University of BRISTOL

Systems Security

COMSM1500

bristol.ac.uk

University of BRISTOL

Web Security

…continued

bristol.ac.uk

How would you…

- CCS-based
- Setting: Attacker can get victim to visit his/her website
- Goal: Learn what other website the victim has visited
- Exploit: ???
- Fix: ???

How would you…

- CCS-based
- Setting: Attacker can get victim to visit his/her website
- Goal: Learn what other website the victim has visited
- Exploit: Link colour (visited/not visited)
- Fix: ???

How would you…

- CCS-based
- Setting: Attacker can get victim to visit his/her website
- Goal: Learn what other website the victim has visited
- Exploit: Link colour (visited/not visited)
- Fix: Lie to JS about this (always say unvisited)

How would you…

- Cache-based
- Setting: Attacker can get victim to visit his/her website
- Goal: Learn what other website the victim has visited
- Exploit: ???
- Fix: ???

How would you…

- Cache-based
- Setting: Attacker can get victim to visit his/her website
- Goal: Learn what other website the victim has visited
- Exploit: Time it takes to load candidate objects (e.g. google map tiles)
- Fix: ???

How would you…

- Cache-based
- Setting: Attacker can get victim to visit his/her website
- Goal: Learn what other website the victim has visited
- Exploit: Time it takes to load candidate objects (e.g. google map tiles)
- Fix:
  - No-cache client side (slow website ☹)

How would you…

- Cache-based
- Setting: Attacker can get victim to visit his/her website
- Goal: Learn what other website the victim has visited
- Exploit: Time it takes to load candidate objects (e.g. google map tiles)
- Fix:
  - No-cache client side (slow website ☹)
  - Origin-based cache (seems a good compromise)
  - Maybe non-local cache (company level?)

How would you…

- Rendering engine
- Setting: Attacker can get victim to visit his/her website
- Goal: Learn what other website the victim has visited
- Exploit: ???
- Fix: ???

How would you…

- Rendering engine
- Setting: Attacker can get victim to visit his/her website
- Goal: Learn what other website the victim has visited
- Exploit:
  - Attacker make a copy of target.com/login (when user is logged off)
  - Display login page of victim.com in a frame
  - CCS blur applied on frame
  - Measure how long it takes for the blurring effect to apply (on “real” and copy frame)
  - Difference would indicate if user was already logged in or not
- Fix: ???

How would you…

- Rendering engine
- Setting: Attacker can get victim to visit his/her website
- Goal: Learn what other website the victim has visited
- Exploit:
  - Attacker make a copy of target.com/login (when user is logged off)
  - Display login page of victim.com in a frame
  - CCS shader effect (e.g. blur) applied on frame
  - Measure how long it takes for the blurring effect to apply (on “real” and copy frame)
  - Difference would indicate if user was already logged in or not
- Fix:
  - Apply same-origin policy on shader effect…
  - … prevent possible genuine use cases?

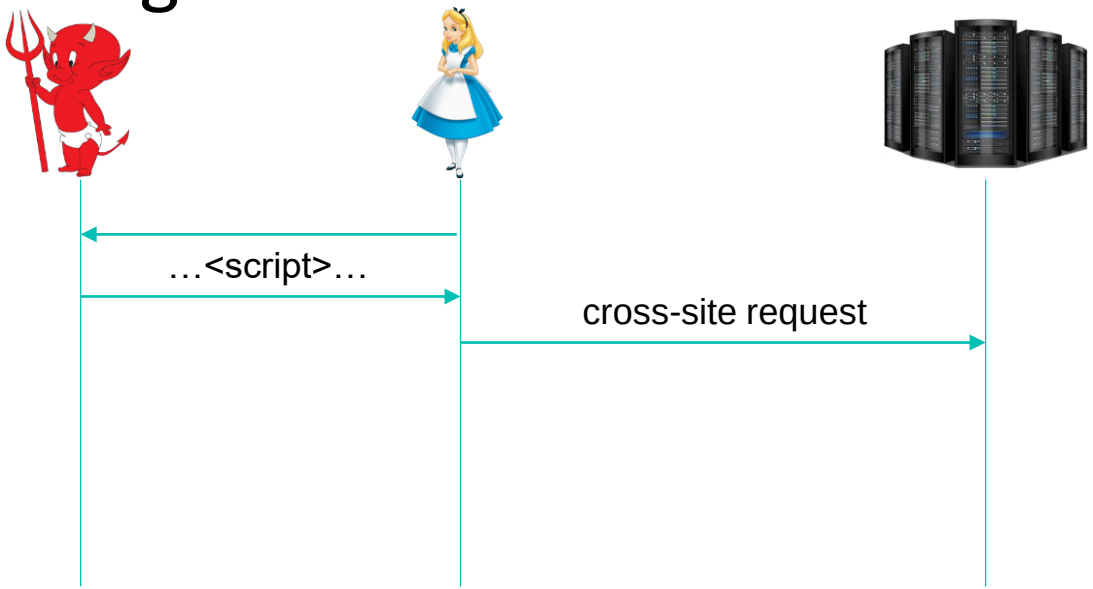
University of BRISTOL

Timing attack

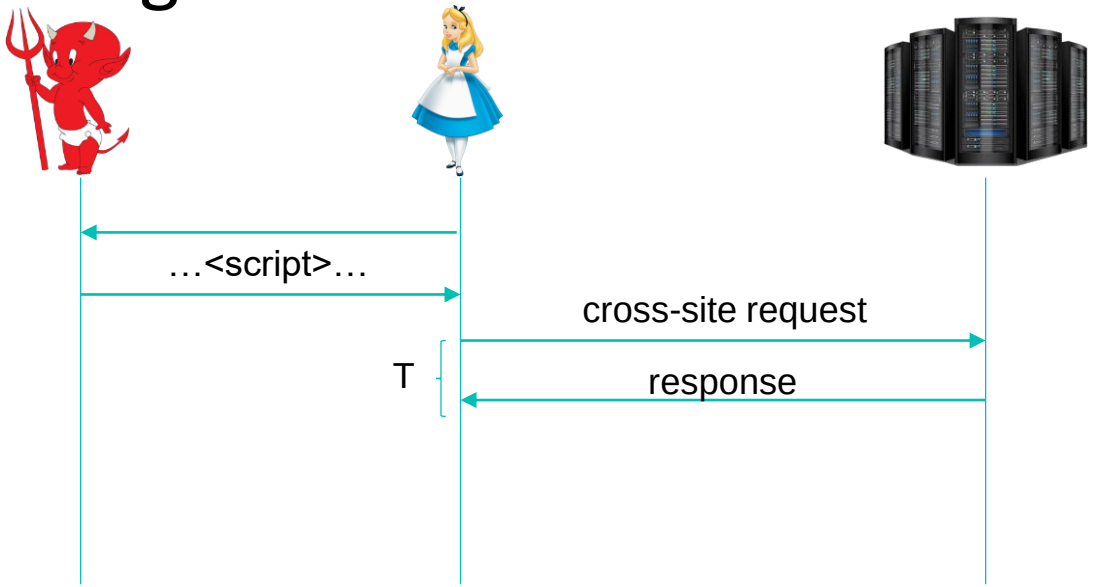
The cache example

bristol.ac.uk

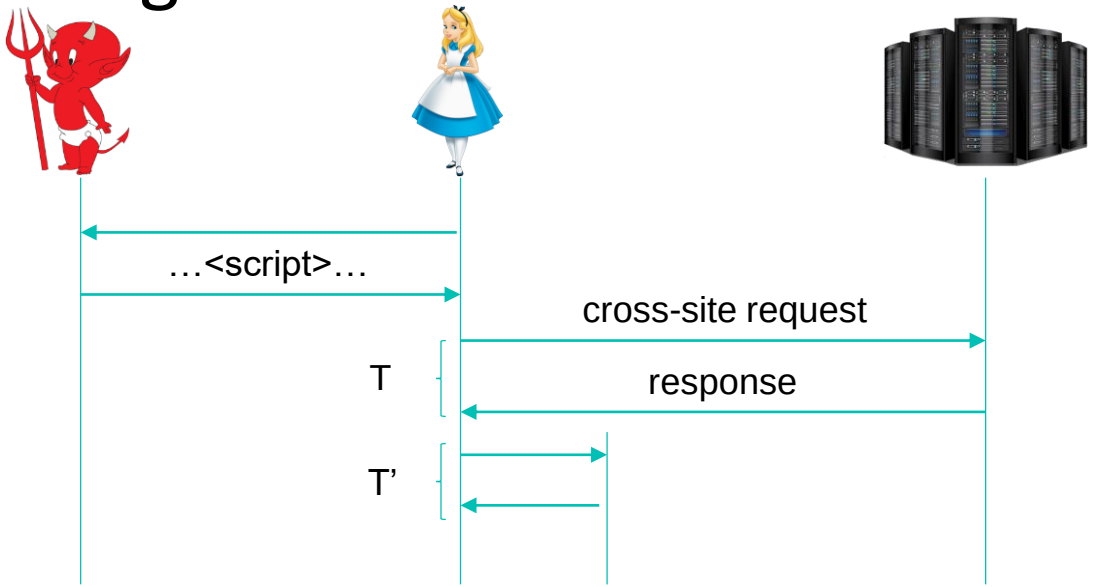
Timing side-channel



Timing side-channel



Timing side-channel



Attacker cannot see the response (same-origin policy), but it can measure delay.

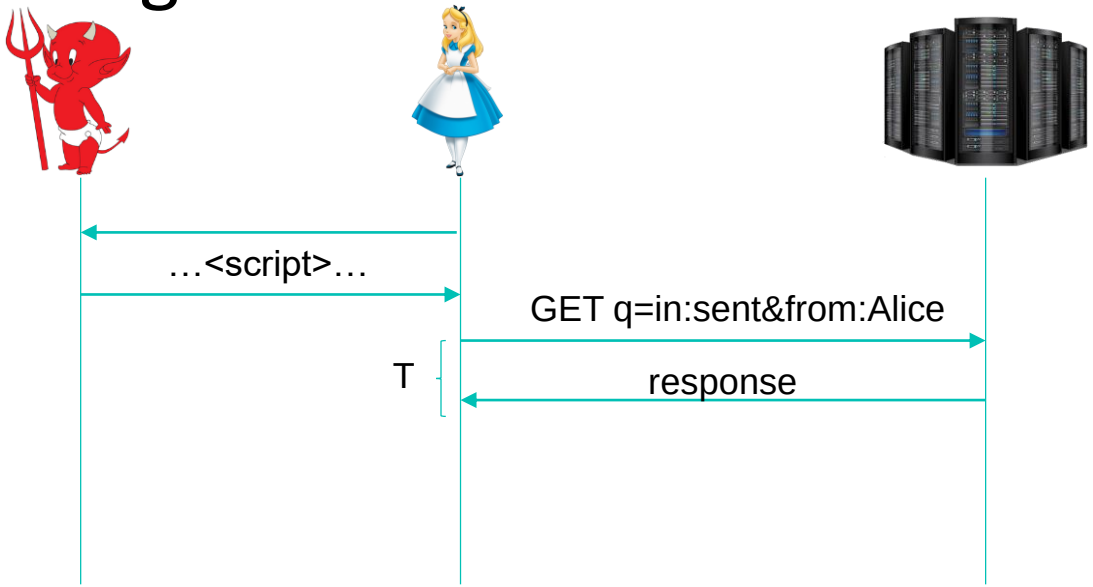
University of BRISTOL

How to build more complex query?

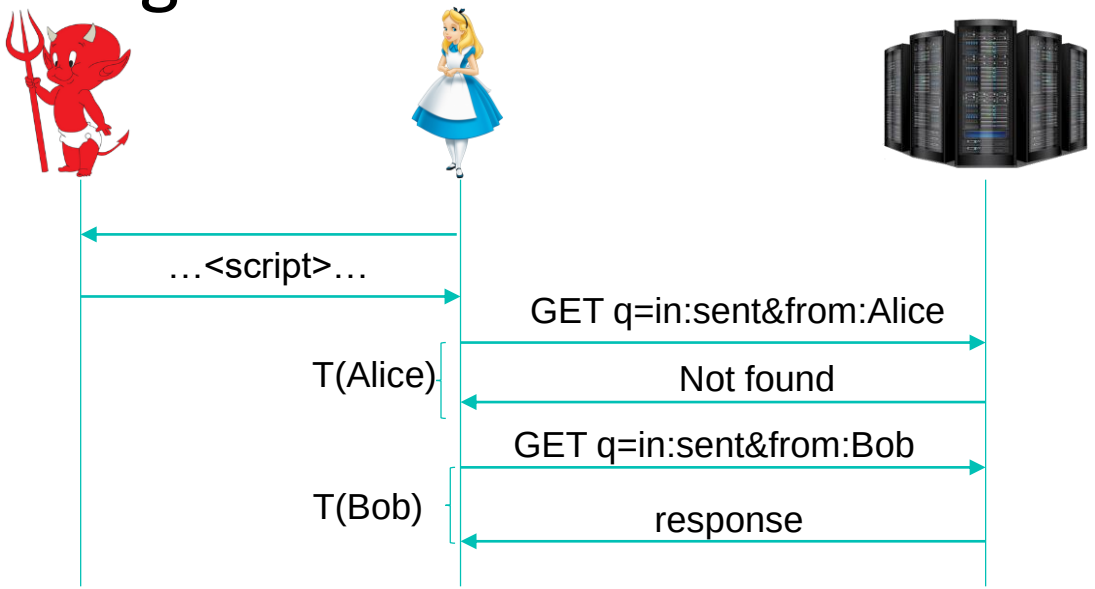
Cross-Site Search Attacks

bristol.ac.uk

Timing side-channel



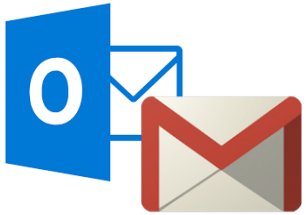
Timing side-channel



- Is user Alice or Bob?
- Compare:
  - T(Bob)
  - T(Alice)

What can we learn?

structured data



e-mail content

– Countermeasure to the attacks discussed today are in place on major platforms… however, this was true 3-4 years ago.

Bing

Search history

Gooooooooooogle
1 2 3 4 5 6 7 8 9 10

relationships
(follows…)

XS-search basic flow

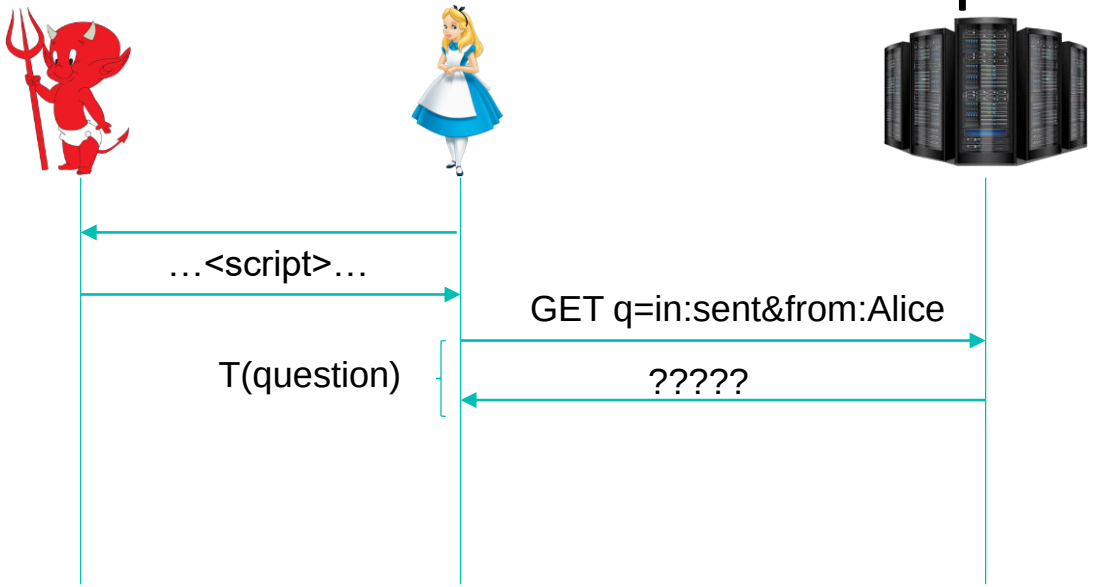
- Goal: find the answer for a boolean question

- Three steps:
  1. Transform the question into a search request
  2. Send search requests and collect samples
  3. Analyse response time -> Answer the question!

XS-search basic flow – 1st step

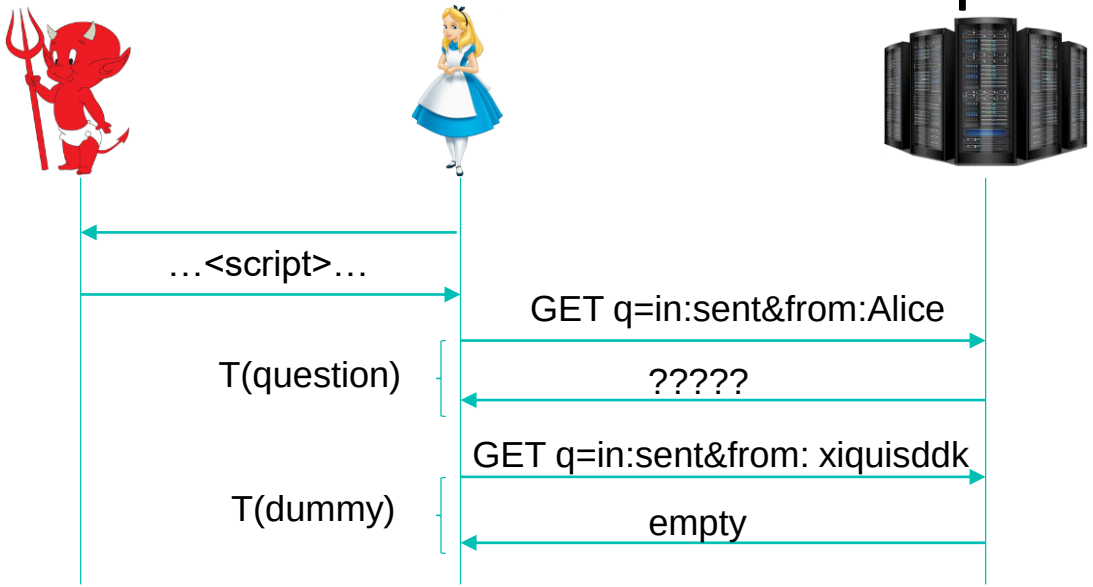
- Is the name of user Alice?
  - in:sent from: Alice
- Is she related to bob@gmail.com ?
  - bob@gmail.com&st=100
- Does Alice have an affair with Charlie?
  - “I love you” to Charlie
- etc…

XS-search basic flow – 2nd step



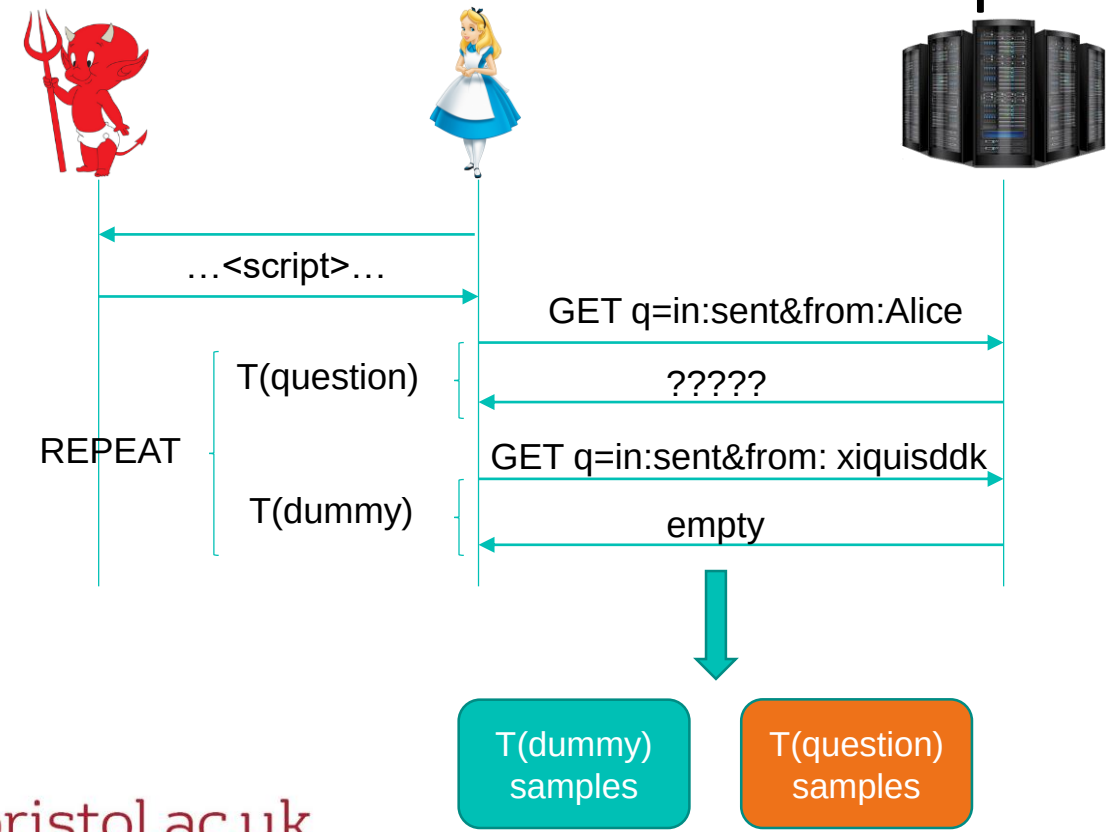
- Send our question
- Is the user name Alice?
  - True: a full response is returned (has some content)
  - False: an empty (short) response is returned

XS-search basic flow – 2nd step



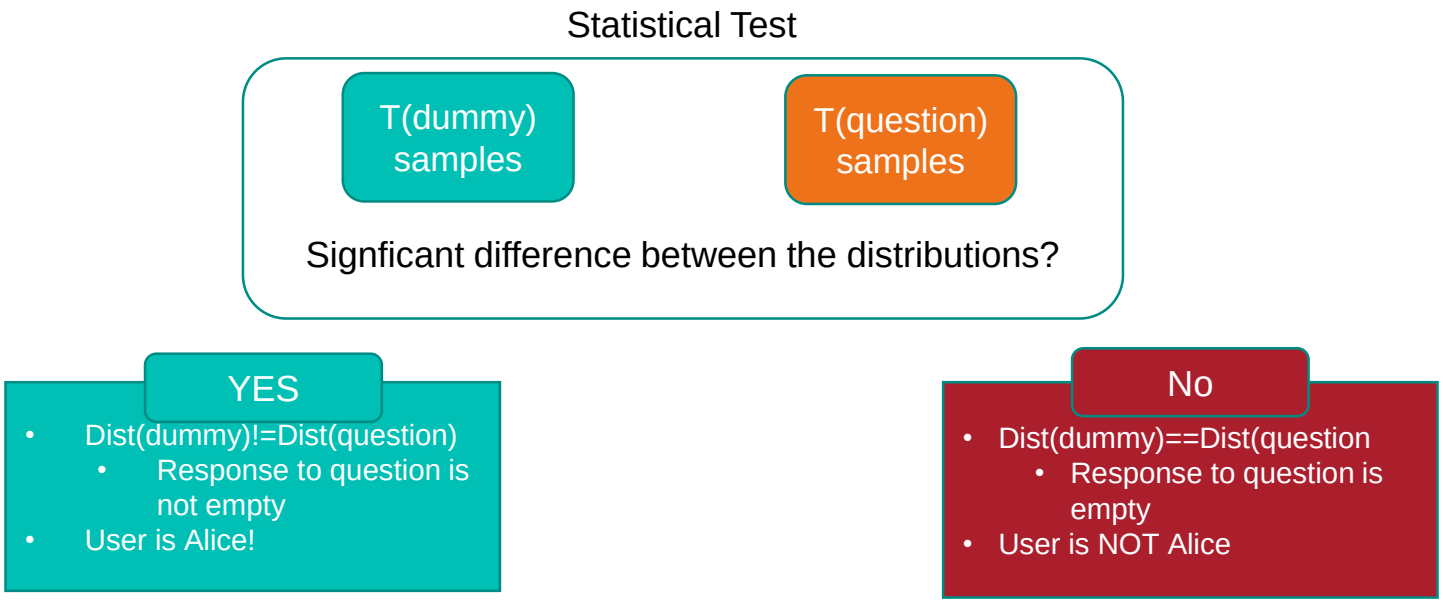
- Send a dummy request
- Is the user name xiquisddk?
  - Expected to be false

XS-search basic flow – 2nd step



- Send a dummy request
- Is the user name xiquisddk?
  - Expected to be false

XS-search basic flow – 3rd step



Practical timing attacks: challenges

- Delays depends dynamically changing factors: congestion and concurrent request on client and server
- Minimal time
  - Exploit even short visit of urers
- Minimal number of requests
  - Avoid detection and blocking
    - e.g. server anti-DOS defenses

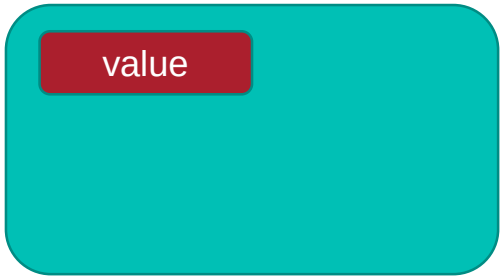
Solution: response inflation!

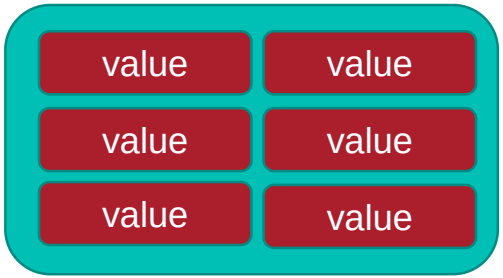
- Idea: make it easier to distinguish between empty and full response
- Increase the difference in size between the two
- Larger difference in size -> larger difference in time

Response inflation

- Search requests have many parameters
- Some are reflected in the response as a function of the number of results

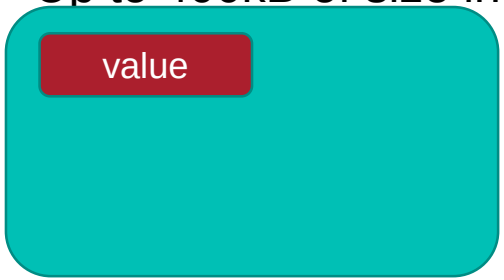
https://example.com/search?reflected_parameter=value

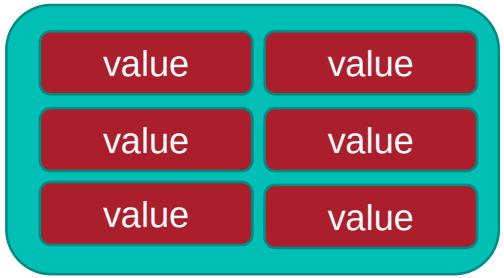




Response inflation

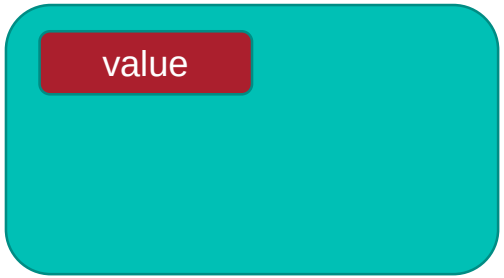
- Send very long string as reflected parameter
- In Gmail used to be the query itself
  - Appear once for each entry (50 max per default)
  - Can be inflated to 8kB
- Up to 400kB of size inflation!

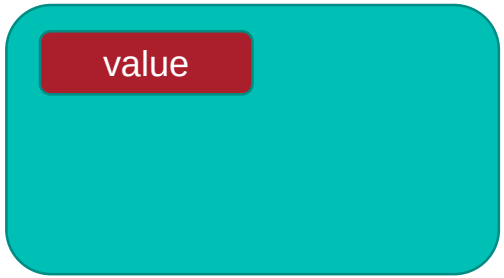




Response inflation

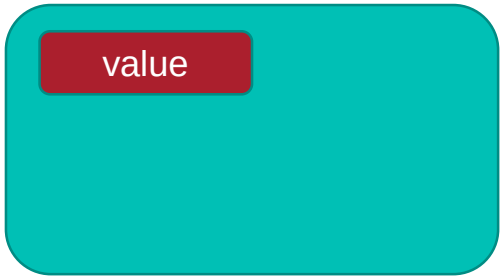
- What if there is no reflected parameter available?
- Empty or full have nearly identical size.

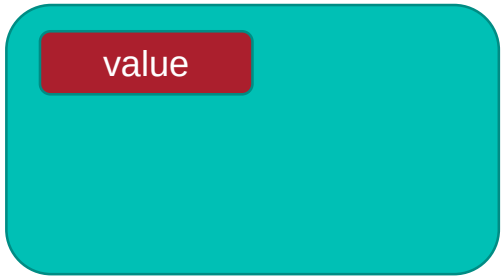




Response inflation

- What if there is no reflected parameter available?
- Empty or full have nearly identical size.
- Computational inflation (use expense query)
  - dummy: in:sent&from:xiquisddk&hasnot:{rjew+...+iqejh}
  - question: in:sent&from:alice&hasnot:{rjew+...+iqejh}





Avoiding detection

- Classical timing attack:
  - Attacker send request directly to the server several time to do the measurement
- serviceWorker.cache
- Browser-based timing attack
  - Load from the server once
  - Do the statistical analysis based on cache return time

Avoiding detection (limitation)

- Measure affected by:
  - Classic: network delay, server processing time, browser processing time
  - Browser-cache: browser processing time

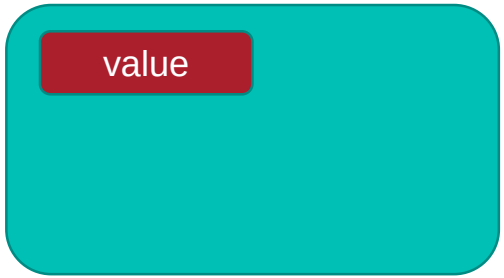
- Can be used to differentiate between:
  - Classic: large/small resources; long/short processing time
  - Browser-based: large/small resources

Optimized Multiple Terms Identification

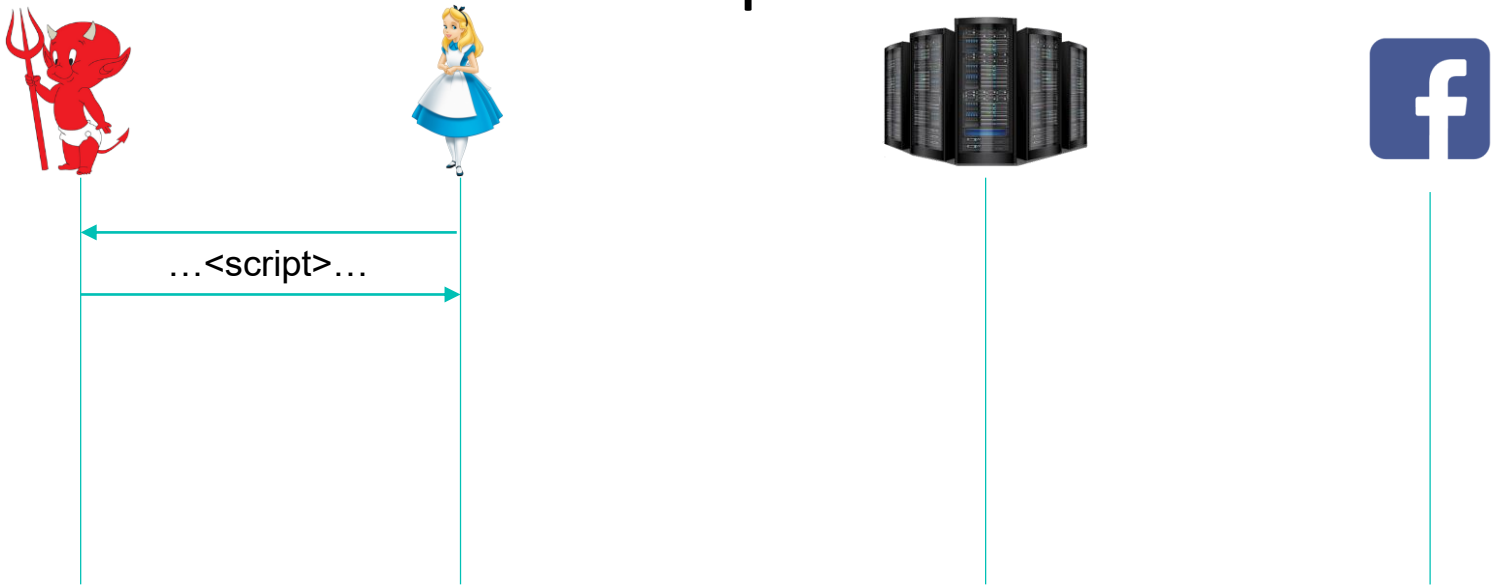
- Term from a list of candidate terms
  - e.g. list of family name / given name combination
- Classic divide and conquer
  - from:michael+OR+dan+OR+.... Up to the URL limit
- Identify each term individually
  - e.g. first name then given name
- Demo https://www.youtube.com/watch?v=9wgLbUet5Wk
  - 2000 entries dictionary

What if inflation is not possible?

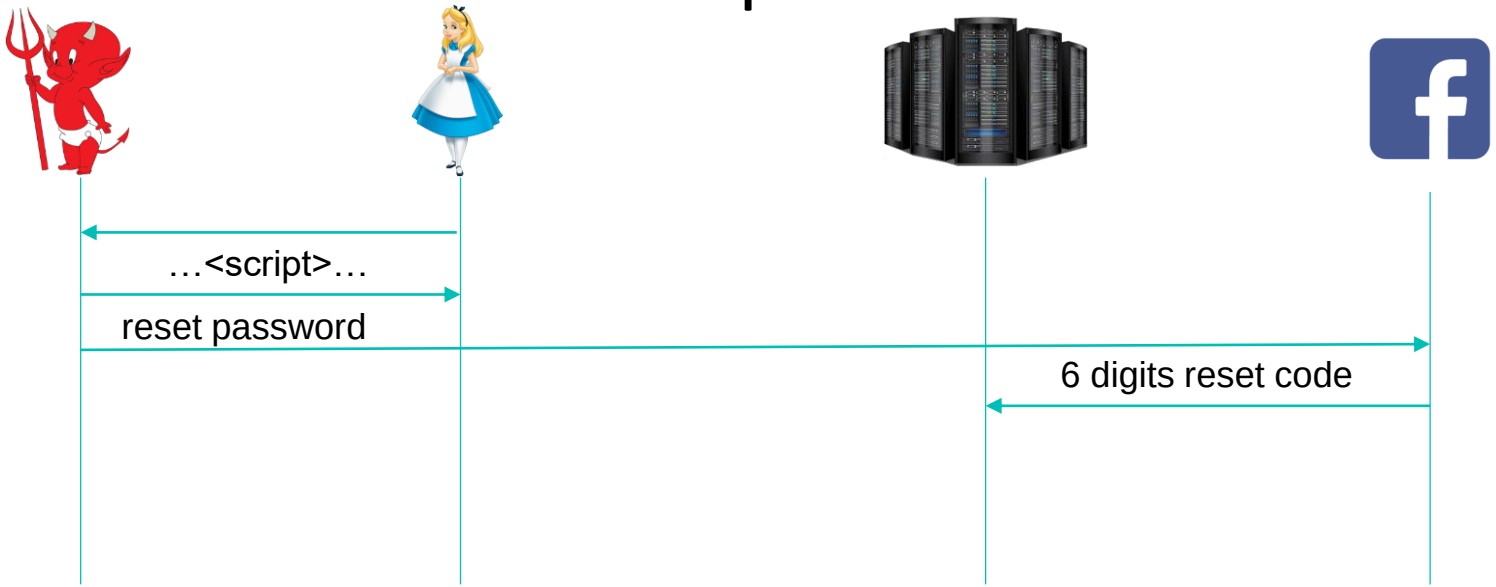
- Barely no difference between full or empty
- The attacker can manipulate the target by making the value appear several time?
  - e.g. e-mail (anyone can send you an e-mail)



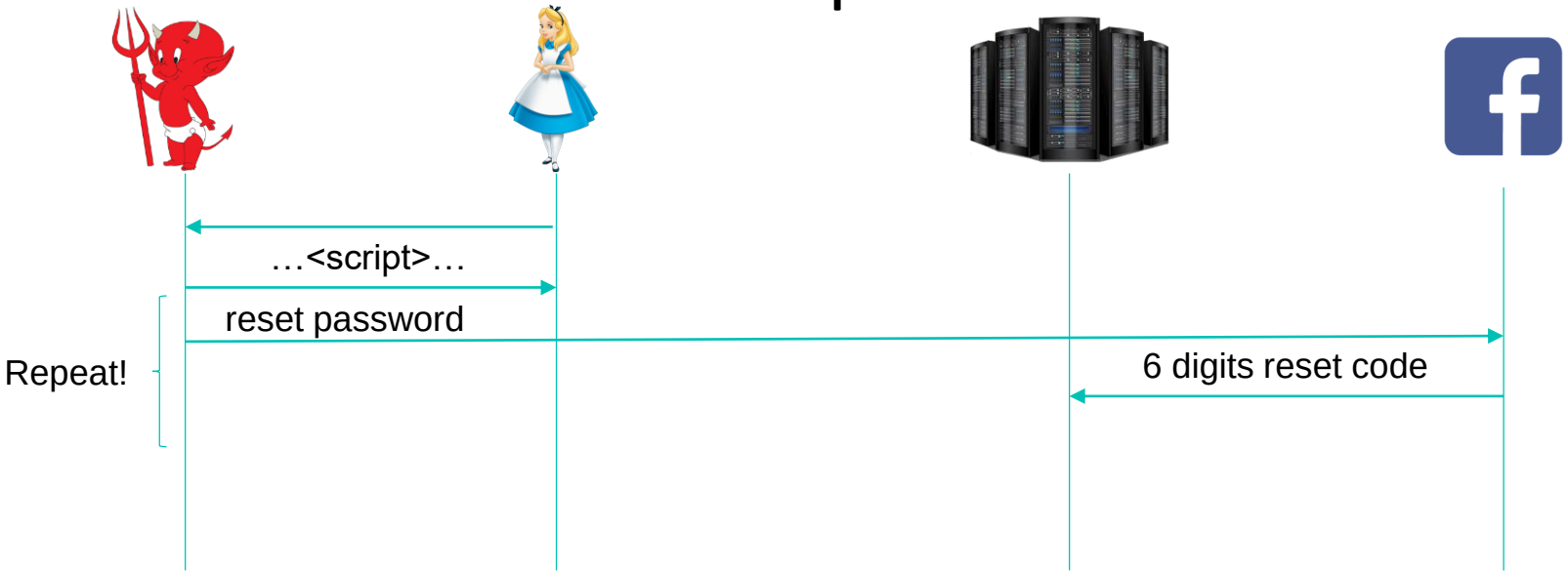
What if inflation is not possible?



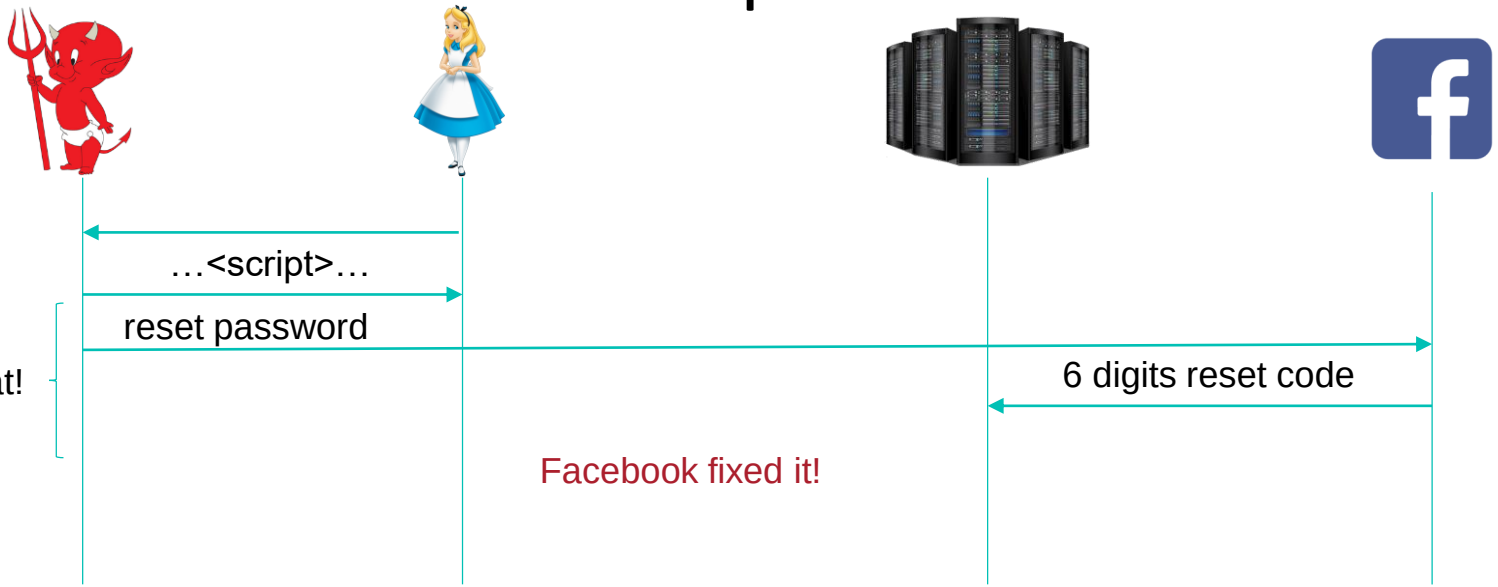
What if inflation is not possible?



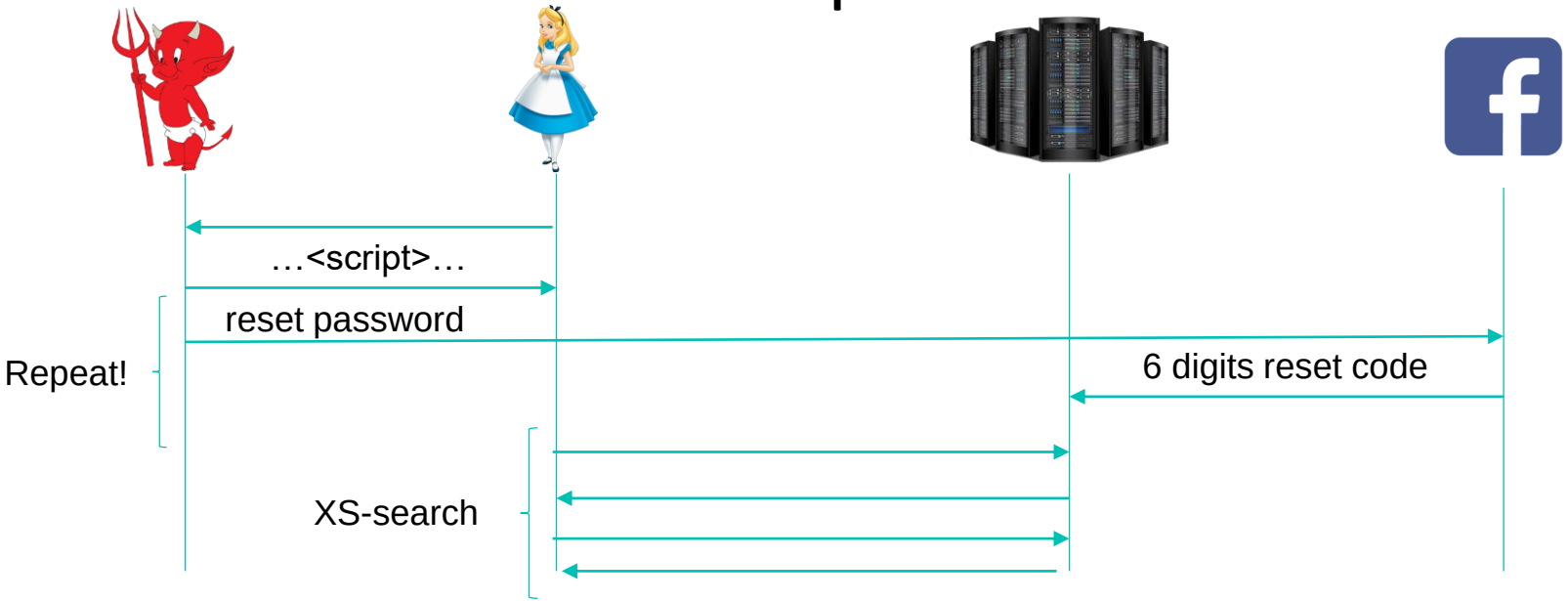
What if inflation is not possible?



What if inflation is not possible?



What if inflation is not possible?



Problem?

- Rely on a vulnerability in the target service
  - e.g. send the same code multiple time
- What if there is no vulnerability?

Problem?

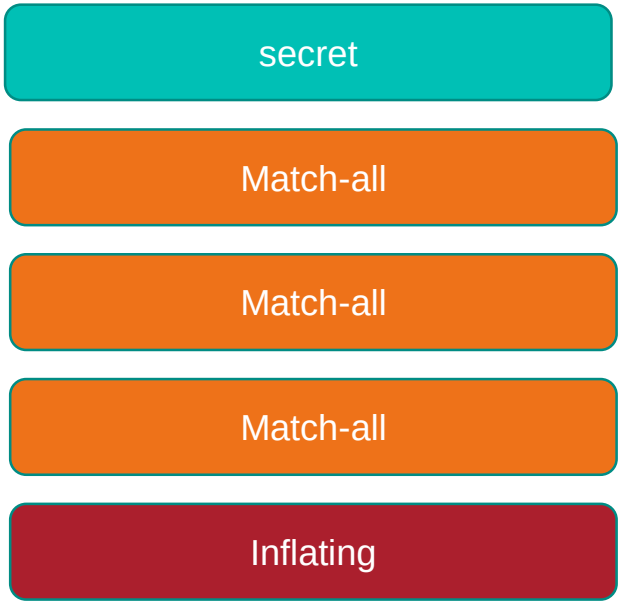
- Rely on a vulnerability in the target service
  - e.g. send the same code multiple time
- What if there is no vulnerability?
- Vocabulary
  - M: maximum number of entry in a response
  - Match-all record: a record that matches all possible entries
  - Inflating record: a match-all record that significantly increase the message size

Solution

- Attack process
  - Part 1
    - Plant one match-all inflating record
    - Plant M-1 match-all record
    - Reset the password (or whatever)
  - Part 2
    - Perform XS-Search

Solution

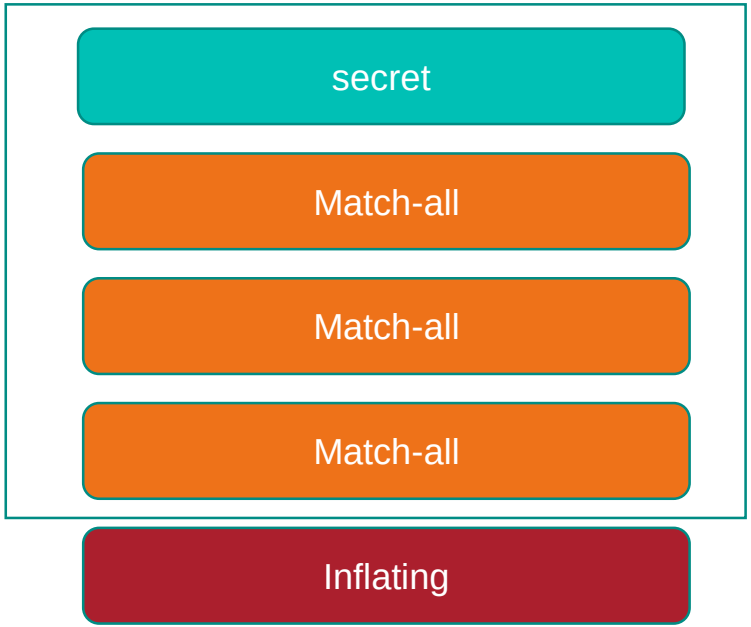
M=4



Solution

query = secret

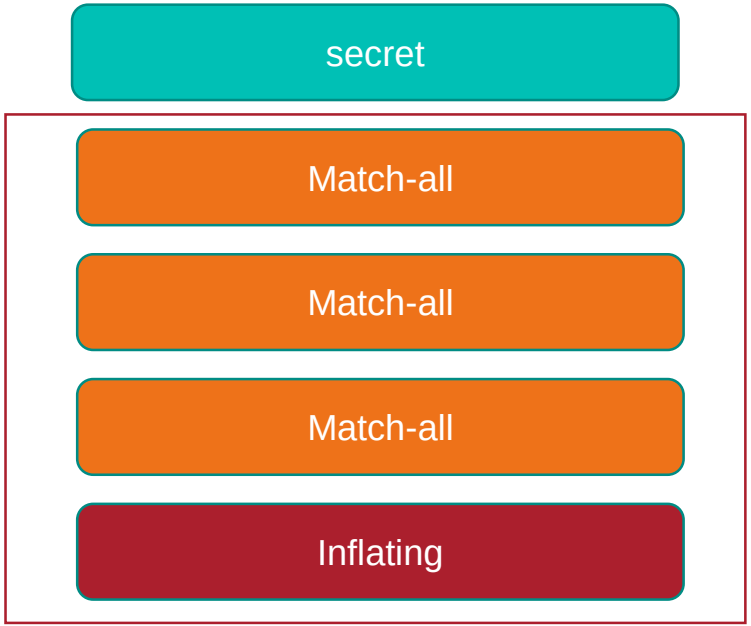
M=4



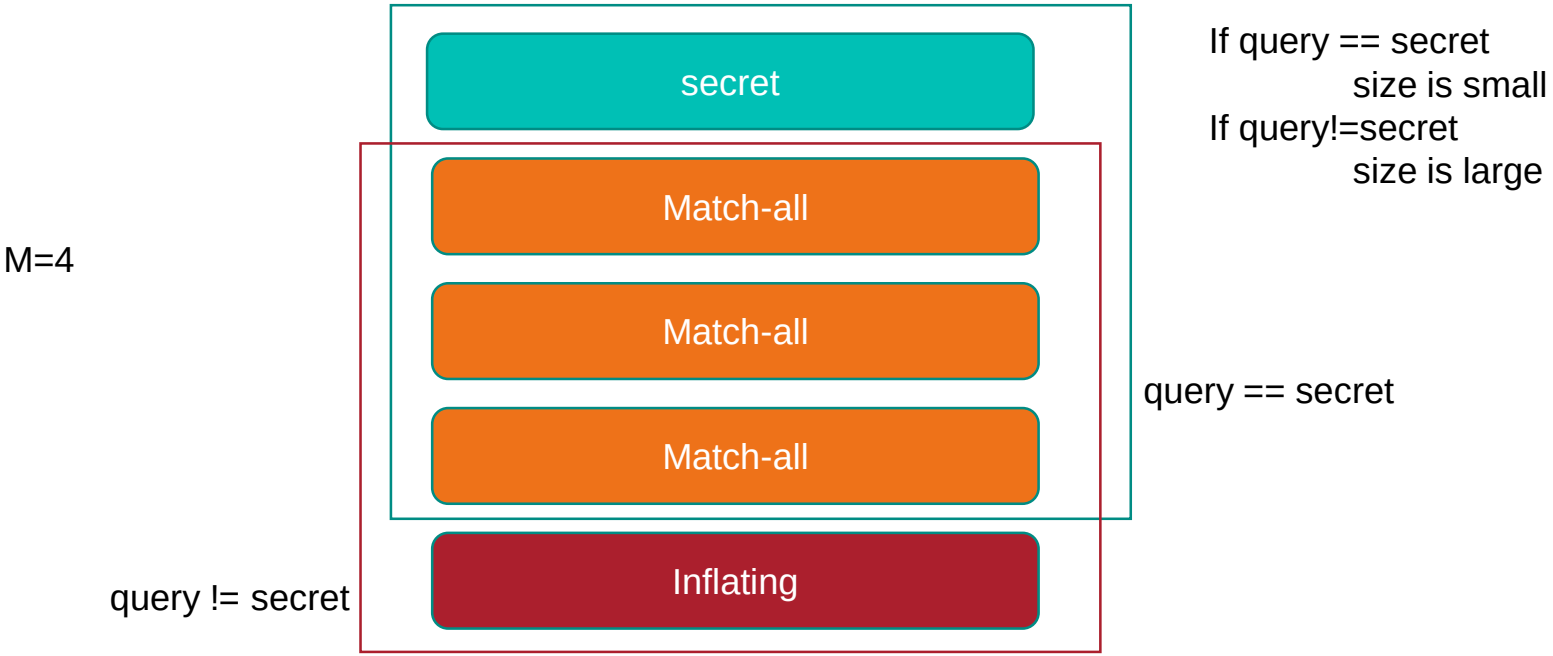
Solution

query != secret

M=4



Solution


secret
Match-all
Match-all
Match-all
Inflating
M=4
query != secret
query == secret
If query == secret
size is small
If query!=secret
size is large

Checkout papers on github

- https://github.com/bris-sys-sec/docs/tree/master/timing

- Read the papers it is useful for exams

Solutions?

- Don’t allow external queries
  - Does not work for a large class of applications
- Reduce query expressivity
  - Has impact on usability/utility of a system
- Rate limit query
  - Attacker can go around this using browser cache (as we have seen)
- Detection?
  - Attack detection is an active area of research, but it is difficult

Timing attack outside the web

- You should read about it ;)
- Active area of research in crypto
- UNIX Login (to identify existing users)
- On GPU (they are not only used for gaming)
- etc…

Conclusion

- Even program with no vulnerability can be compromised
- Timing attacks are an example of side channel attacks
  - They leverage implementation properties to extract secrets
- Side channel attacks are an active area of research
  - Power
  - Electromagnetism
  - etc…
- They do not always have easy fix

University of BRISTOL

Thank you

Office MVB 3.26

bristol.ac.uk