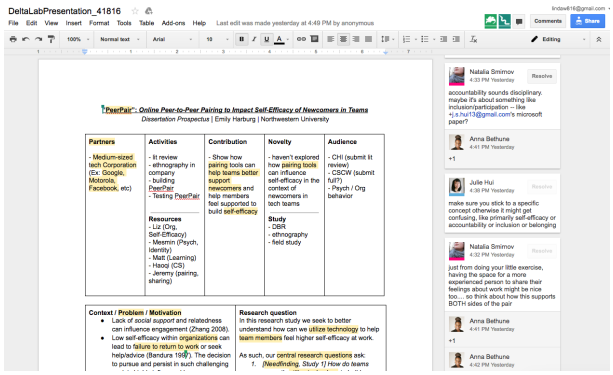

Figure 4: We found a Google Document of a dissertation, where we can view and edit the document as well as see the full names, pictures, and comments of all collaborators of the document.

We also found several Google Documents and Google Slides that had view permissions for anyone with the URL. For some of these view-only documents, even though we were unable to edit them, we were able to share the documents with others. For these cases, a malicious user could easily access these view-only documents or slides and then share it to himself or herself with edit permissions, giving the attacker edit permissions to modify the document.

In addition to Google Documents, we also found Google Forms and Doodles that allowed anyone with the link to submit answers to the form. This allows anyone with the URL to view the form questions and malicious users to potentially spam the form or submit bad input. Some of these forms and Doodles also included information regarding logistics for running an event, which people outside of the organization should not have access to. We also found many download links, and anyone who goes to the URL are able to download the file.

There were also ShoutKeys that linked to unlisted Youtube videos. Unlisted Youtube videos cannot be found through search on Youtube, and can only be accessed by people who have the URL. Therefore, by conducting a dictionary attack on ShoutKey, attackers can get access to these URLs and thus access to these videos that are meant to be private.

Other examples of private data that we found includes the source code of Scratch projects and coding pad where users can collaborate and code together. These platform allows us to view and modify the source code, although we would have to sign in to our own accounts in order to save any changes.

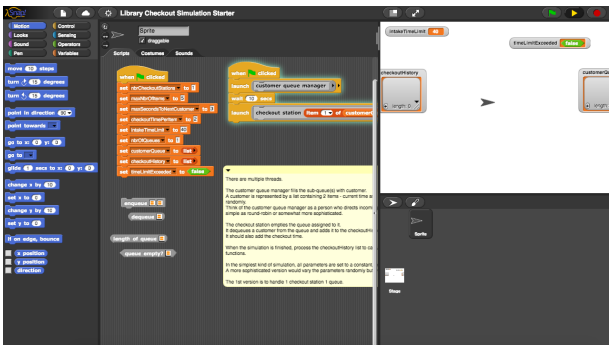

Figure 5: An example of a Scratch project that we found with view and edit access to any user with the URL.

Our script takes approximately 10 hours and 38 minutes to make one pass through our dictionary. This could be further optimized by removing the logging statements. This means that, on average, we can expect to find one successful redirect every 1.876 minutes.

4.2 Optimized Dictionary Attack

During our security analysis of ShoutKey, we discovered that ShoutKey prioritizes simple English words, in order to make it more user-friendly. According to the creator of ShoutKey, ShoutKey stores a dictionary of 596 "preferred" words, and once it uses up all the words in the preferred dictionary, then it uses a modified version of the standard Unix dictionary, which contains 13,569 words. During our dictionary

attack, we found that at a given time, there are usually only around 40 active ShoutKeys. Therefore, we can be fairly certain that the keys would be coming from the preferred dictionary.

4.2.1 Enumeration

For the optimized dictionary attack, we ran the same script used in the dictionary attack described previously, but using only the 340 words that successfully redirected during the first attack. Since we are certain that these words are in the ShoutKey dictionary, the probability that these successfully redirect is much higher. If we continue to run our dictionary attack, we can eventually reconstruct the entire ShoutKey dictionary and just run the dictionary attack using the ShoutKey dictionary.

4.2.2 Results

We found that the optimized dictionary attack was significantly faster than our original dictionary attack. Using our script, running the optimized dictionary attack with 340 simple English words in the ShoutKey dictionary takes only approximately 208.74 seconds to pass through the dictionary once, and we found approximately 42.5 words per pass. This means that, on average, we were able to find one successful redirect every a word every 0.2036 seconds, which is 9.2 times more efficient than the regular dictionary attack. Even if we do not have the full ShoutKey dictionary, we can find keys that redirect with a lot higher probability.

4.3 DOS Attacks

4.3.1 Too many requests

ShoutKey is vulnerable to a few types of Denial of Service (DoS) attacks. Our script is single-threaded and only runs one request at a time. We tried to speed up our script by multithreading it using the Go language, which simplifies multi-threading through the use of goroutines, and were able to traverse the entire dictionary in about 15 minutes. However, when we upped our script to 500 requests per second, we accidentally DOS'd the server. We emailed the website's creator, explaining what happened and promised not to run this new script again. Shoutkey was down for about a day, but after restarting the server and the database, it was once again functional.

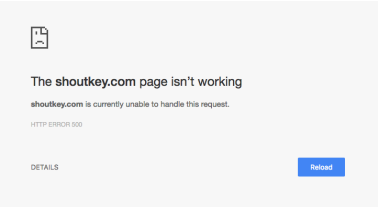

Figure 6: We accidentall8 took down the Shoutkey server when we sent 500 requests/s.

4.3.2 Limited Possibility Space

Since ShoutKey uses a dictionary of 14,165 words, ShoutKey risks running out of words if all words in its dictionary all redirect to another url at the same point in time. Therefore, if an attacker wants to take down the ShoutKey service, he or she could write a script to rapidly create a lot of ShoutKeys, and thus take up all the available words ShoutKey uses. This is not a problem right now since the number of ShoutKeys active at a time is much smaller than the number of available keys. However, it could potentially become a problem if ShoutKey becomes popular and gains a large user base.

We did not conduct this attack, as we didn't want to purposefully take down Shoutkey's server, and instead emailed the creator of ShoutKey. He informed that this is indeed a problem that he has considered, and if this happens, the user will just be denied a Shoutkey. In fact, he noticed that there were people who were creating a lot of ShoutKeys in order to spam on forums. To prevent this, he added a CAPTCHA, which requires users to check a box that says "I am not a robot," in order to distinguish between human users and bots. If a user creates five ShoutKey links in rapid succession, they are faced with a picture CAPTCHA that asks the user to

select images containing a certain randomized object.

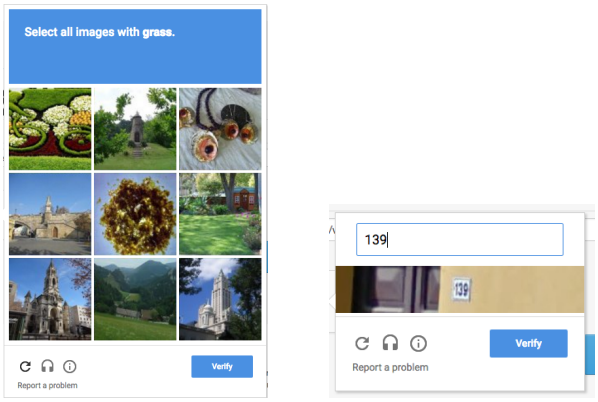

Figure 7: Users who create 5 ShoutKeys in rapid succession are tasked with one of these CAPTCHAs to show that they "are not a robot."

While the CAPTCHAs prevent users from writing a script to spam the website, ShoutKey can still be DoS'd if a persistent attacker decides to create many links by hand, or if the website just becomes very popular and has a large user base. As long as there are more requested keys than available words at any point in time, the user trying use ShoutKey will be denied a key.

4.3.3 ShoutKey Expiration

Another type of DoS attack would be to create immortal ShoutKey links. Upon creation, users can choose the lifespan of their ShoutKey up to a maximum of 12 hours. Inspecting the source, we can see that we can easily edit the value for these choices to be something very high, essentially creating an "immortal" link. However, when we tried to do this, ShoutKey did not create a link and instead refreshed the page. ShoutKey only accepted these changed values if they were another option, e.g. changing the value of 300 for the 5-minute option to be 3600, which is the value for the 1 hour option. It seems that ShoutKey has an extra check to only accept one of these four valid values, preventing anyone from creating immortal ShoutKey links.

Figure 8: Users have four options for how long their link will last. To the right is the source code.

5 WhisperKey Design

After analyzing the design of Shoutkey and running our attack against it, we came up with some designs to improve the security of the URL shortening site without compromising the original goal of having an easily share-able URL. We named our app WhisperKey, because instead of "shouting" one's URL for everyone to hear, we are taking extra measures to limit access to only the people who were meant to hear the key.

We designed WhisperKey to be more secure than Shoutkey by (1) increasing the length of an dictionary attack, and (2) decreasing the probability of a successful attack. We describe ways to approach to these ideas in the remainder of this section.

5.1 Increasing Time of Attack

5.1.1 Expanding the Dictionary

As mentioned in Section 4.1, ShoutKey has a primary dictionary of about 596 words and then draws random words from the standard Unix dictionary if these primary words are all taken in a single point in time. From our dictionary attacks in section 4.1, we discovered only about 40 active links at a time. This guarantees that any active ShoutKey key is from the set of the primary 596 words. The Unix dictionary has about 60,000 words after filtering for proper nouns and apostrophes, so we can greatly expand the space of possible keys for WhisperKey.

5.2 Requiring Two Keys

If ShoutKey's dictionary is of size n, it takes $O(n)$ to run a dictionary attack against Shoutkey by checking each possible word. By requiring two keys, we can increase the space of possibilities and subsequently the time it takes to attack to $O(n^2)$. In this example, instead of shoutkey.com/[key], the shortened WhisperKey URL will be of the form whisperkey.com/[key1][key2]. This does not compromise the user-friendliness of the site, as the words being used are still common English words.

5.3 Decreasing Probability of Success

Another approach is to require a password upon trying to visit a WhisperKey. This is essentially the same concept as having two keys, with the second key as a "password", but this also allows us to introduce restrictions and penalties for incorrect guesses and therefore limit the number of guesses the attacker can make.

5.3.1 Limiting Attempts

One additional security measure to passwords would be limiting the number of attempts that a user can try to guess for each key. After m attempts, the user is no longer allowed to try anymore for that key, and will not be able to access the redirect URL. Since the attacker can not run an unlimited dictionary attack against this URL, we can calculate the probability that he can randomly guess the password for a given key in a dictionary of size n:

$$\frac{1}{n} + \frac{(n-1)(1)}{n^2} + \frac{(n-1)(n-2)(1)}{n^3} = \frac{3n^2 - 4n + 2}{n^3}$$

For a dictionary of size 596, this translates to a 0.3% probability of guessing the password for a given key, which is significantly smaller than the 100% chance of finding the password after trying every word in the dictionary.

5.3.2 Adding Exponential Delays to Password Attempts

While a user who knows the password types it correctly with a high probability, people frequently make errors because they are human. For example, many students mistyped the password for this year's 6.857 student's portal because it was very similar to a real dictionary word, but was not an actual word, and students attempted with the same dictionary word multiple times. If someone thinks they know the password but in actuality does not know how to spell it correctly, they could very likely mistype the password several times even if the word is right in front of them.

One approach to penalize multiple password attempts without entirely locking out the user would be to add an exponential delay before verifying the user's credentials. For example, on the first attempt, the website would wait 1 ms before responding, on the second attempt: 10 ms... on the nth attempt: $10^(n-1)$ ms. A human's perceptual processor has a cycle time of about 100 ms [2], meaning that for up to 3 attempts, the response will seem instantaneous. After about 1 second (4th attempt), the user will notice the delay, and after 10 seconds (5th attempt), the average user will lose patience and look for another task to work on while waiting.
The time the user is expected to wait on the mth attempt for a dictionary of size n is

$$\frac{(n-1)!}{(n-m)!n^m} \sum_{1}^{m} [10^{m-1}]$$

By the 9th attempt, the user would need to wait over a day for the server to respond. Since ShoutKeys (and therefore WhisperKeys) expire after a maximum of 12 hours, this essentially limits a user to only 8 guesses.

5.3.3 Two-Factor Authentication

Two-Factor Authentication adds an extra layer of security to an application by requiring two of three types of credentials: 1) something a user knows (e.g. a password), 2) something a user has (e.g.

a hardware token), or 3) something a user is (e.g. fingerprint). We can further increase the security of WhisperKey by requiring the user to provide the first two.

Recent startups such as SoundPays[3] and SlickLogin[4] use sound as a medium for something a user has. Based on this idea, we could allow the creator of the WhisperKey to emit a high frequency sound from their computer (outside the range of human hearing), and have everyone else in the room enable the microphones on their computer. The WhisperKey link will not redirect until the microphone on the computer registers the sound being emitted by the creator of the link. This ensures that only those who are within the same room as the creator can access the link. Because ShoutKey was originally designed for people to share links verbally or on a board with those in the same room, this idea fits in very well with the original intended use case of the URL shortening site.

5.4 Concept Page

We implemented a concept page where one can play an attacker who is searching for a secret page that they know is linked to on ShoutKey. For the purposes of the concept, we limited the dictionary to just 5 words, and demonstrate the original ShoutKey page, WhisperKey with two keys, WhisperKey with a password, and WhisperKey with limited password attempts. We created a proof-of-concept WhisperKey with exponential-delay password attempts at whipserkey.herokuapp.com, detailed in the next section. The concept page can be accessed at http://web.mit.edu/lindaw16/www/857/concept/

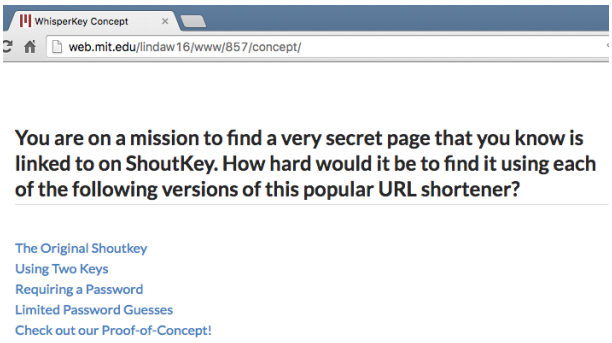

Figure 9: A concept page to allow users to try out the approaches we detailed in this section, and compare them to the original ShoutKey.

5.5 WhisperKey Implementation

We created a limited-attempt password-protected version of ShoutKey as a proof-of-concept of WhisperKey, deployed at whisperkey.herokuapp.com. Upon entering a private link, users are given two words: a key that is the redirect link; and a password that is the password required for the redirect link to work. We modeled our site to look like the original ShoutKey page, and it uses simple dictionary words just as ShoutKey does.

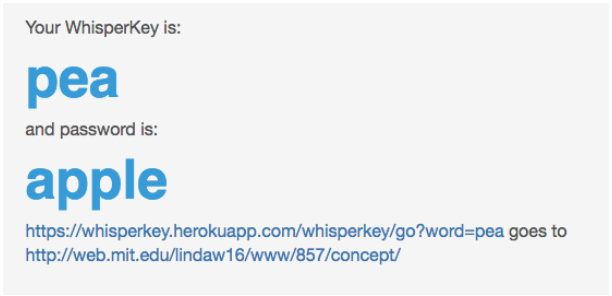

Figure 10: A WhisperKey with "pea" as its key and "apple" as its password. Going to whisperkey.herokuapp.com/whisperkey/go?word=pea will require the user to correctly enter "apple" before redirecting the user to the given website.

5.6 Future Work

Currently we are checking the password on the client side, so if a user examines the page source, they can see the correct password. In the future, this will be moved to the server side so that the user cannot see the password in the page source.

We are also keeping track of the number of

password attempts on the client side, so that the number of attempts can be reset by refreshing the page. We will also move this to the server side so that the user cannot reset the number of attempts in order to reduce their delay time. If we move the counter to the server side, this becomes a counter per WhisperKey instead of a counter per user. Therefore, if a malicious user attempts the password multiple times, this will cause everyone to be subject to the exponential delay. We plan to limit these attempts according to IP addresses or by using cookies.

6 Conclusion

Our findings from this project show that the ShoutKey URL shortening service turns out to have serious consequences for the security and privacy of users. We exploit a key vulnerability with the ShoutKey website by conducting a simple dictionary attack. The resulting URLs that we found demonstrate how easily a third party could access private URLs and view sensitive information.

As we probed the ShoutKey website, we found no Privacy Policy statement making any guarantees about the privacy of the URLs submitted to the service. Given that nearly a third of the links we found led to private data, we conclude that users place too much trust in the website; they should be more aware of the type of data that they are sharing, as it is quite vulnerable to being viewed by prying eyes. More generally, users should pay close attention to the Privacy Policies (or lack thereof) listed on URL shortening websites before they share any sensitive data.

References

[1] Georgiev, M. and Shmatikov, V. Gone in Six Characters: Short URLs Considered Harmful for Cloud Services. April 2016.

[2] https://www.nngroup.com/articles/response-times-3-important-limits/

[3] http://www.soundpays.com/solutions2

[4] http://tech.firstpost.com/news-analysis/google-acquires-slicklogin-sound-based-passwords-become-mainstream-218342.html

7 Dictionary Attack Script

```
import mysql.connector
import urllib2
import datetime

# USE THIS FOR REGULAR DICTIONARY ATTACK
#wl = open('/usr/share/dict/words', 'r')
# USE THIS FOR SHOUTKEY DICTIONARY ATTACK
wl = open('/Users/eunice/Dropbox (MIT)/MIT/Year4/Spring/6.857/elf/shoutkey_dictionary.txt', 'r')
wordlist = wl.readlines()
wl.close()

def get_redirected_url(url):
    try:
        opener = urllib2.build_opener(urllib2.HTTPRedirectHandler)
        request = opener.open(url)
        return request.url
    except:
        print "ERROR OCCURRED"

# set up a database to store the redirect urls
url_data = mysql.connector.connect(host='localhost',
        user='root',
        password='',
        db='urls') # The name of the database we are using is urls

cursor = url_data.cursor()

#drop_stmt = (
#        'DROP TABLE IF EXISTS urls'
#)
#cursor.execute(drop_stmt)

# USE THIS FOR REGULAR DICTIONARY ATTACK
#create_stmt = (
#        'CREATE TABLE IF NOT EXISTS urls'
#        ' (WORD CHAR(30), REDIRECT_URL CHAR(255), TIMESTAMP CHAR(30))'
#)

# USE THIS FOR SHOUTKEY DICTIONARY ATTACK
create_stmt = (
        'CREATE TABLE IF NOT EXISTS shoutkey_attack'
        ' (WORD CHAR(30), REDIRECT_URL CHAR(255), TIMESTAMP CHAR(30))'
)

cursor.execute(create_stmt)
# USE THIS FOR REGULAR DICTIONARY ATTACK
#insert_stmt = 'INSERT IGNORE INTO urls (WORD, REDIRECT_URL, TIMESTAMP) VALUES (%s, %s, %s)'
```

```
# USE THIS FOR SHOUTKEY DICTIONARY ATTACK
insert_stmt = 'INSERT IGNORE INTO shoutkey_attack (WORD, REDIRECT_URL, TIMESTAMP) VALUES (%s, %s

starttime = datetime.datetime.now()
num_redirected = 0
for word in wordlist:
        word = word[0:len(word)-1] # strips off newline at the end
        original_url = "http://shoutkey.com/" + word
        redirected_url = get_redirected_url(original_url)
        if redirected_url == original_url or redirected_url == "http://shoutkey.com/" or redirec
                # did not redirect
                print word, " did not redirect"
                cursor.execute(insert_stmt, (word, None, datetime.datetime.now()))
                url_data.commit()
        else:
                # redirected
                print word, " redirected to ", redirected_url
                num_redirected += 1
                cursor.execute(insert_stmt, (word, redirected_url, datetime.datetime.now()))
                url_data.commit()

totaltime = datetime.datetime.now() - starttime
print totaltime
print "TOTAL NUMBER OF WORDS = ", len(wordlist)
print "NUMBER OF REDIRECTS = ", num_redirected
url_data.close()
```