

Timing Attacks Have Never Been So Practical:

Advanced Cross-Site Search Attacks

Nethanel Gelernter

CYBERP♟ON



About me: Nethanel Gelernter

- Security Researcher / Hacker
 - Web application security
 - Ph.D., hacks, research papers, talks, etc.

- Cyberpion



- Exploring new attack vectors & developing defenses against them

- Leading the cyber-security studies & research in the College of Management, Israel



The College of Management
Academic Studies

Agenda – practical timing attacks

- Cross-site search (XS-search) attacks & Response inflation
- Challenges
 - When response inflation is impossible
- Browser-based XS-search attacks
- Second-order XS-search attacks

Cross-Site Search Attacks

- Gelernter & Herzberg, CCS' 2015
- Exploit 'search' timing side-channel
- 'Search' in private-data kept by web-service
- **Practical:**
 - Tested on hundreds of Gmail users
 - Real world conditions
- Example: find **user name**
 - From lists of 2000 common (first and last) names
 - Takes about a minute

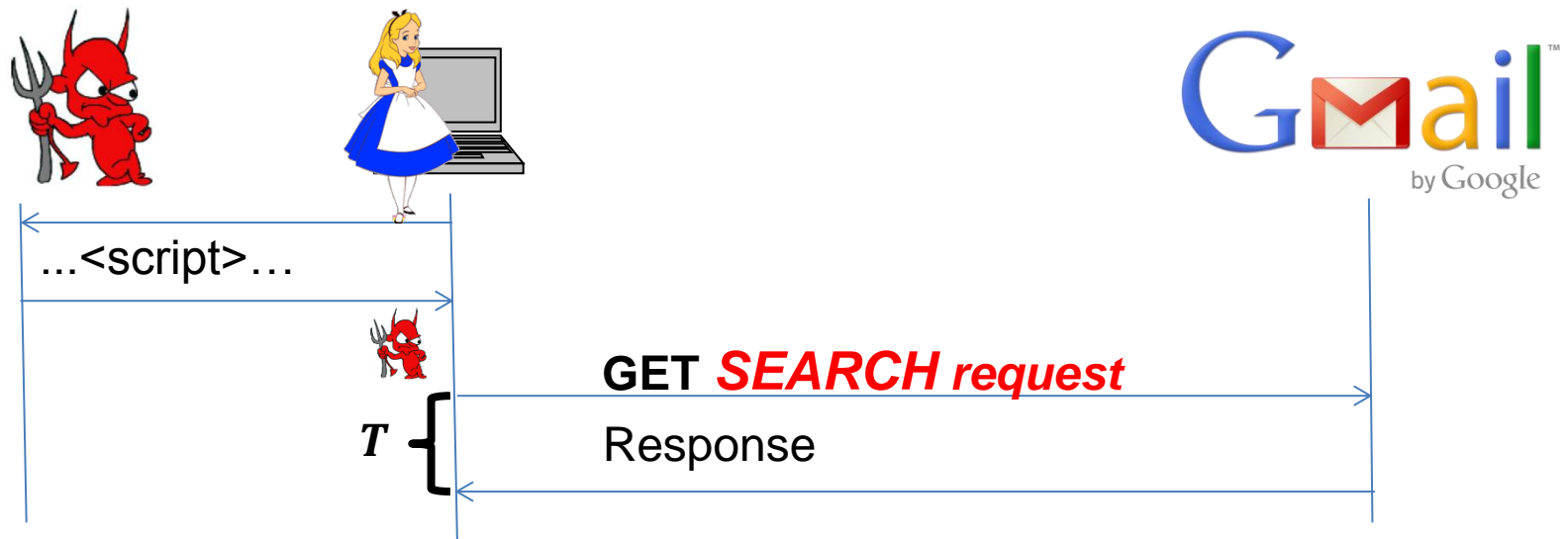
Cross-site attacker model

- Main model for web attacks
- The victim's browser is authenticated to services that hold private records (e.g., Gmail)
- The victim visits the attacker's website



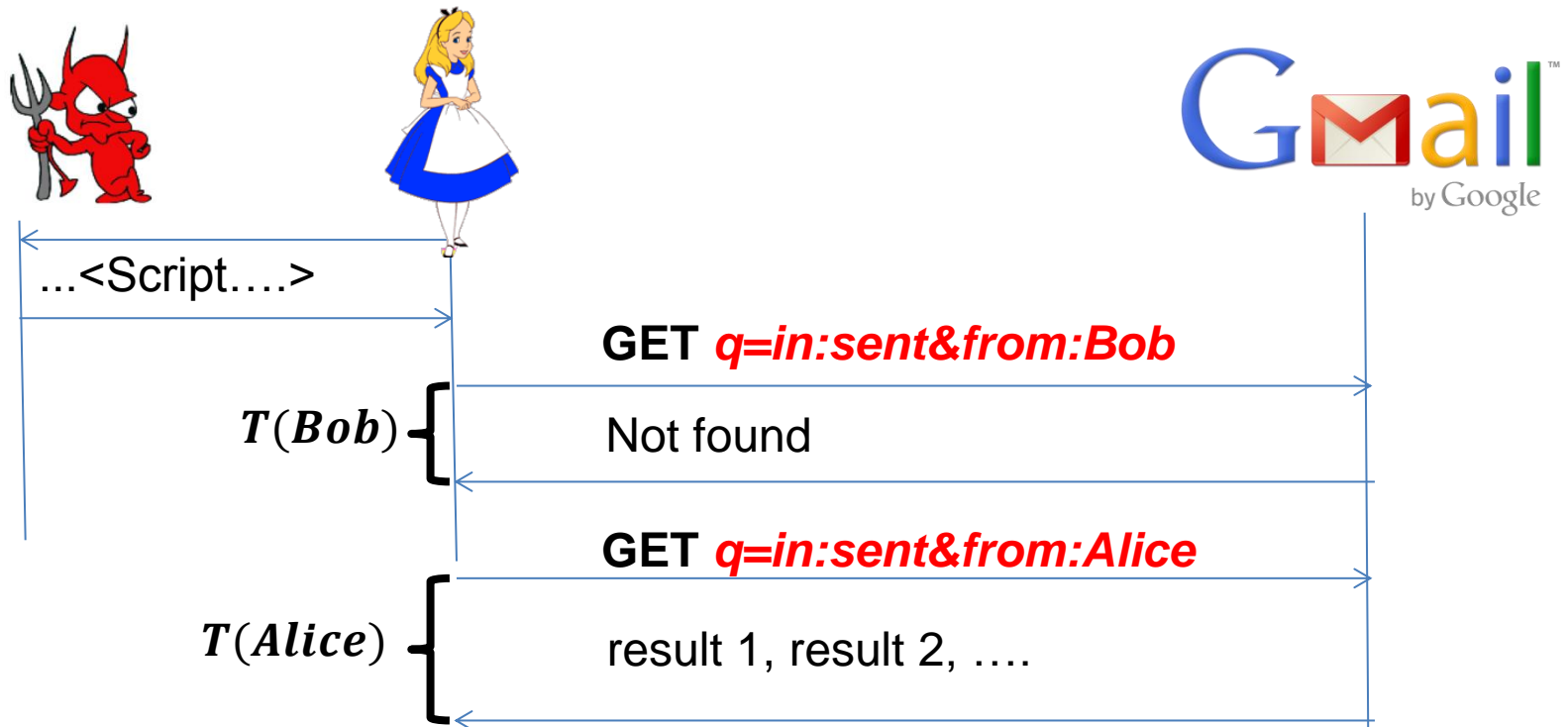
Cross-site attacker model

- Cross-site search over user's data in service
 - Attacker cannot access the content of the response
 - Same Origin Policy
 - The attacker can measure the response *time* (T)



XS-Search example: user name

- Find out whether the user is Alice or Bob...
- Compare:
 - $T(\text{Bob})$: response time for 'messages sent by Bob'
 - $T(\text{Alice})$: response time for 'messages sent by Alice'



What else can XS-Search expose?

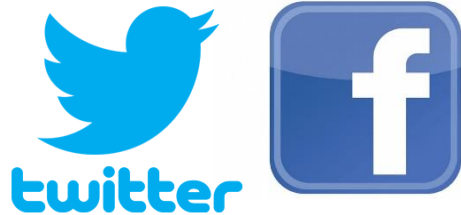
Structured information



Contacts

Name

Email content



**Relationships
(follows, ...)**



Search History

XS-Search: Basic Flow

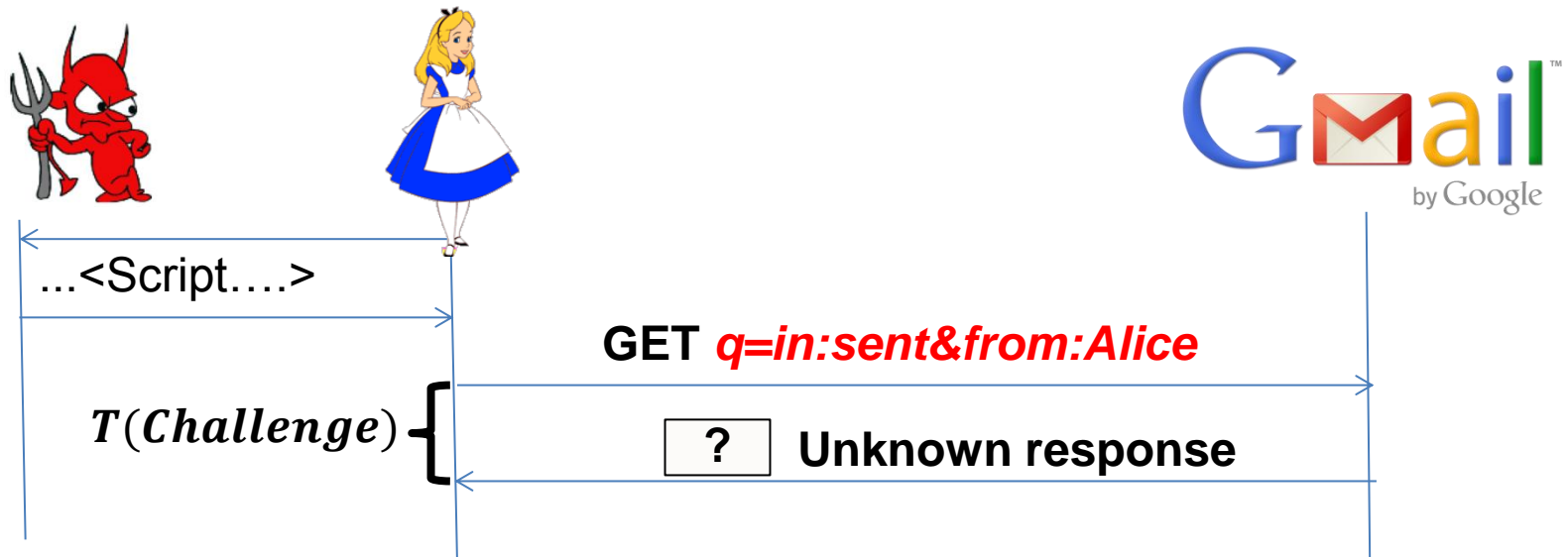
- Find the answer for a Boolean question
- Three steps:
 - Transform the question into a search request
 - Send search requests and collect samples
 - Analyze response times → answer the question!

XS-Search: Basic Flow – 1st Step

- Is the name of the 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 from:Alice

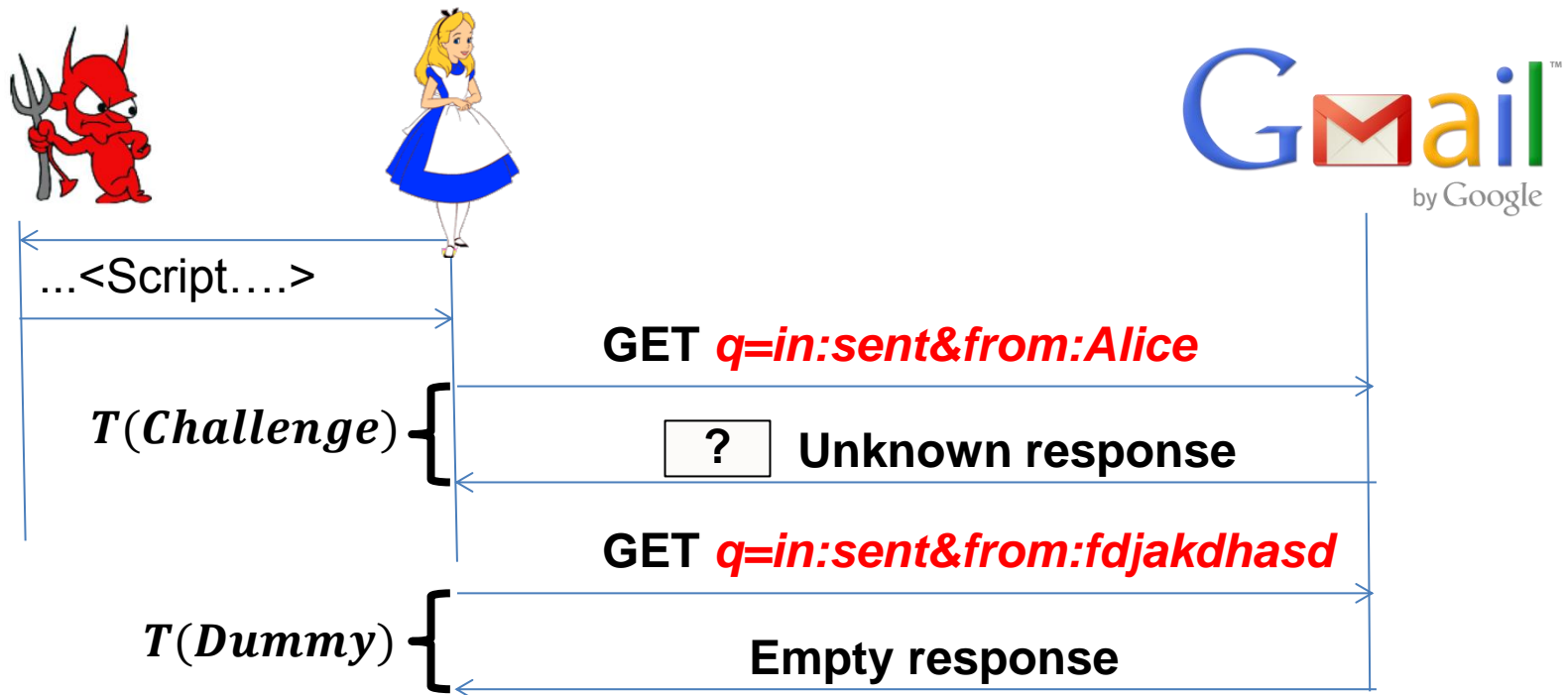
XS-Search: Basic Flow – 2nd Step

- Send a **Challenge** request
 - Is the user name *Alice*?
 - True: a **Full** response is returned (has some content)
 - False: an **empty** response is returned

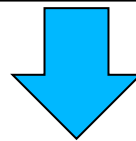
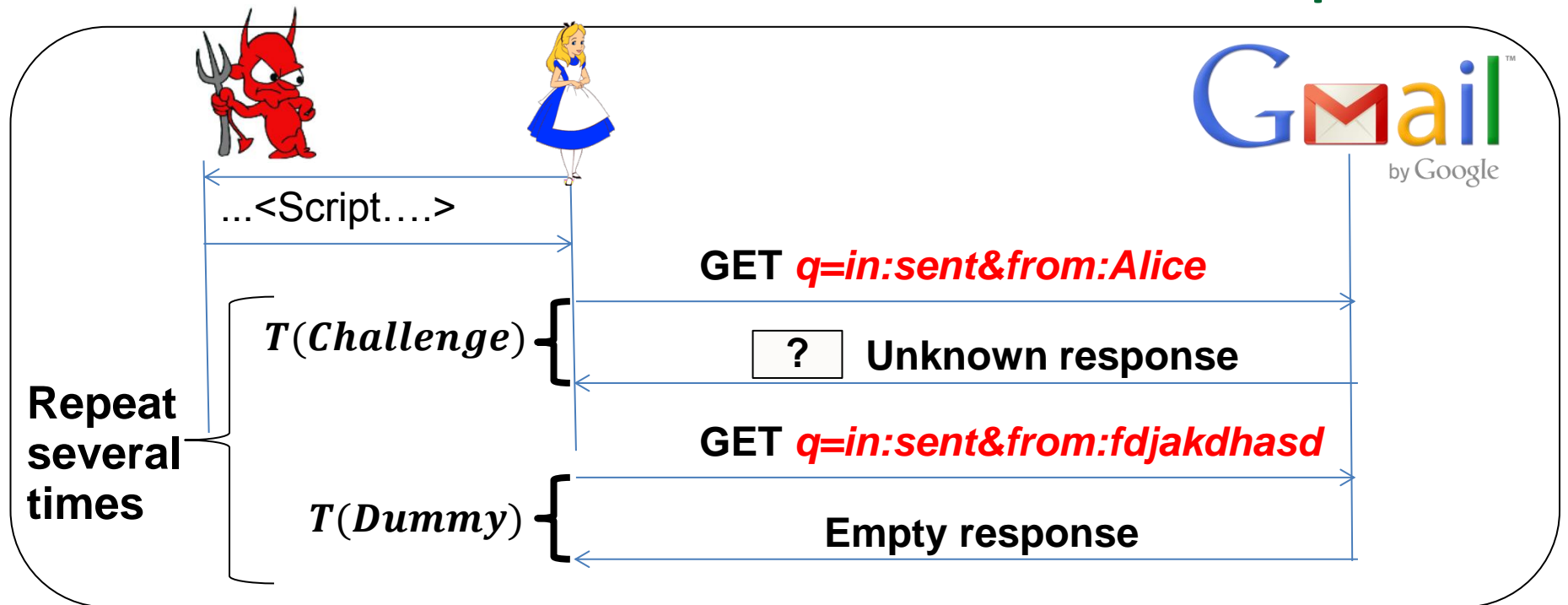


XS-Search: Basic Flow – 2nd Step

- Send a **Dummy** request
 - Is the user name *fdjakdhasd*?
 - The response is expected to be **empty**



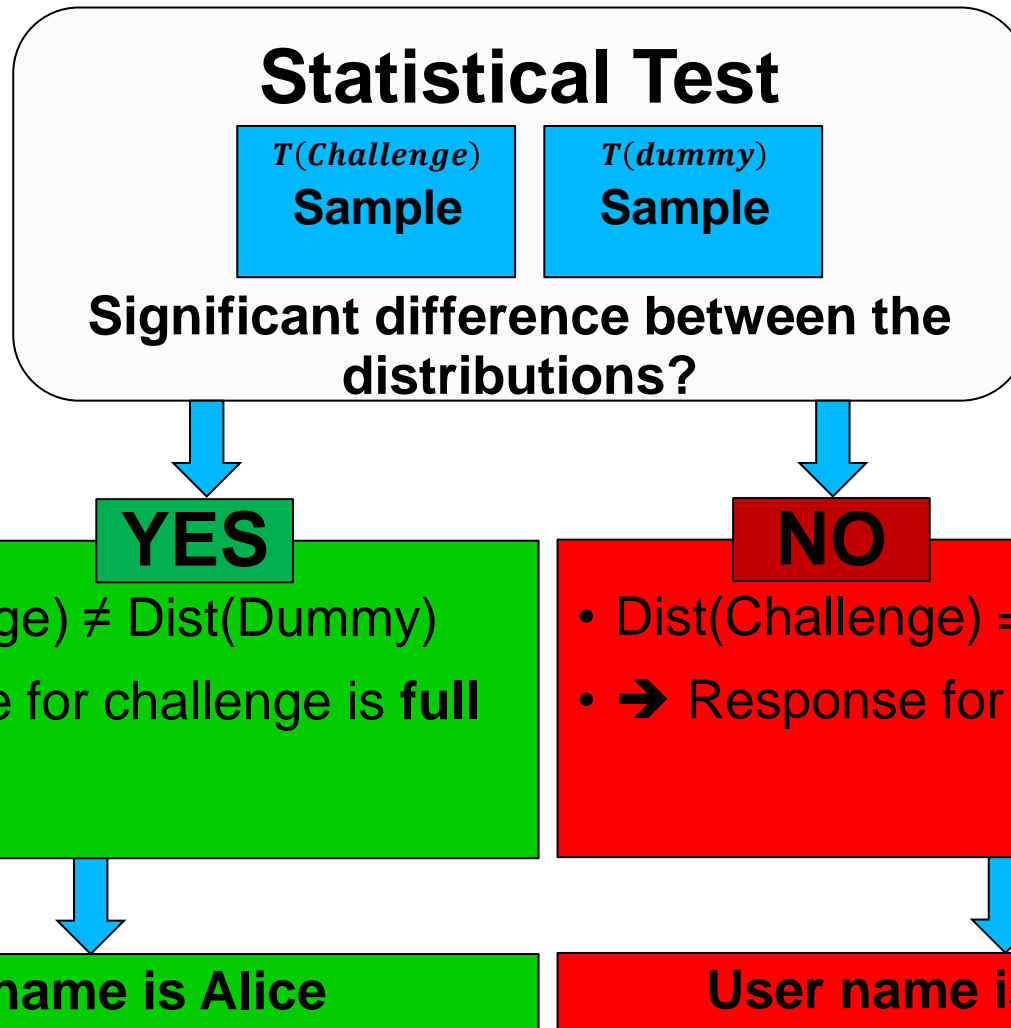
XS-Search: Basic Flow – 2nd Step



T(Challenge)
Sample

T(dummy)
Sample

XS-Search: Basic Flow – 3rd Step



Practical timing attacks: challenges

- Timing attacks
 - Delays depend on dynamically-changing factors, e.g.:
Congestion and concurrent processes in client and server
- Practical attacks
 - Minimal time
 - Exploit also short visits of users
 - Minimal number of requests
 - Avoid detection and blocking
 - E.g., by server's anti-DoS defenses

Response Inflation

- Increase the size difference between **full** and **empty** responses
- Larger difference in size → Larger difference in time

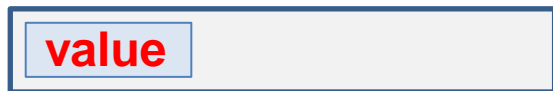


Larger → Slower

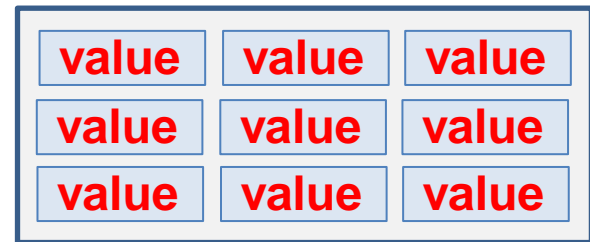
Response Inflation

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

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



Empty response



Full response

Response Inflation

- Sometimes, the attacker send **very long strings** as the value of the reflected parameter

https://example.com/search?reflected_parameter=Long string

Long string.....

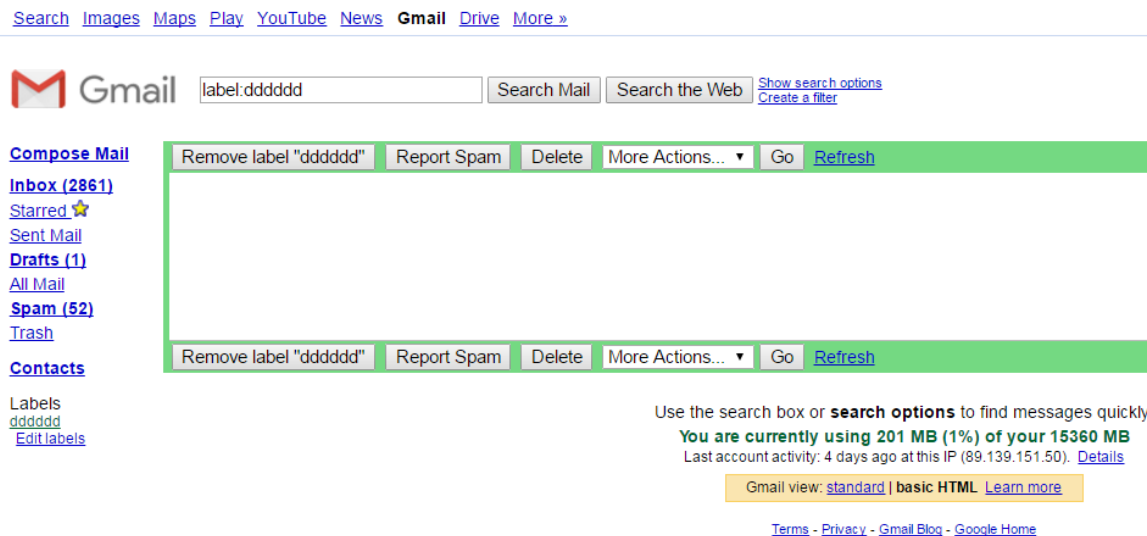
Empty response

Long string.....
Long string.....
Long string.....
Long string.....
Long string.....
Long string.....
Long string.....
Long string.....
Long string.....

Full response

Response inflation example

- Exploiting Gmail search in the HTML view
- The query itself!
 - Appears once for each entry (50 max by default)
 - Can be inflated to 8KB
- Up to 400KB response size inflation!



The screenshot shows the Gmail search interface. At the top, there are navigation links: Search, Images, Maps, Play, YouTube, News, Gmail, Drive, and More. The Gmail logo is on the left, and a search box contains the text 'label:dddddd'. To the right of the search box are buttons for 'Search Mail' and 'Search the Web', along with links for 'Show search options' and 'Create a filter'. Below the search box, there are two identical message entries. Each entry has a green header bar with the following actions: 'Remove label "dddddd"', 'Report Spam', 'Delete', 'More Actions...' (with a dropdown arrow), 'Go', and 'Refresh'. The main content area of each message is empty. At the bottom of the page, there is a footer with the text: 'Use the search box or search options to find messages quickly! You are currently using 201 MB (1% of your 15360 MB) Last account activity: 4 days ago at this IP (89.139.151.50). Details'. Below this is a yellow box with the text: 'Gmail view: standard | basic HTML Learn more'. At the very bottom, there are links for 'Terms - Privacy - Gmail Blog - Google Home'.

But...



What if there is no response inflation?



in:sent from:{Lionel Lisa Litzy Livia L Search Mail Search the Web Show search options Create a filter



We cannot complete the action at this time. Please try again using the search action above.

Compose Mail

Inbox (3)

Starred

Sent Mail

Search results for: in:sent from:{Lionel Lisa Litzy Livia Lizbeth Logan Lola London Londyn Long Lorelai Lorelei Lorenzo Louis Lu Luca Lucas Luc Shiloh Shokri Sidney Siena Sienna Sierra Silas Simeon Simon Simone Sincere Sky Skye Skyla Skylar Skyler Sloan Sloane Sofia Sofie Solomon Son ihzqopynegbvzkuioijgtlzxoqlmentadoxmkafvsallfeiemucpbzmqcxltvrwyejrmtgoahaadepmoaqobtilompnrkkyasufcttkmlfxsijpnvtqnlauapafpwasg

Report Spam Delete More Actions... Go Refresh

We cannot complete the action at this time. Please try again using the search action above.

What if there is no response inflation?

- Browser-based XS-search
 - When there is **some** difference in the response size
- Second-order XS-search
 - When there is **no** difference in the response size!

Browser-based (BB) XS-Search

- Statistical tests and divide and conquer algorithms
 - Gelernter & Herzberg, CCS' 2015
- Browser-based timing side channel
 - Van Goethem et al., CCS' 2015
- Algorithmic improvements



Classical vs. BB timing attacks

- Classical timing attacks:
 - Load the resources from the server several times to collect time measurements
- Browser-based timing attacks:
 - Load all the resources from the server once and cache them
 - Then load them from the cache many times to collect time measurements

Classical vs. BB timing attacks

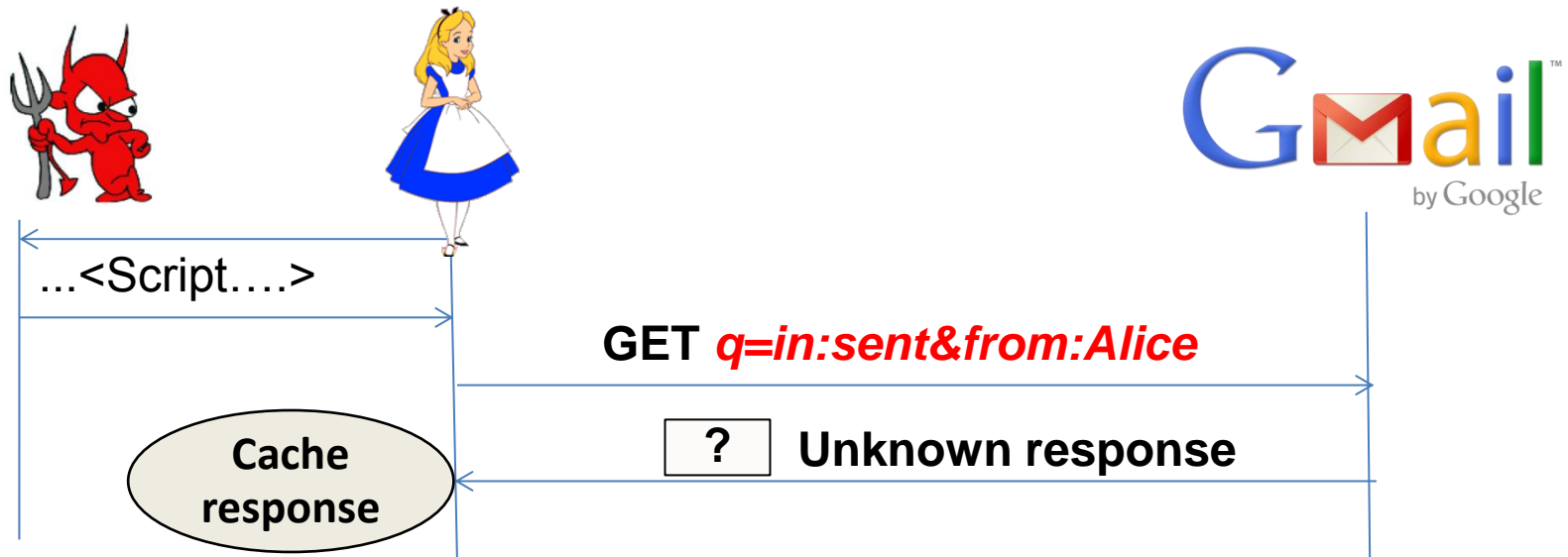
- Exploiting / measurements affected by
 - Classical: network delay, server processing time, browser processing time
 - Browser-based: browser processing time
- Can be used to differentiate between
 - Classical: large/small resources, high/low server processing time
 - Browser-based: large/small resources

BB XS-Search: Basic Flow

- Find the answer for a Boolean question
- Changing only the second step of the original XS-Search attack

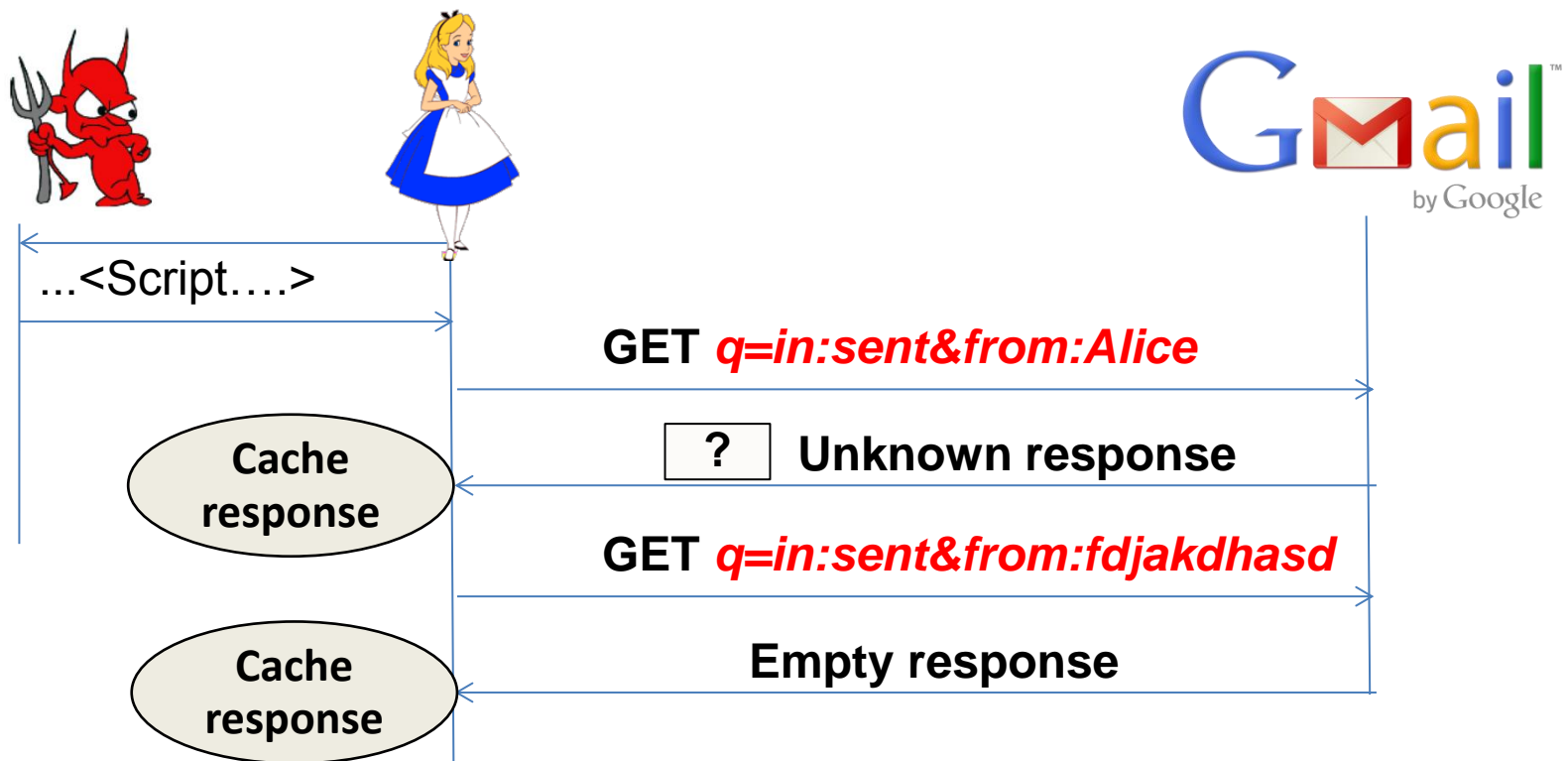
BB XS-Search: Basic Flow – 2nd Step

- Send a **Challenge** request
 - Is the user name *Alice*?
 - True: a **Full** response is returned (has some content)
 - False: an **empty** response is returned

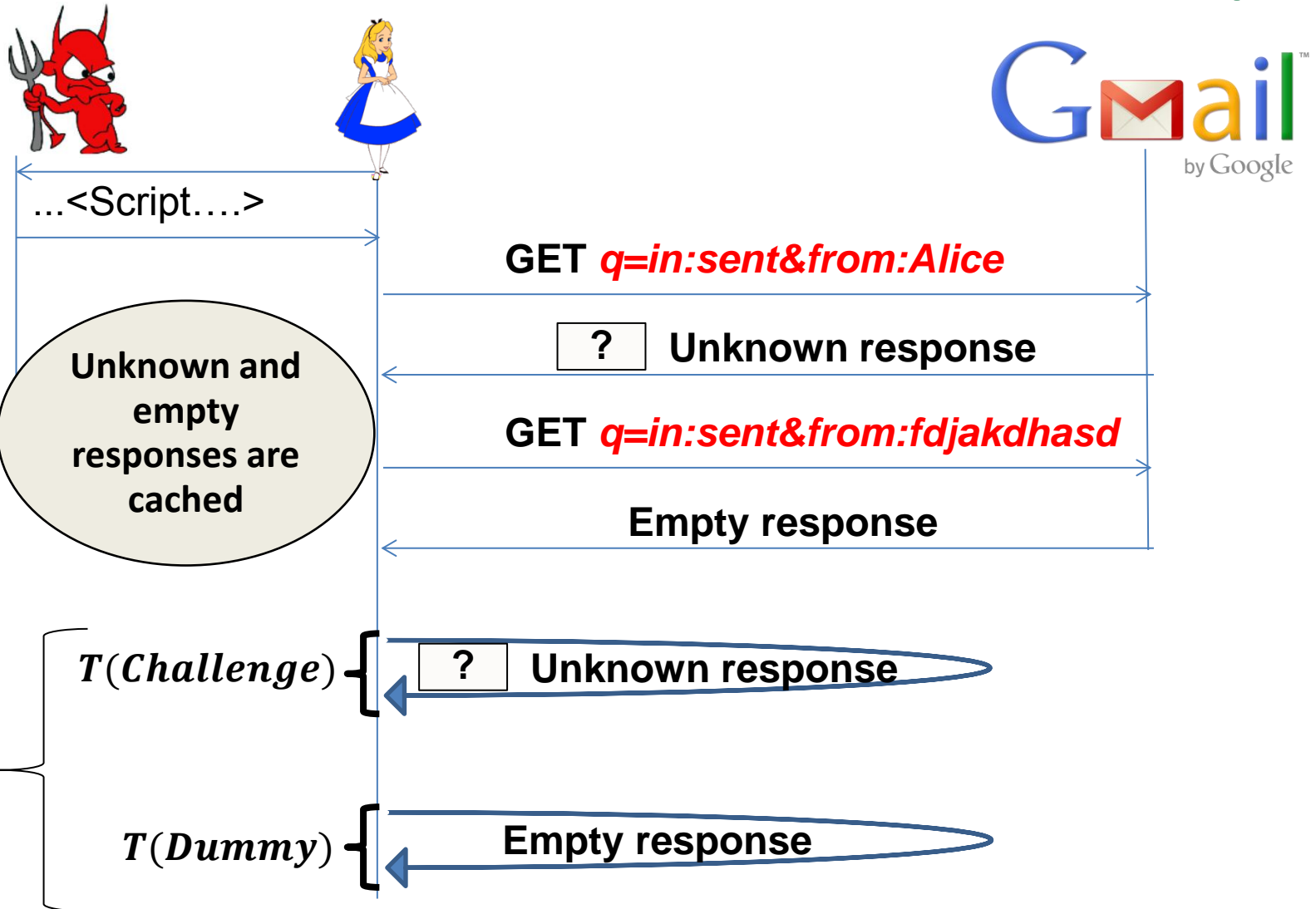


BB XS-Search: Basic Flow – 2nd Step

- Send a **Dummy** request
 - Is the user name *fdjakdhasd*?
 - The response is expected to be **empty**



BB XS-Search: Basic Flow – 2nd Step



Browser-based (BB) XS-Search

- Algorithmic improvements
- Not for Boolean questions
 - Basic flow – only Boolean questions
 - Is the victim's name Alice?
- Answering multiple choice questions
 - E.g., which names out of many options are matching the victim?
- Optimally use the browser-based timing side-channel

Browser-based (BB) XS-Search

- Evaluation compared to both the previous works
- Repeating attacks/experiments done in each of them
 - Original XS-Search: extract victim's names from Gmail
 - BB timing attacks: extract victim's age from Facebook
- Significant improvement!
- In this talk: only one example

BB XS-Search vs. original XS-Search

- Gmail example
 - The goal of the attacker: extract the first and last names of the victim out of a list of 2000 names
 - XS-Search results:
 - 90% success rate (both first and last name found)
 - 1 minute on average
 - 2.6% false positive

BB XS-Search vs. original XS-Search

- How to answer multiple-answer questions efficiently?
- The optimized multiple term identification (OMTI) algorithm
 - Divide and conquer algorithm
 - Relying on the OR operator
 - **Different dummy search request is sent every round**

BB XS-Search vs. original XS-Search

- Rely on browser-based timing side-channel to optimize the OMTI algorithm
- Observation: empty responses are (almost) identical
 - No need to send dummy requests in every round
 - No need to reload the empty response in every round
 - Rely on previous measurements!

BB XS-Search vs. original XS-Search

- Evaluation of the attack on 5 different Gmail accounts
 - 15-16 times on each of them
- Significant improvement!
 - **41.6 seconds on average (compared to 1 minute)**
 - 92.3% success (compared to 89.7%)
 - 1.3% false positive (compared to 2.6%)

BB XS-Search vs. original XS-Search

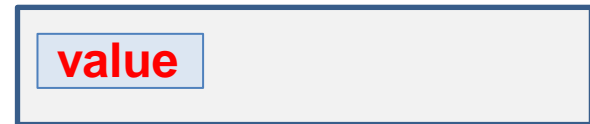
- DEMO

Second-order (SO) XS-Search attacks

- The problem: sometimes the size difference is negligible
- For example: a sentence that appears in a single email



Empty response



Full response

Second-order (SO) XS-Search attacks

- Second-order attacks
 - First, manipulate the attacked web application
 - Make it (more) vulnerable
 - Exploit the vulnerability
- Second-order XS-search attacks
 - First manipulate the attacked storage
 - Create significant response inflation
 - Launch browser-based XS-search attack

Second-order (SO) XS-Search attacks

- Two SO XS-search attacks
 - Simple
 - Inflating

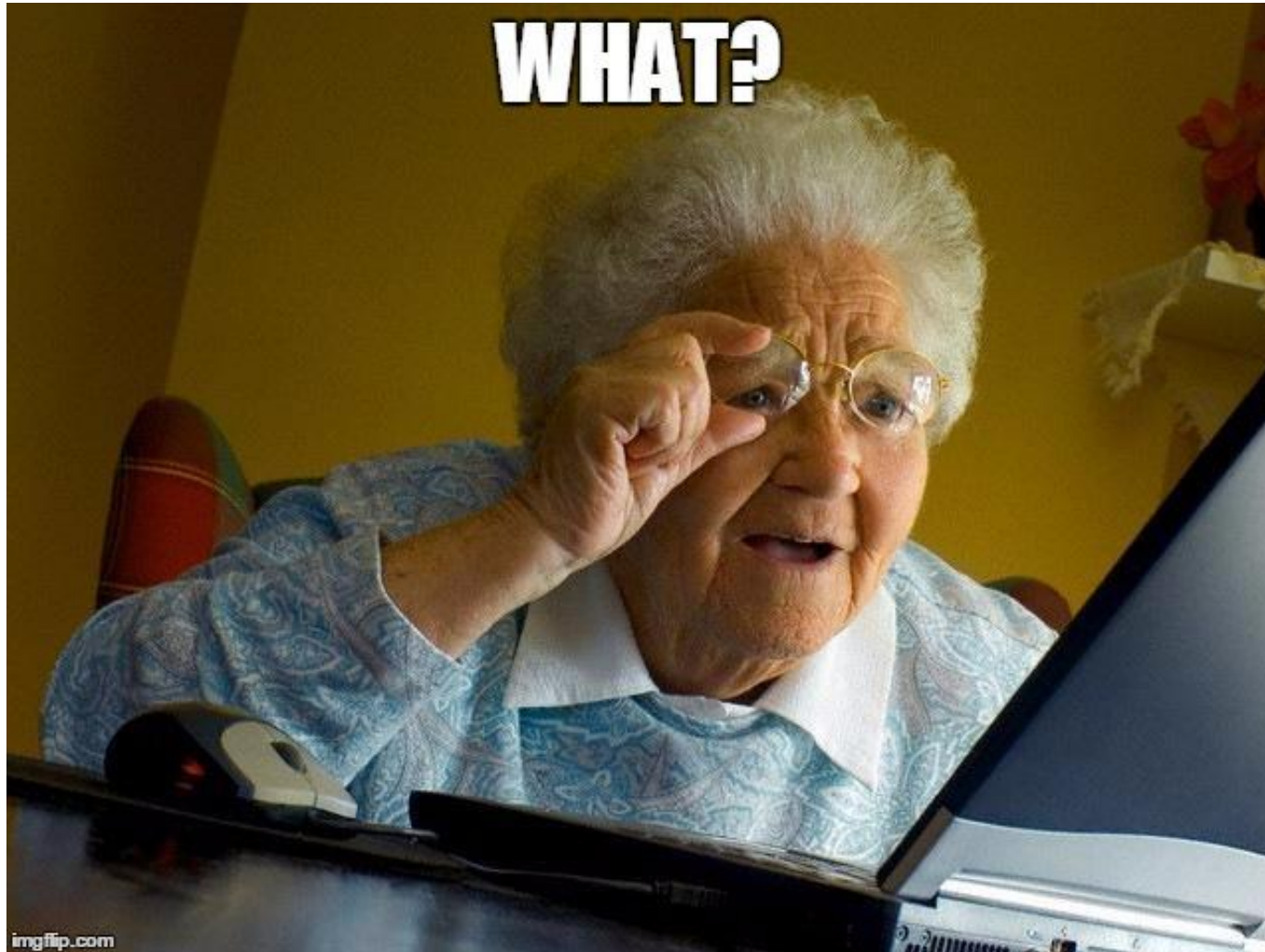
Second-order (SO) XS-Search attacks

- Model
 - Storage
 - Many records
 - A secret appears in one of the records
- Attacker can manipulate the storage remotely
 - E.g., email accounts
 - Another example later...

Simple SO XS-Search attack

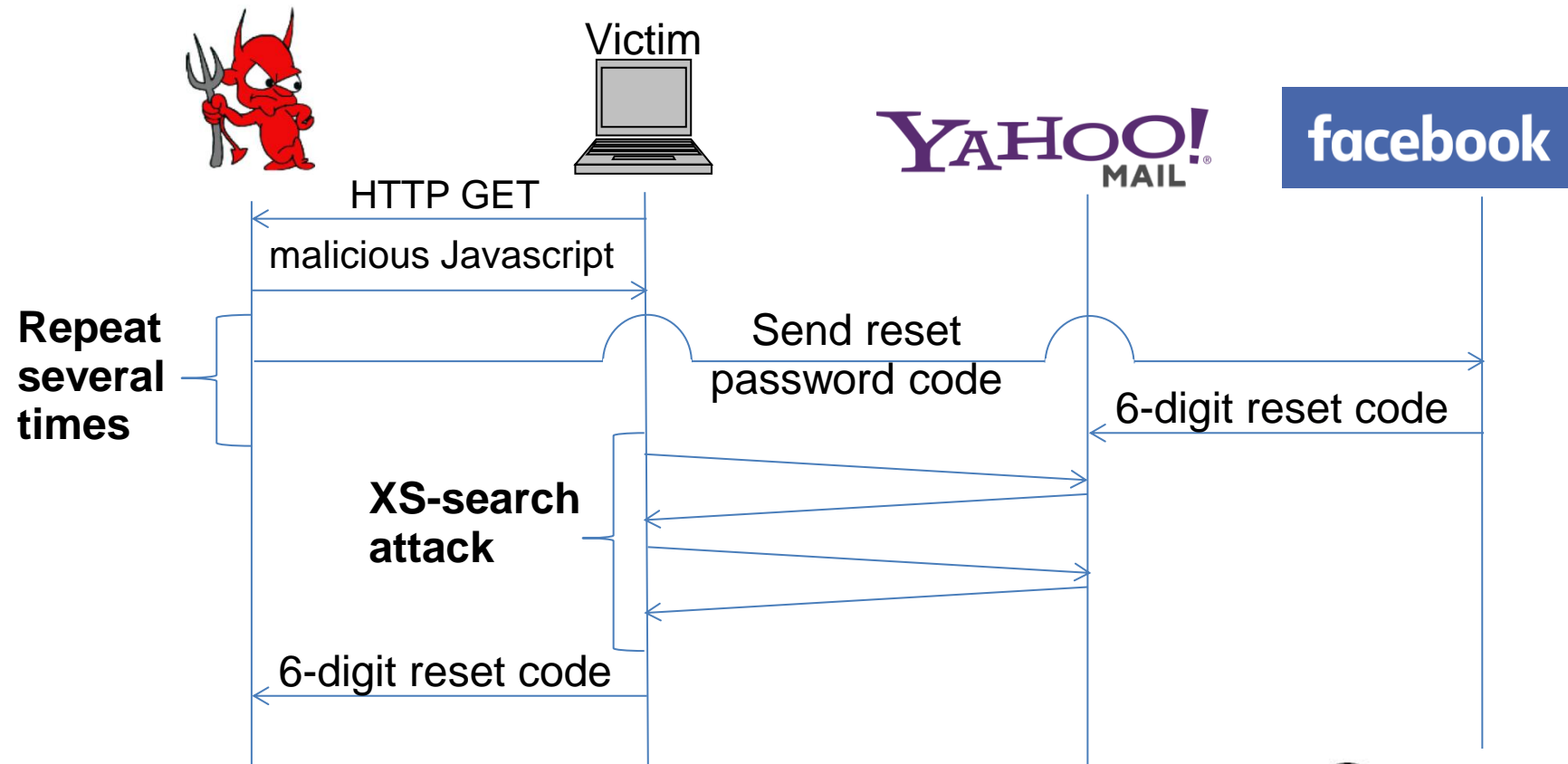
- The problem: the secret appears only once in the storage
- Simple solution: the attacker will add additional records that contain the secret!

Simple SO XS-Search attack



Simple SO XS-Search attack

- Example: extracting Facebook password-reset code from Yahoo! email



Inflating SO XS-Search attack

- Creates significant response inflation effect
 - Increase the size difference between empty and full response
- Unlike all the previous attacks: the empty response will be (significantly) larger than the full response

Inflating SO XS-Search attack

- The challenge of the attacker:
 - Find a secret out of a large dictionary of possible values
- Notations
 - *M* - maximal number of results
 - *Match-all record* – a record that contains all the possible values for the secret
 - *Inflating record* – a record that significantly inflates the size of every response containing it

Inflating SO XS-Search attack

- Attack process

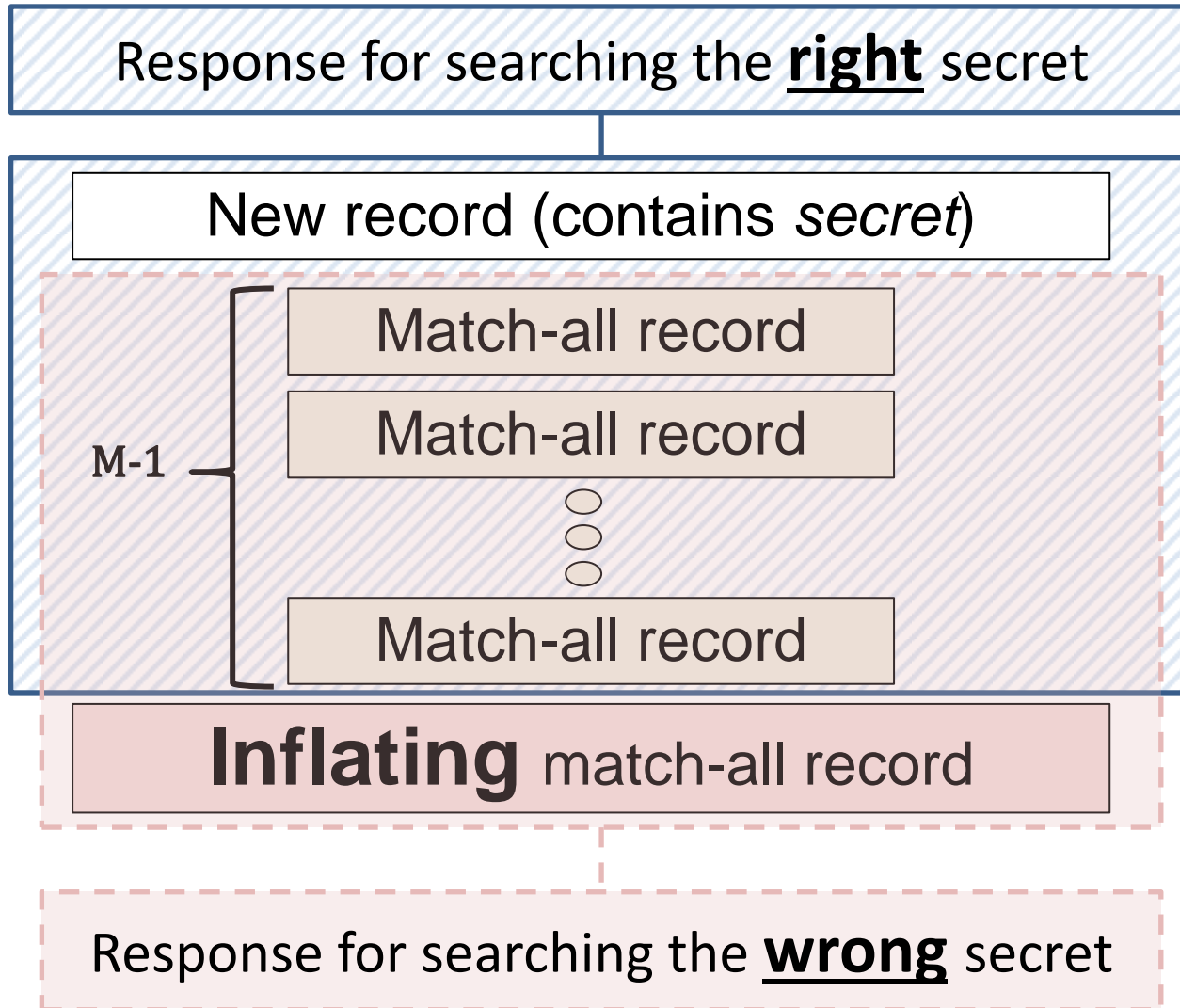
First part:

- Plant one ***match-all inflating*** record in the storage
- Plant additional ***M-1 match-all*** records
- Additional record(s) may be added as a result of the victim's operations, or via operations triggered by the attacker

Second part:

- Launch BB XS-search attack!

Inflating SO XS-Search attack



Inflating SO XS-Search attack

- Inflating record in email service providers
 - Email headers
 - From
 - To

Inflating SO XS-Search attack

- Example: extracting Visa/Mastercard credit card number
 - Structured information
 - VVVV-XXXX-YYYY-ZZZZ
- First and last names: extract 2 out of 2000
 - Done successfully!
- Credit card number: extract 4 out of 10000
 - Should not be much harder

Inflating SO XS-Search attack

- Example: extracting Visa/Mastercard credit card number
- ***Match-all record*** – a record that contains all the possible 4-digit sequences
 - Possibly as an attachment
- ***Inflating match-all record*** – a *match-all* record with very long *From* field

Inflating SO XS-Search attack

- Gmail example
- How?
 - Cross-site search requests are now blocked in both the HTML and standard views
- Cross-site search attack without sending cross-site search requests?

Inflating SO XS-Search attack

- Gmail example
- Exploiting the autocomplete feature!

The screenshot shows the Gmail search interface. The search bar contains the query "from:Anony Mous". Below the search bar, the search results are displayed. A bracket on the right side of the results list indicates that there are 4 results, labeled as $M = 4$ (where M is the maximal number of results).

From	Date
Anony Mous	Jan 11
unsubscribe me	11/30/15
Unsubscribe me	11/9/15
Hi! What's up? me	11/9/15
asdadasd me	10/11/15
grades Anony Mous	10/7/15
Untitled spreadsheet Anony Mous	

Search the web for "from:Anony Mous" Shift + Enter

Inflating SO XS-Search attack

- Gmail example: the manipulated storage

Google

Gmail

COMPOSE

Primary Social Promotions

New record (contains *secret*)

Starred

Sent Mail

M-1

Match-all record

Match-all record

Match-all record

Inflating match-all record

Anony Mous

Attacker.

Attacker.

Attacker.

Attacker.

Your payment details - See in

match-all 3 - visa credit card C

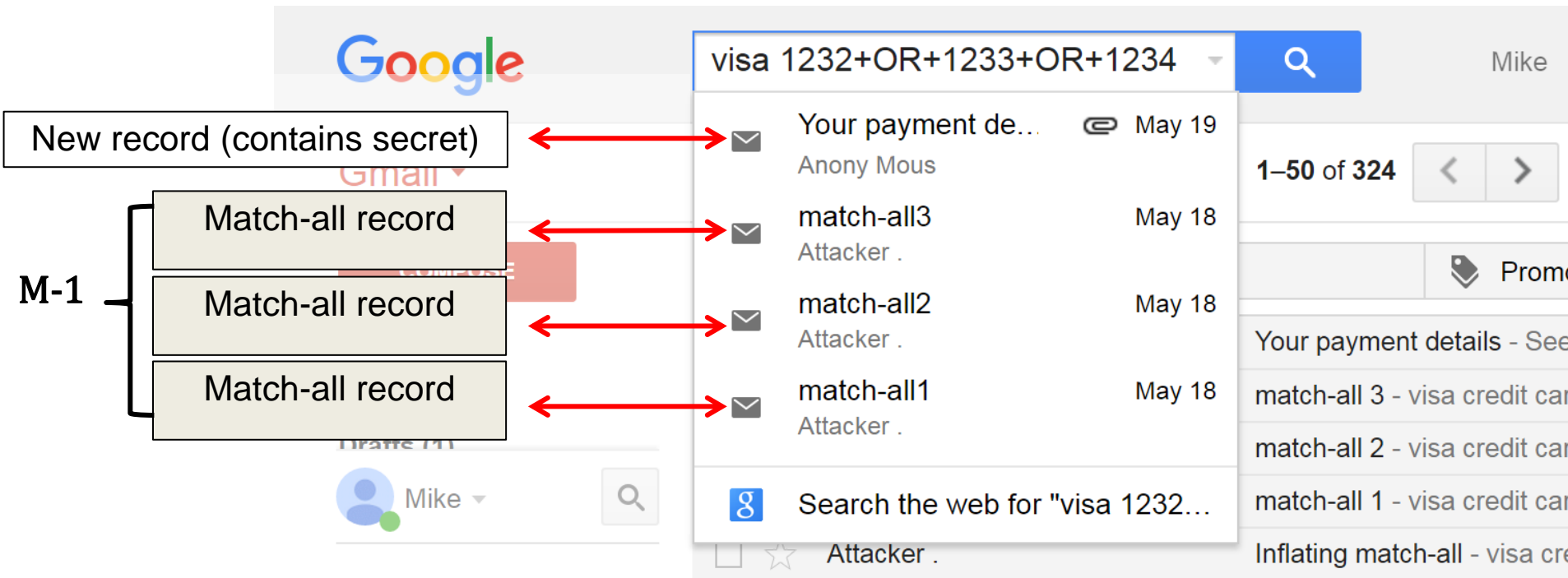
match-all 2 - visa credit card C

match-all 1 - visa credit card C

Inflating match-all - visa credit

Inflating SO XS-Search attack

- Gmail example: full response (size is small)



Inflating SO XS-Search attack

- Gmail example: empty response (size is very large)

The diagram illustrates an inflating match-all record attack on Gmail. On the left, a list of records shows three 'Match-all record' items and one 'Inflating match-all' item. Red double-headed arrows connect these records to a search results page on the right. The search results page shows a search for 'visa 1232+OR+1233+OR+1235' with 324 results, including 'match-all3', 'match-all2', 'match-all1', and 'Inflating match-all'.

Record	Search Results
Match-all record	match-all3 Attacker . May 18
Match-all record	match-all2 Attacker . May 18
Match-all record	match-all1 Attacker . May 18
Inflating match-all record	Inflating match-all Attacker .fluqE9xz1vww... May 18

Inflating SO XS-Search attack

- DEMO

Inflating SO XS-Search attack

- Evaluation results
 - 96% success rate within less than 50 seconds
 - Yet, in the other 4% percent, 3 out of 4 sequences were found, and it was possible to detect the error and to fix it

Stealthy SO XS-Search attacks

- The challenge: manipulations on the storage can be detected!
- Solution: manipulate the storage in a way that will not be detected by the user
- HOW?

Stealthy SO XS-Search attacks

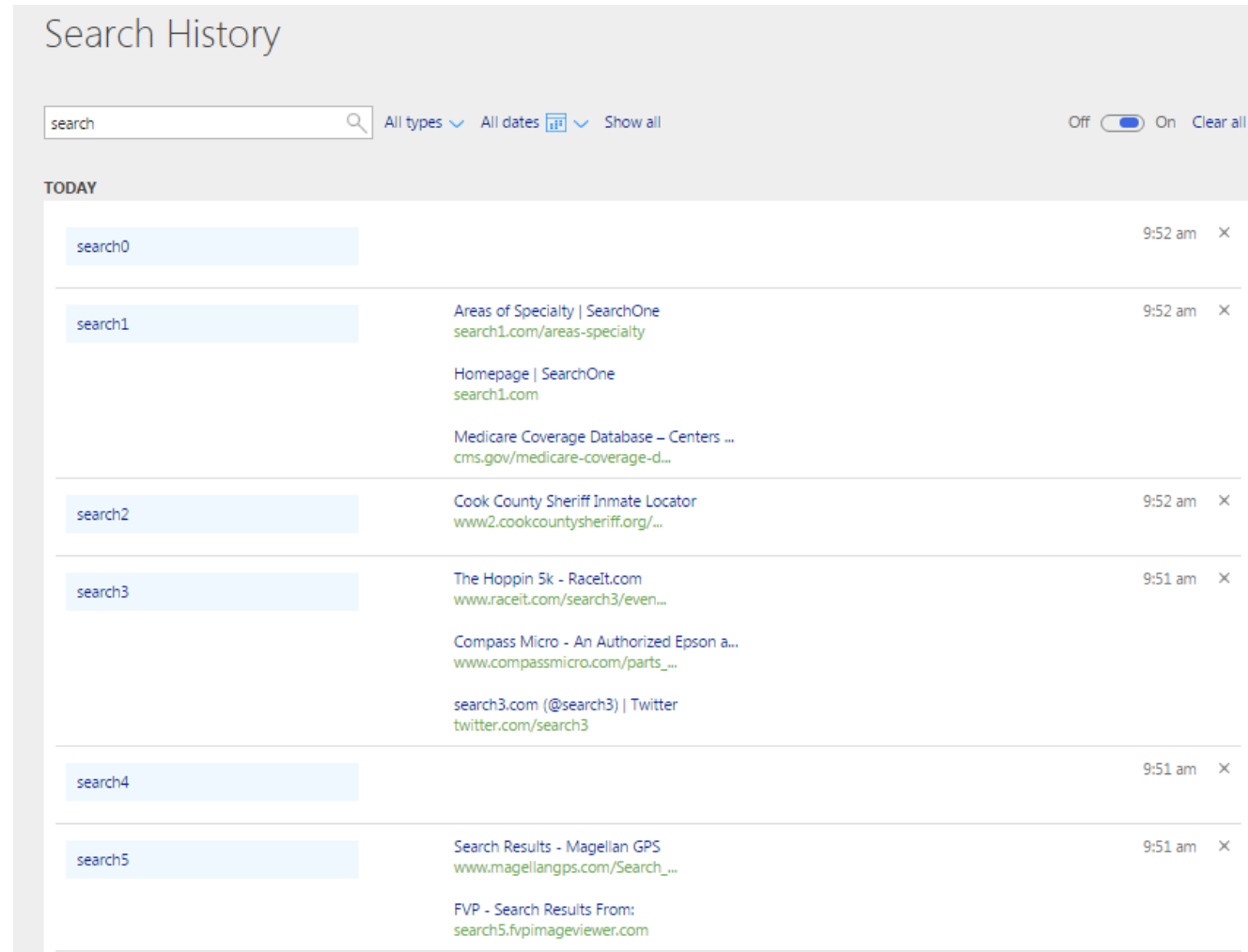
- Emails solution: abuse anti-spam mechanisms
- The planted emails will be marked as spam
 - Users do not get notifications for spam emails
 - Users (usually) do not visit their spam folder
- Only when it is possible to search in the spam and in the other folders using the same request
 - E.g., Gmail
 - in:inbox OR in:spam

Stealthy SO XS-Search attacks

- Search history
- Two requirements for inflating SO XS-Search attack:
 - Inject records to the search history log
 - **DONE**: Gelernter & Grinstein & Herzberg, ACSAC 2015
 - Inject an inflating record

Stealthy SO XS-Search attacks

- Bing example:
inflating SO
XS-Search
attack to
extract
search
history



Defenses (briefly)

- If possible - blocking cross-site search requests
- In other cases – make it harder to exploit
 - Block inflation techniques
 - Rate limit
- Like (almost) every other web-application attack the challenge is to find all the vulnerable spots

Conclusions

- Advanced cross-site search attacks
 - Browser-based
 - Second order
- Practical!
- Many vulnerable websites
 - Including popular ones

Thank you!

Questions?