SlickWraps Penetration Test Results

Preface

This document summarizes my findings during the penetration test of SlickWraps' web service, https://www.slickwraps.com.

> Dear SlickWraps,
>
> Beyond your severe lack of following cybersecurity best practices, I am disgusted that you decided to block me on Twitter when I tried to reach out, ignored all of my attempts at establishing communication, and failed to take timely action even after I shared with you an initial write-up of my results.

Findings

SlickWraps' Website

On their main website, SlickWraps exposes a phone case customizer which allows visitors to upload custom images. Inspecting network requests, I noticed that the customizer uploads these images to the web server by `POST` ing to a PHP script located at `/js/html5/plupload/upload.php`. Uploaded files would (by default) be stored inside the `/media/uploadedImage` directory. Following a quick online search, I found this script, which I assumed (and later confirmed) to closely match the `upload.php` script used by SlickWraps' customizer.

Inspecting the file and testing my findings with initial hand-crafted POST requests, I discovered that this script allows clients to specify a custom filename which is **susceptible to path traversal**. To make matters worse, the script **does not protect against overwriting existing files**. Thus, as long as the web server has sufficient write permissions, this `upload.php` endpoint **allows anyone to write (and overwrite) any file to any location** on the server web root.

Using this vulnerability, I uploaded a custom `.htaccess` file (with `Options +Indexes`) into the `/media` directory. Doing so allowed me to use a directory index, **listing all contents of the `/media` folder, including any and all subfolders**. Inside this `/media` folder, I was able to find copies of:

- **Resumes** of current and past SlickWraps employees, including selfies, e-mail addresses, home addresses, and more

- Back-ups and a 9 GB dump of uploaded print designs, containing **personal customer photos** uploaded via the aforementioned customizer (yes, including dick pics)

Further leveraging the insecure upload script, I managed to deploy a custom `index.php` into an exiting `/media` subfolder. To my surprise, navigating to the subfolder in a browser tricked SlickWraps' web server into running the script. Thus, I had achieved **remote code execution**.

To expedite further testing, I uploaded a copy of the p0wny-shell. (Note that I slightly modified the file to circumvent common anti-malware signatures.) Sure enough, I had unlocked the ability to **execute shell commands** on demand.

Knowing SlickWraps' website was powered by Magento 1.8, I located and decrypted the local configuration file. In here, I found MySQL and Redis credentials, and thus had **full access to their entire database**, including additional SlickLabs WordPress databases.

Investigating the complete 17 GB MySQL dump gave me access to the following:

- All current and historical customer data, from orders and products to **billing/shipping addresses and phone numbers** since 2014
- Admin accounts and their password hashes
- Access and error logs
- **Current API credentials** for
    - MadMimi
    - PayPal Payments Pro
    - Braintree
    - ShipHero
    - Zendesk
    - Facebook
    - Twitter
    - Instagram

Despite already owning a full copy of the files and database, I tried to gain access to SlickWraps' Magento admin dashboard. Initial attempts revealed that their admin path, located at `/shocker`, was protected behind a WAF. However, via the shell I was able to relocate the admin URL, circumventing said WAF. After injecting a new account into the `admin_user` table, I gained **full control over the admin dashboard**, too.

Taking a step back to the uploaded web shell, I was due for an upgrade. Once I set up a local netcat listener, I was able to initiate a reverse shell and thus **start a bash shell session** on their server.

Via the bash shell, I briefly ran a few interactive scripts to scan their system for additional vulnerabilities. Early findings hint that their system appears to be vulnerable to Dirty COW, despite this being a known and fixed exploit since 2016.

After further investigation of the codebase, I also found what seems to be a hidden (albeit dormant) backdoor in a `hello.php` plugin inside a WordPress backup. However, the current WAF is preventing execution of the presumed backdoor from the web.

Zendesk Account

SlickWraps is using Zendesk to manage (or actively ignore) incoming email support tickets. The Zendesk API credentials I retrieved earlier allowed us to create a brand new user on their Zendesk platform and upgrade its role to **gain full Zendesk admin permissions**.

Once logged in, I gained access to to all **historical and current support conversations**, including (but not limited to) closed tickets, open tickets, and forwarded emails that did not trigger a ticket (*Suspended Tickets*).

Twitter Account

SlickWraps is typically present on Twitter via one of two handles: `@SlickWraps` and `@SlickWrapsHelp`. The latter is set up to share the same email address as their Zendesk email configuration. Thus, with the help of Zendesk's forwarded email inbox, I was able to initiate a password reset request from Twitter and intercepted the password reset email. I set up a new password and immediately gained **access to the `@SlickWrapsHelp` handle**.

Internal Communication Account

Aside from the above, I further gained **access into their Slack portal** via the forwarded inbox exposed in Zendesk.

Payment Gateways

Using their PayPal Payments Pro and Braintree API credentials, I managed to retrieve current **account balances** and read **transaction logs**.

Conclusion

All the above exploits and vulnerabilities can be traced back to a single, insecure `upload.php` file. Besides breaking several other cybersecurity best practices, this mistake is absolutely inexcusable. That this mistake requires nothing short of imponderable ignorance with regards to private, customer data becomes very apparent when reading the first couple of lines clearly present at the beginning of the script:

```
#!! IMPORTANT:
#!! this file is just an example, it doesn't incorporate any security checks
and
#!! is not recommended to be used in production environment as it is. Be
sure to
#!! revise it and customize to your needs.
```

Furthermore, it is inexcusable that SlickWraps decided to both ignore and block my attempts at reaching out and informing them about these vulnerabilities. Even after sharing the initial report of my findings, they failed to address any of them in a timely manner.

Needless to say, SlickWraps' malpractices towards cybersecurity basics means they are not a parter you should share your personal information with.

As of today, it is unknown for how long these vulnerabilities have been accessible through the web to everyone, and if or how often private data has been retrieved from malicious parties.