WAFFle: Fingerprinting Filter Rules of Web Application Firewalls

Isabell Schmitt, Sebastian Schinzel*

Friedrich-Alexander Universität Erlangen-Nürnberg
Lehrstuhl für Informatik I
IT-Sicherheitsinfrastrukturen

Email: sebastian.schinzel@cs.fau.de
Twitter: @seecurity

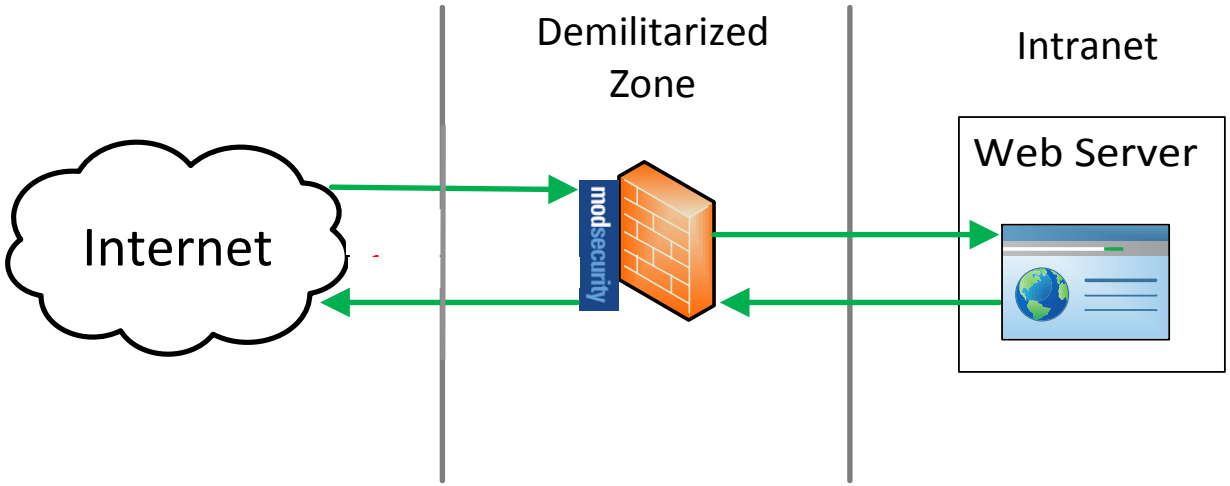
FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

*supported by Deutsche Forschungsgemeinschaft (DFG) as part of SPP 1496 "Reliably Secure Software Systems"

USIFES

Web Application Firewalls

- intercept web requests
- filter requests to prevent attacks
- uses filter rules for detecting common attack patterns
- blind for "new" attack patterns



- If attacker knows active filter rule set, he can search for loopholes in the rule set
- ***What can the attacker learn about the active filter rule set of a WAF?***

Introduction: Related Work

How can the attacker learn active filter rule set?

- **WAFW00f** detects if a web page is protected by a WAF and can differentiate between 22 different WAF producers (no active rule set)
  - analyses HTTP status codes, cookies, etc.
- **WAF Tester** fingerprints WAF filter rules by analyzing the HTTP status codes and whether the WAF drops or rejects the HTTP request on the TCP layer
  - analyses error conditions
  - no visible error condition == no fingerprinting possible

Visibility of Web Application Firewalls

1. Response shows **WAF error message**

a) the rogue request was blocked by the WAF or

b) the WAF passed the request to the web application that responded with an error message and which was then cloaked by the WAF

2. Response shows **Webapp error message**

a) WAF neither blocked the request, nor cloaked the web application's error message

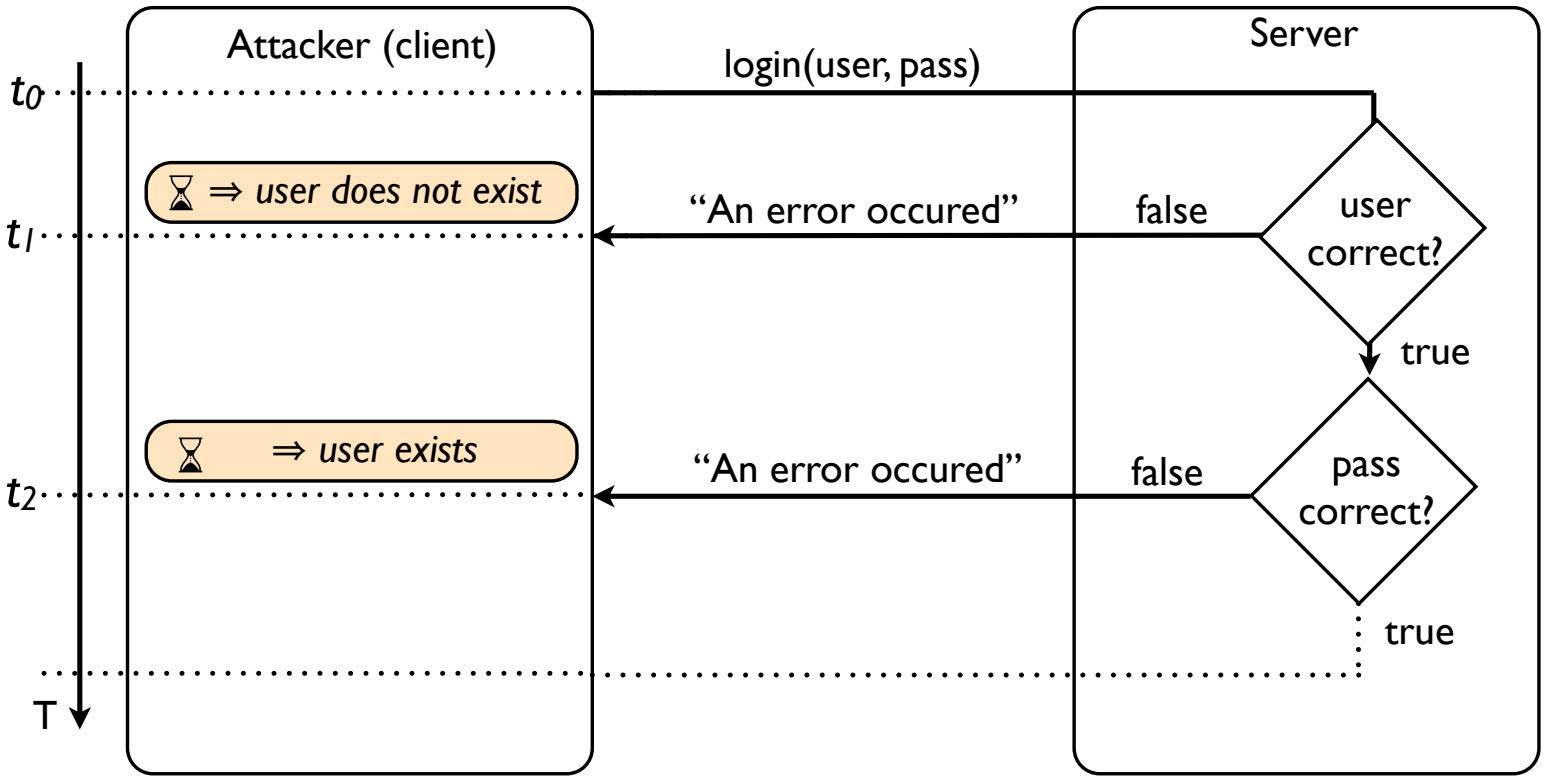
3. Response shows **Normal response**

a) WAF removed the malicious part of the rogue request

b) WAF passed the rogue request but webapp ignored the malicious part of the request

c) WAF passed the rogue request and the malicious part was executed, but it produced no visible result

Introduction to Timing Side Channel Attacks



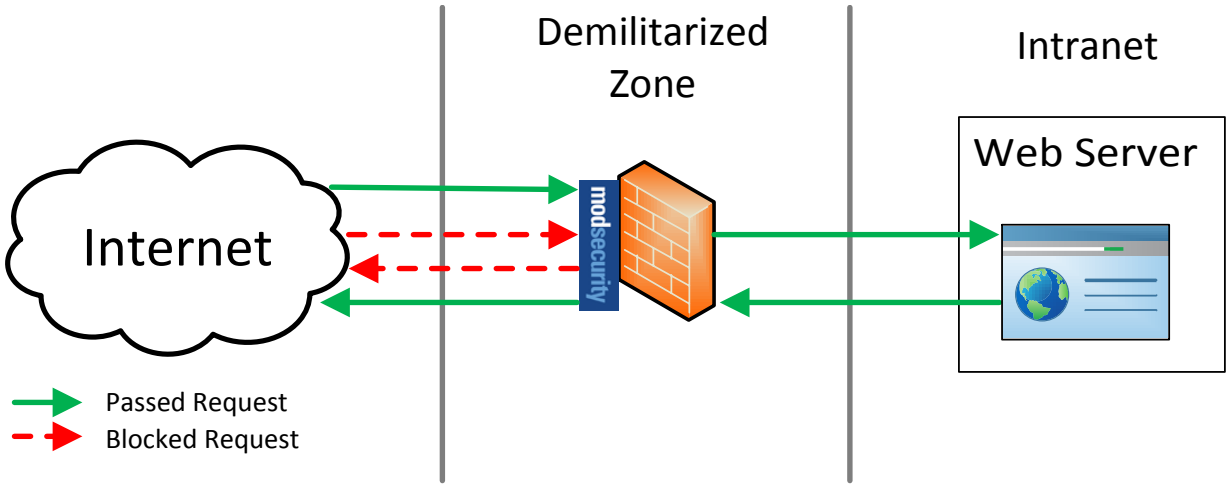
Other examples for side channels:

- sound
- visuals
- emissions
- power consumption
- motion (mobiles)
- size of encrypted packages

1. Scenario: mod_security as Reverse Proxy

mod_security filtering on reverse proxy

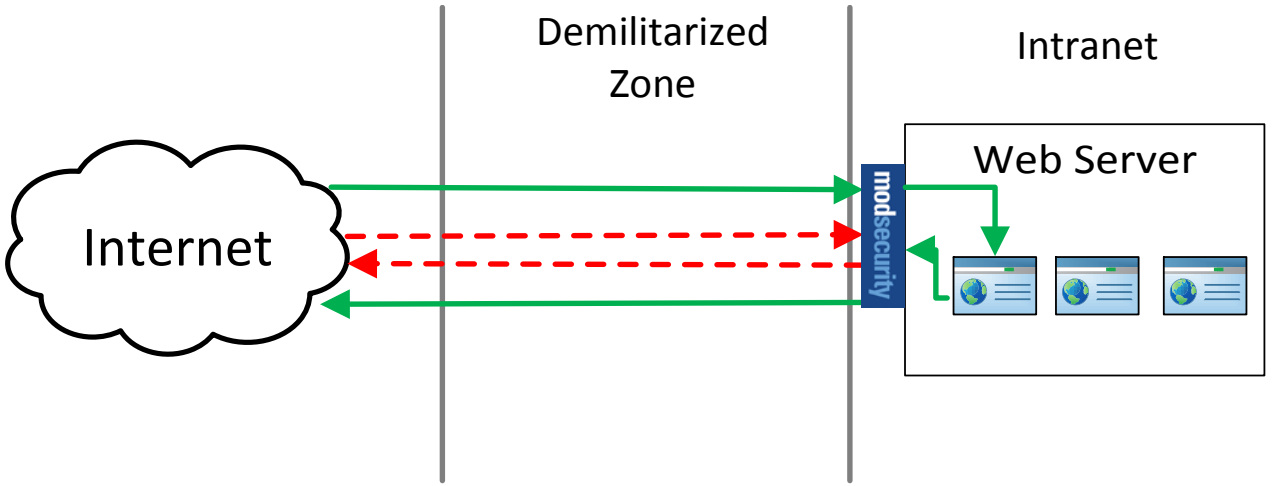
- Request gets passed to Web server iff request is not blocked by filter set
- Blocked requests are never passed to Web server



2. Scenario: mod_proxy as Web Server Plugin

mod_security filtering as web server plugin

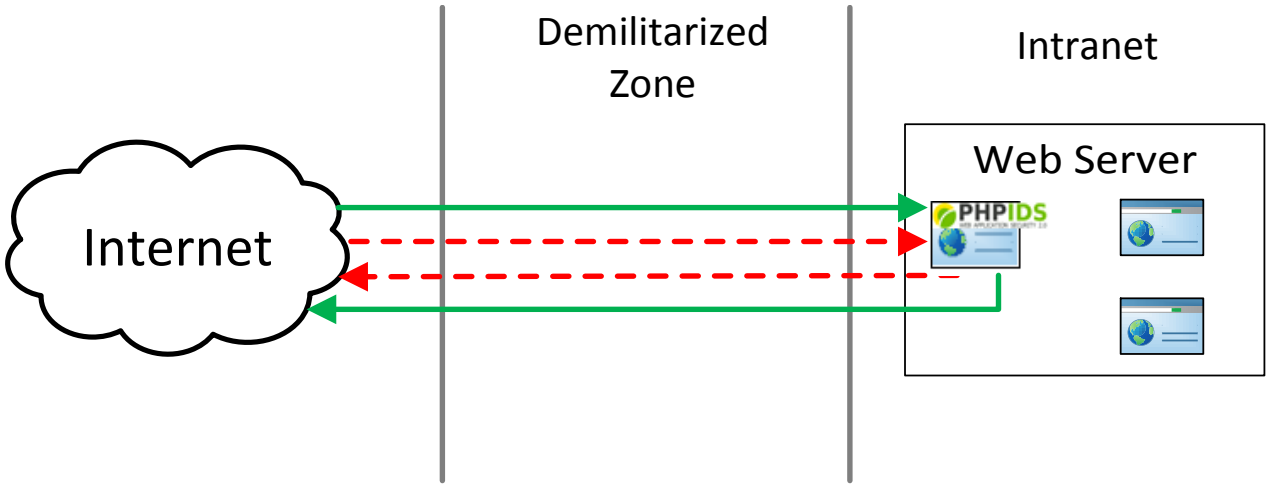
- Request gets passed to Web application iff request is not blocked by filter set
- Blocked requests are never passed to Web application



3. Scenario: PHPIDS as Programming Library

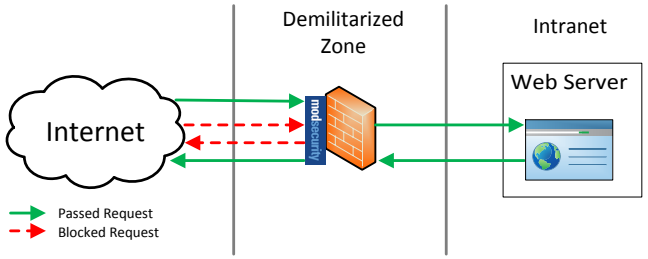
mod_security filtering as web application plugin

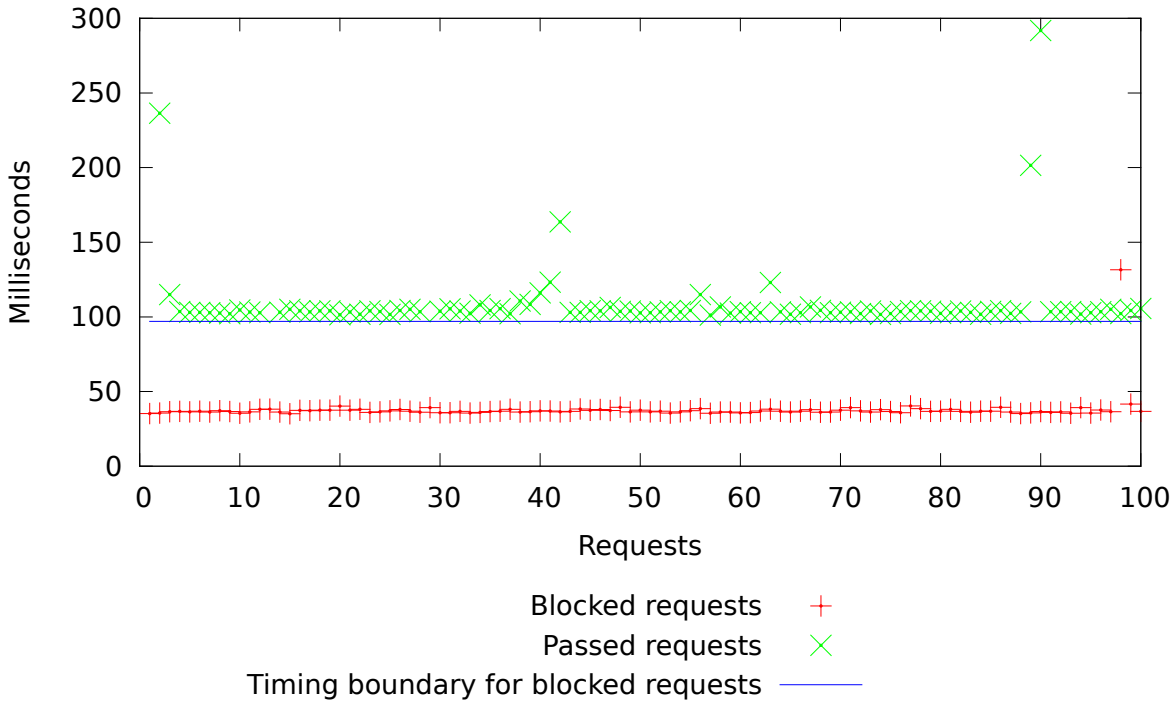
- Request gets passed to business logic iff request is not blocked by filter set
- Blocked requests are never passed to business application



Idea behind WAFfle

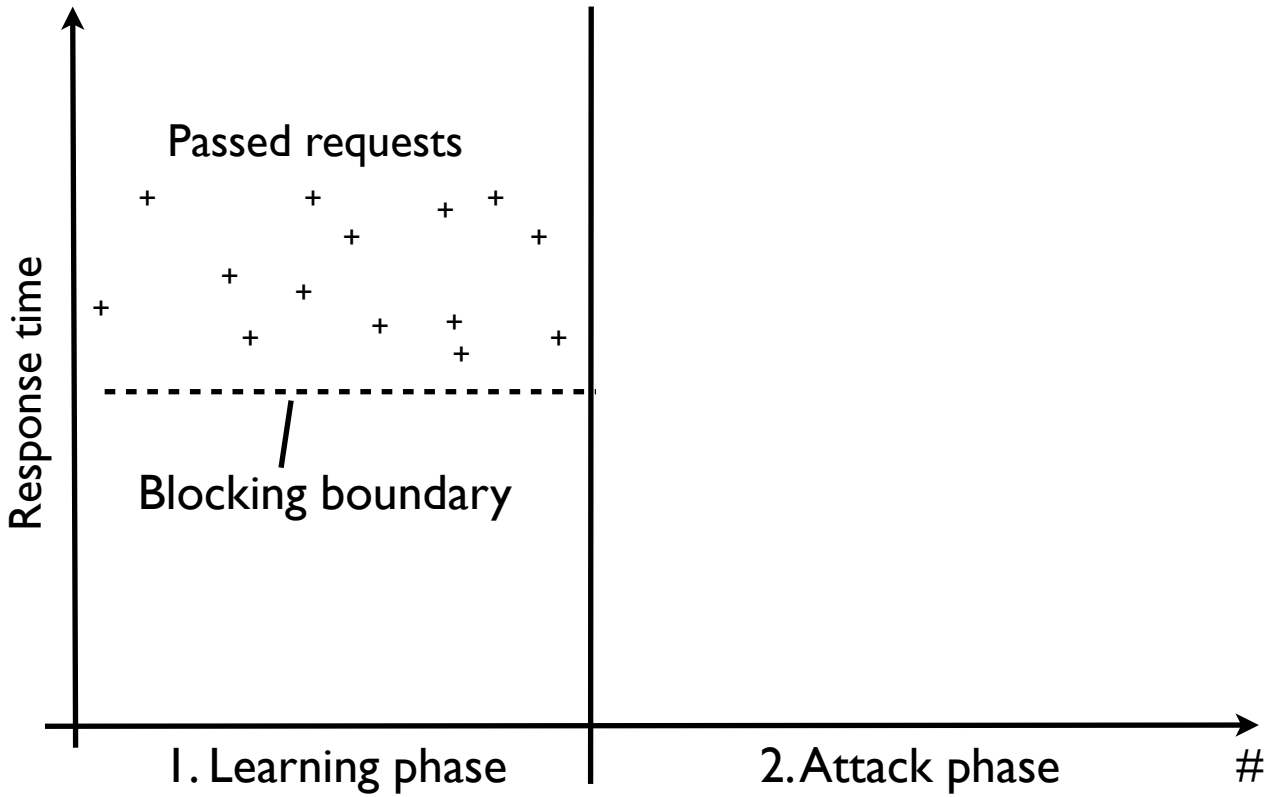
1. Generate polymorphic representations of exploit code (e.g.
   <script>alert(23);</script>,
   <script_>alert(23);</script>,
   <script__>alert(23);</script>
   <script___>alert(23);</script>)
2. Send to web app and measure response time
3. Analyse response time



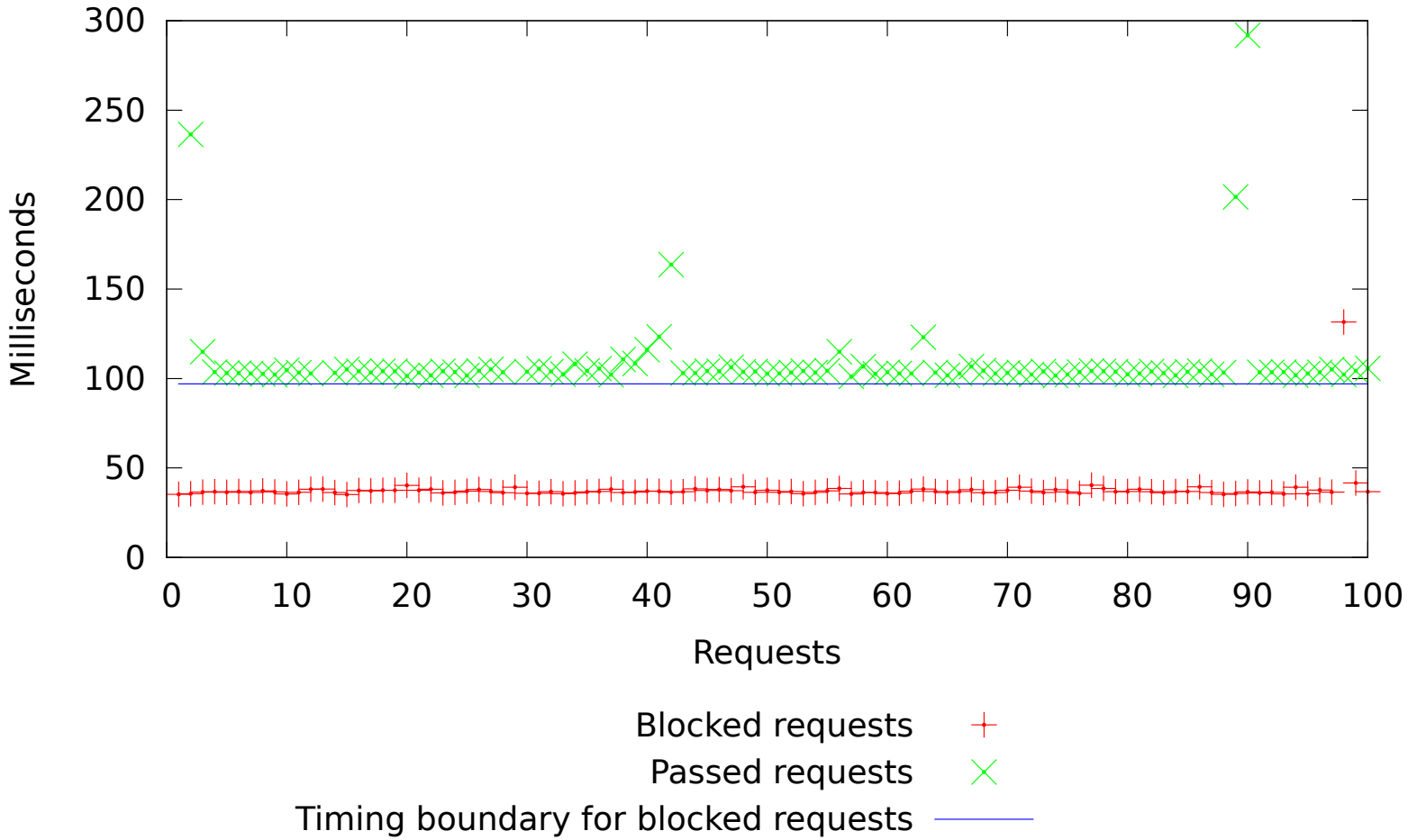


Two phases of WAFle

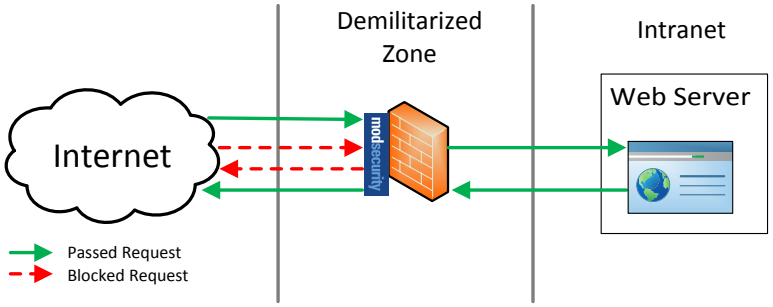
1. Learning phase
   a) measure response times *T* of *n* passed requests
   b) define "blocking boundary" as $b = min(T) - \varepsilon$
2. Attack phase
   a) send probe and measure *t*
   b) blocked request if $t < b$



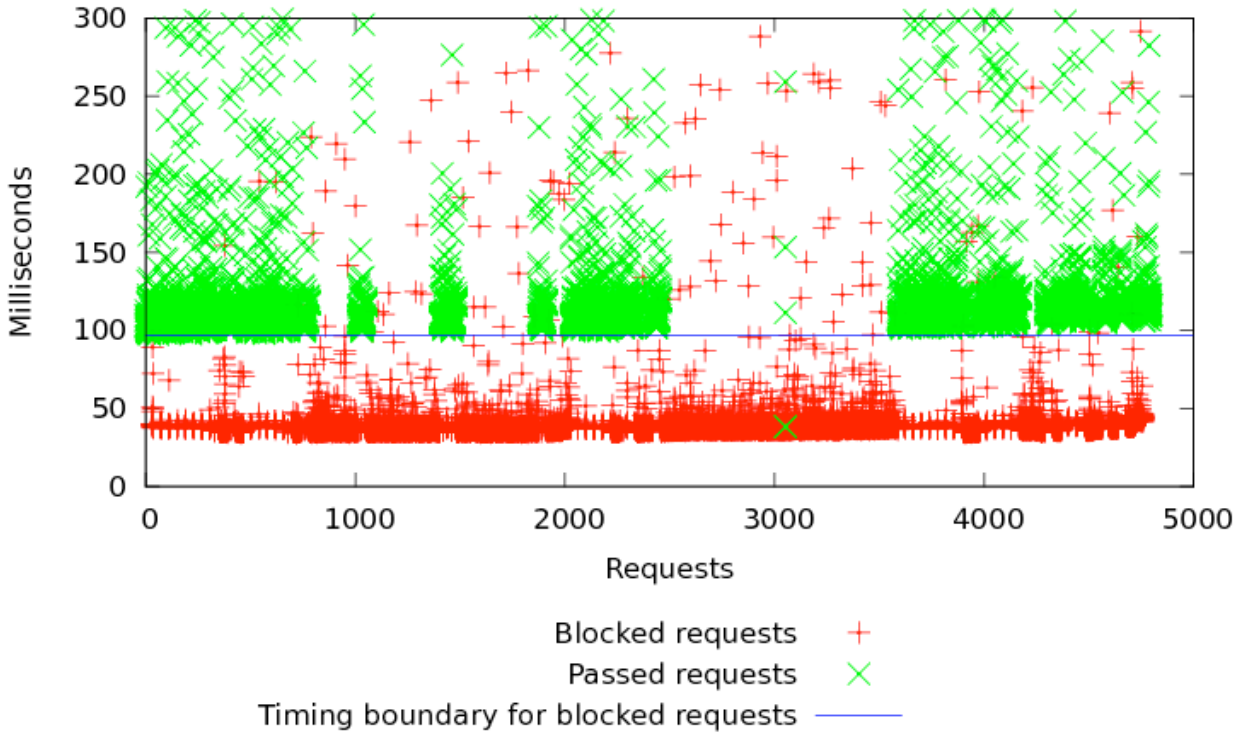
WAFfle: Fingerprinting Filter Rules of Web Application Firewalls



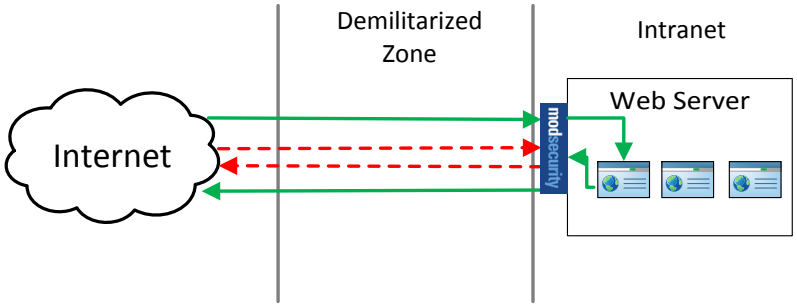
WAFfle: Results



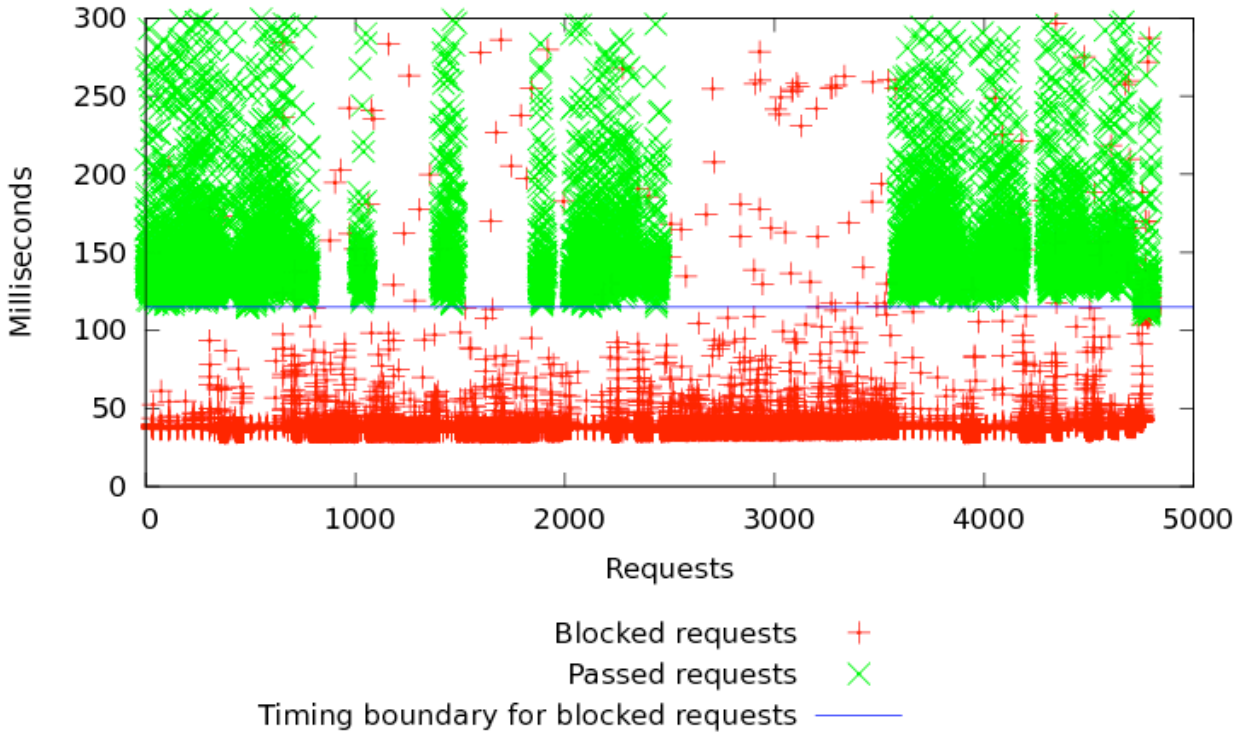
mod_security filtering on reverse proxy



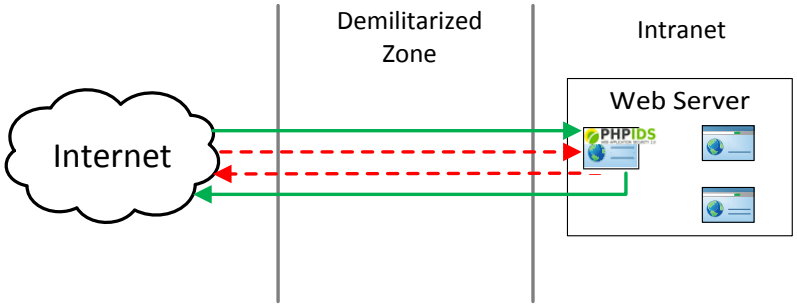
WAFfle: Results



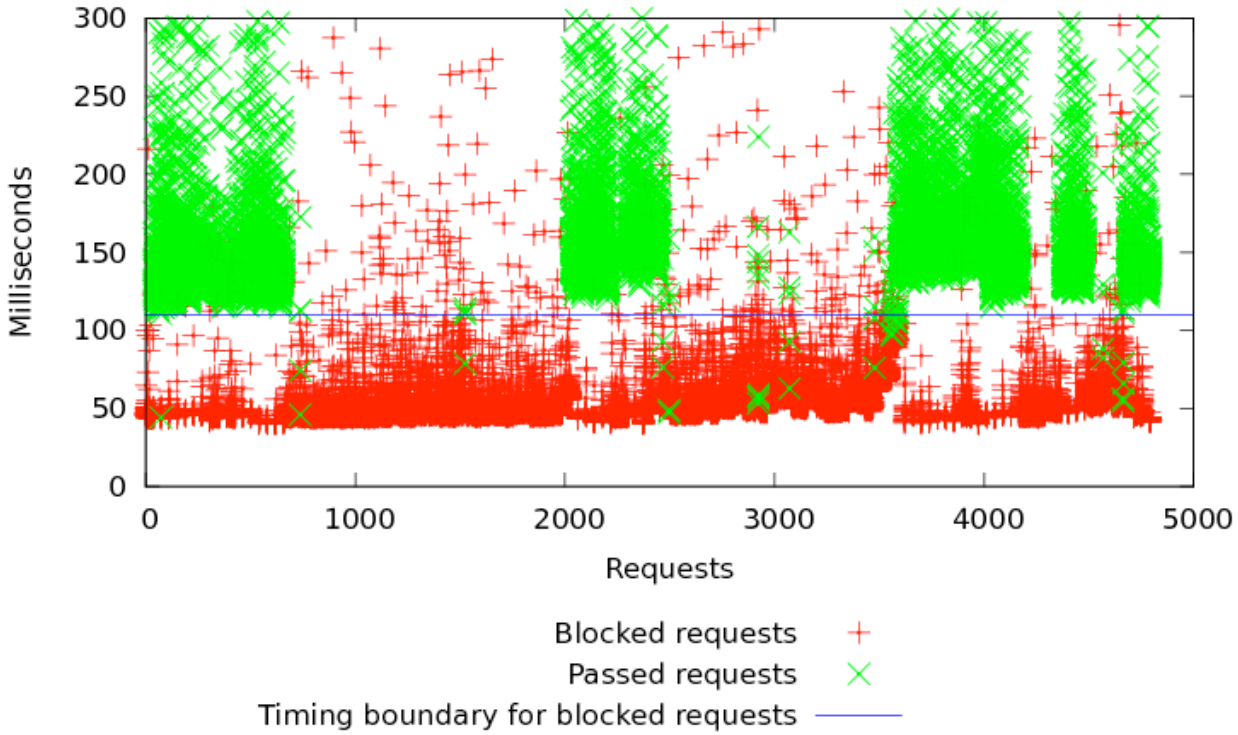
mod_security filtering as web server plugin



WAFfle: Results



mod_security filtering as web application plugin



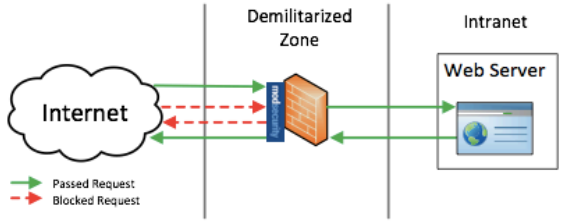
WAFfle: Results

Results

- All three scenarios allow to distinguish blocked from passed requests by observing response times
- With no repetitions, >95% of single requests already correctly determine blocked and passed requests

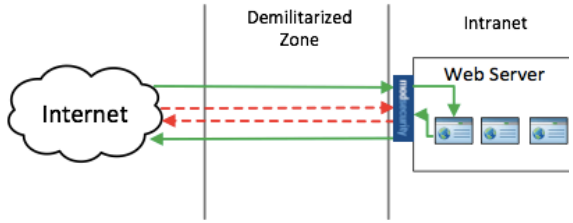
WAF topology	Figure	Timing difference	Correct
Standalone	1(a)	62.63 ms	95.2 %
Web server plugin	1(b)	81.86 ms	95.4 %
Programming library	1(c)	48.22 ms	96.3 %

1(a)



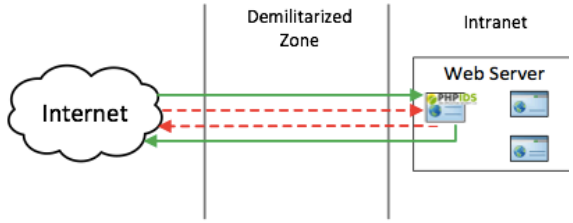
(a) Standalone WAF

1(b)



(b) WAF as web server plugin

1(c)

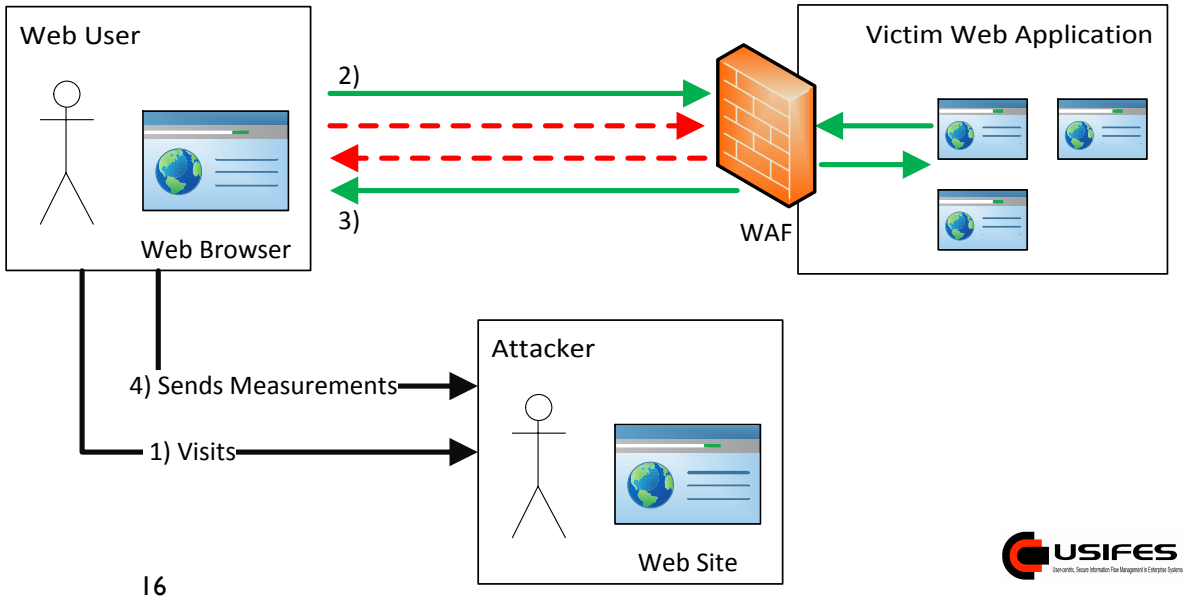


(c) WAF as programming library

WAFfle: Cross Site Timing Attack

One more thing...

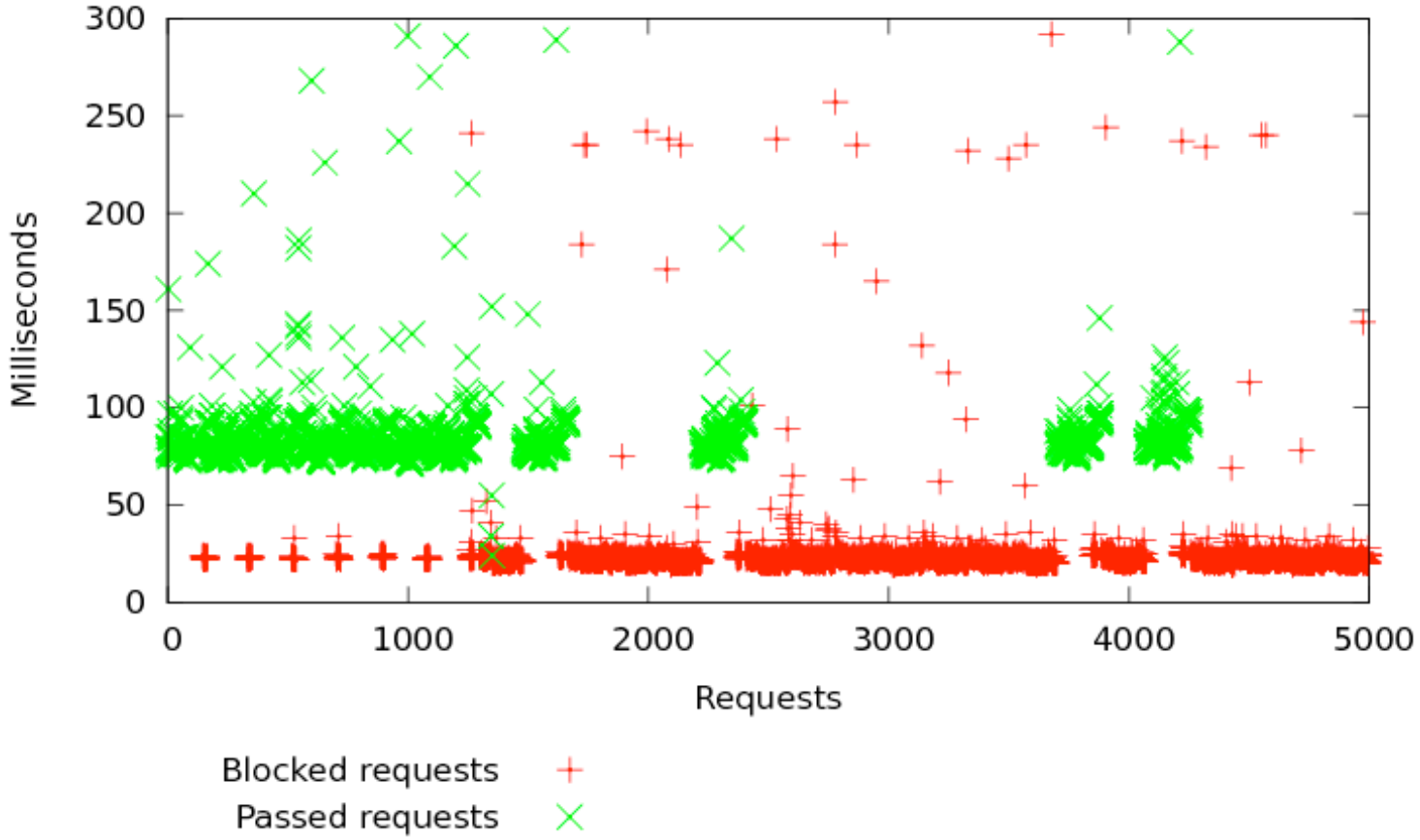
- We're on the web, and the web allows cross site requests
- Extend WAFfle for Cross Site Request Forgery (Cross Site Timing Attack)
- Generate Javascript code that attacker embeds on web page
- Attacker tricks other users to visit web page
- other users perform measurement and send measurements to attacker



Cross-Site Timing Attack

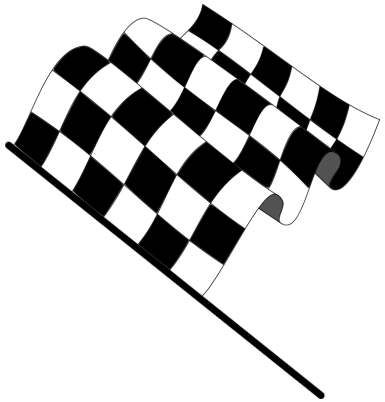
```
1 <script>
2    var time;
3    var img = document.createElement('img');
4    img.onerror = function() {
5         var end = new Date();
6         time = end - start;
7         sendResult(time); // send result to attacker
8    }
9    img.style.display = 'none';
10   document.body.appendChild(img);
11   var start = new Date();
12   img.src = "http://domain.tld/path?" + parameter
          + "=" + exploit;
13 </script>
```

Cross Site Timing Attack



Summary

- Introduced a new timing attack against WAFs that directly distinguishes passed requests from blocked requests without relying on ambiguous error messages
- Tested the attack over an Internet connection against three common WAF deployment setups and showed that the attack is highly practical
- Combined our timing attack with XSRF,
  - hides the attacker's identity
  - prevents the WAF from blocking the attack (assuming that the attacker distributes the attack to many other users)



Thanks!

Discussion.

Email: sebastian.schinzel@cs.fau.de

Twitter: @seecurity