#690   The Prevalence of E-Mail Header Injections on the World Wide Web

Review-690C

> Expanding the investigation to email forms which do not allow user specification of the "to:" field would greatly increase the substance of this finding

> Unfortunately, limiting the crawl to only the "to:" field greatly attenuates the contribution here. I believe this paper could be substantially improved by taking a broader look at header injection vulnerabilities. For instance, if the date or subject headers are user input (or even input via the form submission from the client), can an attacker add a "cc:" or "bcc:" header at that point? That would allow a form (e.g. a "contact us" form) to be used as a spamming vector, as it would allow arbitrary destinations rather than specific destinations.

> 690A: The rest of the fields (e.g., name, password, etc.) the fuzzer fills out in the form being tested are not key to discovering email header injection vulnerabilities - that is, their presence is necessary to pass the form input validation.

> 690B: On the attack side maybe looking at hard to exploit injection such as the one in the body would have make the paper stronger.

As noted in section 3.3.6 “Analyzing regular e-mail”, we would like to point out that we did indeed check whether any of the fields which took in the user input we injected (any data, not just malicious payloads) appeared as part of either the headers or the body of the resulting e-mail. If these user-input-based fields were found on the resulting email, we then injected these fields (which could be fields like subject, d.o.b, name, etc.) with the payloads to check for the presence of e-mail header injection. The forms that we fuzzed were also not limited to only forms that allowed users to specify ‘To:’ fields, but also regular forms (e.g. signup/signin/contact-us).

> Another concern I had is regarding the correlation between finding 106 hosts on spamming blacklists and header injection. As noted above, it's unclear (and unlikely) that these hosts are on this blacklist because of header injection, and the simplest explanation is that their email form is not rate limited in any way

We concur, and have noted the same in Section 5.6.

> Minor: In section 5.1, by "domains" do you mean fully qualified domain names, or "registered" domains (e.g. google.com or yahoo.co.jp)?

We refer to fully qualified domain names

Review-690A

> They built an “email form receiver” (see section 3.3.4) that reconstructs the previously identified email forms from the bits and pieces they store in their database. Unfortunately, it is not clear why they did not just “remember” the URL of the page where that form can be found on the Internet and navigate to it directly, instead of reconstructing it from their database. This fact appears to be an oversight on their part.

We would like to note that this was done in order to avoid additional requests to the same web page, as we inject the payloads in two stages (Section 3.3.5 discusses this), and to avoid parsing forms twice.

> Moreover, since they can reconstruct these forms and submit them at a later time, that means that all those vulnerable forms must also be vulnerable to CSRF attacks. That angle was never discussed in the paper.

> Why didn’t the authors test all forms is not clear from the paper. Similarly, they found 1,132,157 forms with email fields, but tried the regular (non-malicious payload) flow with 934,016 forms only. The difference in numbers is not accounted for. - Out of the 934,016 forms they submitted via the regular flow they received only 52,724 emails. Did they try to test some of the forms that didn’t send out emails manually so that they can verify that their code is not buggy? Not clear. How many of the “broken” forms were not working because of captcha/CSRF protection/etc? Not clear.

We respectfully agree that these points could have been made much clearer in the ‘limitations’ section of the paper. We did indeed test many of the forms manually, but concede that we do not have data about how many forms were not working due to CAPTCHA/CSRF Protection, etc.

> Use Selenium and analyze dynamically generated forms, too. Then you can strengthen your analysis section by drawing a comparison between static vs. dynamic forms. By leveraging Selenium, you will also solve the “malformed HTML” problem you had using Soup.

We make this observation in the limitations section of our paper, and chose not to use Selenium to avoid the performance overhead involved in parsing each of the 21,675,680 URLs.

Review-690B

> The attack is known for a while and already tested by many security scanners for years e.g Acunetix is scanning for it since 2013

While it is true that tools like Acunetix have been testing for this since late 2013, we would like to remark that our system is the first open-source implementation that detects the presence of email header injection vulnerabilities in a blackbox manner.

> This leave us with a paper which is the first academic and large scale measurement of a known attack.

We concur.

> For example a stronger technical contribution like a code source analyzer that detect the vulnerability along with a study of which version of popular open source projects are affected.

Our goal was to develop a black-box approach that did not require the source code of a website to detect email header injection.