Web Application Firewall (PyWAF)

Julian Hirn
Anand Hegde
Lucas McCanna

About WAFs

- ➔ Defense on Layer 7 (Application)
- ➔ Frontline of defence against HTTP and other application level attacks
  - ◆ Some functionalities are lower level
- ➔ Software and Appliances (Hardware)
- ➔ Methods of passing traffic through the WAF
  - ◆ In-line transparent bridge, reverse proxying traffic, or being embedded in the web server itself

Why WAFs?

- ➔ Deploy & configure solution
- ➔ Easy to scale in the age of virtualization
- ➔ Application agnostic

- ➔ Downside: promote bad habits
  - ◆ Shouldn’t be used as a bandaid

The Attacks

- SQL Injection
- XSS (Stored, Reflected)
- Denial of Service

Methodology

- ➔ Criteria of success for WAF
  - ◆ Web Application Security Consortium
- ➔ Followed guidelines to design our implementation
- ➔ Target attacks mention in their criteria
- ➔ Evaluate implementation using common versions of the attacks we aimed to defend

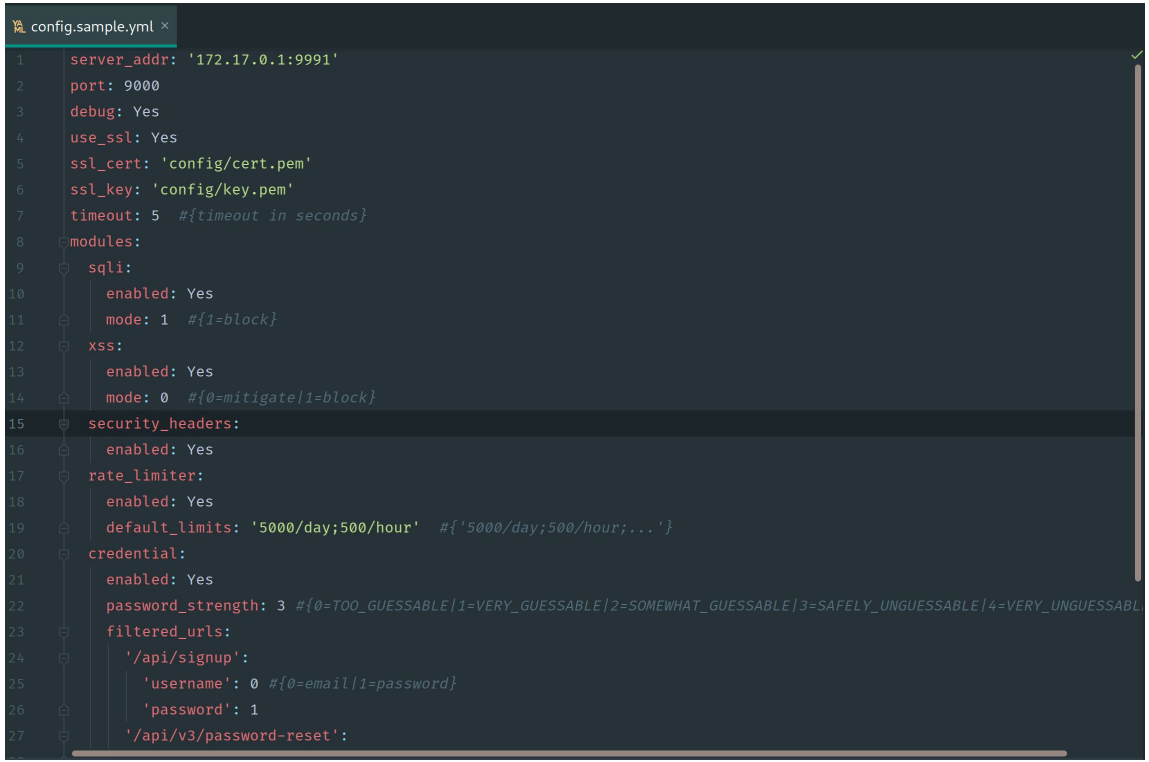
Our implementation

- ➔ We decided to go with the following for our implementation:
  - Software based approach (Can be virtualized)
  - WAF can act as a reverse proxy for traffic to the web server
  - Safe defaults - All security features are explicitly enabled and as strict as possible, unless configured otherwise
  - WAF handles lower layer interactions with the client

Features

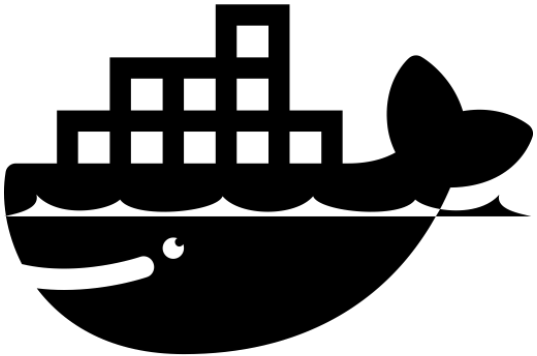
- ➔ Detects and prevents Reflected and Stored XSS attacks by dropping requests or escaping dangerous characters
- ➔ Detects and prevents SQL injections by matching form input in vulnerable forms to SQL commands and escape characters
- ➔ Prevents DoS for the web server by rate limiting
- ➔ Supports TLS with HTTPS for requests
- ➔ Adds security headers requests
- ➔ Checks for breached emails/passwords

Features (cont.)



Deliverables

github.com/Nineluj/py-waf



docker.io/nineluj/py-waf

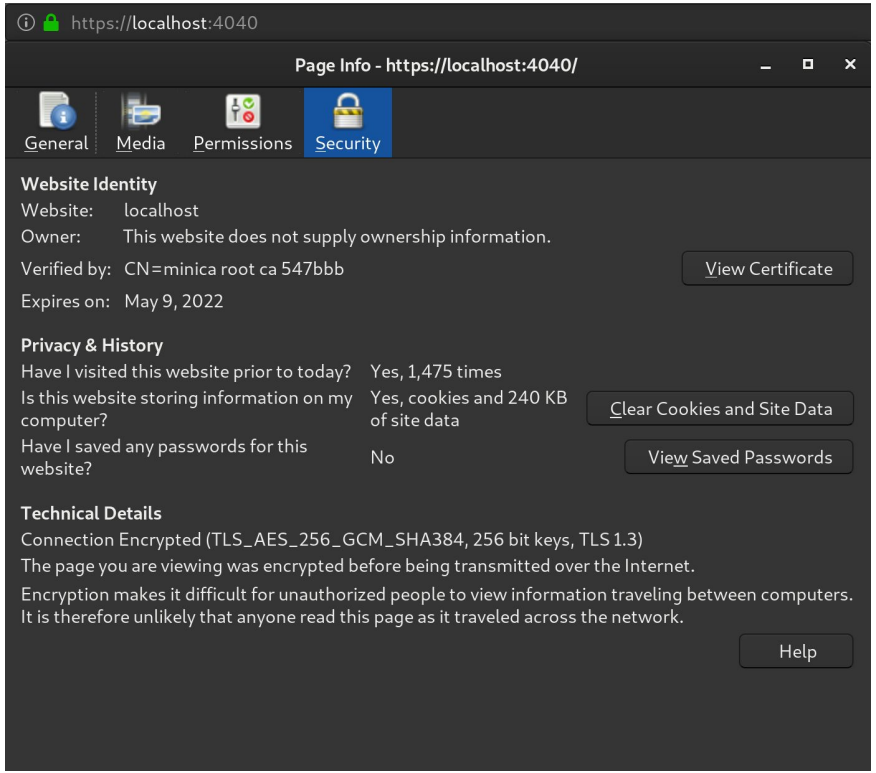
Our environment

- ➔ All of our containers and applications were running on the same machine for testing, but can be separated for distribution to different hosts.
- ➔ Out WAF was coded using Python and relied on a Flask implementation to handle incoming requests and for use of its various libraries
- ➔ These requests would be filtered/checked by our implemented protections and if there were no issues, would be forwarded to the real web server as a cleaned up request.
- ➔ Any malicious request or other traffic would be blocked/mitigated at the WAF and the web server would not be served the malicious request

Vulnerable Server

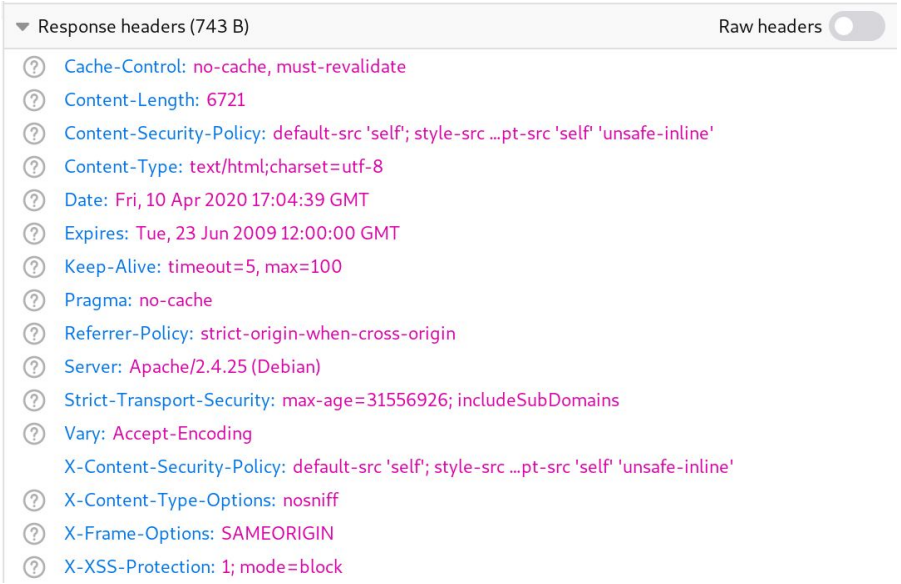
- In order to test that our WAF actually protects against attacks, we needed a server to launch attacks against
- One that use is the OWASP Vulnerable Web Application (VWA) as the application running on the web server we protect
  - Has vulnerabilities for all of the features we needed to test
- Another is the Damn Vulnerable Web App (DVWA), which has a few more advanced tests
- The WAF provides the attacker with access to the web app as if the attacker was directly connecting with it, and only filters/blocks content when it is detected as malicious

Examples in use: HTTPS



- ➔ Our WAF implements HTTPS, regardless of what the vulnerable web server uses it (Secures insecure web servers)
- ➔ WAF accepts any given (and valid) .pem certificate with key

Examples in use: Security Headers



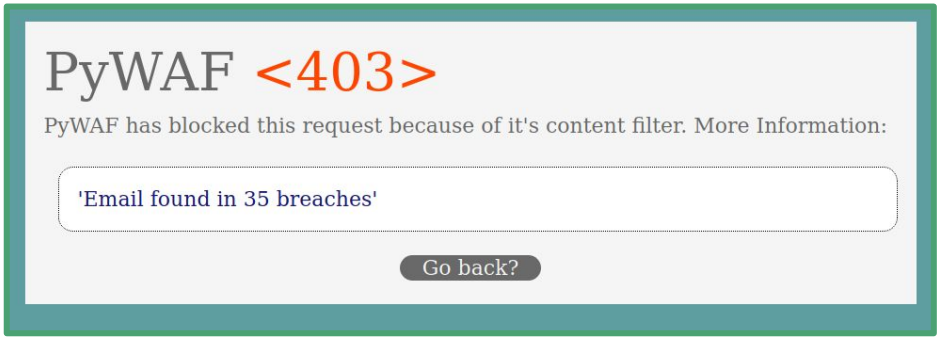
➔ Our WAF enforces security with security headers for every response:

- (X-)Content-Security-Policy
- X-XSS-Protection
- X-Frame-Options
- HSTS

Examples in use: Rate Limiter

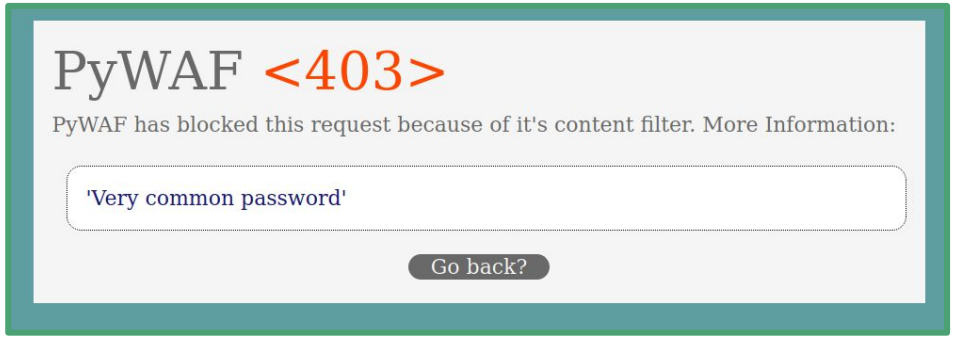
```
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3CA%250aONPOINTEReNtER%2509%3D%2509%5B8%5D.find%28confirm%29%2F%2Fv3dm0s3nd HTTP/1.1" 200 -
[2020-04-10 10:11:50,629] INFO in reverse_proxy: Retrieving URL: http://localhost:8000/www/XSS/XSS_level1.php?username=st4r7s%26lt%3BD3V%250aONPoIntERENTER%250a%3D%250aconfirm%28%29%2F%2Fv3dm0s3nd
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3CD3V%250aONPoIntERENTER%250a%3D%250aconfirm%28%29%2F%2Fv3dm0s3nd HTTP/1.1" 200 -
[2020-04-10 10:11:50,641] INFO in reverse_proxy: Retrieving URL: http://localhost:8000/www/XSS/XSS_level1.php?username=st4r7s%26lt%3BdETaIlS%250donpoiNTErEnteR%250a%3D%250aconfirm%28%29%250dx%2F%2F3nd
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3CdETaIlS%250donpoiNTErEnteR%250a%3D%250aconfirm%28%29%250dx%2F%2F3nd HTTP/1.1" 200 -
[2020-04-10 10:11:50,653] INFO in reverse_proxy: Retrieving URL: http://localhost:8000/www/XSS/XSS_level1.php?username=st4r7s%26lt%3BhTmL%2F%2B%2FoNPoINTEreNTER%250a%3D%250a%28prompt%29%60%60%2F%2F3nd
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3ChTmL%2F%2B%2FoNPoINTEreNTER%250a%3D%250a%28prompt%29%60%60%2F%2F3nd HTTP/1.1" 200 -
[2020-04-10 10:11:50,666] INFO in reverse_proxy: Retrieving URL: http://localhost:8000/www/XSS/XSS_level1.php?username=st4r7s%26lt%3Ba%2F%2B%2FONmOuSEoVER%2509%3D%2509a%3Dprompt%2Ca%28%29%2F%2Fv3dm0s3nd
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3Ca%2F%2B%2FONmOuSEoVER%2509%3D%2509a%3Dprompt%2Ca%28%29%2F%2Fv3dm0s3nd HTTP/1.1" 200 -
[2020-04-10 10:11:50,679] INFO in reverse_proxy: Retrieving URL: http://localhost:8000/www/XSS/XSS_level1.php?username=st4r7s%26lt%3BD3V%250dONMoUsEoVEr%2B%3D%2B%5B8%5D.find%28confirm%29%2F%2Fv3dm0s3nd
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3CD3V%250dONMoUsEoVEr%2B%3D%2B%5B8%5D.find%28confirm%29%2F%2Fv3dm0s3nd HTTP/1.1" 200 -
[2020-04-10 10:11:50,691] INFO in reverse_proxy: Retrieving URL: http://localhost:8000/www/XSS/XSS_level1.php?username=st4r7s%26lt%3BA%2F%2B%2FONMOUseOVer%2B%3D%2Bconfirm%28%29%2F%2Fv3dm0s3nd
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3CA%2F%2B%2FONMOUseOVer%2B%3D%2Bconfirm%28%29%2F%2Fv3dm0s3nd HTTP/1.1" 200 -
[2020-04-10 10:11:50,703] INFO in reverse_proxy: Retrieving URL: http://localhost:8000/www/XSS/XSS_level1.php?username=st4r7s%26lt%3BHtmL%250aoNpOInTERErEntEr%250a%3D%250aa%3Dprompt%2Ca%28%29%2F%2F3nd
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3CHtmL%250aoNpOInTERErEntEr%250a%3D%250aa%3Dprompt%2Ca%28%29%2F%2F3nd HTTP/1.1" 200 -
[2020-04-10 10:11:50,715] INFO in reverse_proxy: Retrieving URL: http://localhost:8000/www/XSS/XSS_level1.php?username=st4r7s%26lt%3Ba%250aoNPOiNtErEnTer%250d%3D%250da%3Dprompt%2Ca%28%29%250dx%2F%2Fv3dm0s
3nd
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3Ca%250aoNPOiNtErEnTer%250d%3D%250da%3Dprompt%2Ca%28%29%250dx%2F%2Fv3dm0s3nd HTTP/1.1" 200 -
[2020-04-10 10:11:50,726] INFO in reverse_proxy: Retrieving URL: http://localhost:8000/www/XSS/XSS_level1.php?username=st4r7s%26lt%3BD3V%250aonmoUSEoVER%2509%3D%2509confirm%28%29%250dx%2F%2Fv3dm0s3nd
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3CD3V%250aonmoUSEoVER%2509%3D%2509confirm%28%29%250dx%2F%2Fv3dm0s3nd HTTP/1.1" 200 -
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3CHTmL%250aoNpoiNTErenter%2B%3D%2B%28prompt%29%60%60%2F%2F3nd HTTP/1.1" 429 -
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3CDETAILS%2509oNTOGgle%2509%3D%2509%28prompt%29%60%60%250dx%2F%2F3nd HTTP/1.1" 429 -
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3CA%2509OnPointerEnteR%250a%3D%250aconfirm%28%29%250dx%2F%2Fv3dm0s3nd HTTP/1.1" 429 -
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3CdetailS%2F%2B%2FONPoInTEReNTer%250d%3D%250d%28confirm%29%28%29%2F%2F3nd HTTP/1.1" 429 -
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3CDeTAilS%250dONPoiNtERENtER%250a%3D%250a%28prompt%29%60%60%2F%2F3nd HTTP/1.1" 429 -
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3CdETAilS%250aoNtOGgLE%250d%3D%250dconfirm%28%29%250dx%2F%2F3nd HTTP/1.1" 429 -
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3CdETaILS%2509onPoiNTeReNTER%2509%3D%2509%28prompt%29%60%60%250dx%2F%2F3nd HTTP/1.1" 429 -
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3CA%2F%2B%2FonpoiNTeReNteR%250a%3D%250aconfirm%28%29%2F%2Fv3dm0s3nd HTTP/1.1" 429 -
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3CD3V%2F%2B%2FonPoiNterEnteR%2B%3D%2B%5B8%5D.find%28confirm%29%2F%2Fv3dm0s3nd HTTP/1.1" 429 -
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3CD3V%250dOnmOuSeoveR%250a%3D%250a%28prompt%29%60%60%250dx%2F%2Fv3dm0s3nd HTTP/1.1" 429 -
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3Cd3v%250aONPOiNTeRENtER%250d%3D%250dconfirm%28%29%250dx%2F%2Fv3dm0s3nd HTTP/1.1" 429 -
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3CDetAIlS%250aOnpoINtErENtER%250d%3D%250d%5B8%5D.find%28confirm%29%2F%2F3nd HTTP/1.1" 429 -
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3ChTmL%250dONPOinTErenTeR%2509%3D%2509a%3Dprompt%2Ca%28%29%2F%2F3nd HTTP/1.1" 429 -
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3Ca%2509OnPOIntERENtEr%250d%3D%250da%3Dprompt%2Ca%28%29%2F%2Fv3dm0s3nd HTTP/1.1" 429 -
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3Cd3V%2509ONpOinTEreNteR%2509%3D%2509%28prompt%29%60%60%250dx%2F%2Fv3dm0s3nd HTTP/1.1" 429 -
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3CDetAIlS%2F%2B%2FONPoiNteRENtER%2509%3D%2509%5B8%5D.find%28confirm%29%250dx%2F%2F3nd HTTP/1.1" 429 -
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3CA%250aONmouseOVEr%2B%3D%2B%28confirm%29%28%29%2F%2Fv3dm0s3nd HTTP/1.1" 429 -
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3Ca%250dOnmouSEoVER%2B%3D%2B%28prompt%29%60%60%2F%2F3nd HTTP/1.1" 429 -
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3CDeTaIls%250doNpOiNTerEnTER%250d%3D%250d%28confirm%29%28%29%2F%2F3nd HTTP/1.1" 429 -
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3CDEtAils%2F%2B%2FOnpOiNTeREnter%250d%3D%250d%28prompt%29%60%60%250dx%2F%2F3nd HTTP/1.1" 429 -
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3CD3V%2509ONPoiNTerEnTEr%2B%3D%2B%5B8%5D.find%28confirm%29%250dx%2F%2Fv3dm0s3nd HTTP/1.1" 429 -
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3CDeTaiLs%250aonToGgle%250d%3D%250da%3Dprompt%2Ca%28%29%2F%2F3nd HTTP/1.1" 429 -
127.0.0.1 - - [10/Apr/2020 10:11:50] "GET /www/XSS/XSS_level1.php?username=st4r7s%3CA%250donmOuSEOVer%250d%3D%250dconfirm%28%29%2F%2Fv3dm0s3nd HTTP/1.1" 429 -
```

Examples in use: Credential Validation



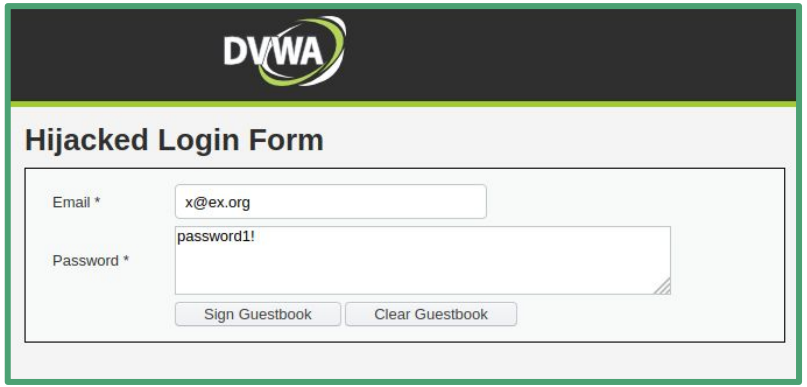
➔ Our credential validation feature allows for any form to have its input validated by to check for breached emails/passwords and password strength

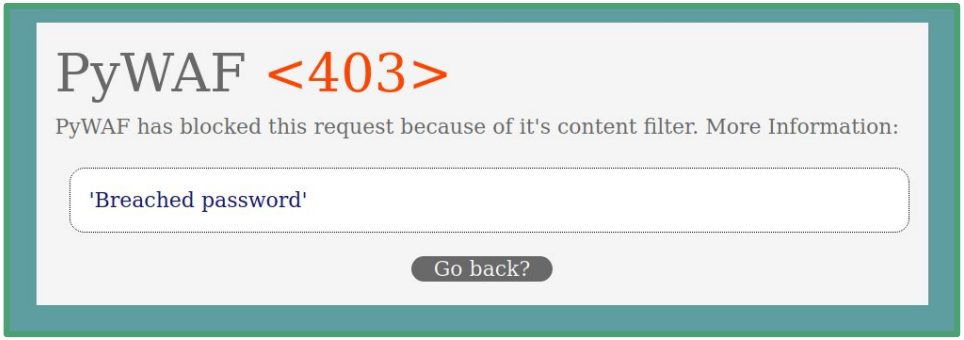
Examples in use: Credential Validation



➔ Given a poor password, the WAF will warn the user. Password strength is measured calling by the pwnedpasswords.com API.

Examples in use: Credential Validation



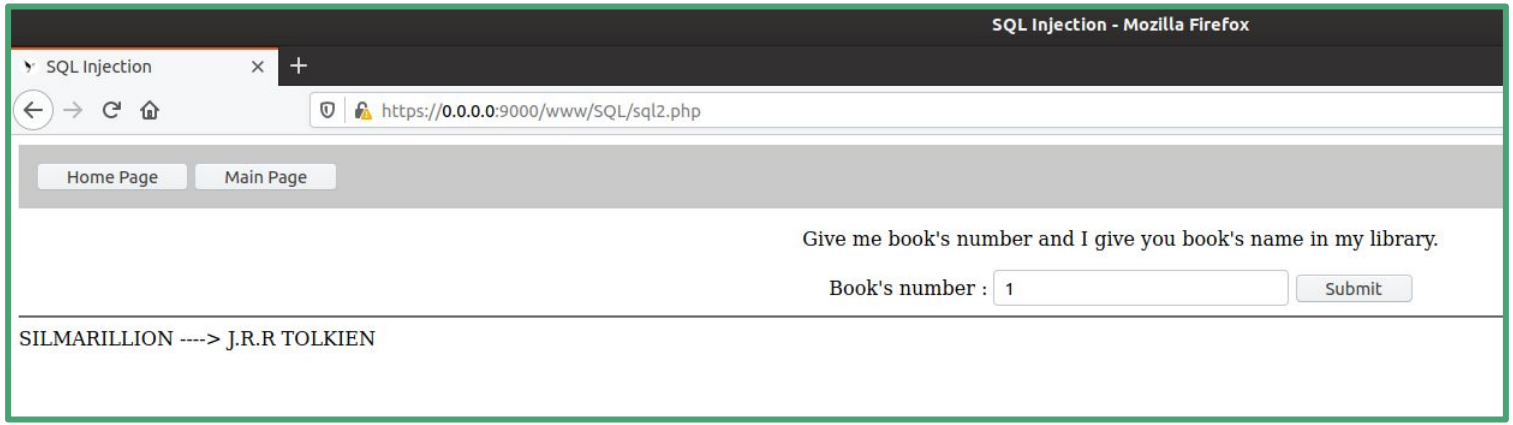


➔ The WAF can also check passwords to see if they are in common breached password lists, and therefore are vulnerable.

Examples in use: Credential Validation

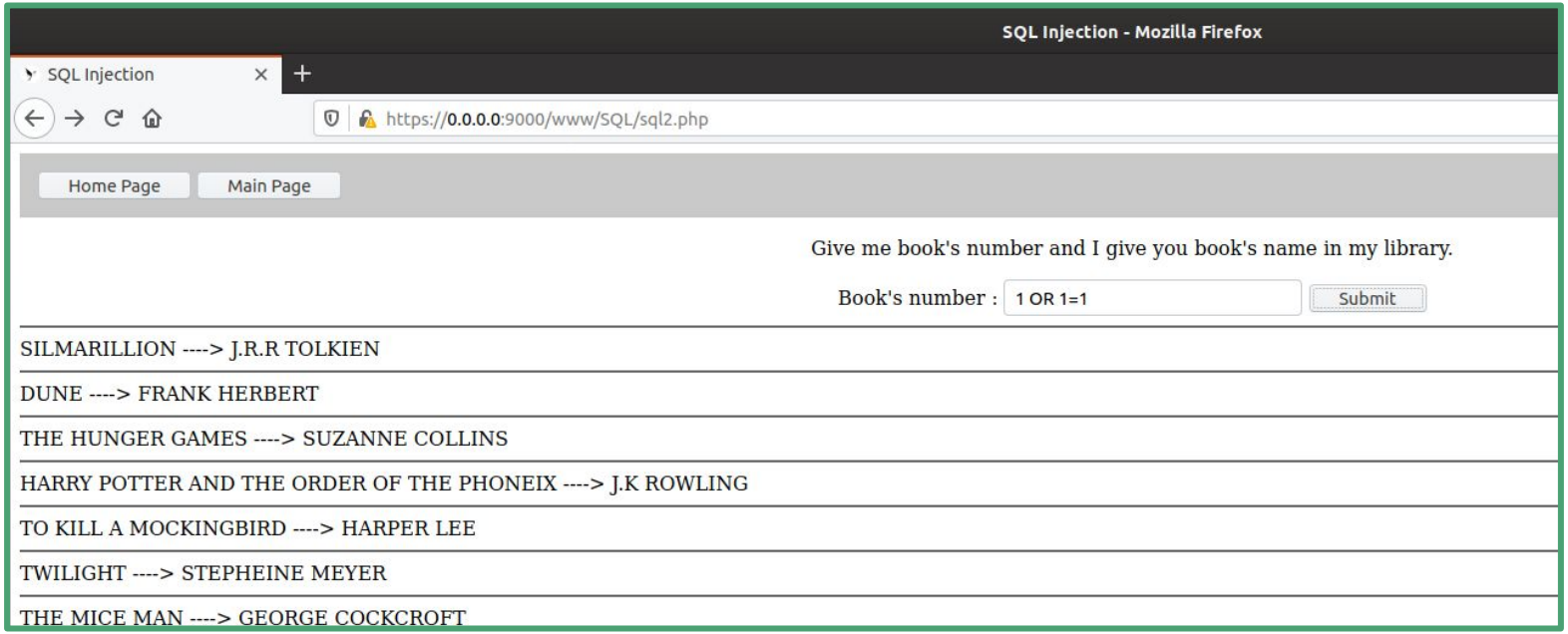
➔ Proper credentials (Not in a breach and strong enough password) will go through the WAF without any error raised

Examples in use: SQLi



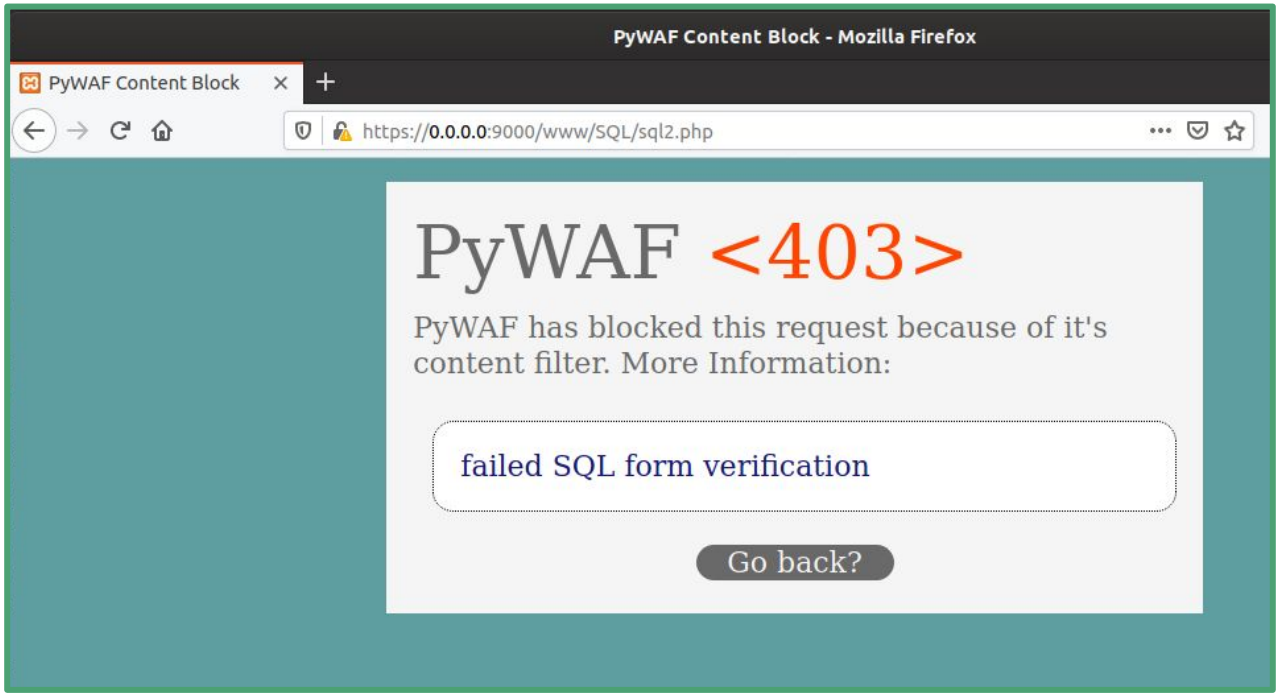
➔ A very simple SQL query form just expecting a single number

Examples in use: SQLi



➔ A successful attack on a very vulnerable form

Examples in use: SQLi



➔ The final image is the result of the same attack with PyWAF protection enabled

Test Data

- ➔ Because each attack takes on a different form, each of the various protections were tested separately, but followed similar structures
- ➔ Method:
  - ◆ Send many requests to the WAF attacking specific weaknesses in the OWASP web app
  - ◆ Input is either flagged by the WAF or allowed to pass through the the web app
- ➔ Input:
  - ◆ A collection of various forms in which the attack can take
  - ◆ A collection of non-malicious input the test as a
- ➔ Output:
  - ◆ The amount of false-negative (Not flagged malicious) and false-positive (Flagged non-malicious) cases from the input

Evaluating SQLi

- ➔ The SQLi protection was evaluated using the pycurl library to create and send requests with the form data populated with the testing payload
- ➔ The payloads were separated into groups based on malicious and non-malicious with two varieties of non-malicious payloads as shown on the upcoming slide.
- ➔ These were then run by the sql_tester script which formatted and sent the requests to the web server through the WAF.
- ➔ Any unexpected responses (False negatives/positives) were then recorded

sql_tester.py snippet:

```
c = pycurl.Curl()
c.setopt(c.URL, 'https://localhost:9000/www/SQL/sql3.php')
fw = open(‘output.txt’, ‘w+’)
with open(‘input.txt') as f:
   for line in f:
       post_data = {'number': f'{line}', ‘submit’: ‘Submit’}
       postfields = urlencode(post_data)
       c.setopt(pycurl.SSL_VERIFYPEER, 0)
       c.setopt(c.POSTFIELDS, postfields)
       c.perform()
       # Unexpected HTTP response code, e.g. 200.
       if c.getinfo(c.RESPONSE_CODE) == 200:
           fw.write(line.strip() + ‘ | RESULT: ‘ + str(c.getinfo(c.RESPONSE_CODE)) + "\r\n")
```

Results of Evaluation

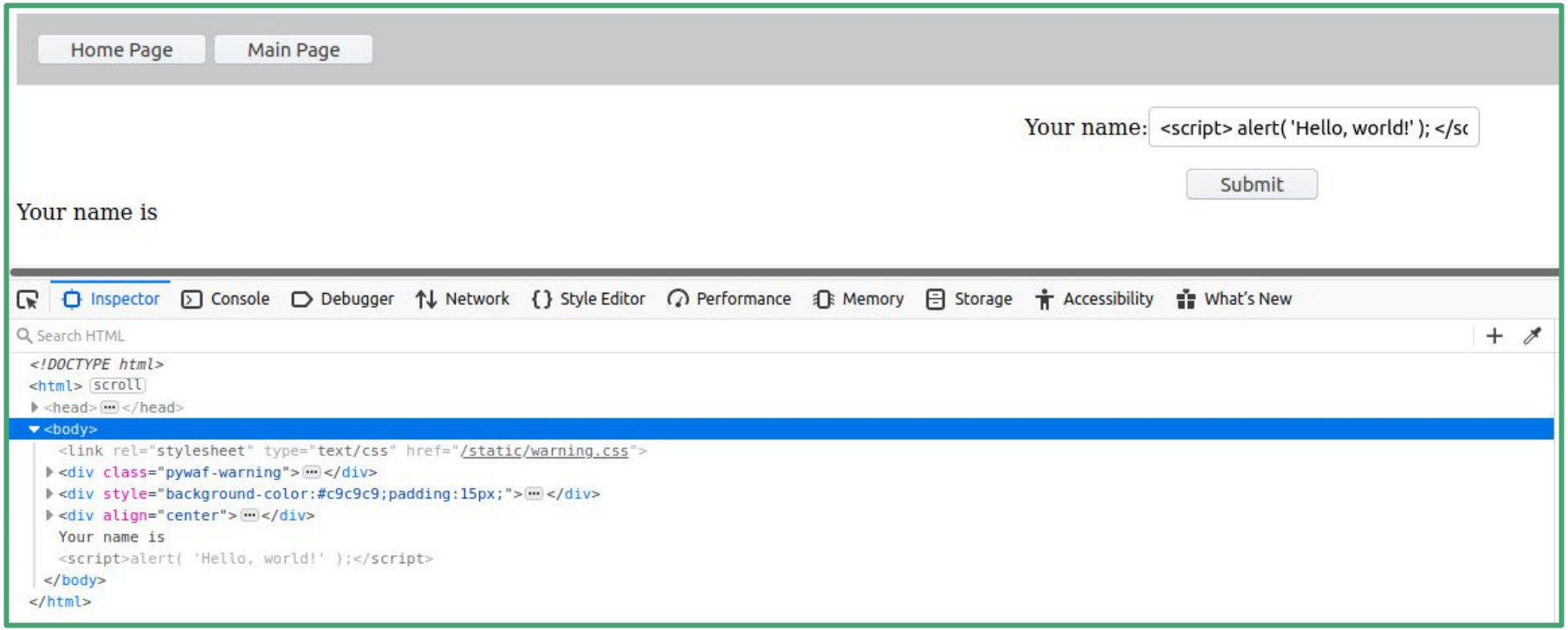
➔ SQL Injection:

Data	Flagged by PyWAF	Unflagged by PyWAF
Attacks (800 samples of injections)	798	2
Letter/Number input (10,010 non-malicious inputs)	0	10,010
Password input (Special characters) (1,500 from rockyou.txt)	6	1,494

Explaining Results

- Any form in which the SQLi protection is enabled on will have its input sent through the SQLi check
- The check matches multiple variations of injection attacks to the input
- If any match is found, the input is flagged
- So what about the missed cases and false-positives?
  - (Special character handling)
  - More on this later

Examples in use: XSS

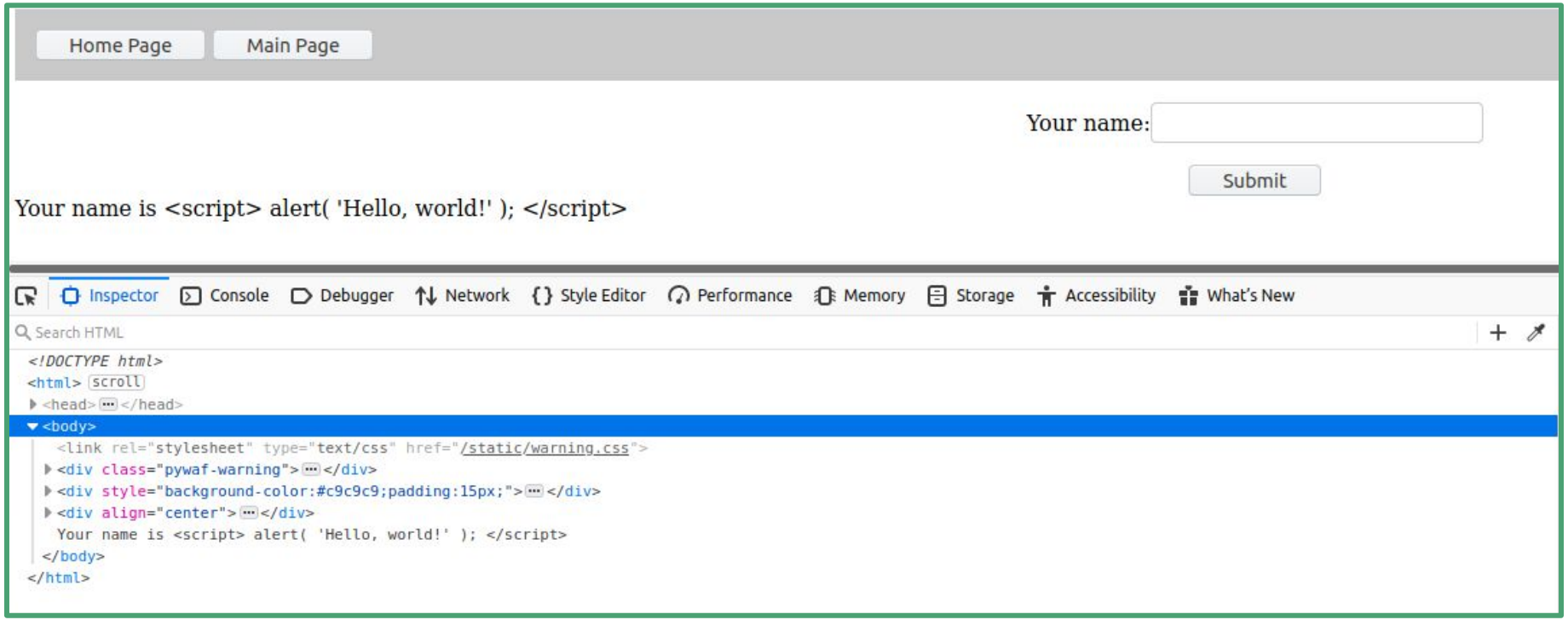


➔ A very simple scripting attack on our vulnerable web server

Examples in use: XSS

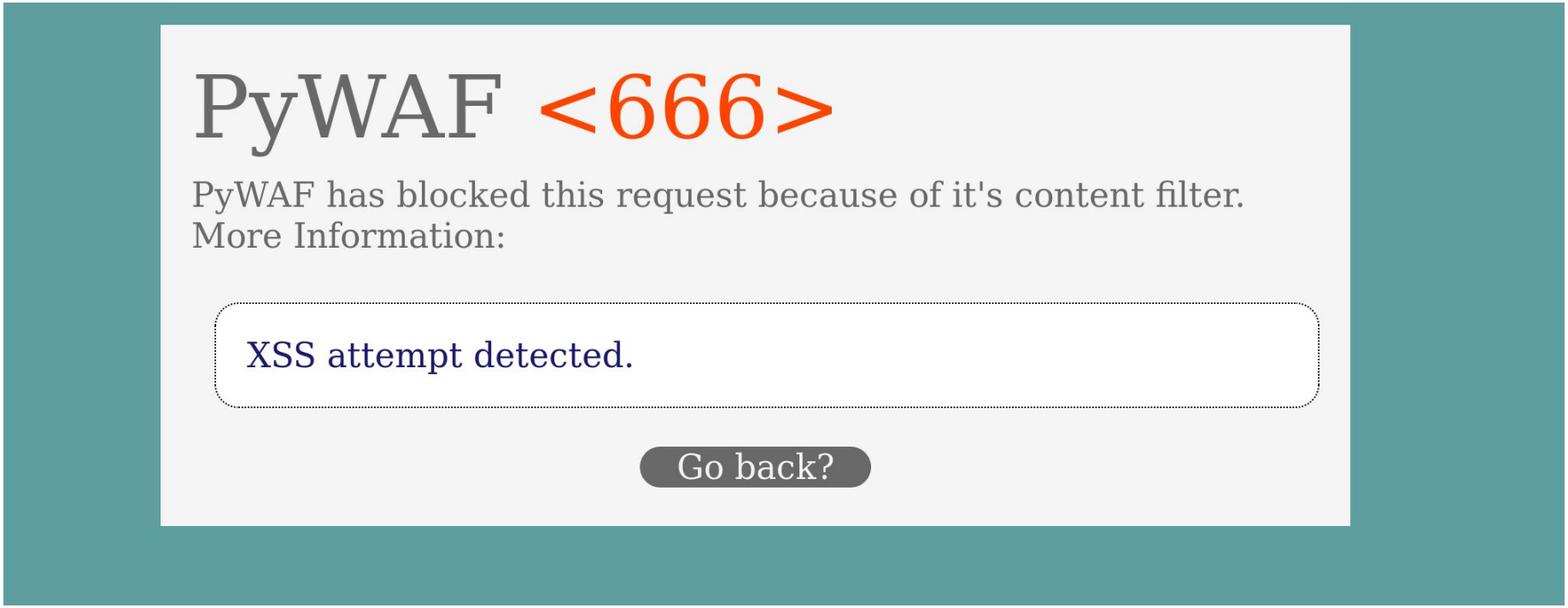
➔ The injected HTML code is executed when submitted without the protection from the WAF enabled

Examples in use: XSS



➔ Mitigated XSS keeps the original text, but in a HTML safe encoding

Examples in use: XSS



➔ The blocked attack serves an error page to the user

Evaluating XSS

- ➔ To test the XSS protection, we used a tool called XSStrike to generate and run various XSS attacks against our vulnerable web server
  - There were 3 different variations of tests run:
    - No WAF intervention
    - Mitigation: The input is filtered but still server to the web server
    - Blocking: Any detected malicious input causes the request to be blocked
- ➔ The XSStrike test suite matches these variations with the terms [passed], [filtered], and [blocked] for their respective categories.
  - Run various iterations of each test shown

XSS Results:

WAF OFF	WAF ON (Mitigation)	WAF ON (Blocking)
Fuzzing parameter: username	Fuzzing parameter: username	Fuzzing parameter: username
[passed] <test	[filtered] <test	[blocked] <test
[passed] <test//	[filtered] <test//	[blocked] <test//
[passed] <test>	[filtered] <test>	[blocked] <test>
[passed] <test x>	[filtered] <test x>	[blocked] <test x>
[passed] <test x=y	[filtered] <test x=y	[blocked] <test x=y
[passed] <test x=y//	[filtered] <test x=y//	[blocked] <test x=y//
[passed] <test/oNxX=yYy//	[filtered] <test/oNxX=yYy//	[blocked] <test/oNxX=yYy//
[passed] <test oNxX=yYy>	[filtered] <test oNxX=yYy>	[blocked] <test oNxX=yYy>
[passed] <test onload=x	[filtered] <test onload=x	[blocked] <test onload=x
[passed] <test/o%00nload=x	[filtered] <test/o%00nload=x	[blocked] <test/o%00nload=x
[passed] <test sRc=xxx	[filtered] <test sRc=xxx	[blocked] <test sRc=xxx
[passed] <test data=asa	[filtered] <test data=asa	[blocked] <test data=asa
[passed] <test data=javascript:asa	[filtered] <test data=javascript:asa	[blocked] <test data=javascript:asa
[passed] <svg x=y>	[filtered] <svg x=y>	[blocked] <svg x=y>
[passed] <details x=y//	[filtered] <details x=y//	[blocked] <details x=y//
[passed] <a href=x//	[filtered] <a href=x//	[blocked] <a href=x//
[passed] <emBed x=y>	[filtered] <emBed x=y>	[blocked] <emBed x=y>
[passed] <object x=y//	[filtered] <object x=y//	[blocked] <object x=y//
[passed] <bGsOund sRc=x>	[filtered] <bGsOund sRc=x>	[blocked] <bGsOund sRc=x>
[passed] <iSinDEx x=y//	[filtered] <iSinDEx x=y//	[blocked] <iSinDEx x=y//
[passed] <aUdio x=y>	[filtered] <aUdio x=y>	[blocked] <aUdio x=y>
[passed] <script x=y>	[filtered] <script x=y>	[blocked] <script x=y>
[passed] <script//src=//	[filtered] <script//src=//	[blocked] <script//src=//
[passed] ">payload<br/attr="	[filtered] ">payload<br/attr="	[blocked] ">payload<br/attr="
[passed] "-confirm``-"	[filtered] "-confirm``-"	[blocked] "-confirm``-"
[passed] <test ONdBlcLicK=x>	[filtered] <test ONdBlcLicK=x>	[blocked] <test ONdBlcLicK=x>
[passed] <test/oNcoNTeXtMenU=x>	[filtered] <test/oNcoNTeXtMenU=x>	[blocked] <test/oNcoNTeXtMenU=x>
[passed] <test OndRAgOvEr=x>	[filtered] <test OndRAgOvEr=x>	[blocked] <test OndRAgOvEr=x>

Explaining Results

- Our XSS filtering is based on checking for any HTML code from possible user input
  - URL's, Forms, etc.
- We check for code by converting the input to all HTML safe characters
- If the input differs from the conversion, we flag it

Reflection: Areas for improvement

- ➔ Configuration:
  - ◆ Some protections/features cannot be applied across all forms or pages, or should be applied differently in different cases. Having some program to automatically crawl all pages of the web server, find vulnerable components, and create a configurable dataset to apply protections onto would be a nice tool to have, but was too large and could stand as its own project.
    - The special character case for SQLi would also benefit by allowing different character-sets for different forms
- ➔ False Positives
  - ◆ The web apps we used for testing were **extremely** insecure. So in order to protect them we had to have the highest level of scrutiny when checking input. This leads to some possibilities for false positives. Using some system like what is mentioned above, it would be nice to be able to configure the security on a situational basis (More than just filter or block)

References:

$ **Web Application Firewall Evaluation Criteria (Web Application Security Consortium)**

> http://projects.webappsec.org/w/page/13246985/Web%20Application%20Firewall%20Evaluation%20Criteria

$ **OWASP Vulnerable Web Application Project**

> https://github.com/OWASP/Vulnerable-Web-Application

$ **XSStrike**

> https://github.com/s0md3v/XSStrike

$ **DVWA (Damn Vulnerable Web App)**

> http://dvwa.co.uk/ (Unironically using HTTP)