

The Threat of Advanced Cross-Site Search Attacks

Nethanel Gelernter

CYBERPON



The College of Management
Academic Studies



OWASP

AppSec Israel 2016

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



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



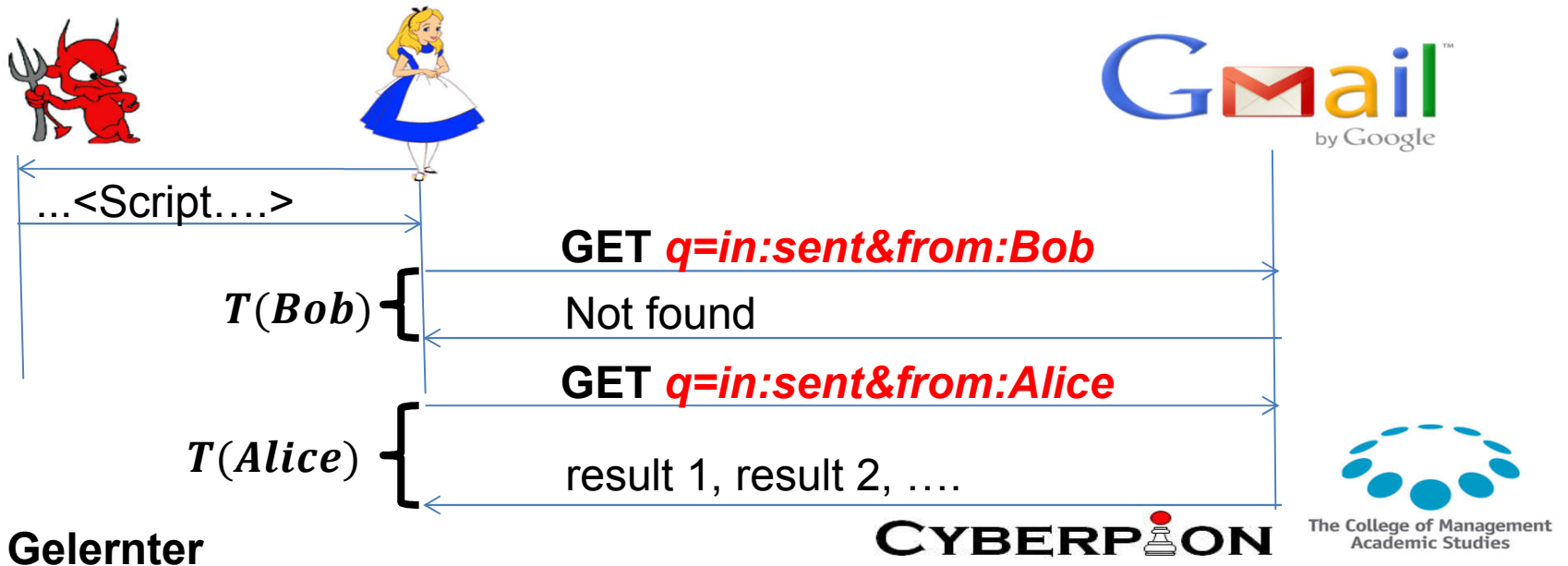
Cross-site attacker model

- 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'



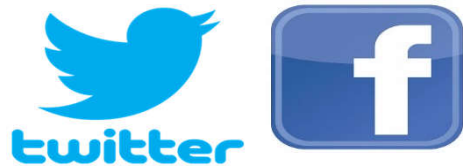
What else can XS-Search expose?

**Structured
information**



Contacts

Name



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



Search History

XS-Search: Basic Flow

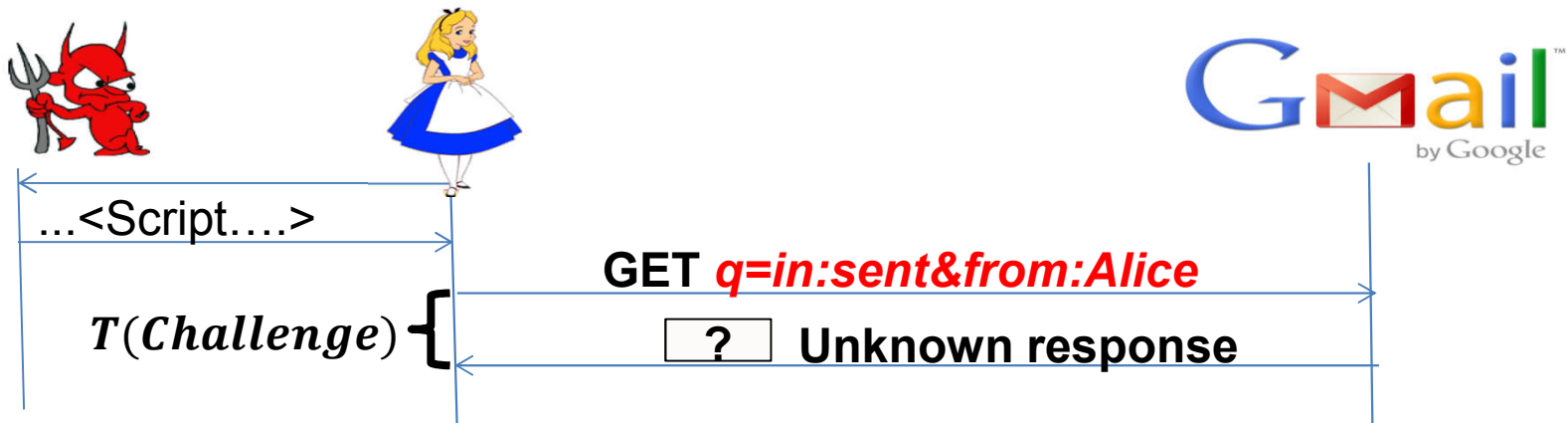
- Find the answer for a Boolean question
- Three steps:
 - Transform the question into a search request
 - Send search requests and collect samples
 - Analyze response times → answer the question!

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

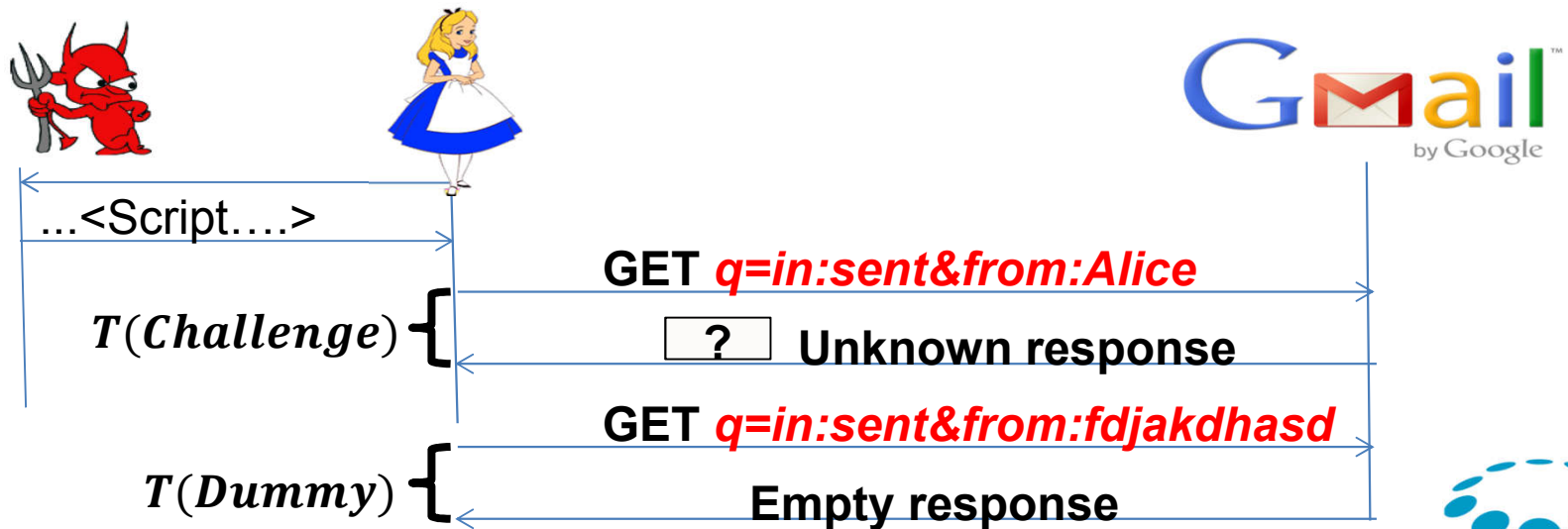
XS-Search: Basic Flow – 2nd Step

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

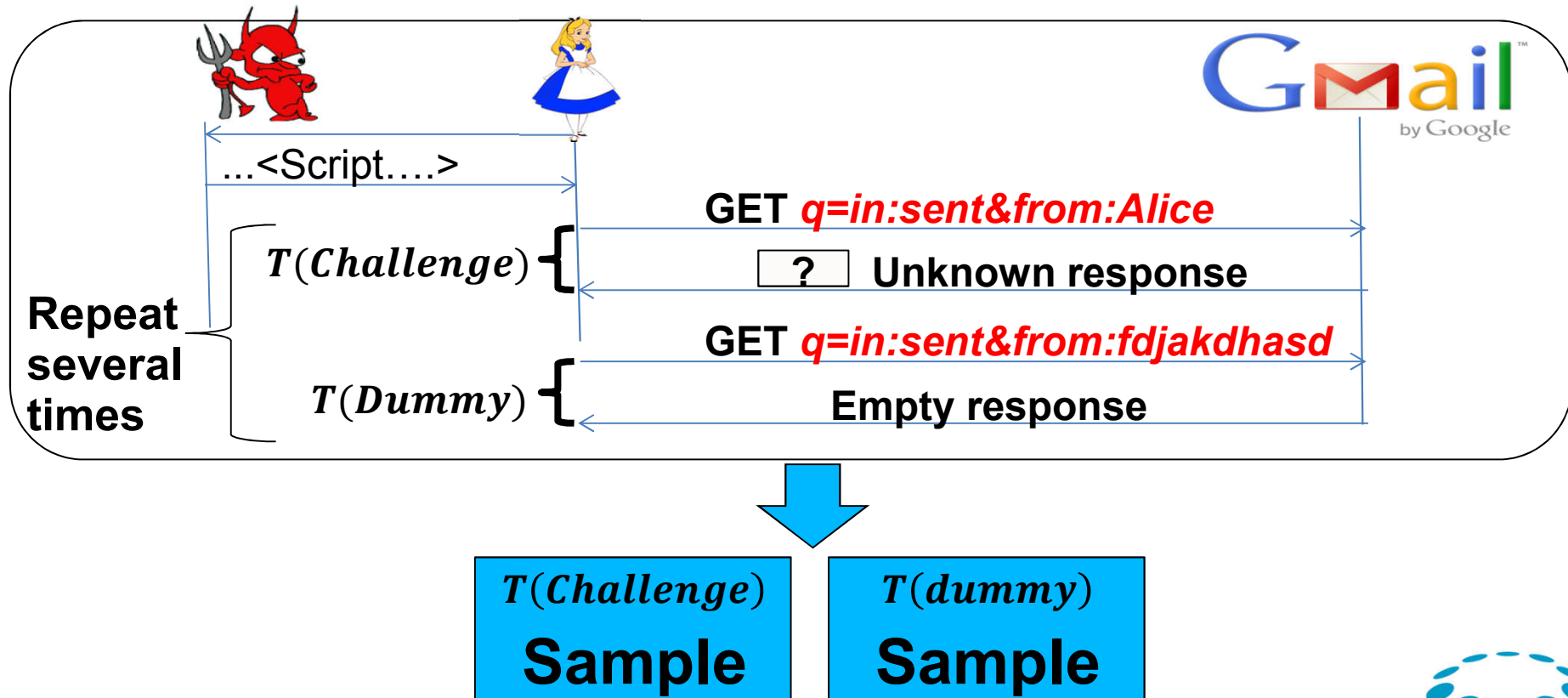


XS-Search: Basic Flow – 2nd Step

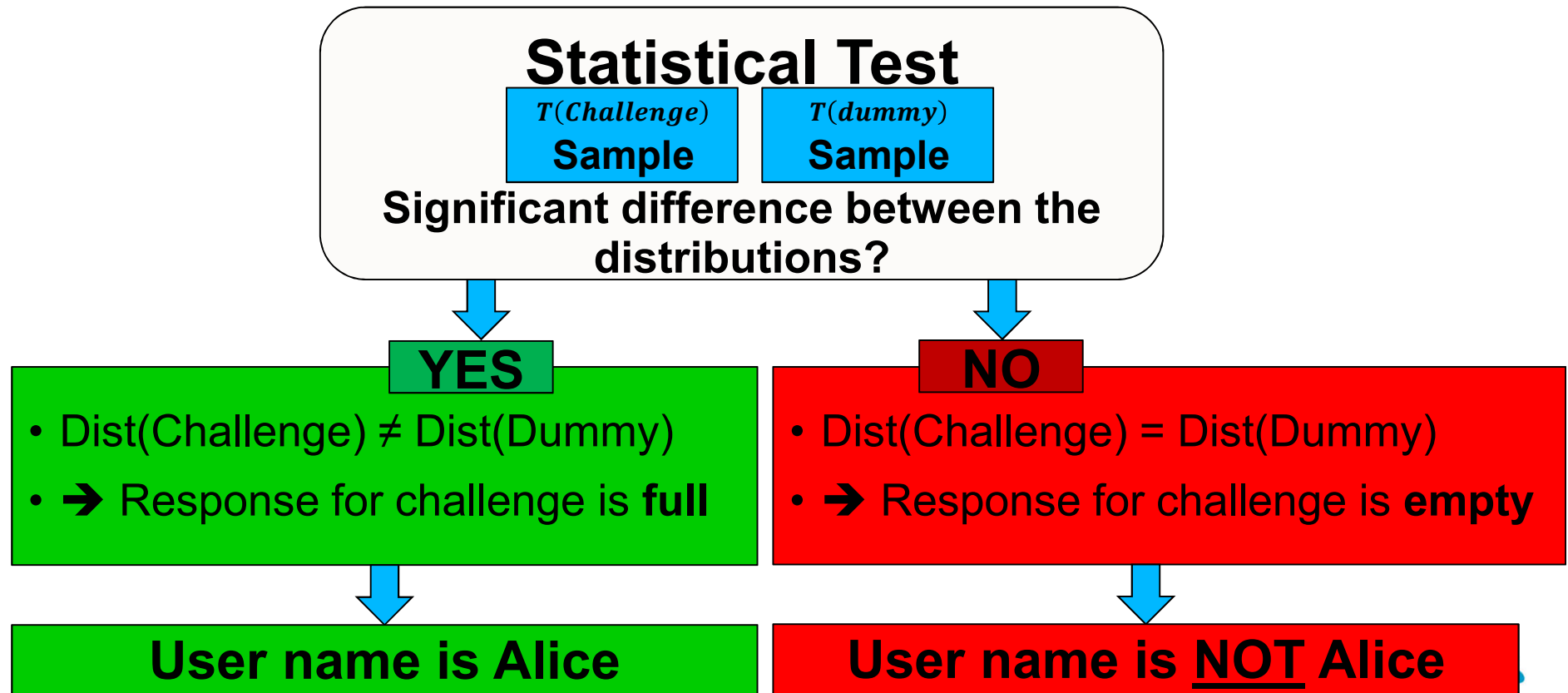
- Send a **Dummy** request
 - Is the user name *fdjakdhasd*?
 - The response is expected to be **empty**



XS-Search: Basic Flow – 2nd Step



XS-Search: Basic Flow – 3rd Step



Practical timing attacks: challenges

- Timing attacks
 - Time measurements 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

value	
-------	--

Empty response

value	value	value
value	value	value
value	value	value

Full response

Response Inflation

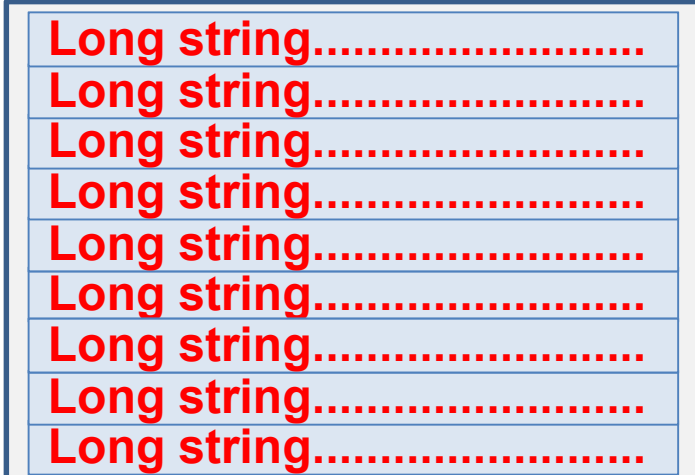
- Sometimes, the attacker can send **very long strings** as the value of the reflected parameter

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

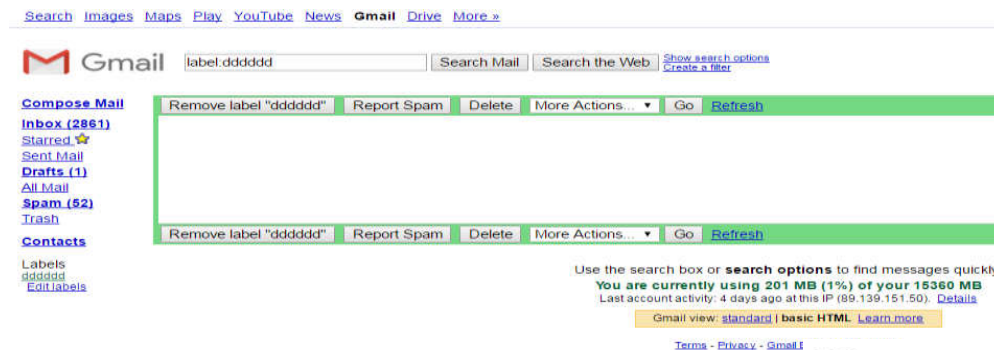
Full response

CYBERP^{ON}


The College of Management
Academic Studies

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!**



But...



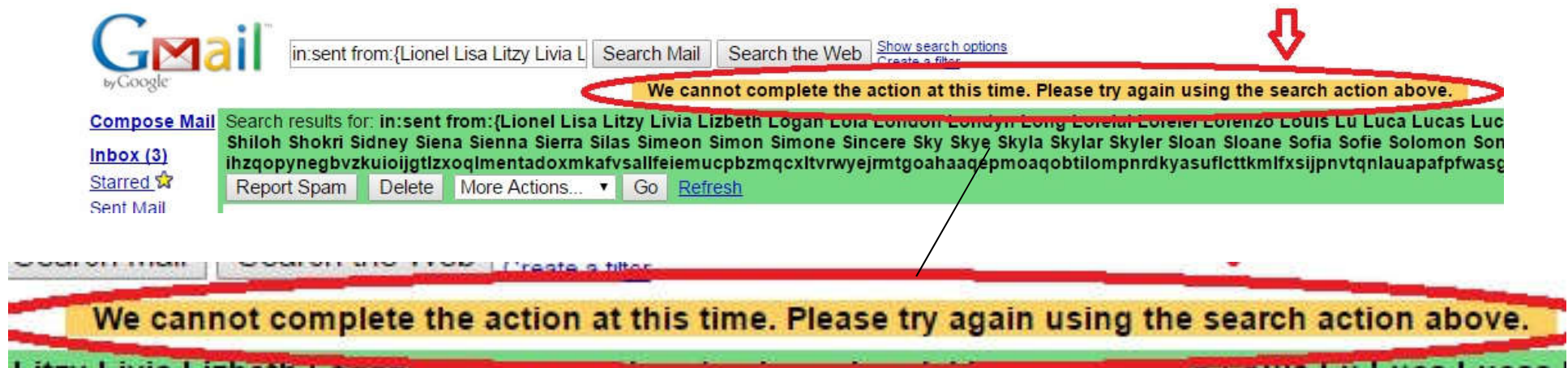
Nethanel Gelernter

CYBERPION



The College of Management
Academic Studies

What if there is no response inflation?



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
 - **Not in this talk**



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

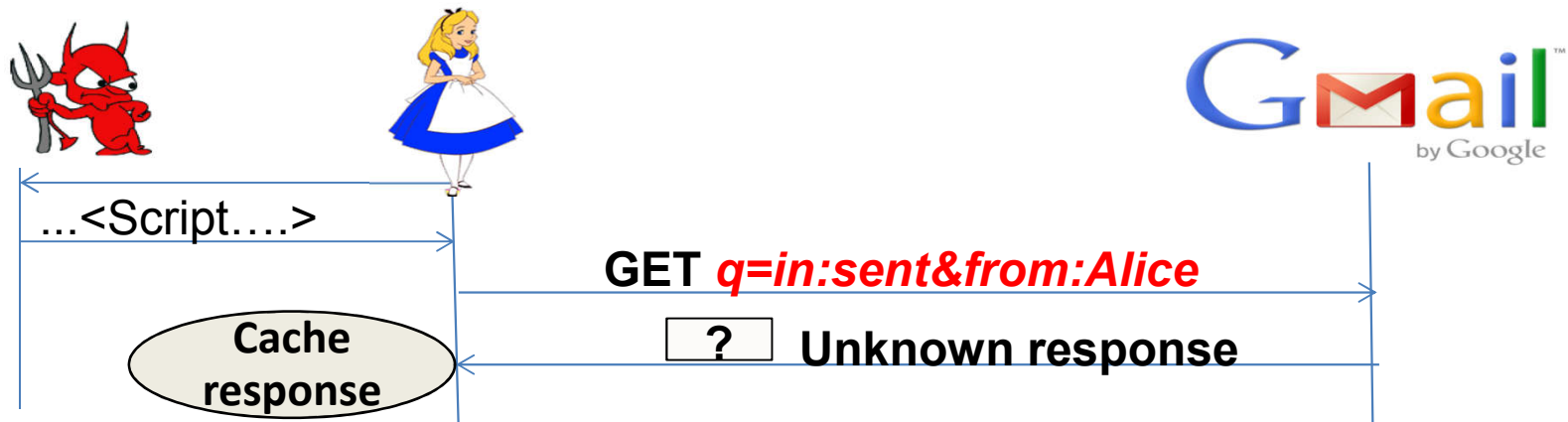
- Exploiting / measurements affected by
 - Classical: network delay, server processing time, browser processing time
 - Browser-based: browser processing time
- Can be used to differentiate between
 - Classical: large/small resources, high/low server processing time
 - Browser-based: large/small resources

BB XS-Search: Basic Flow

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

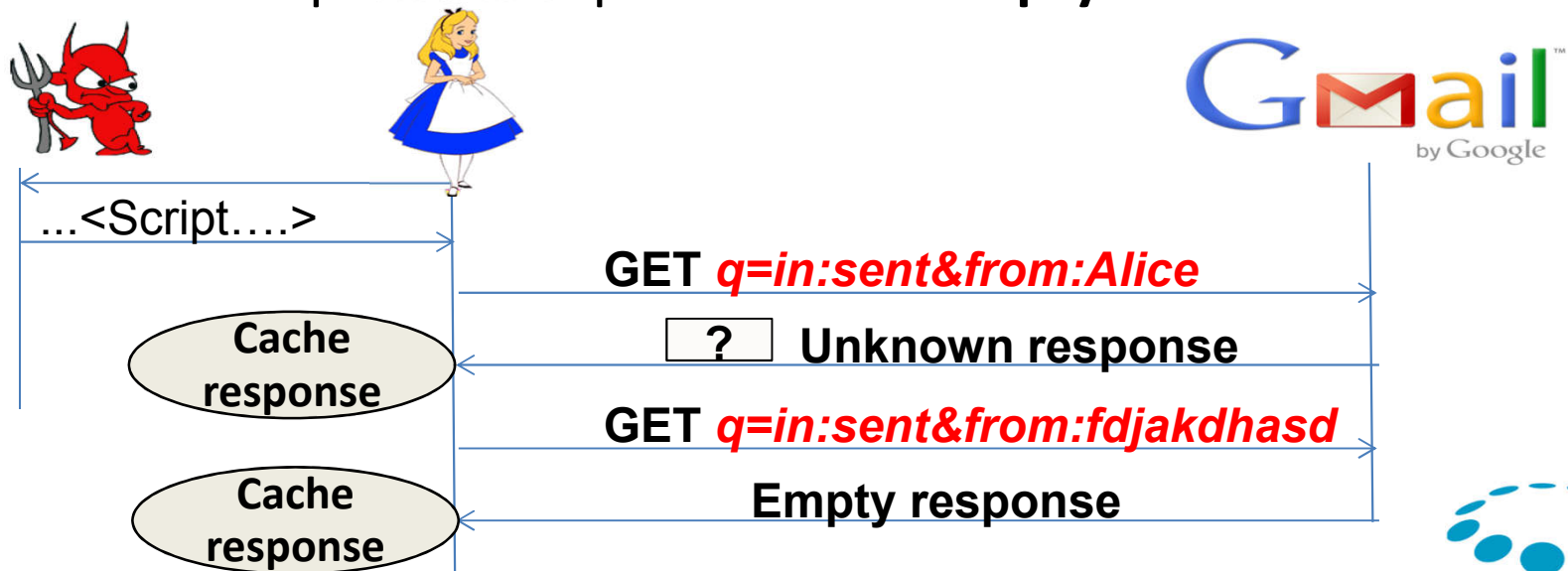
BB XS-Search: Basic Flow – 2nd Step

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

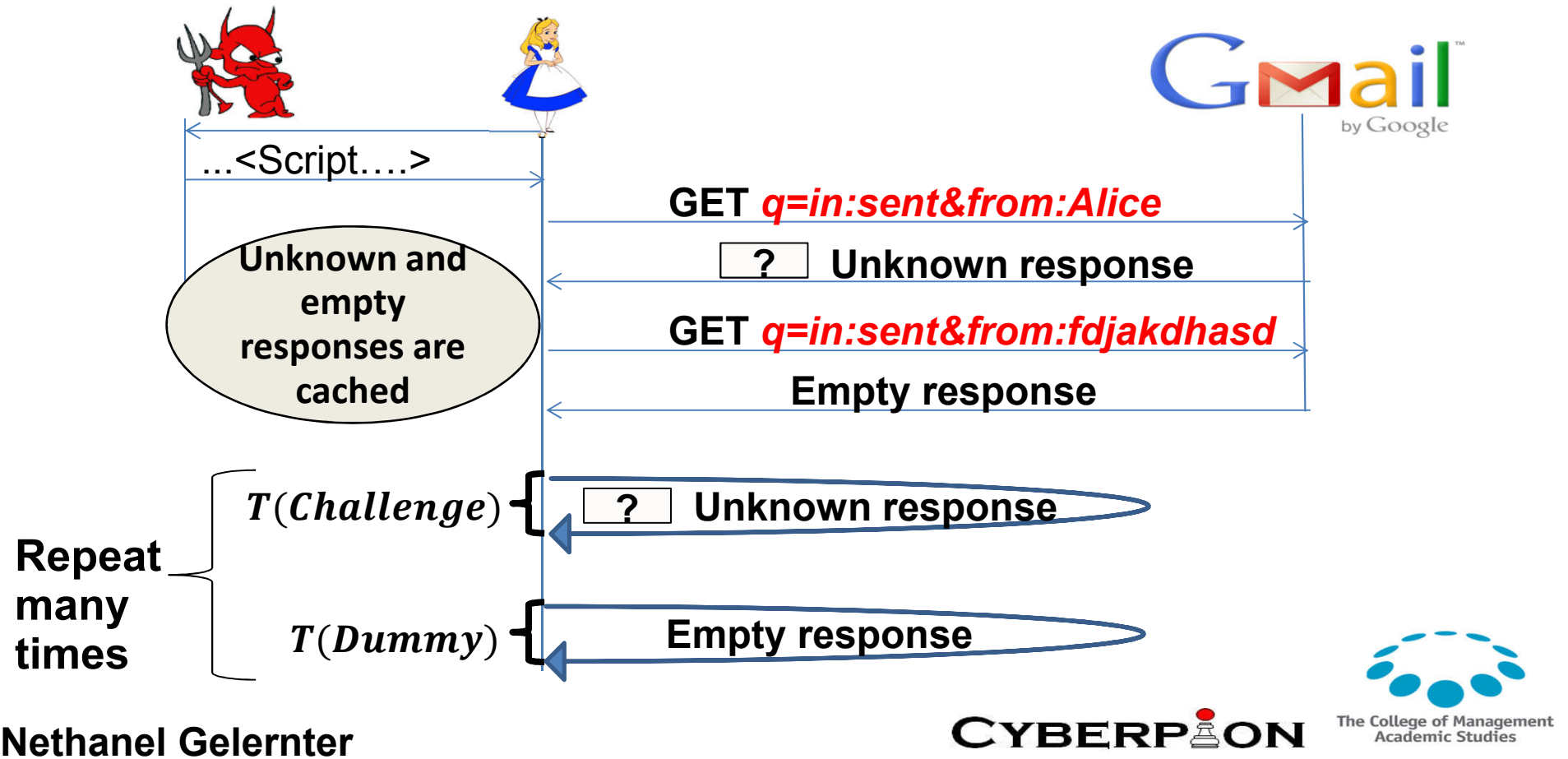


BB XS-Search: Basic Flow – 2nd Step

- 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 in this talk**
- 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

- Evaluation of the browser-based XS-search 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

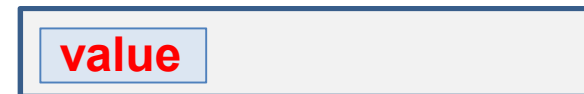
- DEMO

Second-order (SO) XS-Search attacks

- The problem: sometimes the size difference is negligible
- For example: a term 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

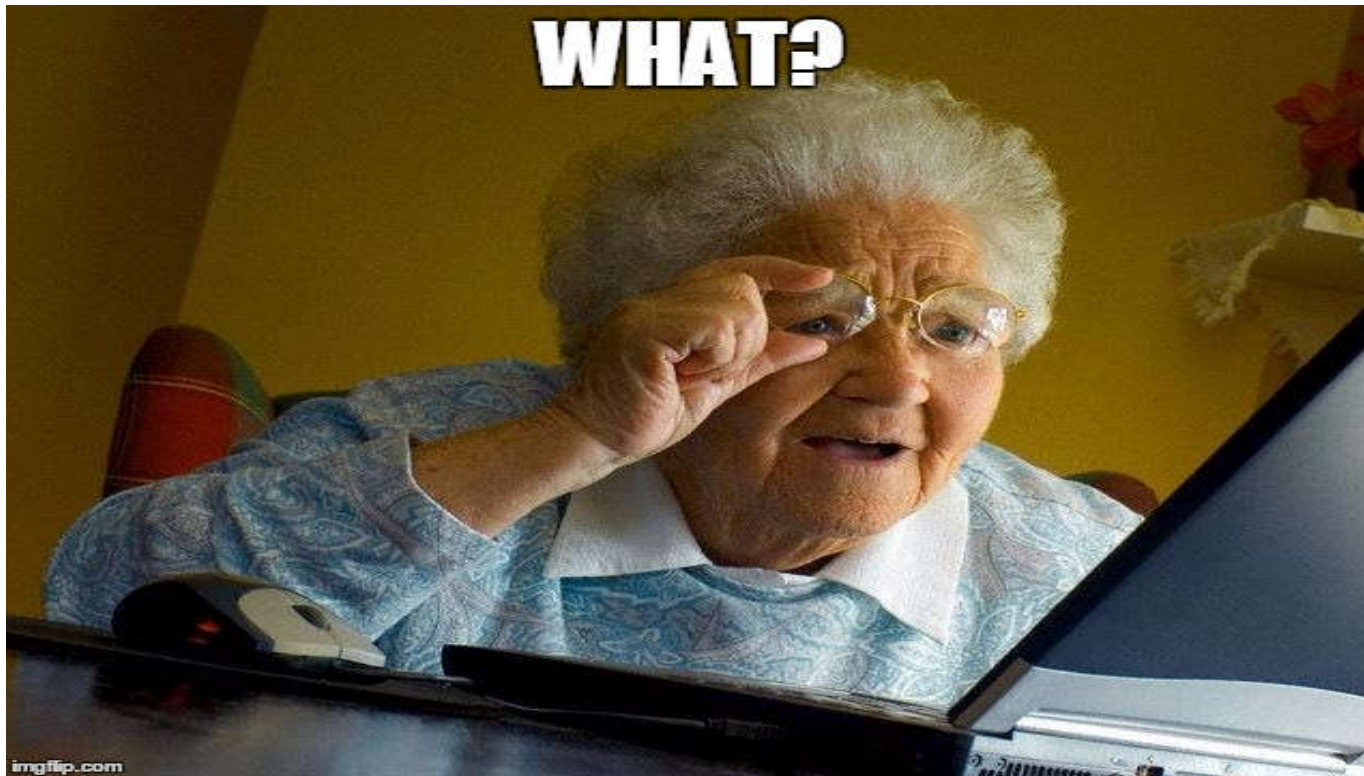
Second-order (SO) XS-Search attacks

- 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



Nethanel Gelernter

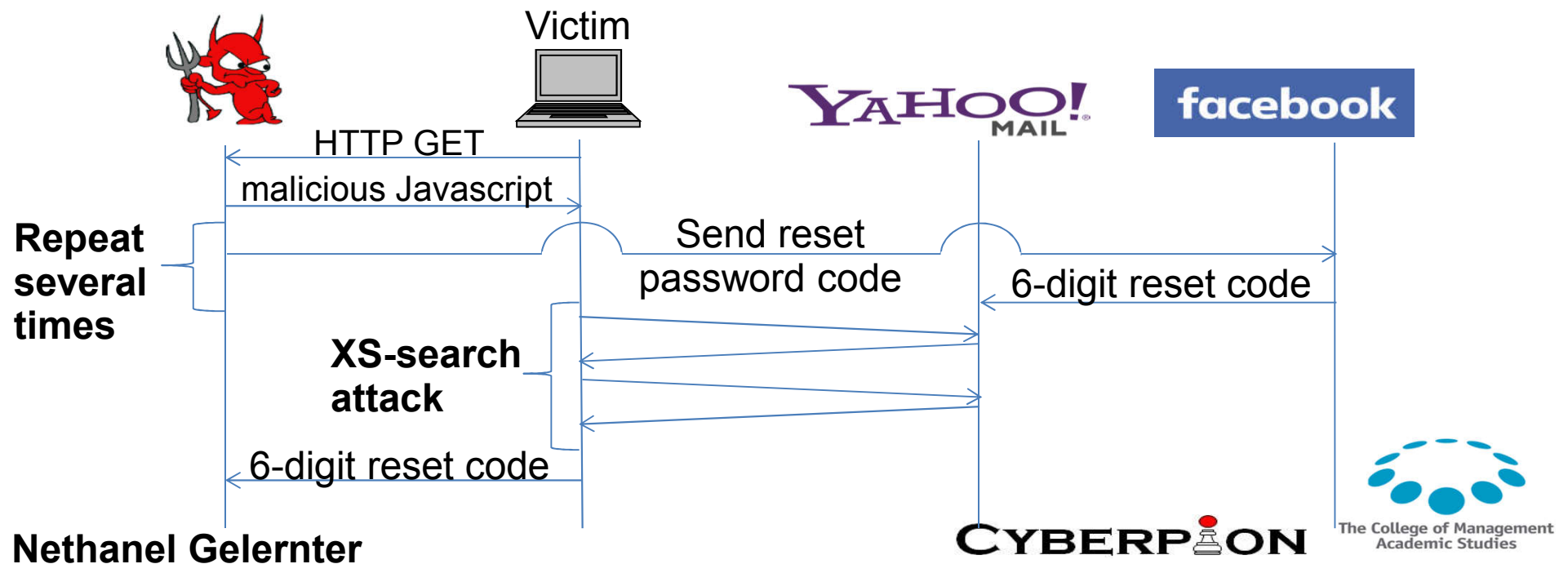
CYBERPION



The College of Management
Academic Studies

Simple SO XS-Search attack

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



Inflating SO XS-Search attack

- Creates significant response inflation effect
 - Increases 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

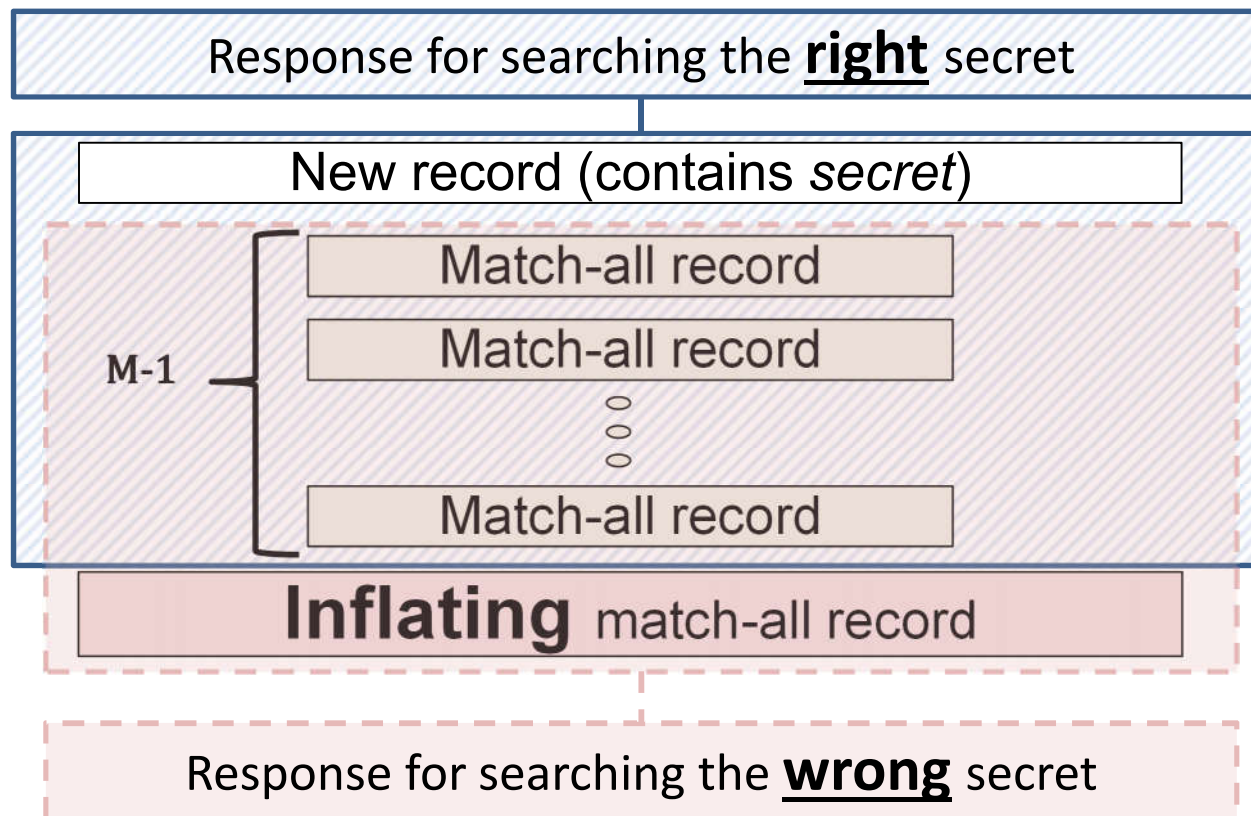
First part:

- 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 (browser-based) 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

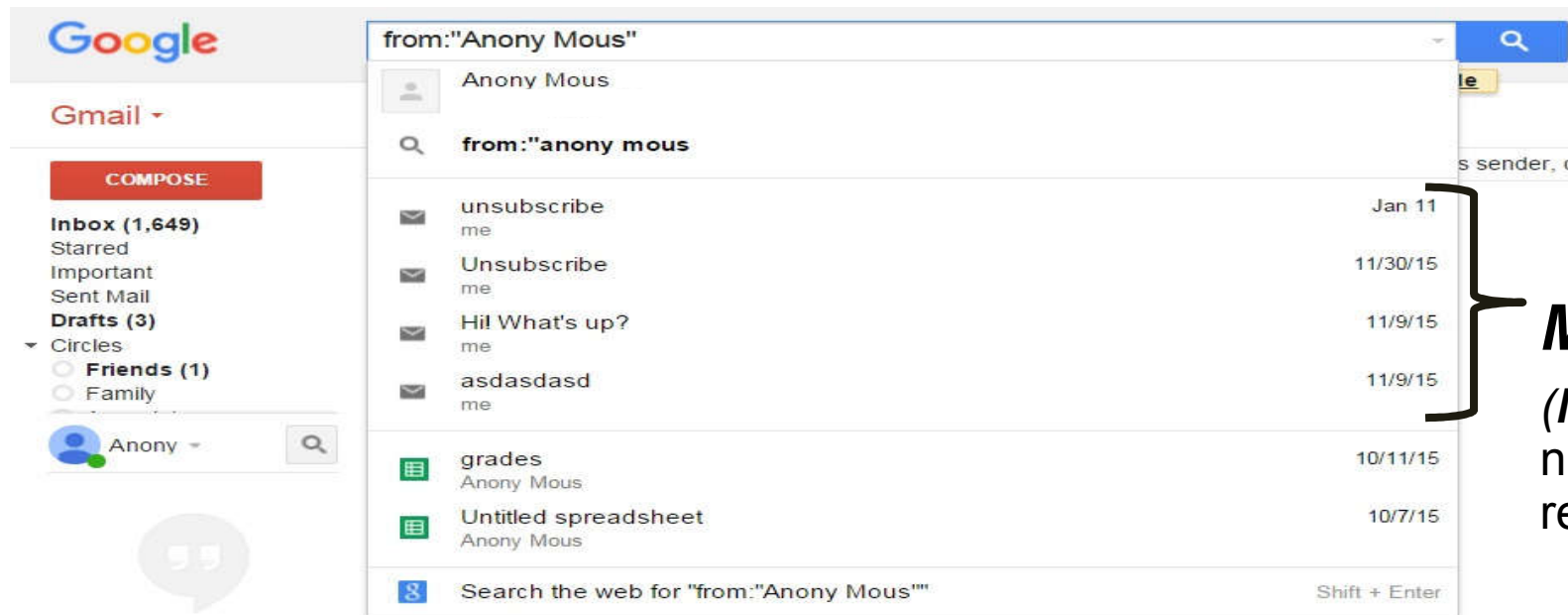
- Example: extracting Visa/Mastercard credit card number
- ***Match-all record*** – a record that contains all the possible 4-digit sequences
 - Possibly as an attachment
- ***Inflating match-all record*** – a *match-all* record with very long *From* field

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!



$M = 4$

(M = maximal number of results)

Inflating SO XS-Search attack

- Gmail example: the manipulated storage

The screenshot shows a Gmail inbox with several email entries. A diagram is overlaid on the left side of the inbox, illustrating an attack strategy. The diagram consists of a vertical stack of boxes. The top box is labeled 'New record (contains secret)'. Below it are three boxes labeled 'Match-all record'. The bottom box is labeled 'Inflating match-all record'. A bracket on the left side of the 'Match-all record' boxes is labeled 'M-1'. Red arrows point from the 'Match-all record' boxes to the email entries in the inbox. The email entries are: 'Anony Mous' (with a red arrow pointing to 'Your payment details - See in'), 'Attacker.' (with a red arrow pointing to 'match-all 3 - visa credit card C'), 'Attacker.' (with a red arrow pointing to 'match-all 2 - visa credit card C'), 'Attacker.' (with a red arrow pointing to 'match-all 1 - visa credit card C'), and 'Attacker.' (with a red arrow pointing to 'Inflating match-all - visa credit').

Google

Gmail

COMPOSE

Primary Social Promotions

New record (contains secret)

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

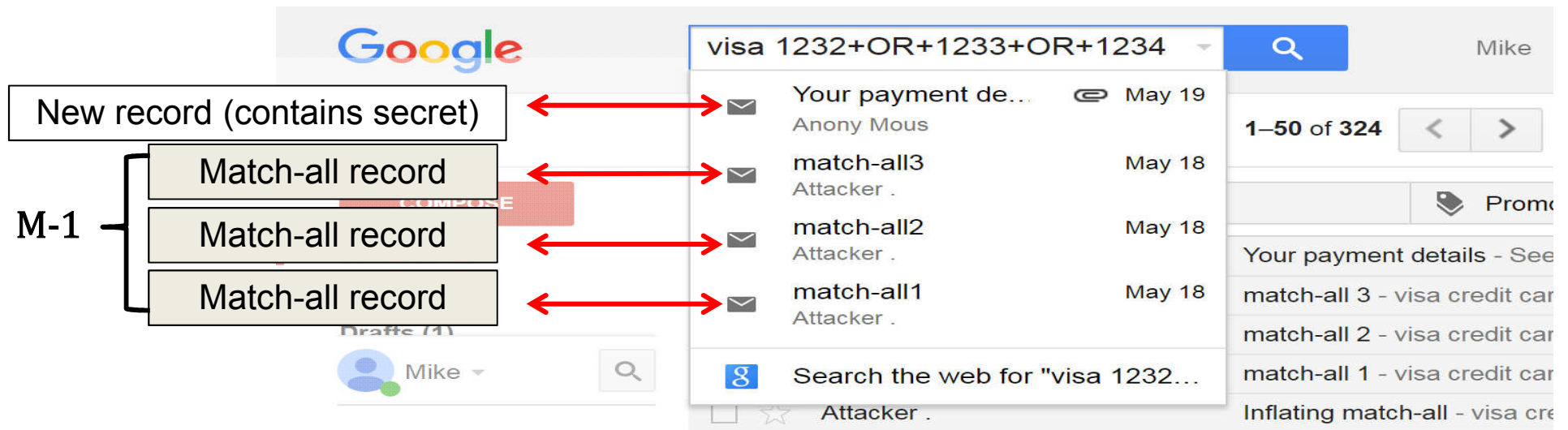
match-all 2 - visa credit card C

match-all 1 - visa credit card C

Inflating match-all - visa credit

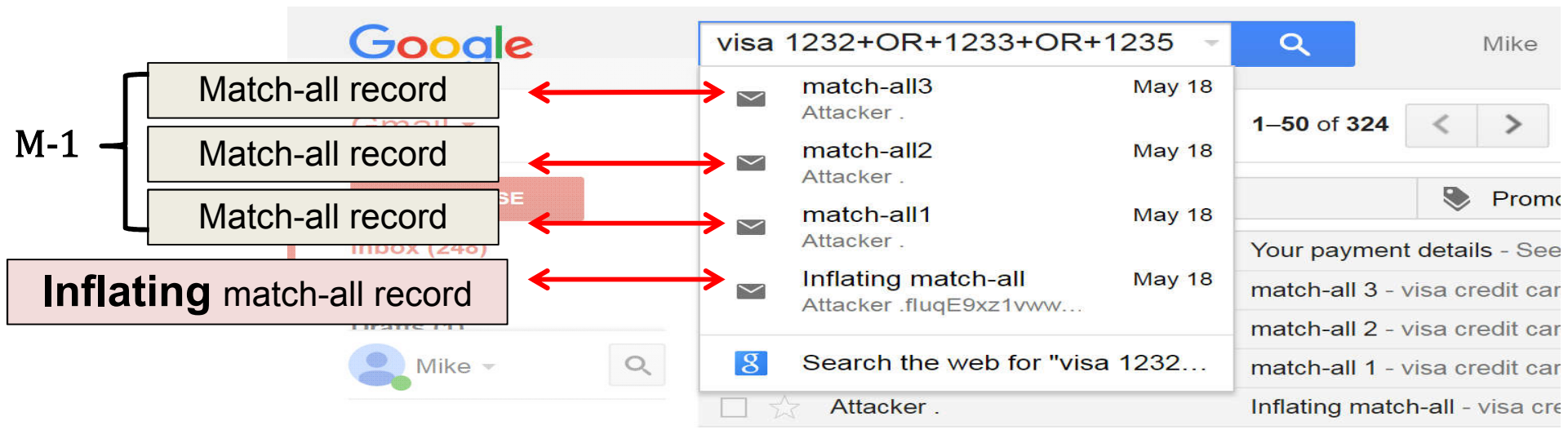
Inflating SO XS-Search attack

- Gmail example: full response (size is small)



Inflating SO XS-Search attack

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



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

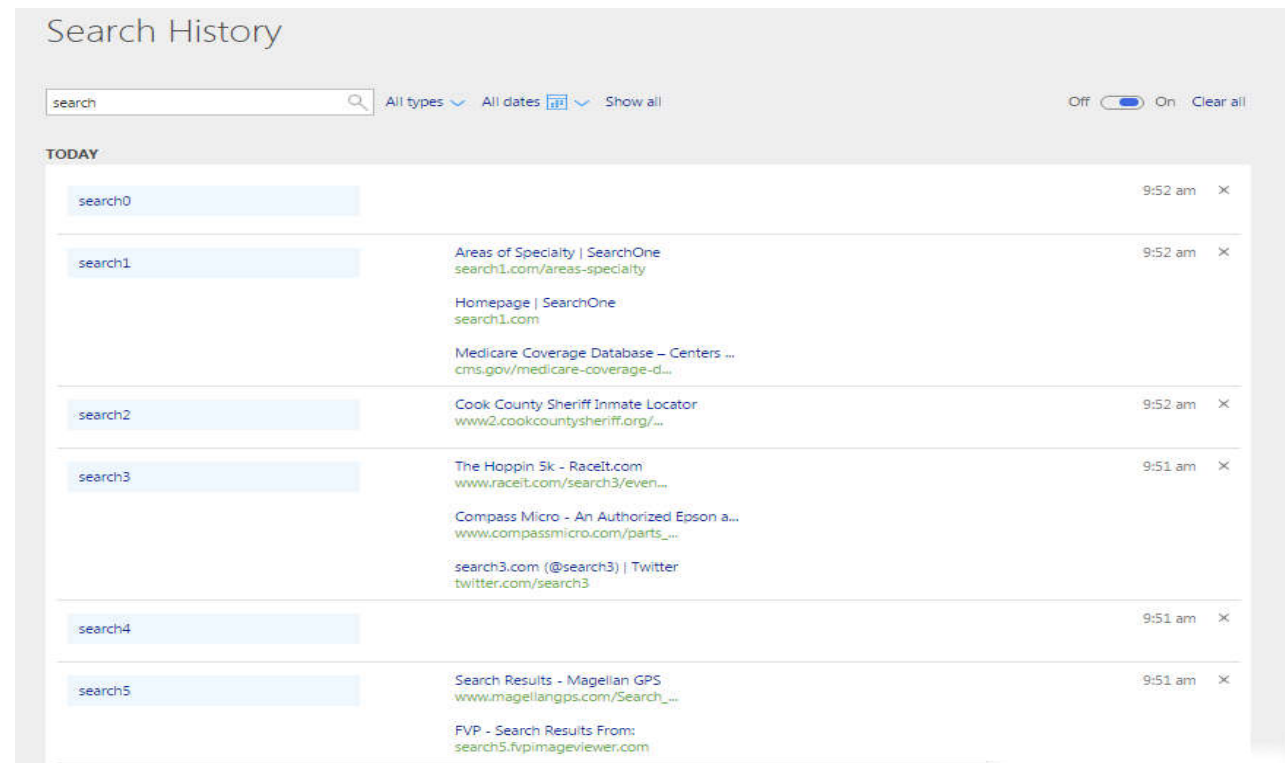
- Emails solution: abuse anti-spam mechanisms
- The planted emails will be marked as spam
 - Users do not get notifications for spam emails
 - Users (usually) do not visit their spam folder
- Only when it is possible to search in the spam and in the other folders using the same request
 - E.g., Gmail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

Stealthy SO XS-Search attacks

- Search history
- Two requirement 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!

Nethanel Gelernter

CYBERP**ON**


The College of Management
Academic Studies

Questions?

Nethanel Gelernter

CYBERP**ON**


The College of Management
Academic Studies