



lmstfu

Virtual Patching: Does it work?

OWASP NZ Day | 22 Feb 2019

Kirk Jackson | @kirkj
lmstfu.com | @LetMeSecureThat

This talk is not about RedShield!

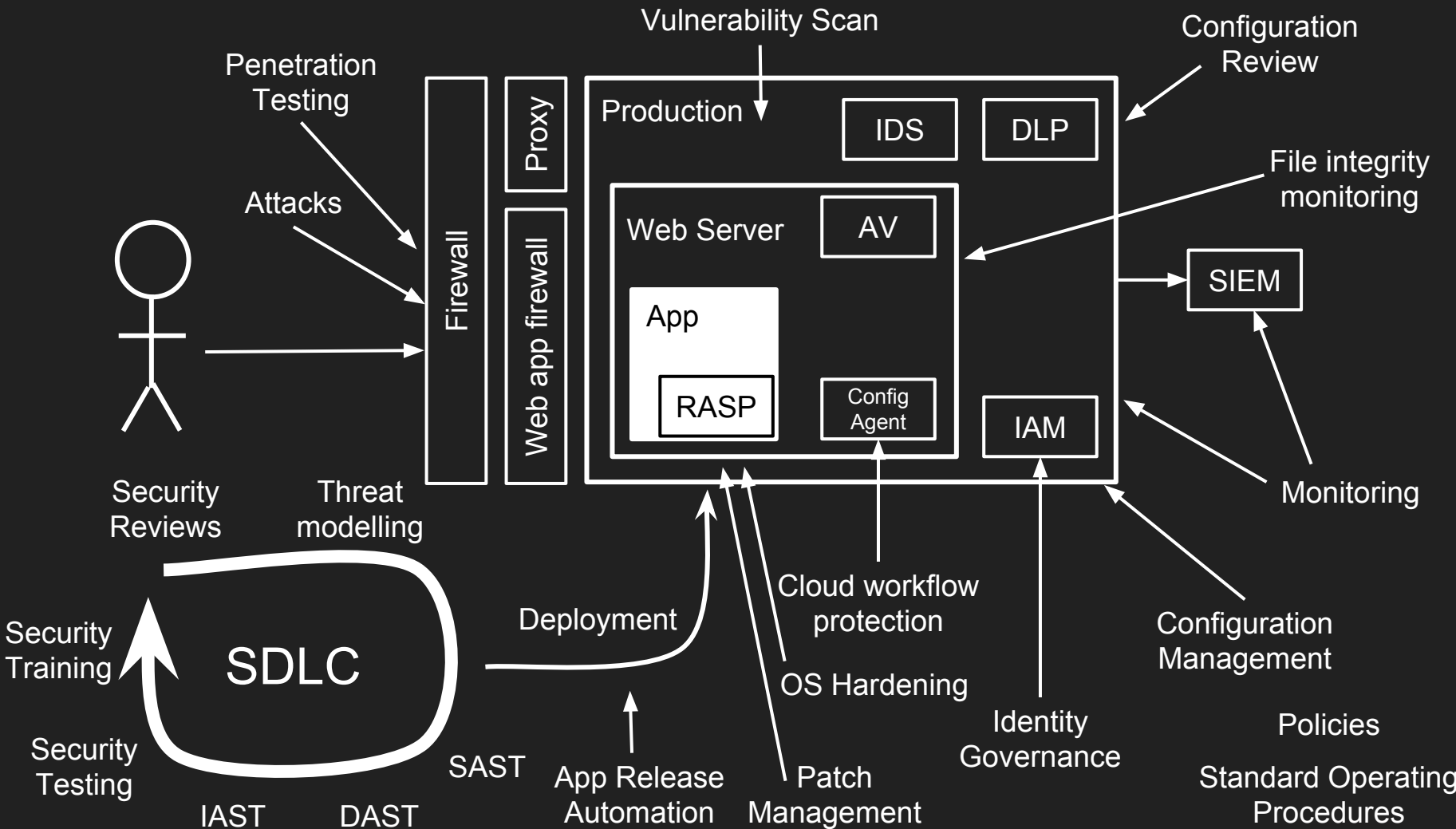
“We are the world's first web application shielding-with-a-service cybersecurity company.”

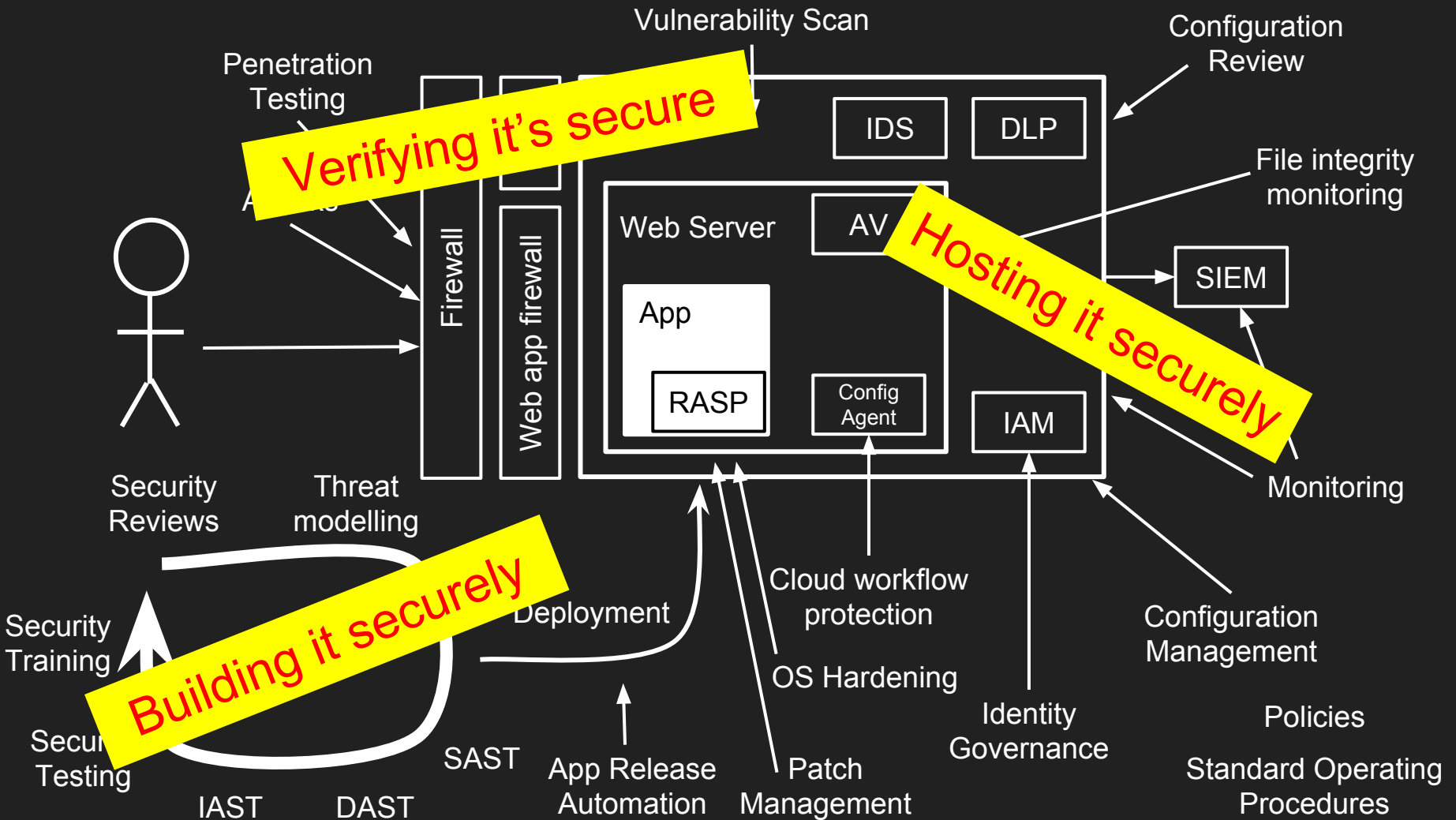
- RedShield don't use ModSecurity or node.js
- We wrap a service around virtual patching
- How does virtual patching work?
- Can it be done “DIY”?

Not an advertisement!



Building a secure web app



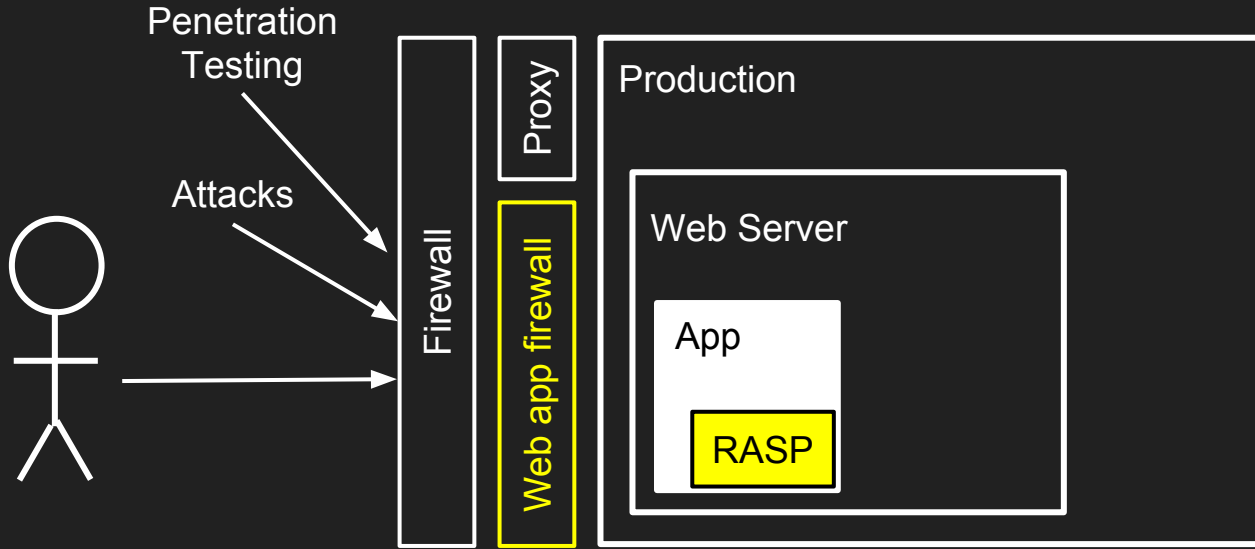


Approximate cost:
\$4.2m

Building a secure web app



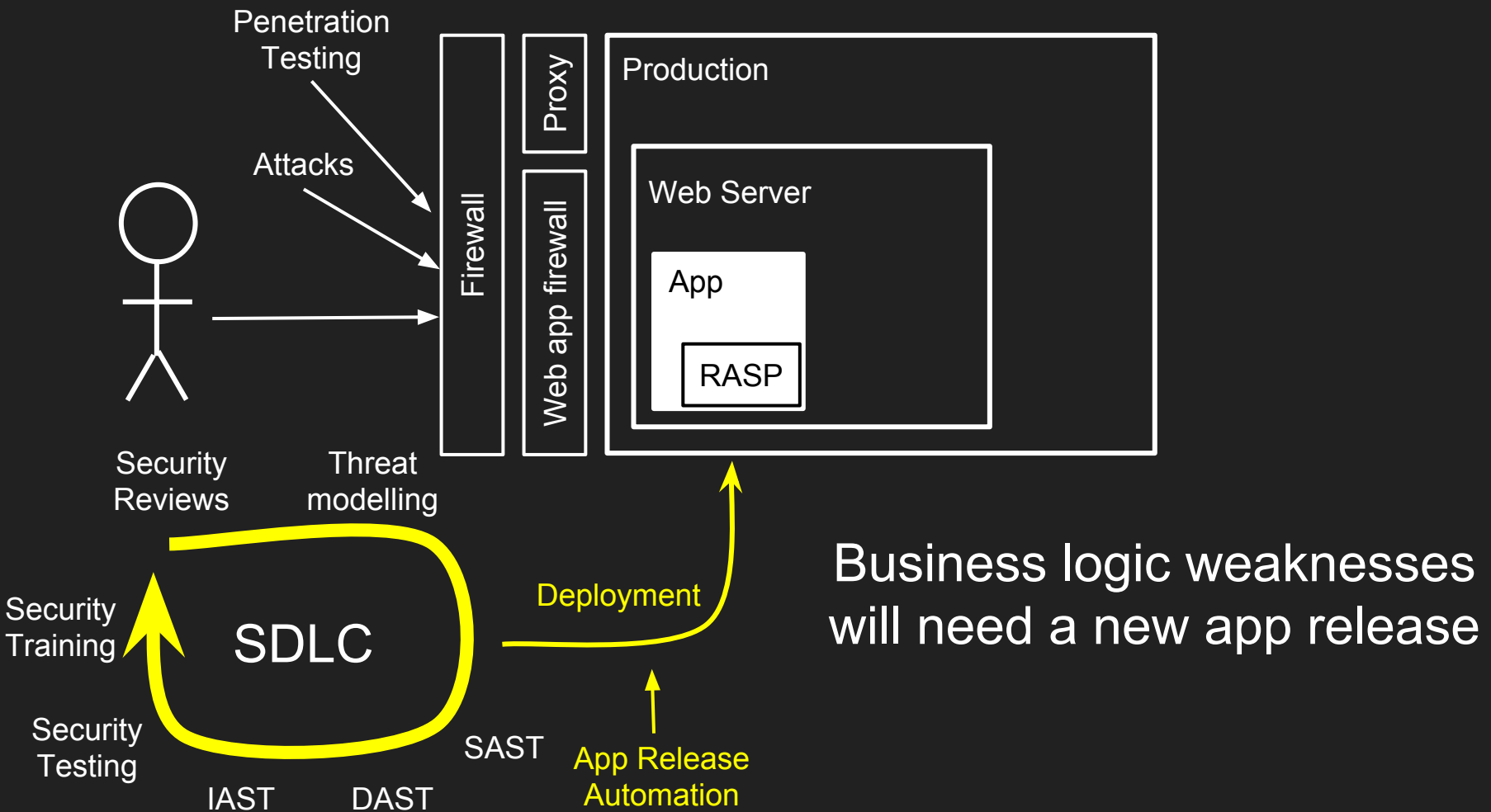
But what if there are bugs?



The WAF might stop signature-based attacks

business logic

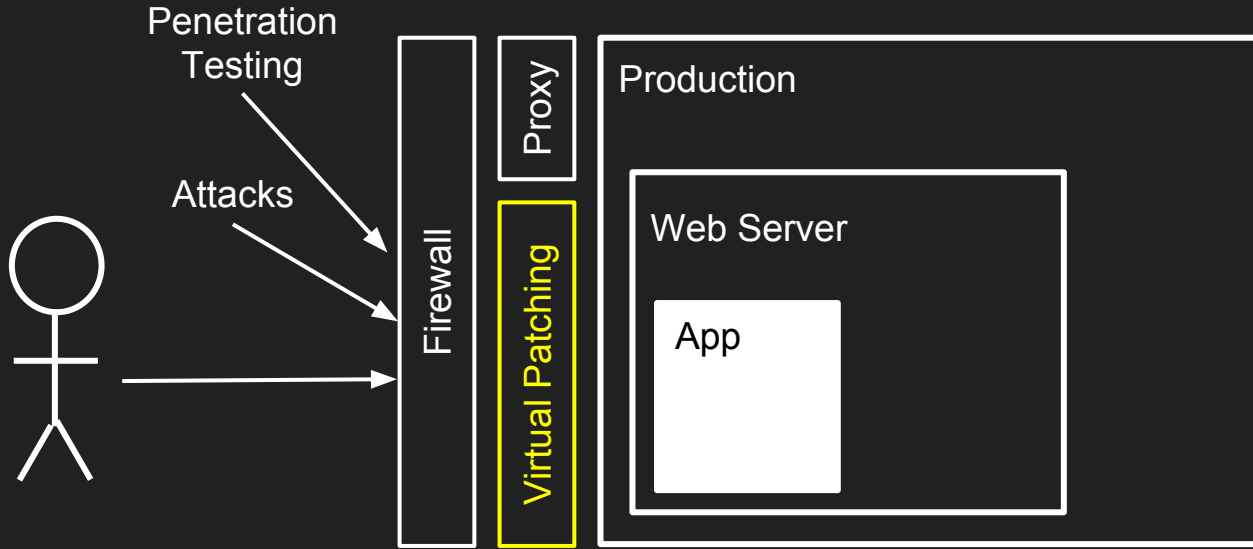
But what if there are bugs?



The cost of releasing updated software

- Divert team from current projects
 - Branch, merge, build, test
 - Release management, change review board
 - Release
-
- Timeframe from discovery to release?
 - Do you even have the source code?
 - What “process” do you need to shortcut?

Can we **patch** the issues
without touching the
underlying website?



Replace the “WAF” with a more capable layer...

“Let me secure that for you!”

Virtual Patching

*“ prevent the exploitation
of a known vulnerability ”*

Virtual Patching

An agile security approach

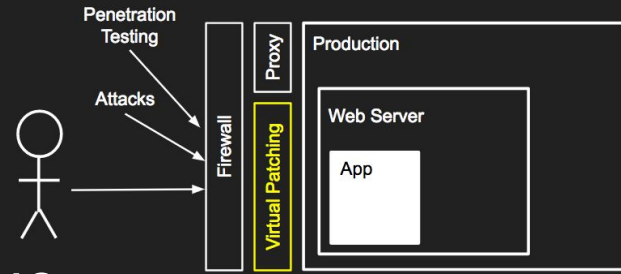
Available to security teams in lieu of software development

React quickly while waiting for the cleanup

Our Virtual Patching Approach

Understand *how to exploit* the security issue

Only patch *known vulnerabilities or weaknesses*



Avoid over-patching or doing things that will cause issues:

- Learning mode, tuning, false positives
- Large blocklists
- Focus *only* on the script, page or parameter affected

What do we need to do?

	Proxy?
Block traffic to certain urls, from IPs, countries	✓
Add headers into responses, modify cookies	✓
Detect SQLi, XSS attacks	
Replace HTML in responses	
Modify requests to neutralise attacks	
Track users and their actions	
Add CSRF protection to a page	
Track application state	
Perform role and privilege checks	

How do we react?

Choose an approach:



Alert - so you know if there's an attack

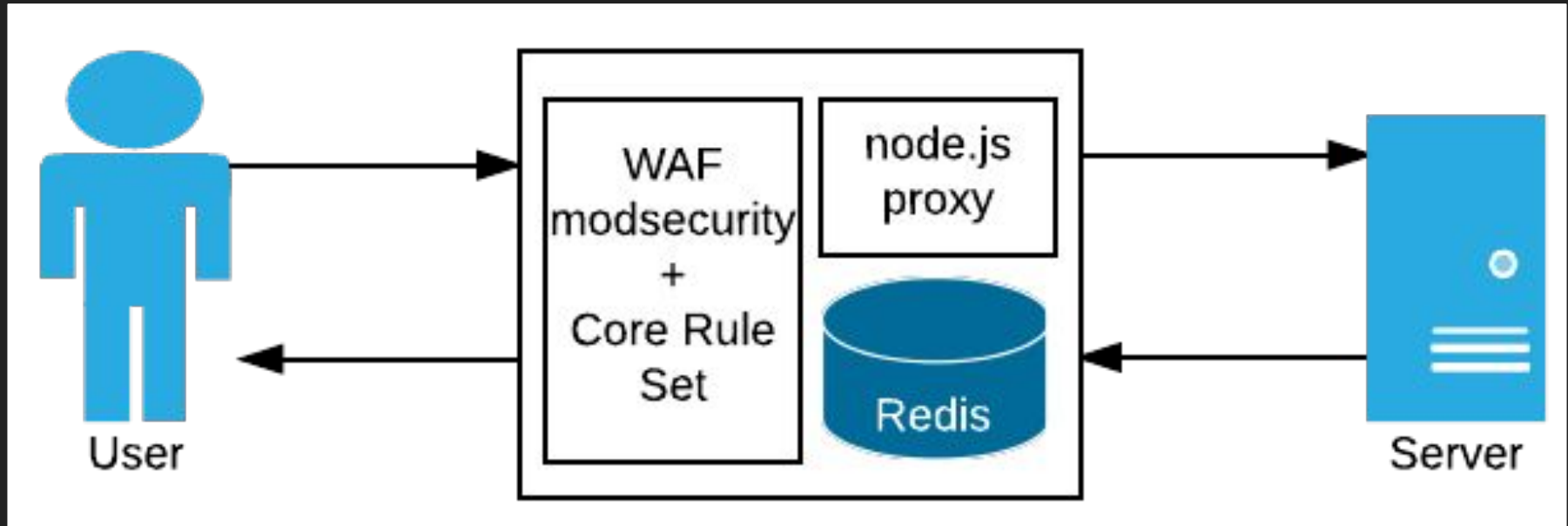
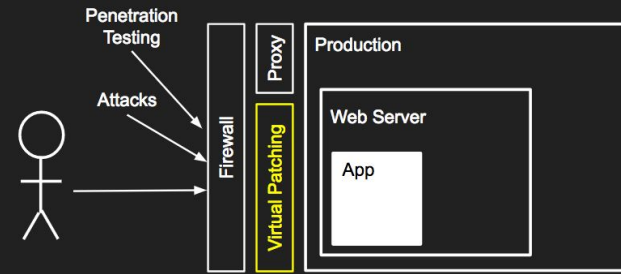
Block - if you're not worried about the user's experience

Redirect - send the user to an error page

Transform - change request or response to make safe

Validation - if you want to give helpful messages to guide users to enter correct values

“Let me secure that for you!”



WAF ALERT!!!!

Our users hate our WAF because they're blocked by false positives

“ Our developers hate our WAF because it slows them down ”

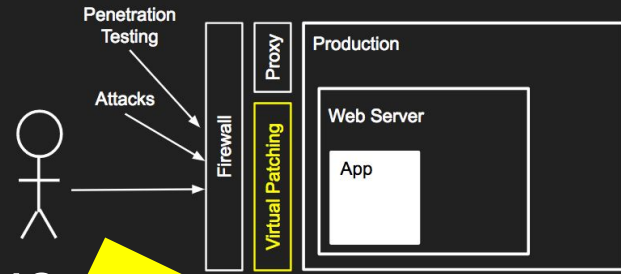
Our sysadmins hate our WAF because it requires constant tuning

Our security team hate our WAF because it doesn't block real vulnerabilities

Our Virtual Patching Approach

Understand *how to exploit* the security issue

Only patch *known vulnerabilities or weaknesses*

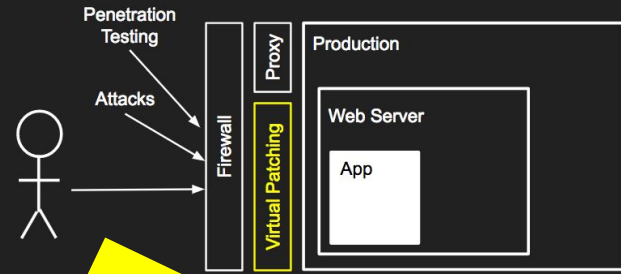


Recap

Avoid over-patching or doing things that will cause issues:

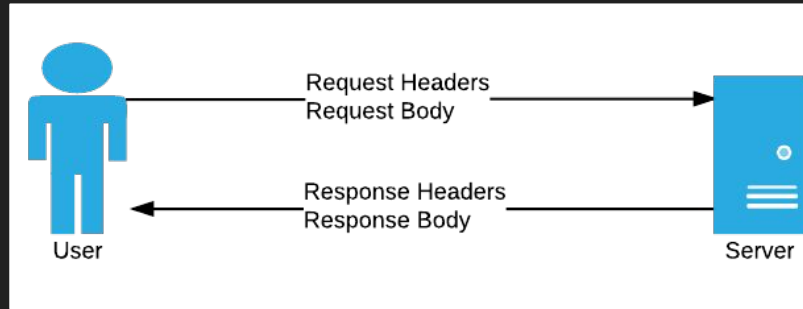
- Learning mode, tuning, false positives
- Large blocklists
- Focus *only* on the script, page or parameter affected

Our Virtual Patching Approach

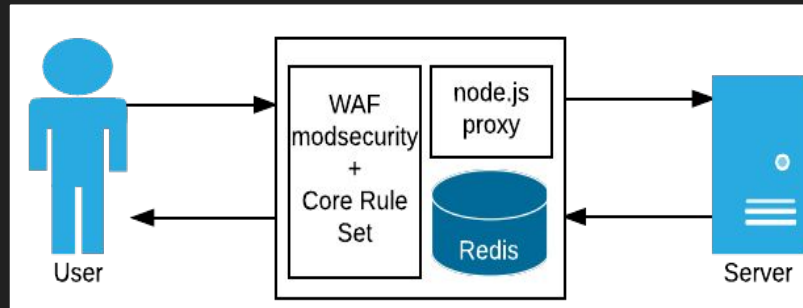


Only patch *known vulnerabilities or weaknesses*

[Pause to demo a vulnerable website]



lmstfu.dev.0-days.net:5000



<https://lmstfu.dev.0-days.net>

Our site has multiple vulnerabilities and weaknesses

OWASP Top 10:

- Access to admin urls
- Purchase negative quantities
- SQL injection in product search
- Cross-site scripting in product comments
- Cross-site request forgery in product comments

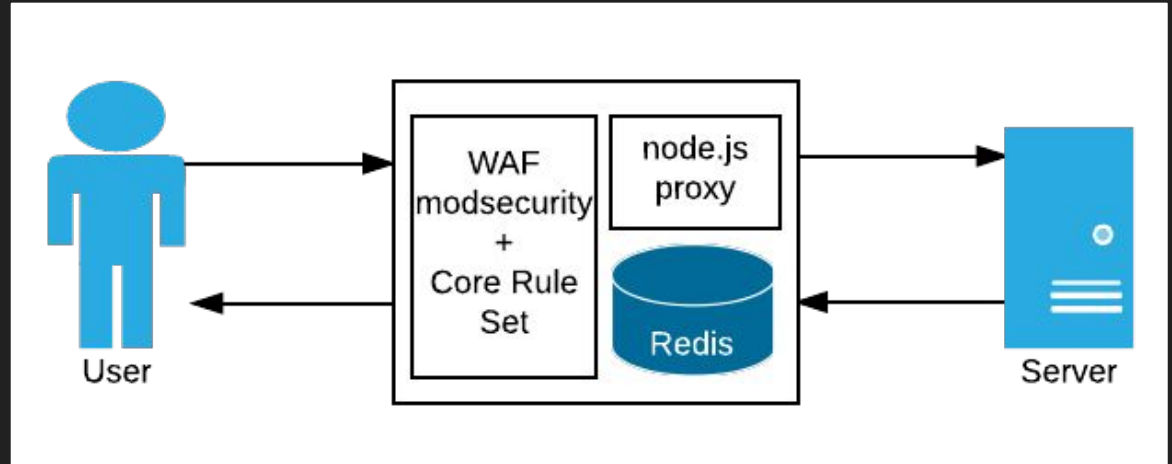
Business-logic vulnerabilities:

- Viewing other people's orders
- Jumping past the payment screen
- Disclosure of credit card numbers
- Password weaknesses

“Let me secure that for you!”

ModSecurity

- Open-source web app firewall



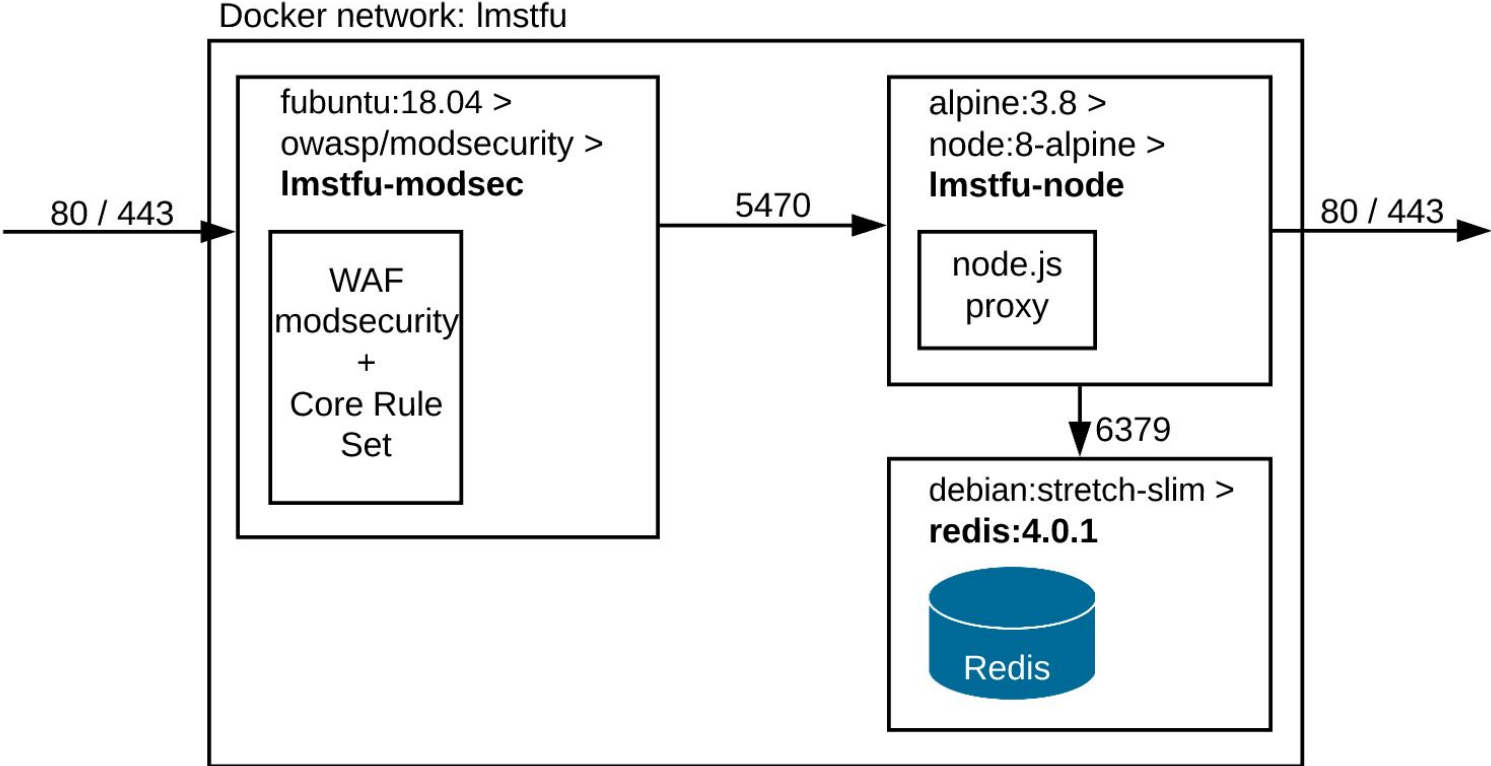
OWASP Core Rule Set

- Signatures for common OWASP attacks

Node.js

- Fast, flexible, event-driven
- State storage in redis

Docker



Demo of Imstfu setup

ModSecurity

- Originally an Apache httpd module
- v2.9.3 also supports IIS and nginx (instabilities)
- v3.0.3 rewritten into libmodsecurity + connector
 - Not all features supported yet
- Doesn't do much out of the box
 - Safe to enable in DetectionOnly mode



```
[[root@677152b32509 modsecurity.d]# wc modsecurity.conf
226 1176 8441 modsecurity.conf
[[root@677152b32509 modsecurity.d]# cat modsecurity.conf
# -- Rule engine initialization -----

# Enable ModSecurity, attaching it to every transaction. Use detection
# only to start with, because that minimises the chances of post-installation
# disruption.
#
SecRuleEngine DetectionOnly

# -- Request body handling -----

# Allow ModSecurity to access request bodies. If you don't, ModSecurity
# won't be able to see any POST parameters, which opens a large security
# hole for attackers to exploit.
#
SecRequestBodyAccess On
```

OWASP ModSecurity Core Rule Set

Ruleset for common attacks:

- SQL Injection (SQLi)
- Cross Site Scripting (XSS)
- Local File Inclusion (LFI)
- Remote File Inclusion (RFI)
- Remote Code Execution (RCE)
- PHP Code Injection
- HTTPoxy
- Shellshock
- Session Fixation
- Scanner Detection
- Metadata/Error Leakages
- GeoIP Country Blocking



OWASP
ModSecurity
Core Rule Set
THE 1ST LINE OF DEFENSE

Tuned to avoid false positives

coreruleset.org

OWASP ModSecurity Core Rule Set

`crs-setup.conf.example` - configure mode, paranoia level

Look at requests

`REQUEST-901-INITIALIZATION.conf`
`REQUEST-903.9001-DRUPAL-EXCLUSION-RULES.conf`
`REQUEST-903.9002-WORDPRESS-EXCLUSION-RULES.conf`
`REQUEST-905-COMMON-EXCEPTIONS.conf`
`REQUEST-910-IP-REPUTATION.conf`
`REQUEST-911-METHOD-ENFORCEMENT.conf`
`REQUEST-912-DOS-PROTECTION.conf`
`REQUEST-913-SCANNER-DETECTION.conf`
`REQUEST-920-PROTOCOL-ENFORCEMENT.conf`
`REQUEST-921-PROTOCOL-ATTACK.conf`
`REQUEST-930-APPLICATION-ATTACK-LFI.conf`
`REQUEST-931-APPLICATION-ATTACK-RFI.conf`
`REQUEST-932-APPLICATION-ATTACK-RCE.conf`
`REQUEST-933-APPLICATION-ATTACK-PHP.conf`
`REQUEST-941-APPLICATION-ATTACK-XSS.conf`
`REQUEST-942-APPLICATION-ATTACK-SQLI.conf`
`REQUEST-943-APPLICATION-ATTACK-SESSION-FIXATION.conf`

Look at responses

`RESPONSE-950-DATA-LEAKAGES.conf`
`RESPONSE-951-DATA-LEAKAGES-SQL.conf`
`RESPONSE-952-DATA-LEAKAGES-JAVA.conf`
`RESPONSE-953-DATA-LEAKAGES-PHP.conf`
`RESPONSE-954-DATA-LEAKAGES-IIS.conf`

Is this an attack?

`REQUEST-949-BLOCKING-EVALUATION.conf`
`RESPONSE-959-BLOCKING-EVALUATION.conf`
`RESPONSE-980-CORRELATION.conf`

Anatomy of a SecRule

`SecRule VARIABLES OPERATOR [ACTIONS]`

Anatomy of a SecRule

SecRule VARIABLES OPERATOR [ACTIONS]

SecRule REQUEST_FILENAME

*ARGS, ENV, FILES, IP, PATH_INFO, REMOTE_ADDR, REQUEST_COOKIES, REQUEST_URI,
REQUEST_HEADERS, ...*

<http://lmstfu.com/SecRuleLayout>

Anatomy of a SecRule

```
SecRule VARIABLES OPERATOR [ACTIONS]
```

```
SecRule REQUEST_FILENAME "@rx /order/details/" \
```

@rx, @streq, @beginsWith, @contains, @gt, @lt,

Anatomy of a SecRule

```
SecRule VARIABLES OPERATOR [ACTIONS]
```

```
SecRule REQUEST_FILENAME "@rx /order/details/" \  
    "id:11101,phase:1,deny,log,\  
"
```

Phases:

1) Request headers, 2) Request body, 3) Response headers, 4) Response body, 5) Logging

<http://lmstfu.com/SecRuleLayout>

Anatomy of a SecRule

```
SecRule VARIABLES OPERATOR [ACTIONS]
```

```
SecRule REQUEST_FILENAME "@rx /order/details/" \  
    "id:11101,phase:1,deny,log,\  
    t:none,t:lowercase,t:normalisePath,\  
    msg:'Blocking access to %{MATCHED_VAR}'"
```

SecRule chaining

Logical AND between rules:

```
SecRule ARGS:p "@rx test1" \  
    id:2000,chain,...
```

```
    SecRule ARGS:q "@rx test2"
```

More complex flow control

SecMarker **IF**

```
SecRule &ARGS:admin "@gt 0" \  
    "id:2000,pass,nolog,skipAfter:ELSE"
```

SecMarker **THEN**

```
SecRule ARGS:p "@rx K1" "id:2001,block,log"  
SecAction "id:2003,pass,nolog,skipAfter:END"
```

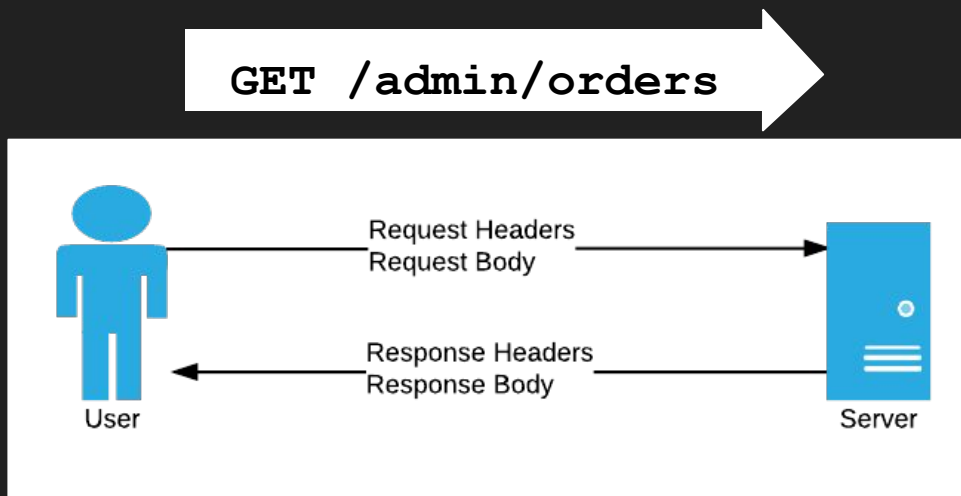
SecMarker **ELSE**

```
SecRule ARGS:p "@rx K3" "id:2003,block,log"
```

SecMarker **END**

Demo of modsecurity

Block a URL



Inspect:

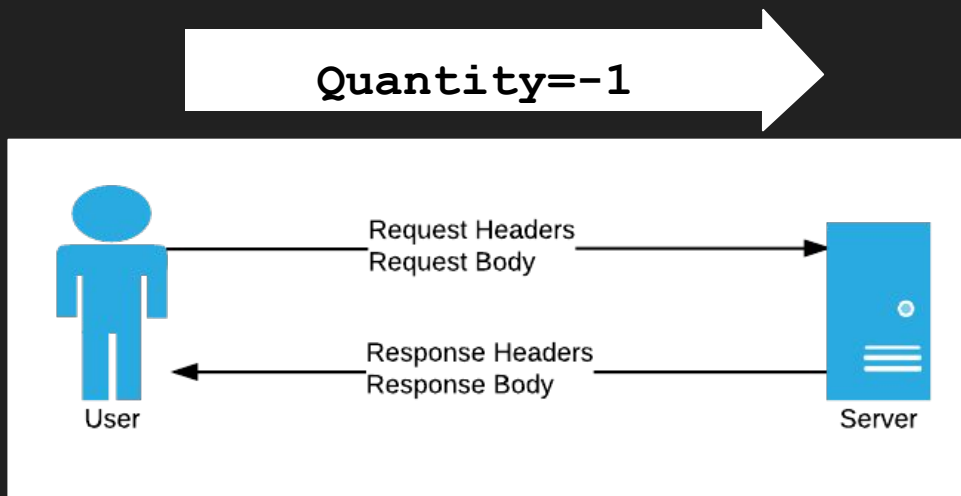
- Inspect Request Headers
 - Is it the right url?

Take action:

- Reject

```
SecRule REQUEST_FILENAME "/admin/orders" \  
  "id:11101,phase:1,deny,log,\  
  t:none,t:lowercase,t:normalisePath,\  
  msg:'Blocking access to %{MATCHED_VAR}'"
```

Validating a parameter



Inspect:

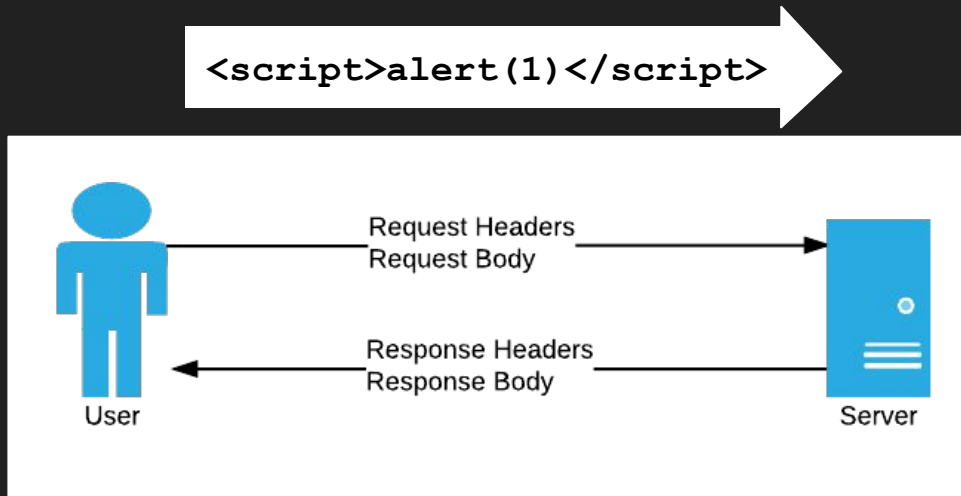
- Inspect Request Headers + Body
 - Is parameter a positive number?

Take action:

- Reject

```
SecRule ARGS:/^ProductChoices\[.*\].Quantity$/ "!@rx ^\d+$" \  
  "id:10050, phase:2, pass, log, \  
  t:none, t:removeWhitespace, \  
  msg:'Invalid quantity entered: %{MATCHED_VAR}'"      (V15)
```

XSS in the input?



Inspect:

- Inspect Request Headers + Body
 - Does parameter look like XSS?

Take action:

- Reject

```
# Configure-Time: Only test XSS for the Comment parameter
```

```
SecRuleUpdateTargetByID 941100-941999 "ARGS:Comment"
```

```
# See V15
```

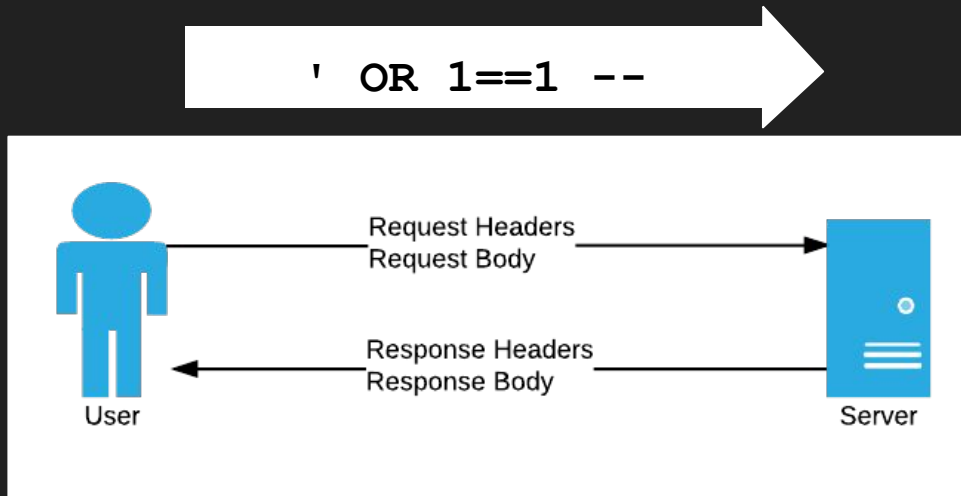

Blocking XSS

ModSecurity also uses libinjection for XSS and SQLi detection

Guns and Butter: Towards Formal Axioms of Validation
Hanson and Patterson

...formally proved that for any regex validator, we could construct either a safe query which would be flagged as dangerous, or a dangerous query which would be flagged as correct

SQL injection in the input?



Inspect:

- Inspect Request Headers + Body
 - Does parameter look like SQLi?

Take action:

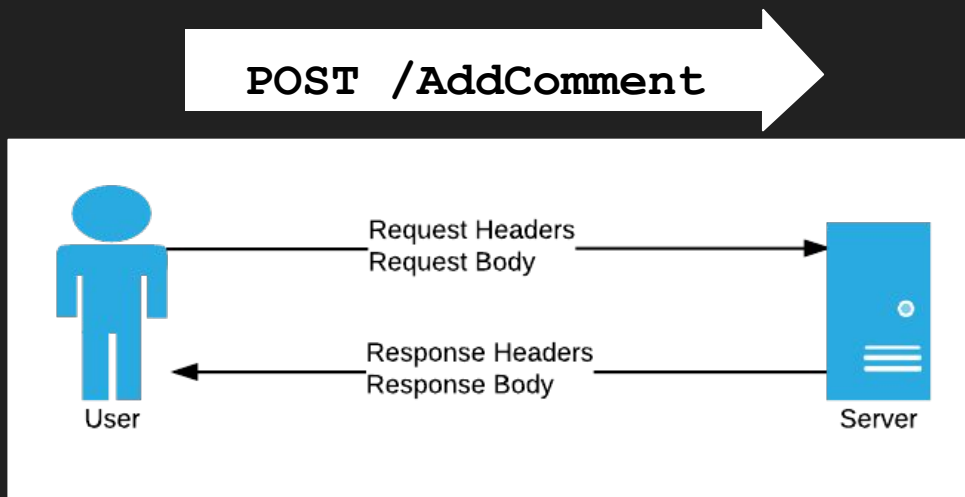
- Reject

```
# Configure-Time: Only test SQLi for the SearchTerm parameter
```

```
SecRuleUpdateTargetByID 942100-942999 "ARGS:SearchTerm"
```

```
# See V17
```

Cross-Site Request Forgery



```
# If no token exists:  
# - Create a CSRF token using ModSecurity  
# - Send the token as a cookie  
# - Add the token to the form post using inserted  
  javascript  
# - On POST, check if the cookie value matches the posted value  
See V16
```

Inspect:

- Inspect Request Headers + Body
 - Is the referrer wrong?
 - Is the origin wrong?
 - Is there a valid CSRF token?

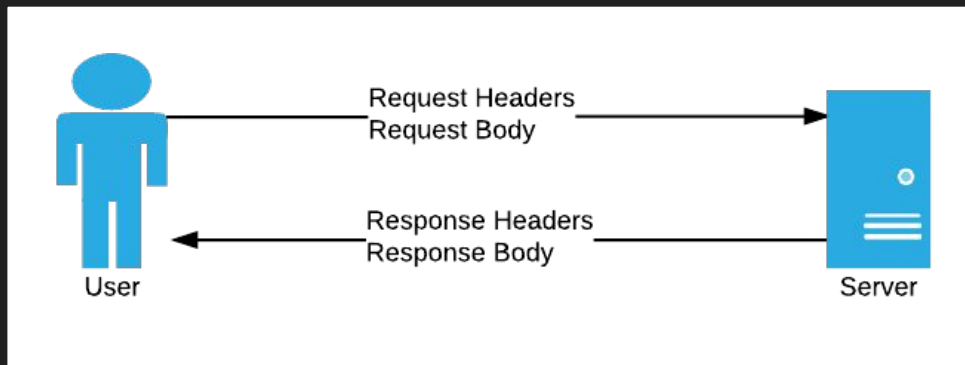
Take action:

- Reject

Missing headers and cookie flags

Take action:

- Alter Response Headers
 - Set cookie flags
 - Add new headers



Set-Cookie: a=b

```
Header edit Set-Cookie "(?i)^(.AspNetCore.Antiforgery.(?:?!httponly).+)$" "$1; HttpOnly"
```

See V13

Why do we use ModSecurity + CRS?

Good, low false positive set of XSS and SQLi rules

Efficient processing and blocking

Allows simple things to be done *relatively* easily

Can be extended to do complex things, but it gets complicated fast!

Limitations of ModSecurity

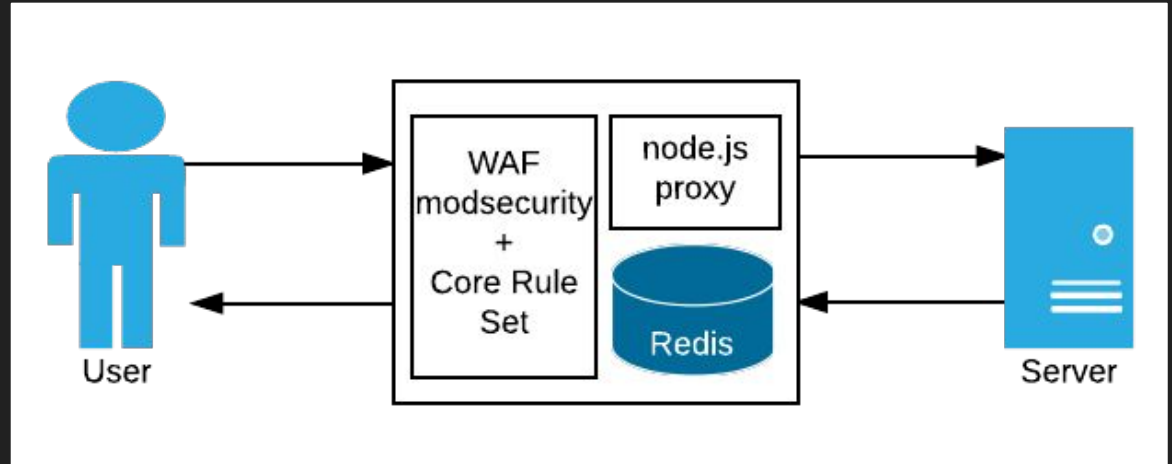
- Daunting syntax
- Limited manipulation of the response body
 - Hard to remove sensitive data, add validation text etc
- Hard to capture *program state*
- Extensible via Lua, but not many examples

Business Logic

“Let me secure that for you!”

node.js proxy

- Business logic
- Storing state in redis
- Transforming HTML



Redbird, http-proxy, harmon, trumpet

Redbird

- Wraps http-proxy with extra features



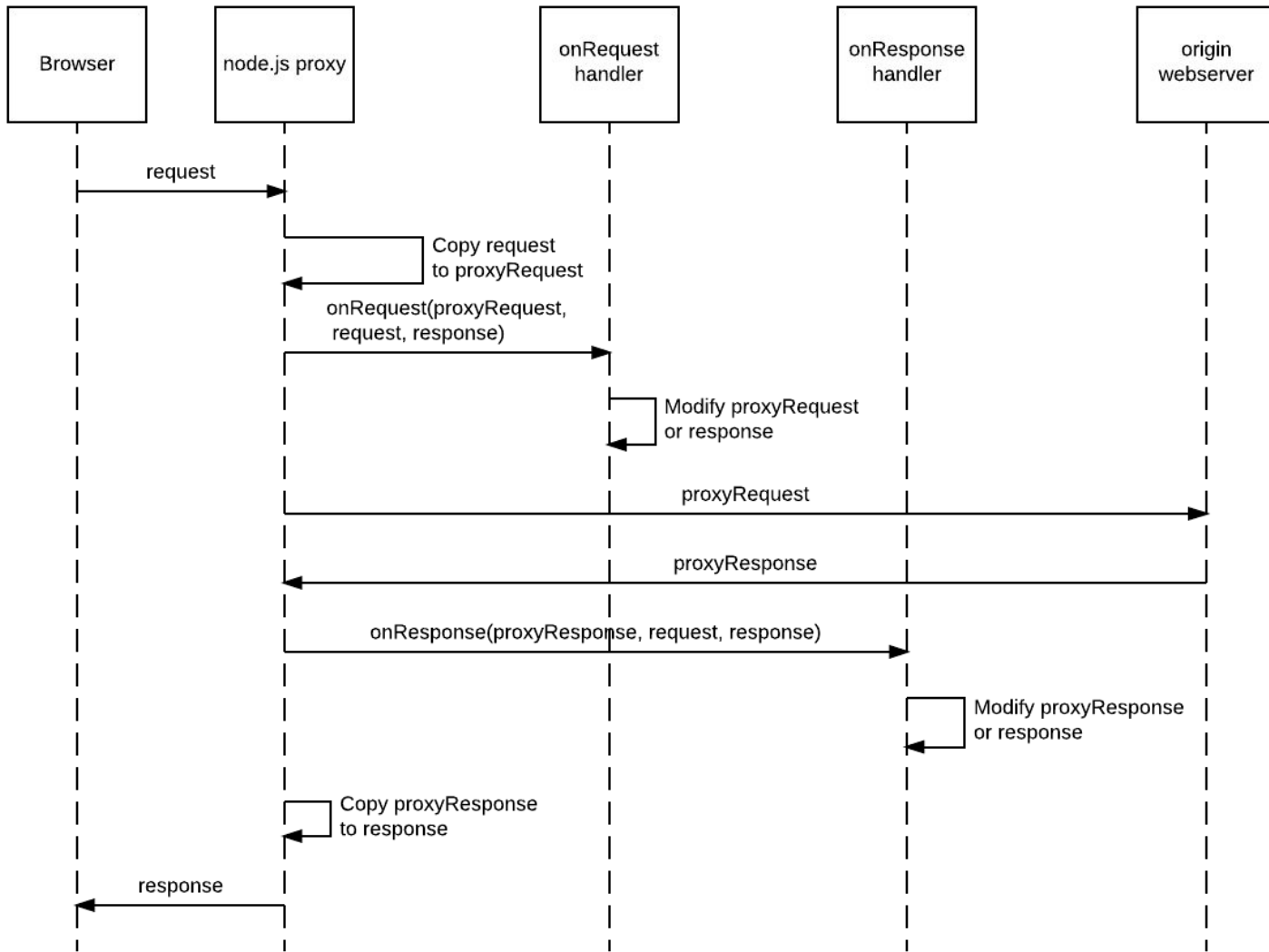
Harmon

- Uses trumpet for streaming HTML manipulation



160+ node modules!

Search www.npmjs.com for Redbird, http-proxy, harmon, trumpet

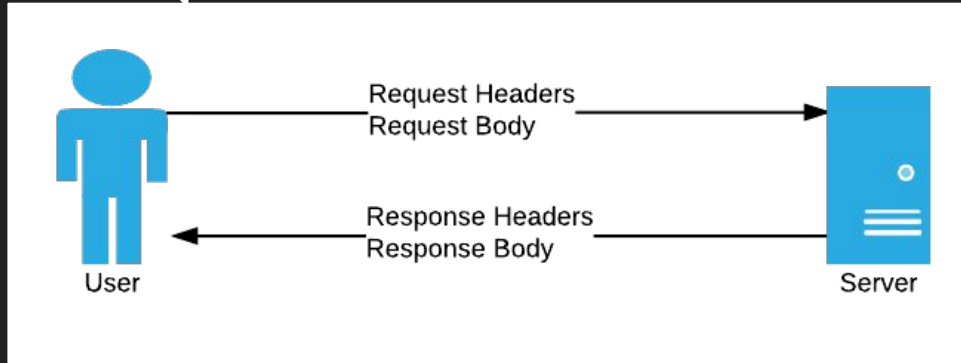


Demo - More complex vulnerabilities

View other people's orders

GET /Order

Orders 1,2,3,5



GET /Order/Details?id=14

ordercontrol.js

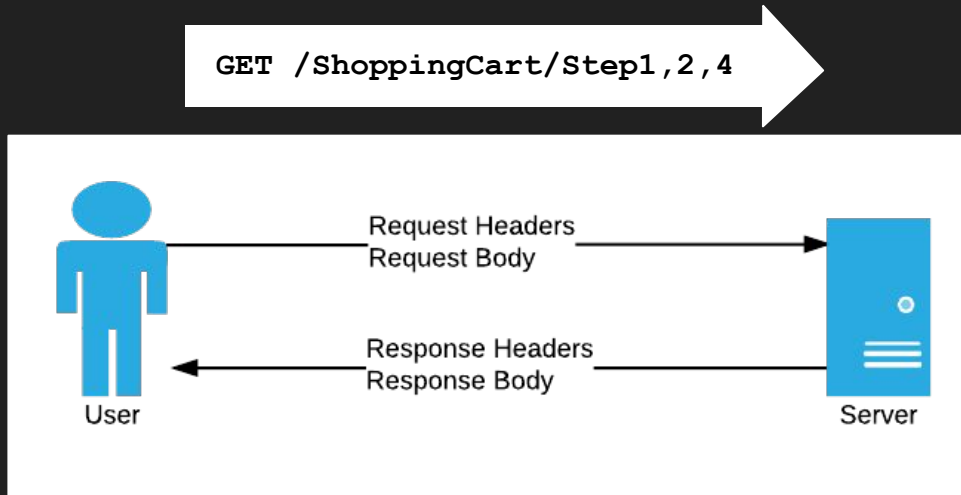
Inspect + State:

- Inspect the response to the “My Orders” request
- Store all the user’s orders persistently
- Check requests for order details to make sure the user can access

Take action:

- Block

Skip the payment step and get free orders



Inspect + State:

- Keep track of which steps have submitted successfully
- Check no step is missed

Take action:

- Redirect to Step1

HTML Manipulation

```
simpleselect.query = 'input[type=password]';  
  
simpleselect.func = function (node) {  
  
    node.setAttribute('autocomplete', 'off');  
  
}
```



User

Request
Request

Response
Response

Server

<html>

PCI Compliance

```
step3cardnumber.query = '#CartPaymentViewModel_CardNumber'  
step3cardnumber.func = function (node) {  
    var attr = node.getAttribute("value");  
    if (attr !== "") {  
        node.setAttribute("value", "****-****-****-****");  
    }  
}
```



User

Server

<html>

creditcard.js

Other examples

- Password strength checking
- HTML manipulation
- Tamper-protection on hidden fields
- Changing validation rules and messages
- XML / json inspection
- API protection

Why use node.js proxy?

Javascript is the language of the internet

Fast, scalable, mature

Often used for node.js load balancing

Performant HTML manipulation

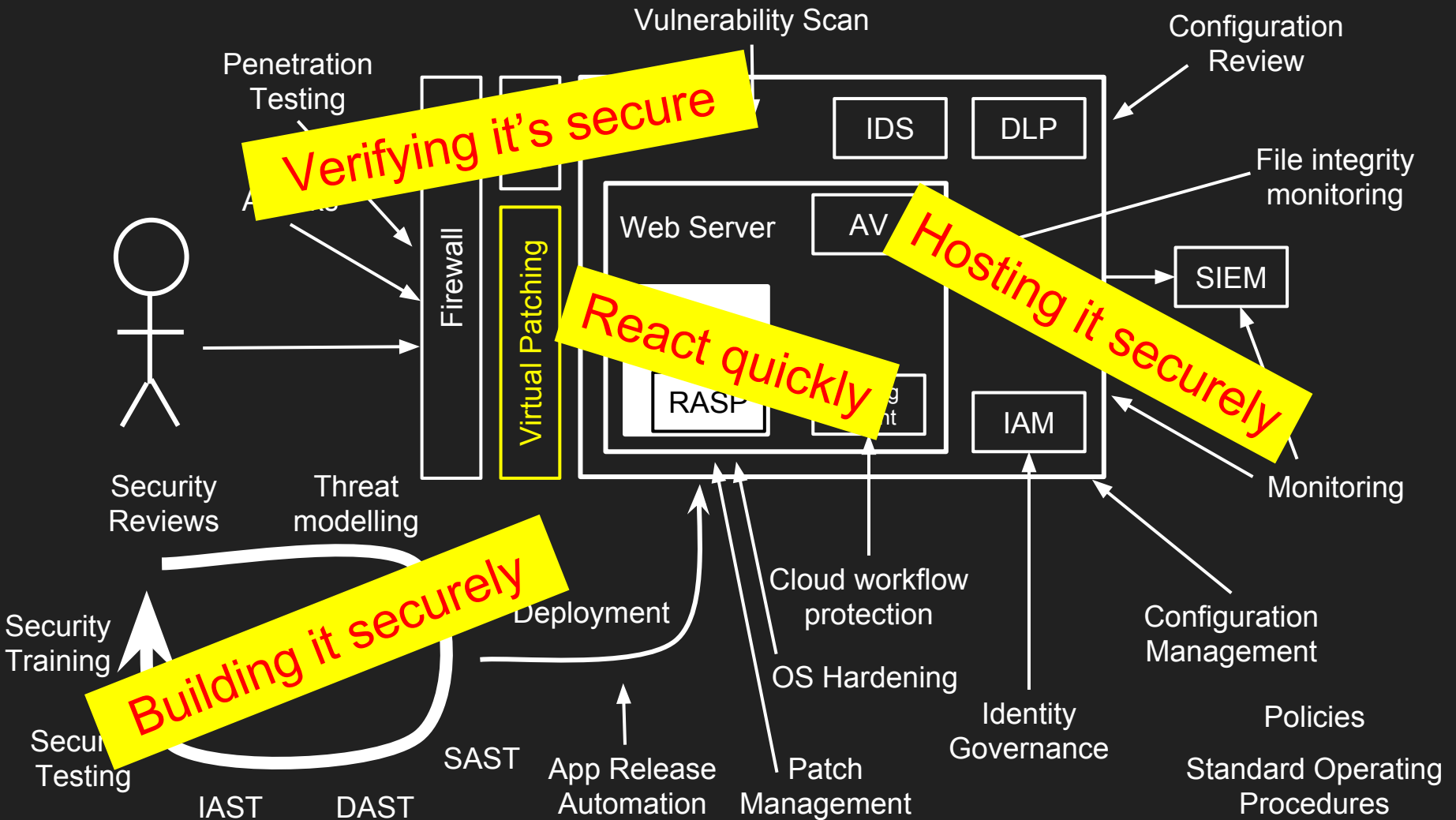
Limitations of node.js proxy

- Asynchronous programming is hard
- Still need modsecurity + CRS for signatures
- Lot of overlaps with modsecurity - which tool is the right one?

Virtual Patching is a thing

Another tool to add to your toolbelt

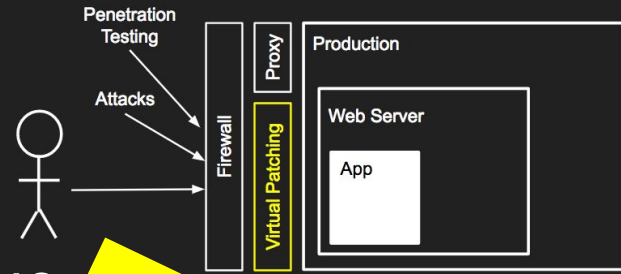
Prepare the infrastructure in advance



Our Virtual Patching Approach

Understand *how to exploit* the security issue

Only patch *known vulnerabilities or weaknesses*

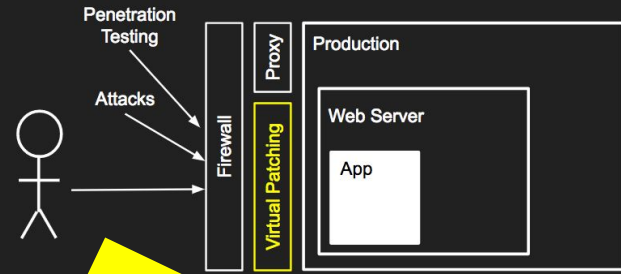


Recap

Avoid over-patching or doing things that will cause issues:

- Learning mode, tuning, false positives
- Large blocklists
- Focus *only* on the script, page or parameter affected

Our Virtual Patching Approach



Only patch *known vulnerabilities or weaknesses*

Recap



lmstfu

Virtual Patching: Does it work?

OWASP NZ Day | 22 Feb 2019

Kirk Jackson | @kirkj
lmstfu.com | @LetMeSecureThat