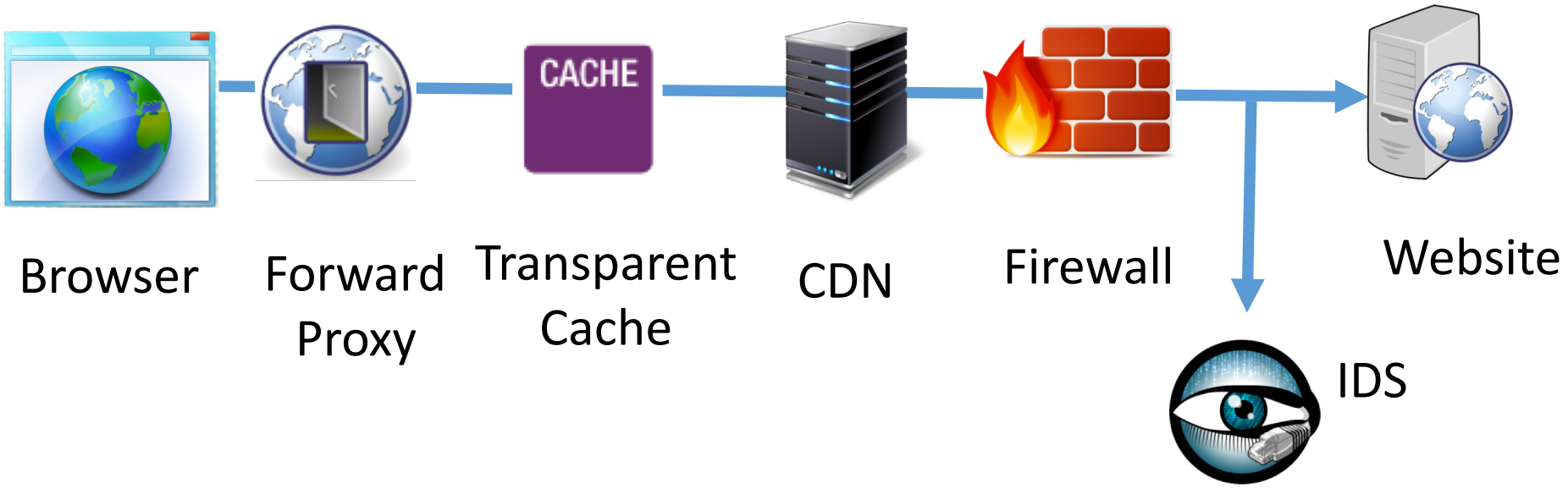
Host of Troubles: Multiple **Host** Ambiguities in HTTP Implementations

Jianjun Chen, Jian Jiang, Haixin Duan,

Nicholas Weaver, Tao Wan, Vern Paxson

Multiparty interactions in current Internet



Ambiguity between different parties could cause security problems.

Previous works about ambiguity

- HTTP request smuggling [Linhart 2005]
  - Exploiting ambiguity of Content-Length header
- HTTP Evader [Ullrich 2013]
  - Exploits multiple ambiguities of HTTP response headers (Content-Encoding .etc)
- Host header attacks [Kettle 2013]
  - Exploiting insufficient input validation of host-related variables in web applications
  - Leading to phishing, cross-site scripting.

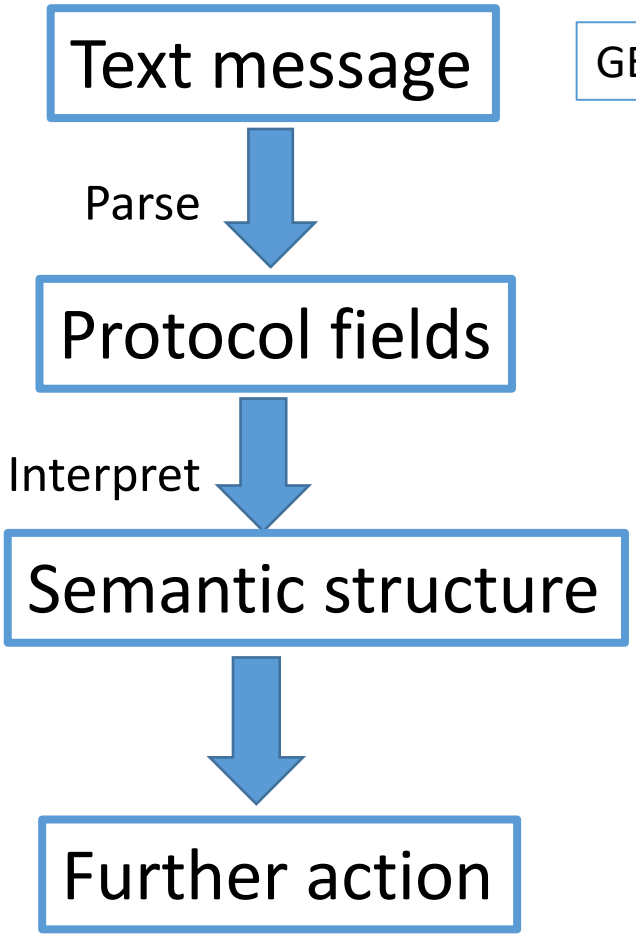
Our work

- We present “Host of Troubles” attacks, that can cause severe security consequences, such as cache poisoning and filter bypass.
  - 3 types of techniques
- We studied 33 popular HTTP implementations, and identified a large range of potential exploits.
- We conducted a large scale measurement and found that around 97% of Internet users served by a transparent cache are subject to cache poisoning attacks.

Outline

- Overview of HTTP Host header
- Three techniques leading to Host header ambiguity
- Five attacks exploiting Host header ambiguity
- Large scale measurement of transparent cache poisoning
- Concluding remarks

How HTTP requests are processed



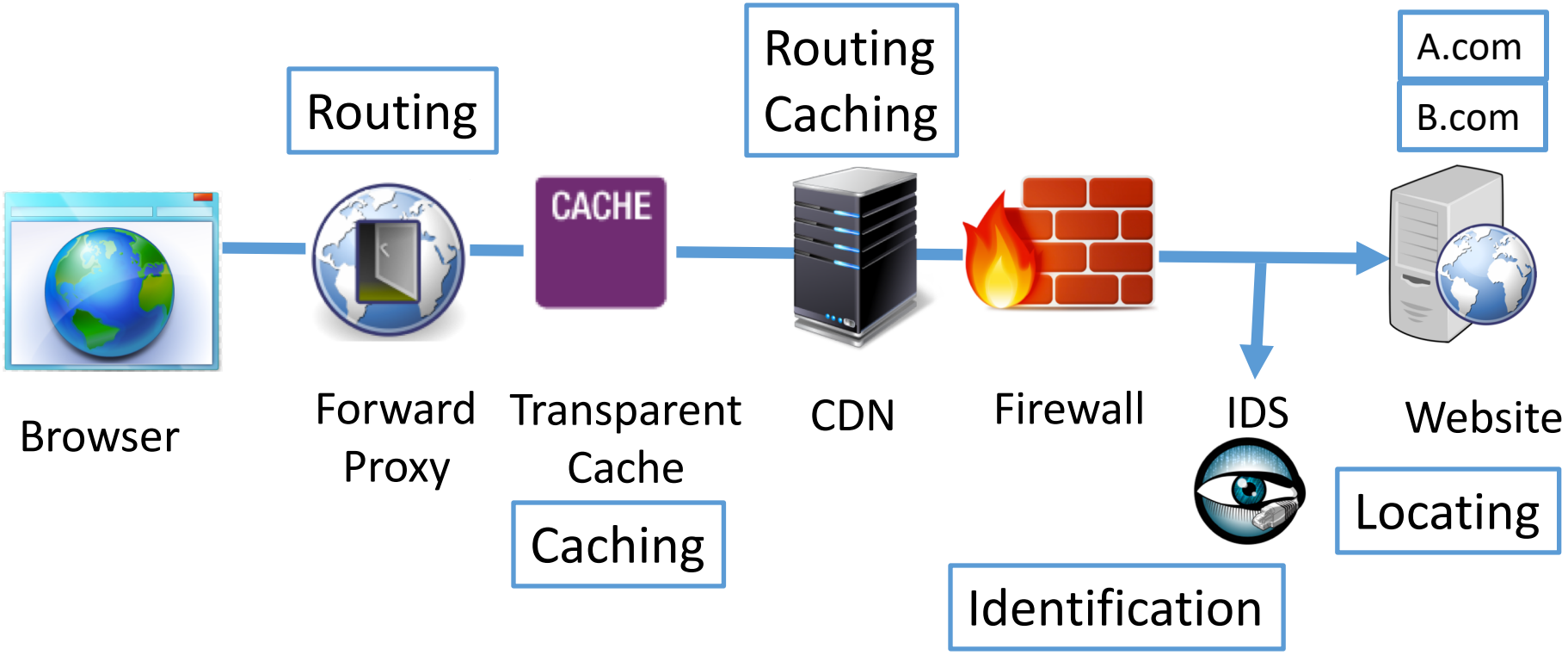
GET / HTTP/1.1\r\nHost: a.com\r\nUser-Agent:Mozilla…

GET / HTTP/1.1
host a.com
user-agent Mozilla
…

```
typedef struct {
    ngx_list_t        headers;

    ngx_table_elt_t  *host;
    ngx_table_elt_t  *user_agent;
    ngx_table_elt_t  *referer;
    ngx_table_elt_t  *content_length;
    ngx_table_elt_t  *content_type;
```

Host – A critical HTTP field

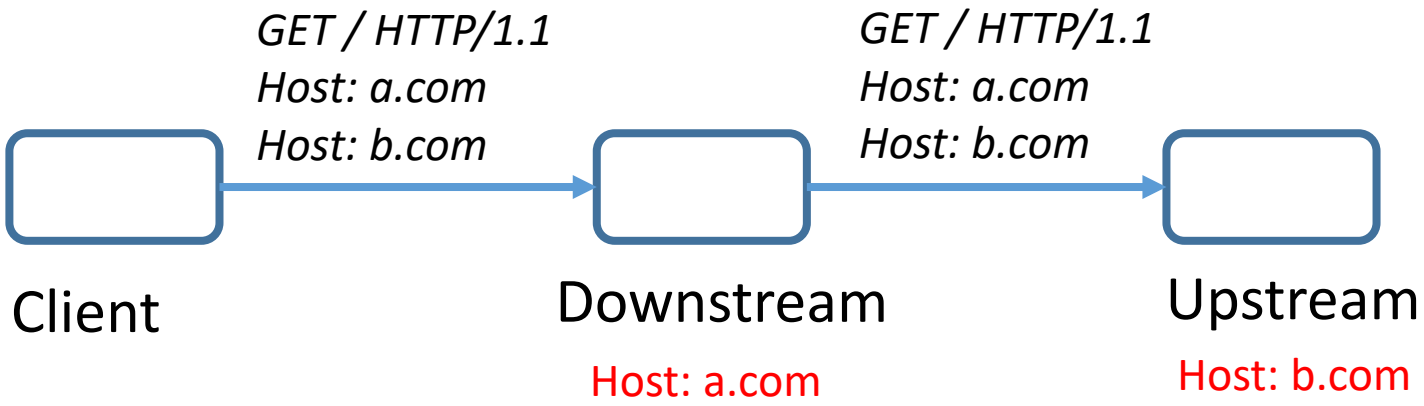


Ambiguity between different parties can cause disastrous consequences

Outline

- Overview of HTTP Host header
- Three techniques leading to Host header ambiguity
- Five attacks exploiting host header ambiguity
- Large scale measure of transparent cache poisoning
- Concluding remarks

Technique 1: Multiple Host header



HTTP standard (HTTP/1.1)

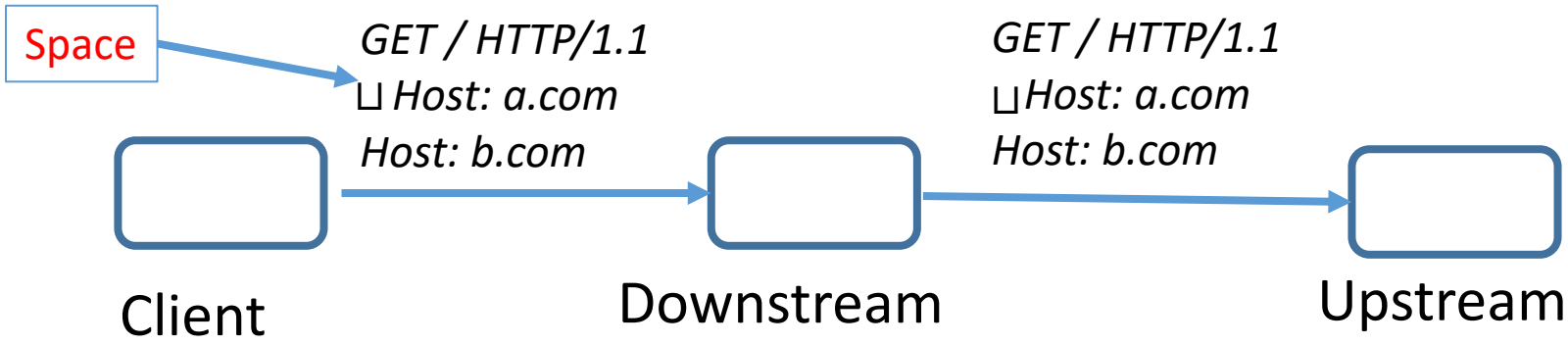
- RFC 2616 (obsoleted), implicitly requires rejection.
- RFC 7230 (latest), explicitly requires rejection.

How do implementations handle requests with multiple Host header?

Implementation	Preference	Implementation	Preference	Implementation	Preference
Apache	Concatenate	Akamai	First	Bitdefender	First
IIS	Reject	Alibaba	First	ESET	Last
Nginx	First	Azure	Reject	Huawei	First
Tomcat	First	CloudFlare	First	Kaspersky	First
ATS	First	CloudFront	First	OS X	Concatenate
Squid	First	Fastly	Reject	PAN	First
Varnish	Reject	Tencent	Last	Windows	First

- Most implementations don't follow RFC7230
- Some implementations are inconsistent with others

Technique 2: Space-surrounded Host Header



HTTP standard

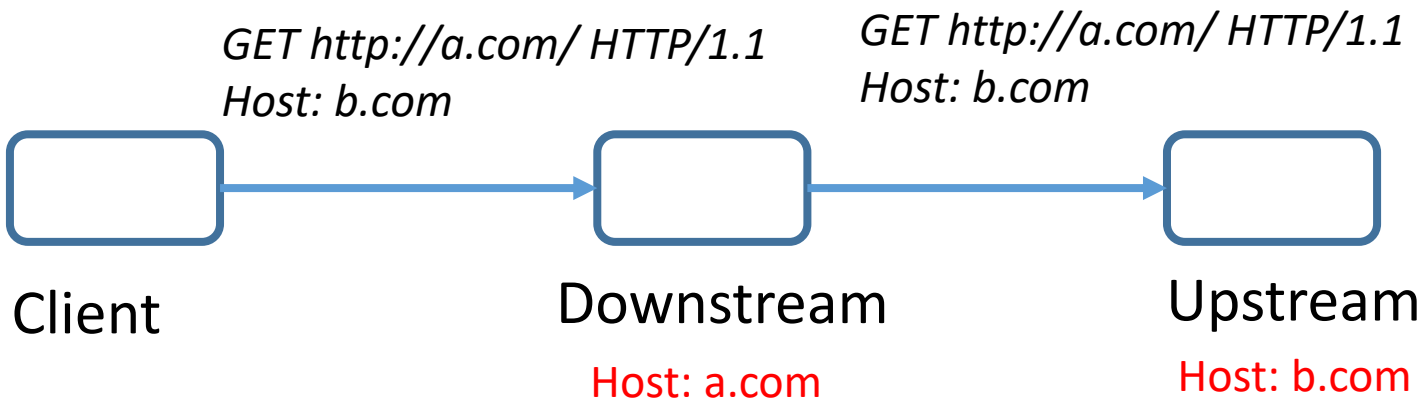
	Space-preceded Host as first header	Other space-preceded Host header	Space b/w Host and ':'
RFC 2616	Reject (implicit)	Line folding	Recognize (implicit)
RFC 7230	Reject	Reject	Reject

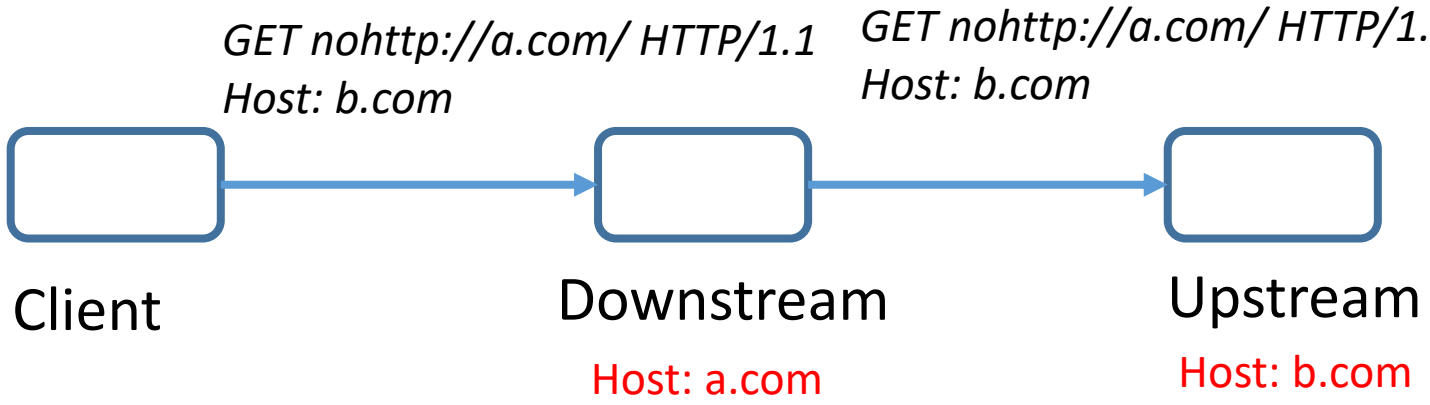
How implementations handle requests with space-surrounded Host Header?

		Space-preceded Host as first header	Other space-preceded Host header	Space-succeeded Host header
Server	Apache	Not recognize	Line folding	Recognize
	IIS	Recognize	Line folding	Recognize
	Nginx	Not recognize	Not recognize	Not recognize
Transparent Cache	ATS	Not recognize	Not recognize	Not recognize
	Squid	Recognize	Recognize	Recognize
CDN	Akamai	Recognize	Recognize	Recognize
	Alibaba	Not recognize	Not recognize	Not recognize
	CloudFlare	Not recognize	Not recognize	Not recognize
	Tencent	Recognize	Recognize	Recognize
Firewall	Huawei	Not recognize	Not recognize	Not recognize
	PAN	Not recognize	Not recognize	Not recognize

- Most implementations don't follow RFC7230 and vary in processing space-surrounded Host headers

Technique 3: Absolute-URI as request-target





Technique 3: Absolute-URI as request-target

HTTP standard

	Preference	Schema
RFC 2616	Absolute-URI	Not specified
RFC 7230	Absolute-URI	Not specified

HTTP implementations

- For preference between absolute uri and Host header
  - Except Akamai， other implementations follow RFC

How do different implementations handle absolute-URI?

Implementation	Schema	Implementation	Scheme	Implementation	Scheme
Apache	HTTP only	Akamai	HTTP/S	Bitdefender	any
IIS	HTTP/S	Alibaba	any	ESET	any
Nginx	any	Azure	HTTP/S	Huawei	any
Tomcat	HTTP/S	CloudFlare	any	Kaspersky	any
ATS	any	CloudFront	any	OS X	HTTP only
Squid	HTTP only	Fastly	HTTP only	PAN	HTTP/S
Varnish	HTTP only	Tencent	HTTP only	Windows	any

The space of Host ambiguity increases once again!

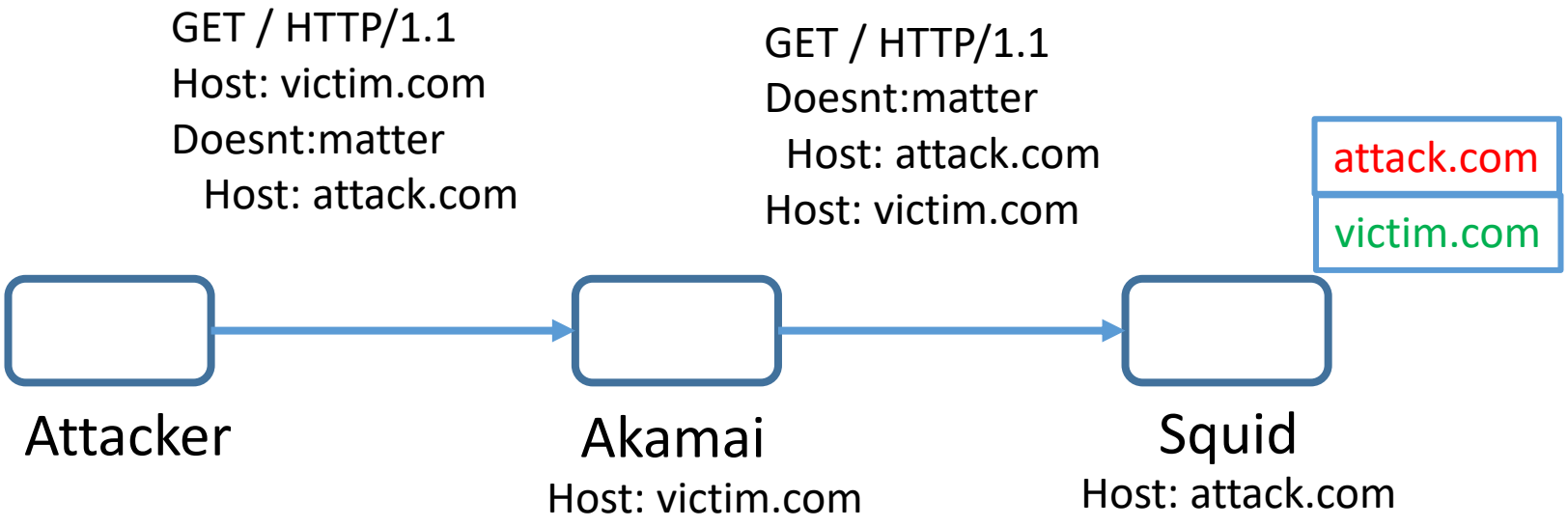
Outline

- Overview of HTTP Host header
- Three techniques leading to Host header ambiguity
- Five attacks exploiting host header ambiguity
- Large scale measure of transparent cache poisoning
- Concluding remarks

Attacks exploiting host ambiguity

- Cache poisoning Attacks
  - Cache poisoning co-hosting website
  - Cache poisoning co-CDN website
  - Cache poisoning any HTTP website
- Bypass security policy
  - Bypass firewall filtering policy
  - Bypass WAF

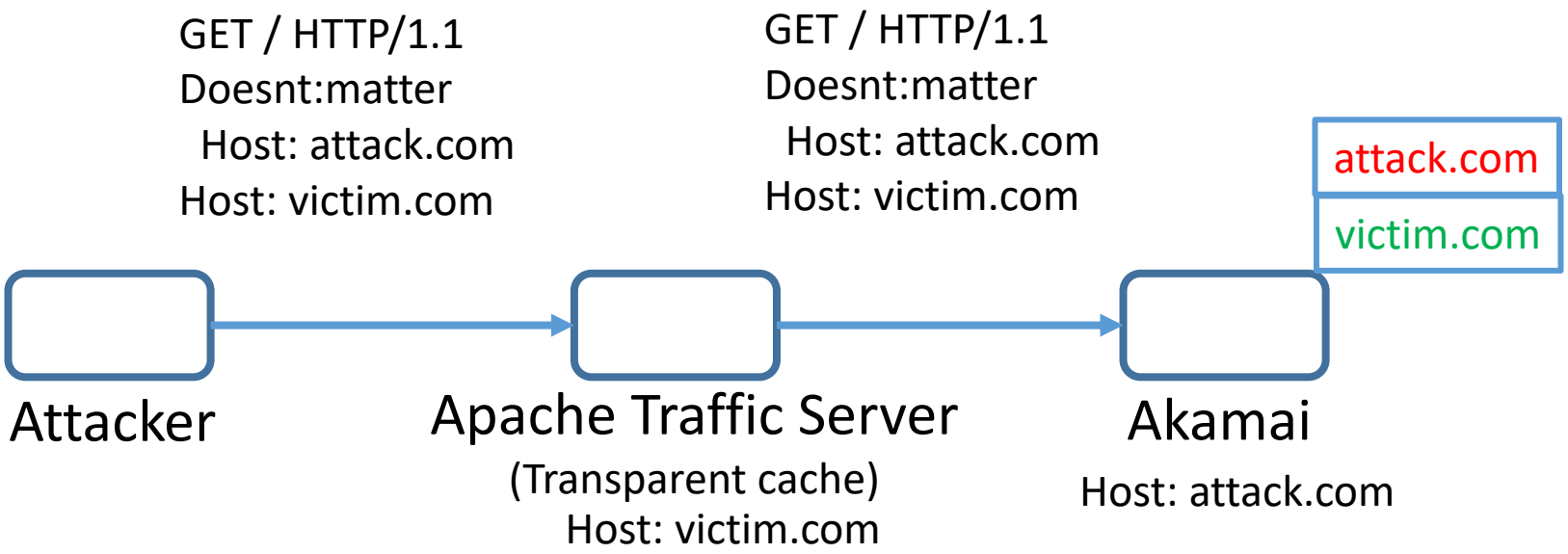
Attack 1: Cache poisoning co-hosting website



Requirement: co-hosting of attack.com and victim.com

Consequence: CDN cache poisoning

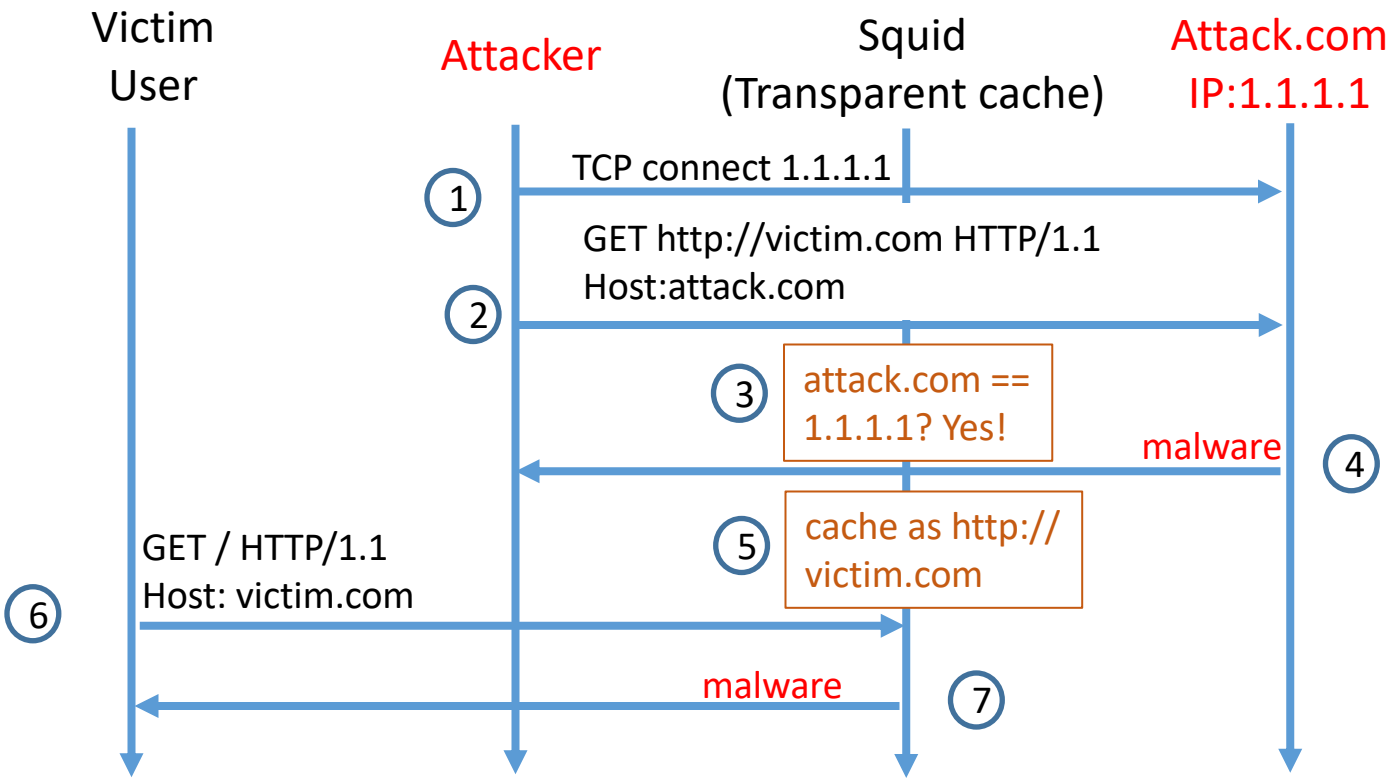
Attack 2: Cache poisoning co-CDN website



Requirement: co-CDN of attack.com and victim.com

Consequence: transparent cache poisoning

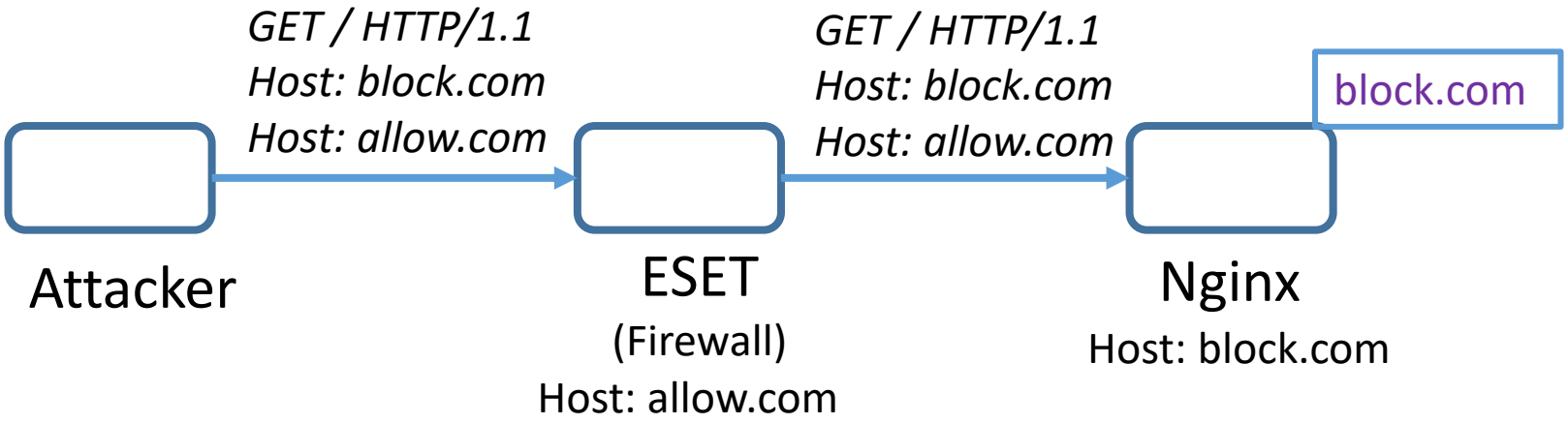
Attack 3: Cache poisoning any HTTP website (*CVE-2016-4553*)



Requirement: no condition for victim website

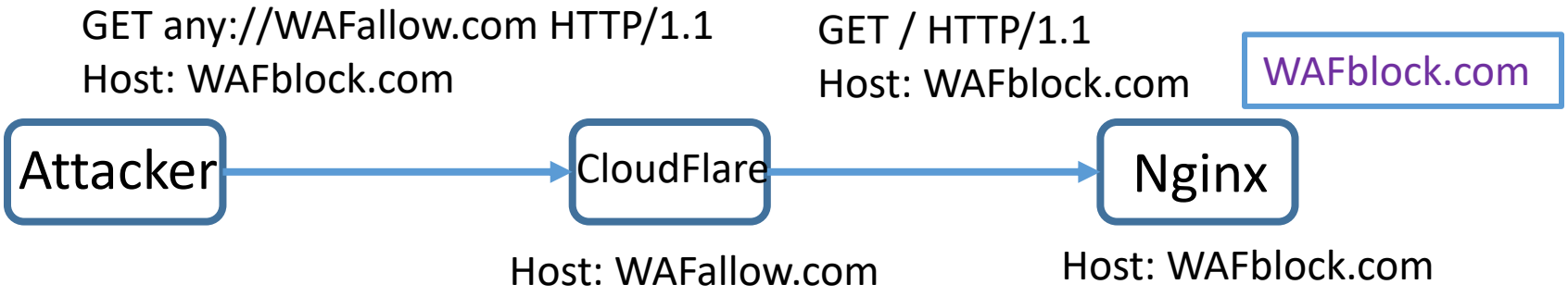
Consequence: transparent cache poisoning

Attack 4: Firewall bypass



ESET firewall doesn't allow client to visit block.com.

Attack 5: WAF bypass



CloudFlare customer *WAFblock.com* uses CloudFlare's Web Application Firewall(WAF) to block SQL injection attacks.

How Prevalent are Upstream/Downstream vulnerabilities?

Downstream \ Upstream		Reverse Proxy							CDN								Server					
		Apache	IIS	Lighttpd	LiteSpeed	Nginx	Squid	Varnish	Akamai	Alibaba	Azure	CloudFlare	CloudFront	Fastly	Level3	Tencent	Apache	IIS	Lighttpd	LiteSpeed	Nginx	Tomcat
Transparent Cache	ATS				✓		✓		✓							✓				✓		
	Squid						✓		✓							✓						
Forward Proxy	Apache															✓						
	Squid						✓		✓							✓						
Reverse Proxy	Apache								—	—	—	—	—	—	—	—						
	Lighttpd				✓	✓			—	—	—	—	—	—	—	—				✓	✓	
	LiteSpeed	✓		✓		✓		✓	—	—	—	—	—	—	—	—	✓		✓		✓	✓
	Squid						✓		—	—	—	—	—	—	—	—						
	Varnish		✓	✓	✓	✓			—	—	—	—	—	—	—	—		✓	✓	✓	✓	✓
CDN	Akamai						✓		—													
	Alibaba				✓		✓		✓	—										✓		
	CloudFlare	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	—	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	CloudFront												—			✓						
	Fastly		✓	✓	✓	✓			✓	✓		✓	✓	—	✓			✓	✓	✓	✓	✓
Firewall	Bitdefender	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	ESET	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Huawei	✓	✓	✓	✓		✓	✓	✓		✓			✓		✓	✓	✓	✓	✓		
	Kaspersky	✓	✓	✓	✓		✓	✓	✓		✓			✓		✓	✓	✓	✓	✓		
	OS X	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
	PAN	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓			✓	✓	✓	✓	✓	✓	✓
	Windows	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

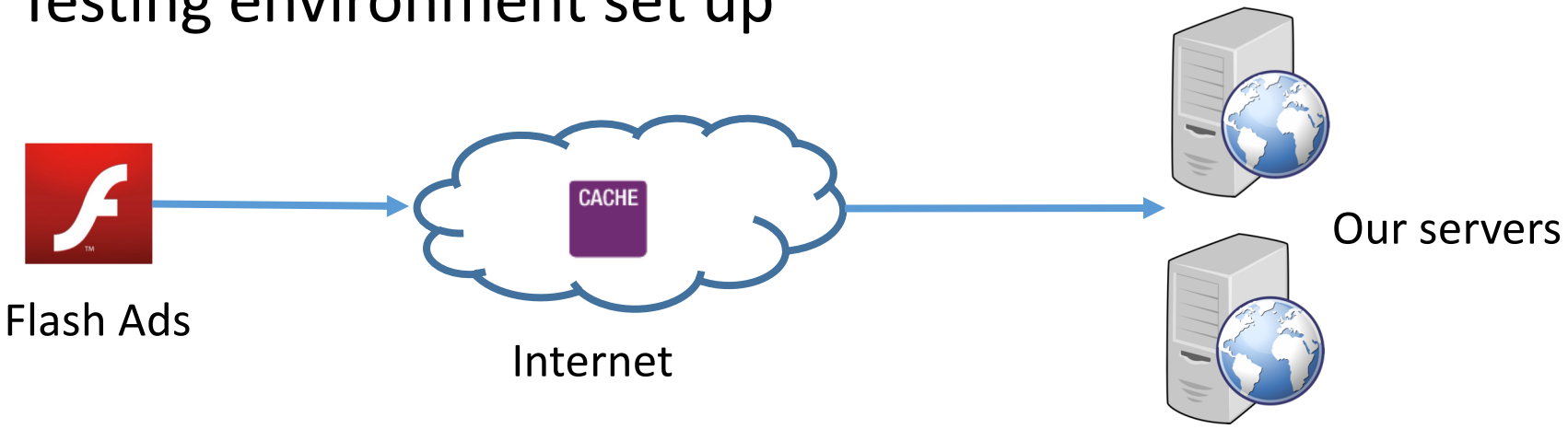
202 different combinations that could be exploited.

Outline

- Overview of HTTP Host header
- Three techniques leading to Host header ambiguity
- Attacks exploiting host header ambiguity
- Large scale measurement of transparent cache poisoning
- Concluding remarks

Measurement set up

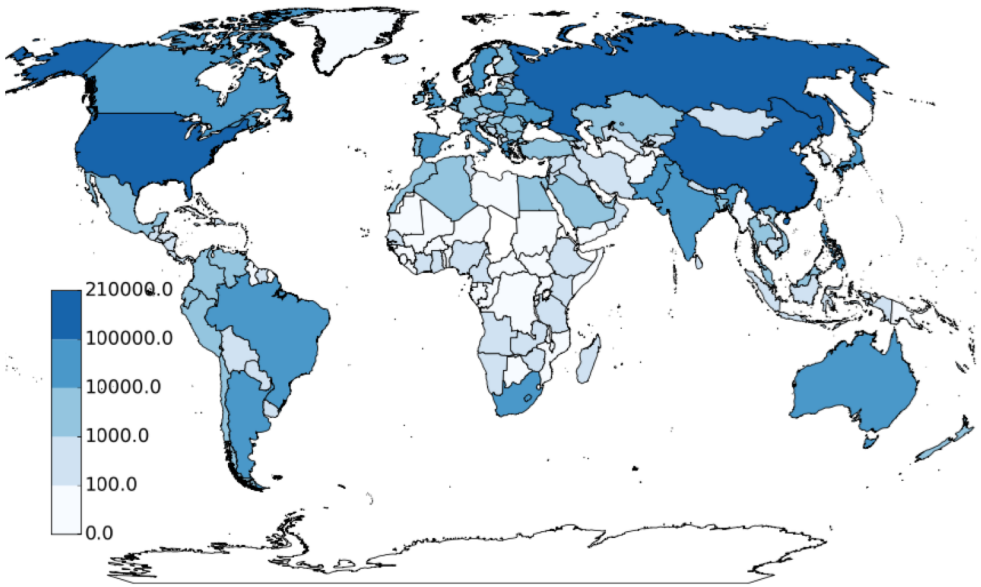
- Online Flash advertisement
- Testing environment set up



- 16 different test cases
  - 11 of them to detect co-hosting cache poisoning
  - 5 of them to detect general cache poisoning

Execution of test cases

- Utorrent PC advertising , 1.5M impressions, $110
- Hosted by a large website, 3/11/2016 to 3/31/2016



Geographical distribution of involved clients

Measurement results

- Utorrent ads
  - 16,168 IPs detected ISP caches
  - Among them, 15,677 (96.9%) IPs can be exploited
- Website ads
  - 1,376 IPs detected ISP caches
  - Among them, 1,331 (96.7%) IPs can be exploited

97% of users served by transparent caches could have been poisoned.

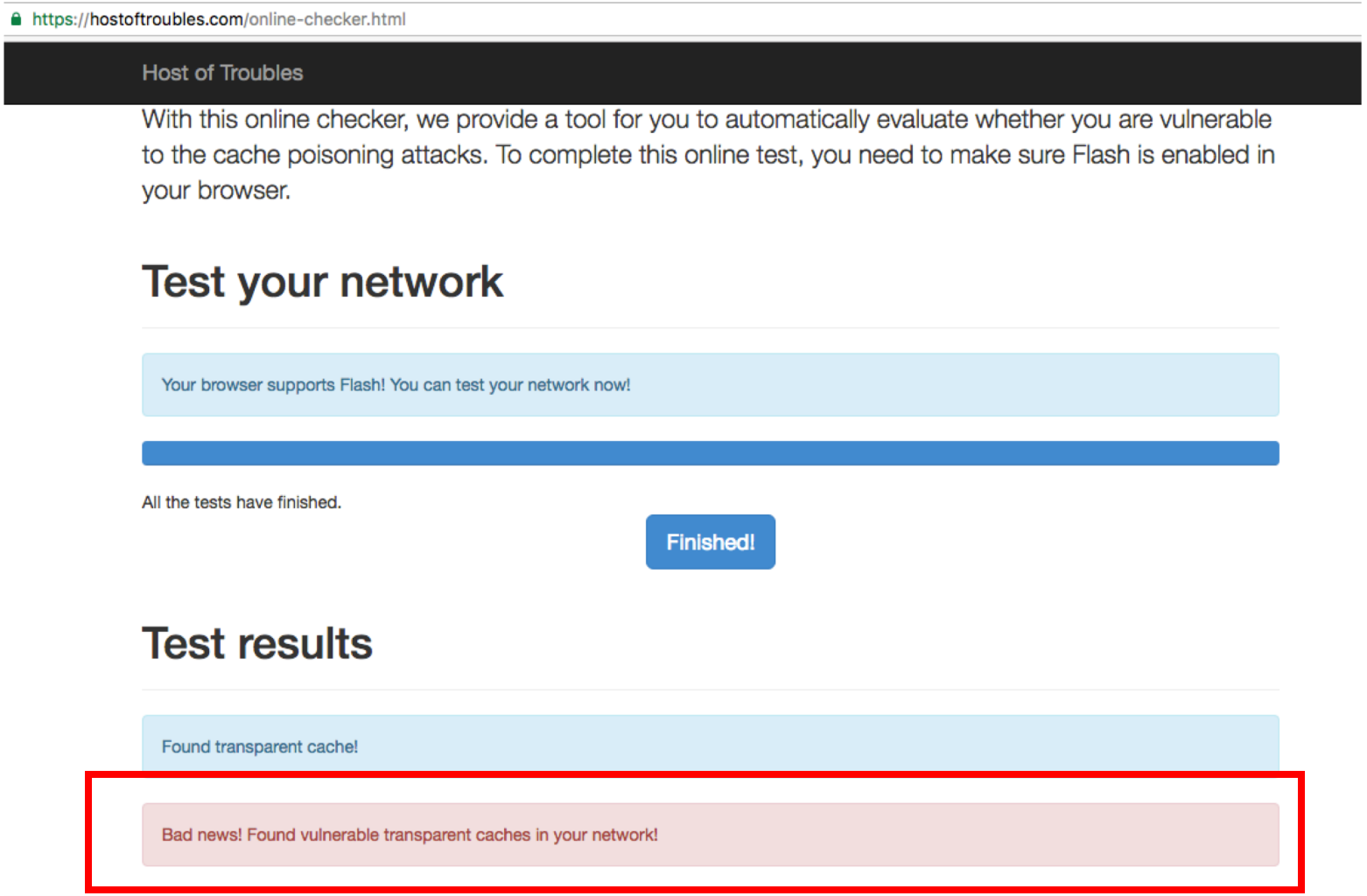
Responsible disclosure

- Cache poisoning
  - Squid： Fixed, CVE-2016-4553, CVE-2016-4554
  - Akamai：Fixed
  - Tencent： Fixed
  - Alibaba：Fixed
  - Apache Traffic Server： Confirmed
- Filter bypass
  - Palo Alto Networks：add new option， Fixed
  - Huawei：add new option， Fxied
  - ESET：Fixed
  - CloudFlare：Fixed
  - Fastly：Fixing

Mitigation

- HTTP standard need to be precise and complete.
- HTTP implementations should fully comply with RFC 7230 to avoid inconsistent.
  - treat multiple Host headers and white-spaces around field-names as errors
- Websites can deploy HTTPS with pre-loaded HSTS to avoid transparent cache.
- For end users, we provide an online tool to check if you are vulnerable to transparent cache poisoning attacks.
  - https://hostoftroubles.com/online-checker.html

A test in my phone's network



Discussion

- Limitations of Postel's law
  - "Be conservative in what you send, be liberal in what you accept"
- Specifications written in natural language inevitably introduce ambiguities
  - Provide reference implementations?
- When designing protocols, we should try to avoid introducing overlapping semantics in protocol fields
  - Rather than resolve such issues by specification rules

- Research Question: Is it possible to automate analysis of consistency between implementation and standard?

Thank you !

Visit https://hostoftroubles.com to see demos.