Top 10 Web Hacking Techniques of 2014
Special Edition

COVERT TIMING CHANNELS BASED ON HTTP CACHE HEADERS

Denis Kolegov, Oleg Broslavsky, Nikita Oleksov

F5 Networks

Tomsk State University Information Security and Cryptography Department

ZeroNights (13-14 November 2014) Moscow, Russia

SibeCrypt (8-13 September 2014) Ekaterinburg, Russia

Who we are?

- Denis Kolegov
  - Sr. security test engineer at F5 Networks
  - PhD, associate professor at Tomsk State University Information Security and Cryptography Department
- Oleg Broslavsky
  - 3rd year student at Tomsk State University Information Security and Cryptography Department
  - Member of TSU's SiBears Capture the Flag team
- Nikita Oleksov
  - 3rd year student at Tomsk State University Information Security and Cryptography Department
  - Member of TSU's SiBears Capture the Flag team

Prologue

This is a presentation of our research devoted to new covert timing channels based on HTTP cache headers

We discovered previously unknown techniques and introduced them on the ZeroNights and SibeCrypt security conferences in 2014

In the current list of «Top 10 Web Hacking Techniques of 2014» there are many valuable and significant attacks and, of course, we don't think that our work is the best. We are considering participation in 2014 Hacks as opportunity for feedback and information sharing

Summary

We found and investigated previously unknown covert timing channels based on main HTTP cache headers

We explored different properties of these covert channels (e.g., throughput, anonymity, reliability)

We implemented most efficient ETag-based covert channel in Browser Exploitation Framework (BeEF) for covert communications

Also we implemented ETag-based covert timing channel providing anonymity property to attackers in Google Drive environment

Introduction

A covert channel is a path that can be used to transfer information in a way not intended by the system's designers (CWE-514)

A covert storage channel transfers information through the setting of bits by one program and the reading of those bits by another (CWE-515)

Covert timing channels conveys information by modulating some aspect of system behavior over time, so that the program receiving the information can observe system behavior and infer protected information (CWE-385)

Introduction

HTTP is one of the most used protocol on the Internet so detections of the covert channels over the HTTP is an important research area

HTTP timing channels have received little attention in computer security

The main HTTP covert timing channel throughput is equal to 1.82 bps [1]. This channel doesn't use any HTTP mechanisms and is based on TCP/IP timing channel

Server-to-Client DNS-tunnel [3] implemented in BeEF has throughput equal to 10 bit/s

Introduction

HTTP Covert Channels' Usage

- Implementation of communication channels in targeted browsers (BeEF)
- Botnet command and control channels
- Key exchange in malicious software
- Transferring of illegal content

General HTTP Cache Headers

RESPONSE (SERVER) HEADERS

- Last-Modified
- ETag

REQUEST (CLIENT) HEADERS

- If-Modified-Since
- If-Unmodified-Since
- If-Match
- If-Non-Match
- If-Range

Directions of Covert Channels

Covert channels can be classified as client – server channels and server – client channels

Client-server covert channels are easier to implement. Server-client channels are more complicated and most of them are timing channels

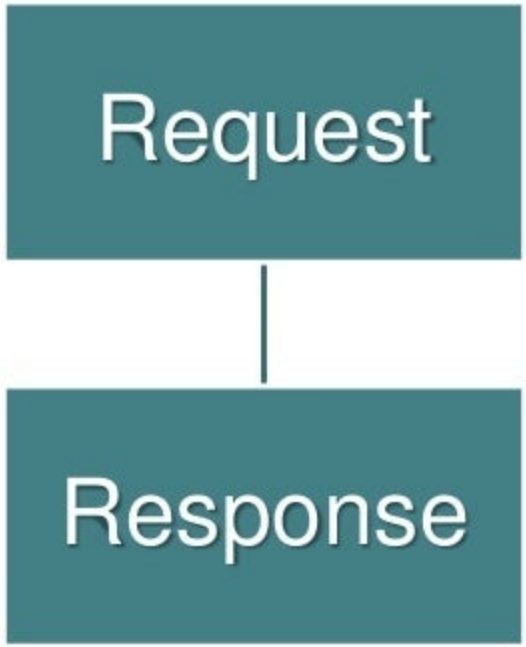
For example, covert storage channel via If-Range header can be implemented by the following way

```
GET / HTTP/1.1
Host: evil.com
If-Range: 120c7bL-32bL-4f86d4105ac62L
…
```

Hex-encoded data

Last-Modified Response Header

Last-Modified HTTP header stores a date of the last web entity's modification



GET / HTTP/1.1
Host: evil.com

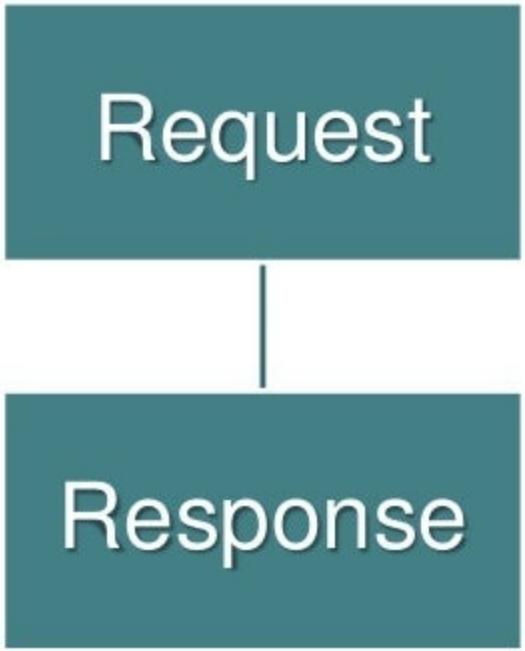
HTTP/1.1 200 OK
Server: nginx/1.1.19
Date: Wed, 02 Apr 2014 14:33:39 GMT
Content-Type: text/html
Content-Length: 124
Last-Modified: Wed, 02 Apr 2014 14:33:39 GMT
Connection: keep-alive
(data)

ETag Response Header

The ETag value is formed from the hex values by he following way

120c7bL-32bL-4f86d4105ac62L

file's inode *size* *last-modified time* (*mtime*)

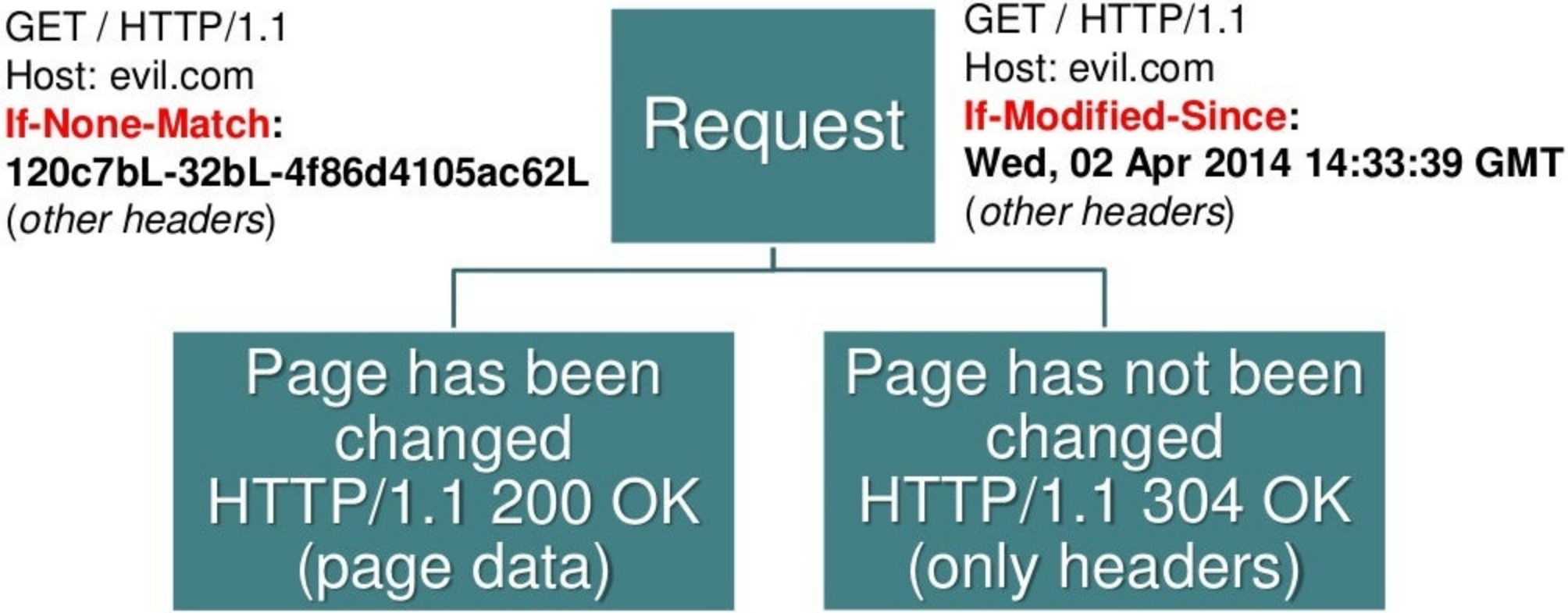


GET / HTTP/1.1
Host: evil.com

HTTP/1.1 200 OK
Server: Apache/2.2.22 (Ubuntu)
Date: Wed, 02 Apr 2014 14:33:39 GMT
Content-Type: text/html
Content-Length: 124
ETag: 120c7bL-32bL-4f86d4105ac62L
Connection: keep-alive
(data)

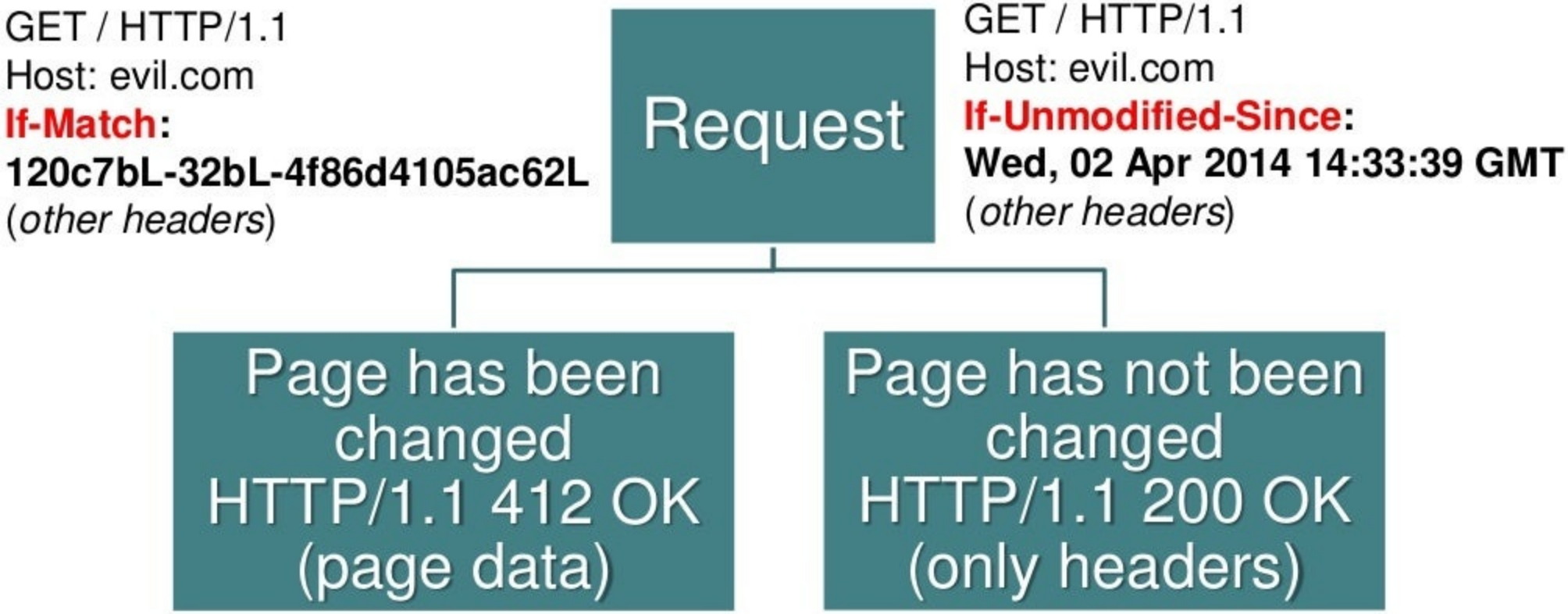
Common Usage of Cache Request Headers

HTTP cache headers allows to web-browsers not to download a page if it hasn't been changed since the certain time

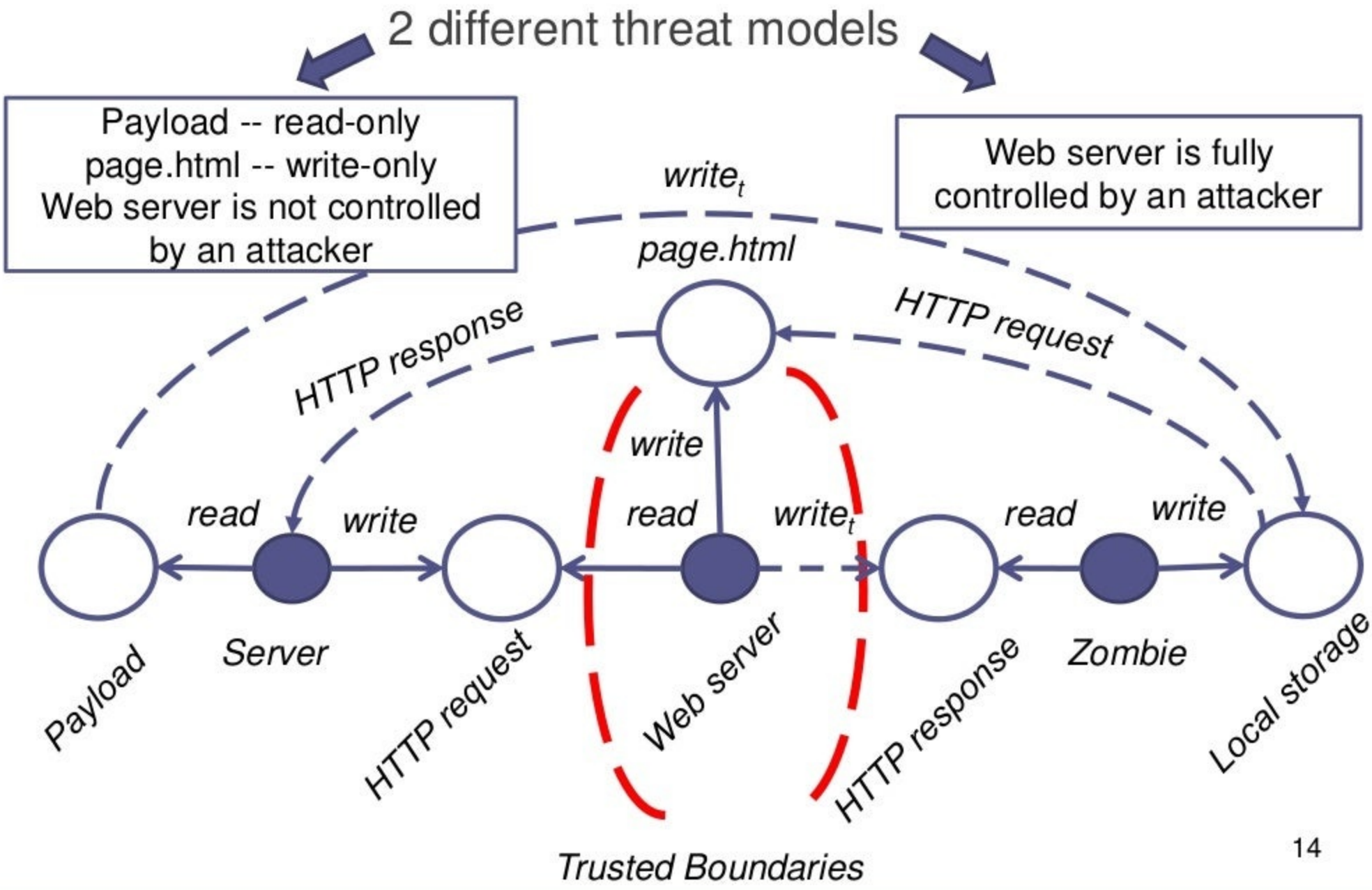


Common Usage of Cache Request Headers

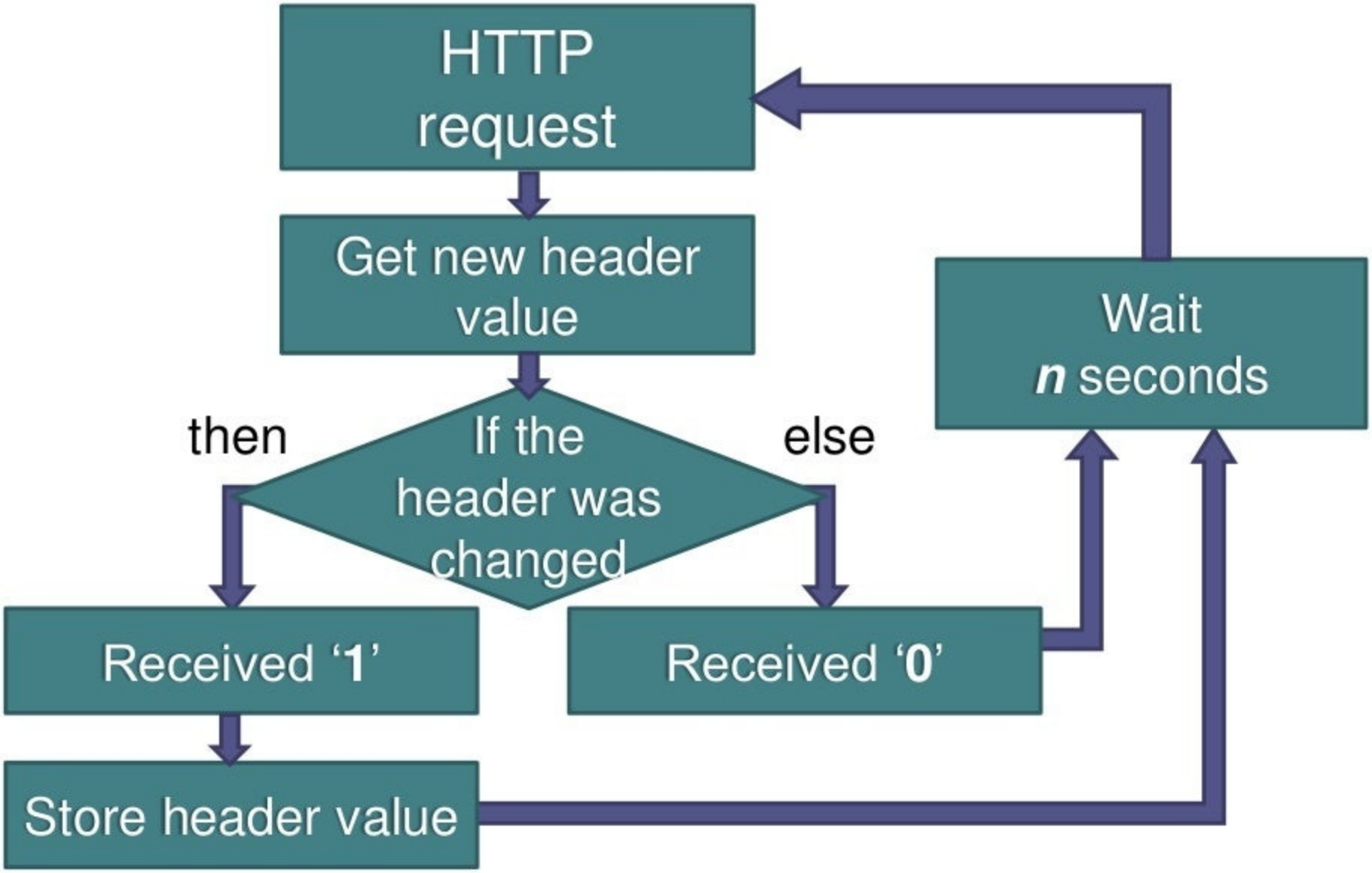
Second pair of headers does the same as previous but with logically inverse condition



DFD Threat Model



General Covert Channels Scheme



General HTTP Cache Headers

RESPONSE (SERVER) HEADERS

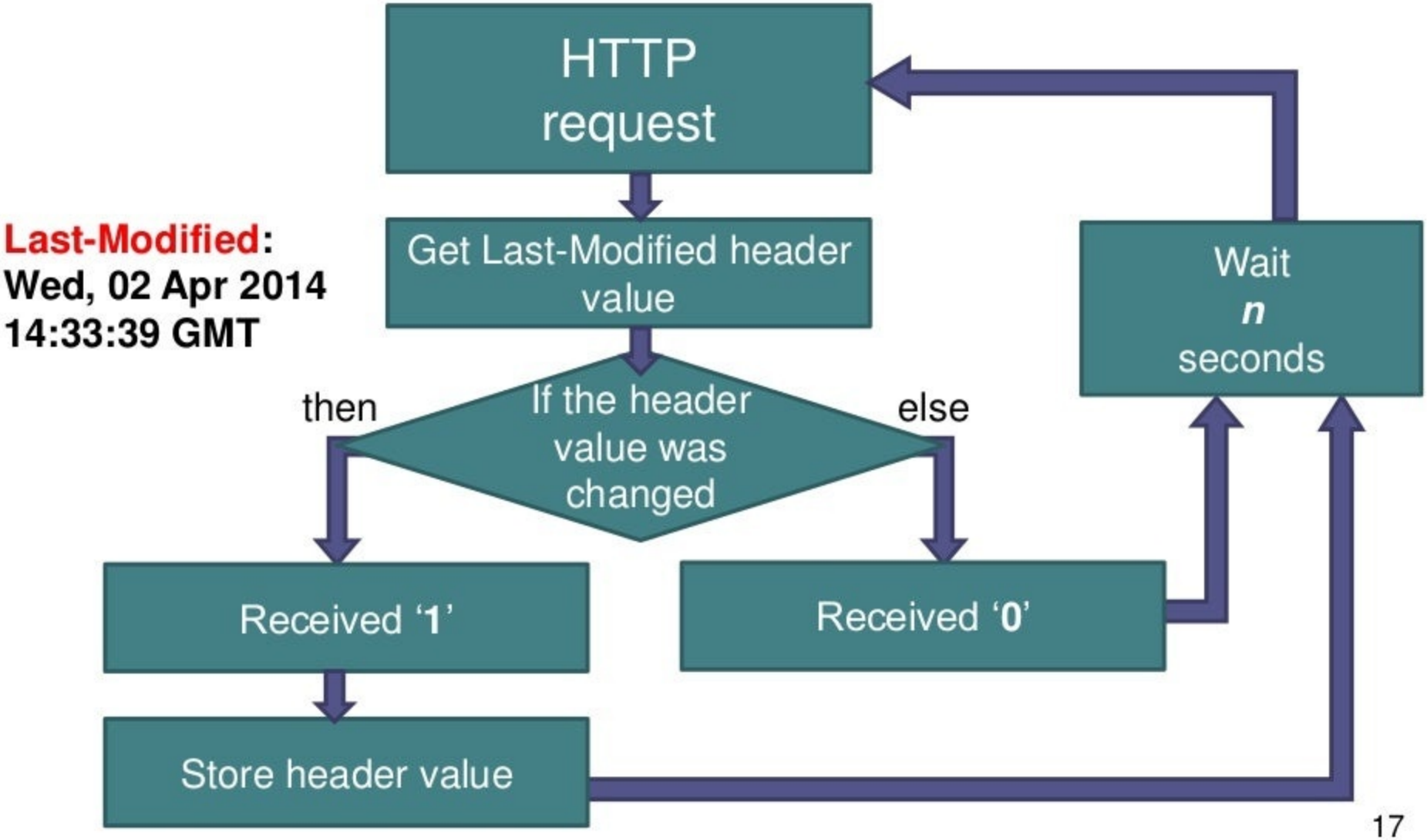
- Last-Modified
- ETag

REQUEST (CLIENT) HEADERS

- If-Modified-Since
- If-Unmodified-Since
- If-Match
- If-Non-Match
- If-Range

Last-Modified Based Channels

Last-Modified header value covert channel



Classification

Covert Timing Channels based on HTTP-date entities

- Based on Last-Modified header
- Based on If-Modified-Since header
- Based on If-Unmodified-Since header

Covert Timing Channels based on ETag entities

- Based on ETag header
- Based on If-Match header
- Based on If-None-Match header

Last-Modified based Channel

Zombie requests page.html and receives the HTTP response that contains initial Last-Modified value *HTTP-date*$_0$

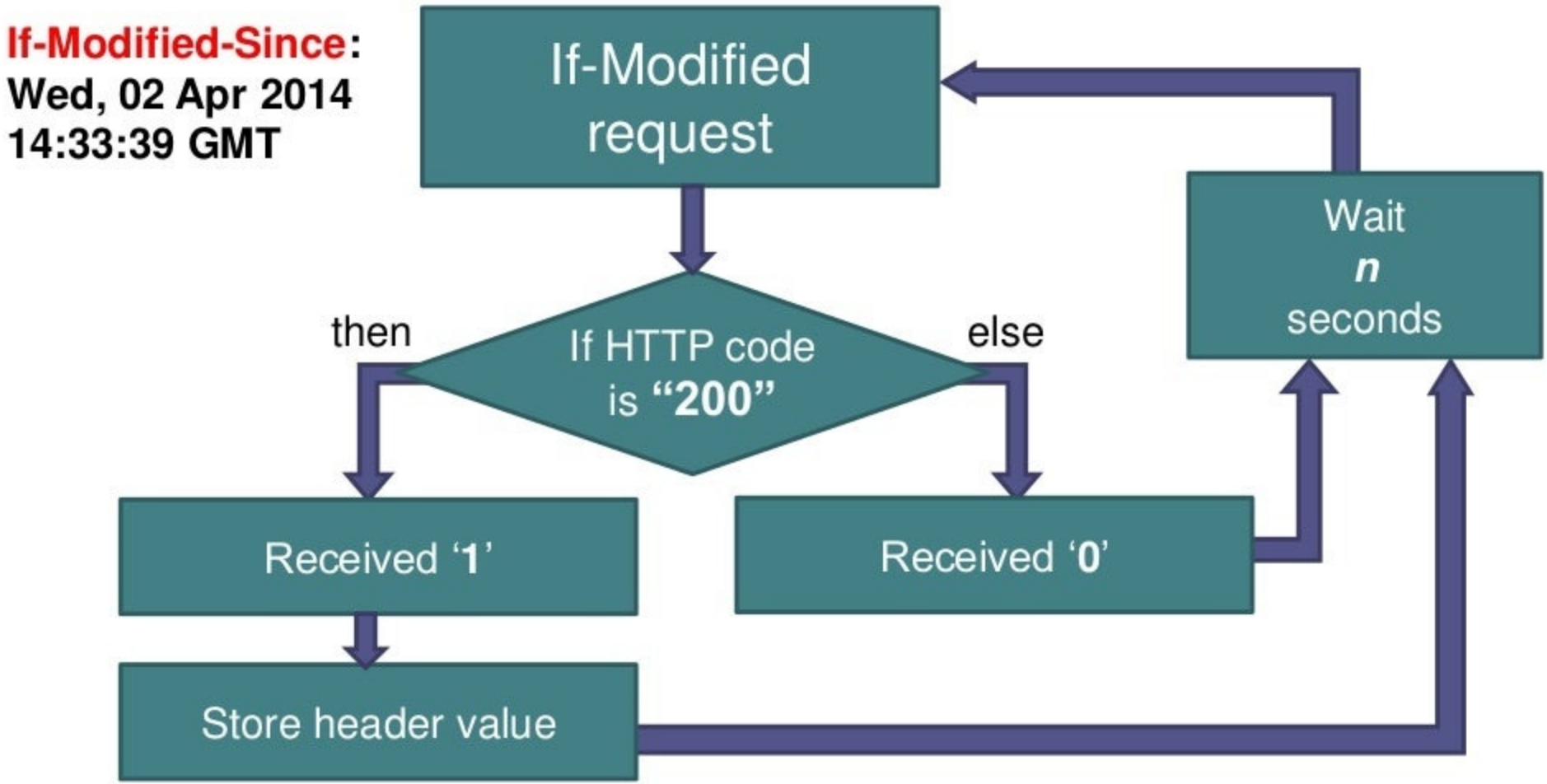
Server performs read or write access to the page.html

To obtain 1 bit of information Zombie request page.html again and compares the new Last-Modified value *HTTP-date*$_1$ with the old one

If *HTTP-date*$_1$ and *HTTP-date*$_0$ is not the same, so the Server has sent 1, otherwise Server has sent 0

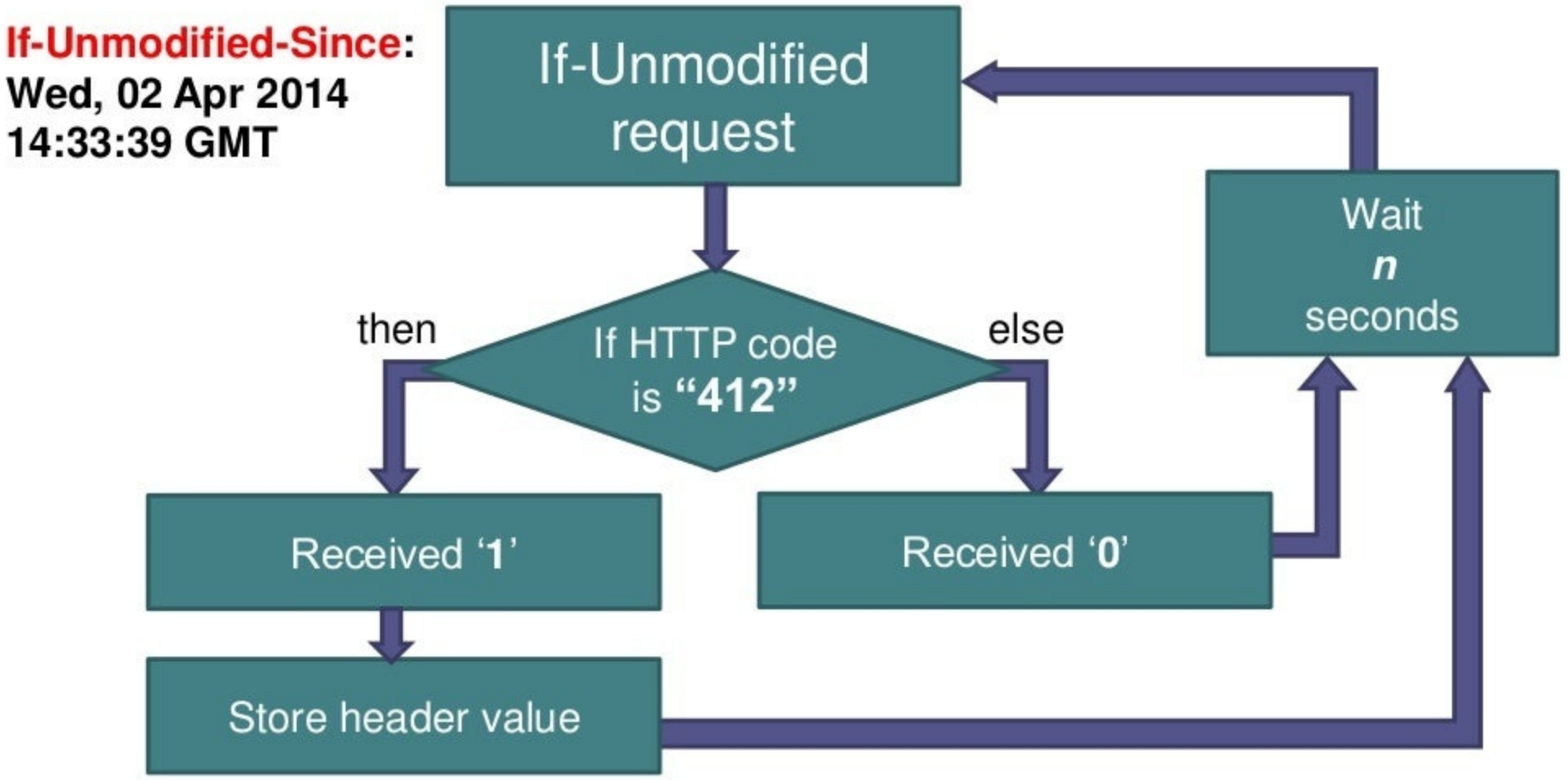
If-Modified-Since based Channel

Covert channel based If-Modified-Since header



If-Unmodified-Since based Channel

Covert channel based on If-Unmodified-Since header



ETag based Channel

Zombie requests page.html and receives the HTTP response that contains initial ETag value *entity-tag*$_0$

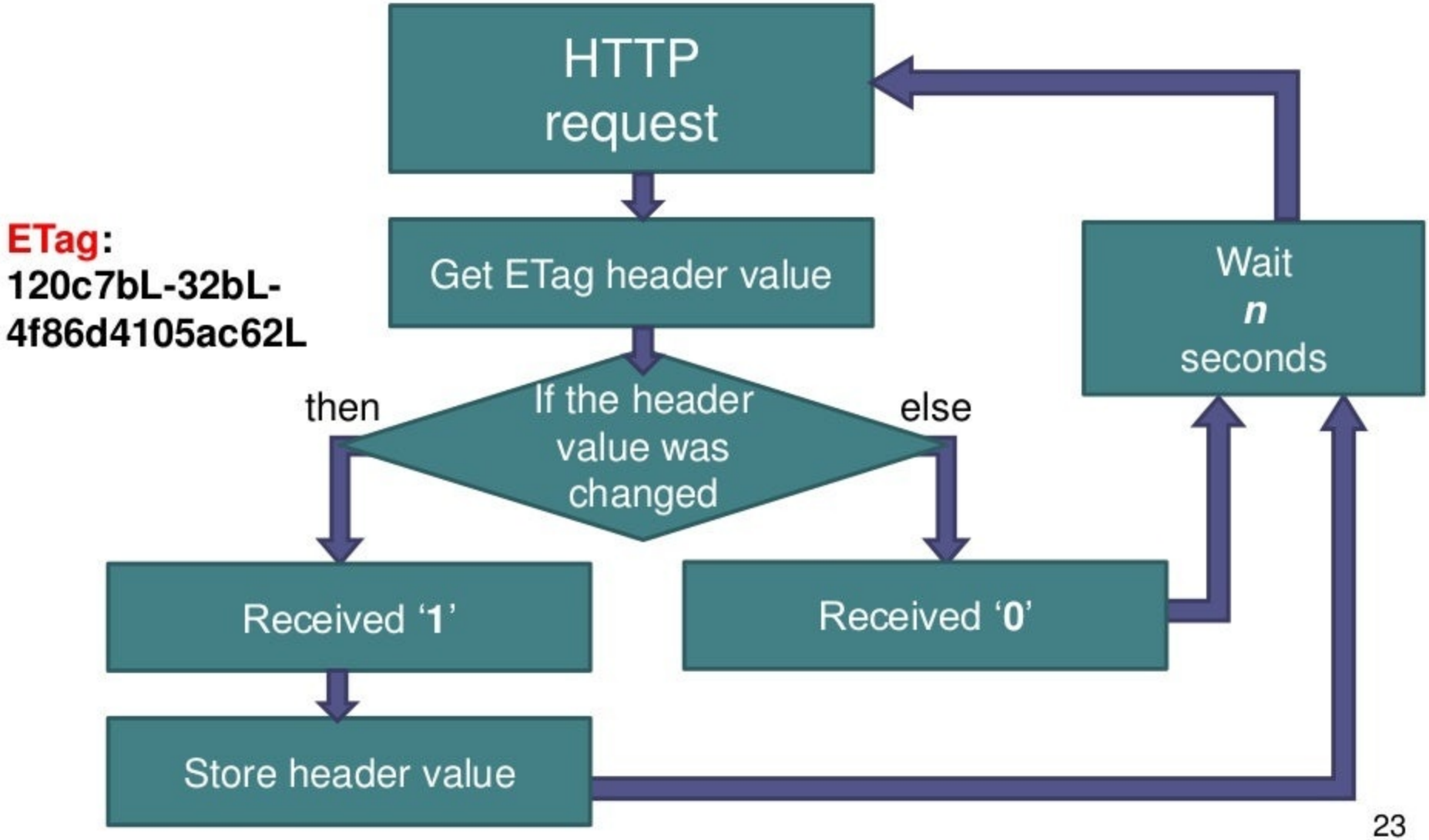
Server performs read or write access to the page.html

To obtain 1 bit of information Zombie request page.html again and compares the new ETag value *entity-tag*$_1$

If *entity-tag*$_1$ and *entity-tag*$_0$ is not the same, so the Server has sent 1, otherwise Server has sent 0

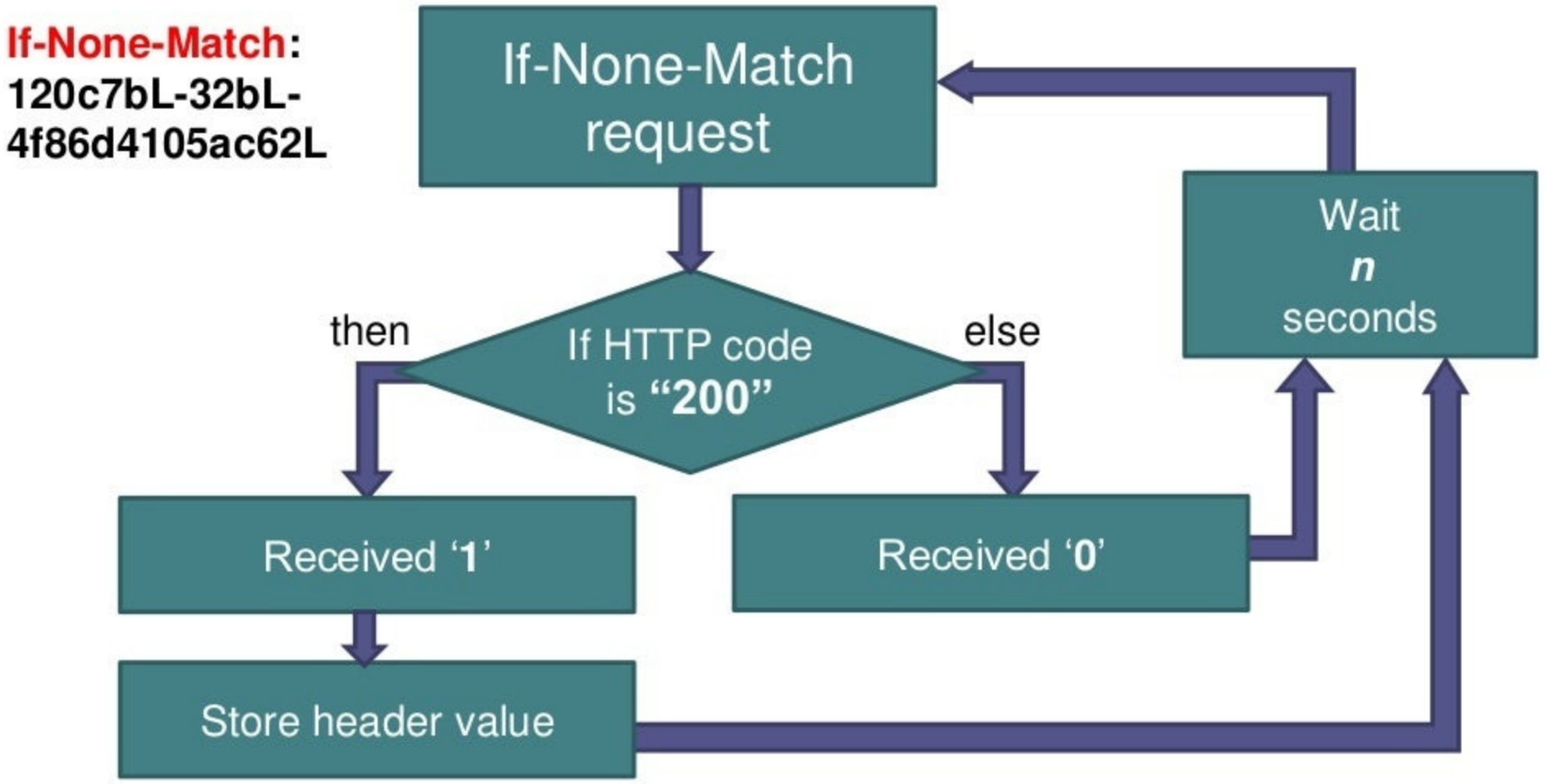
ETag based Channel

Covert channel based on ETag header



ETag based Channel

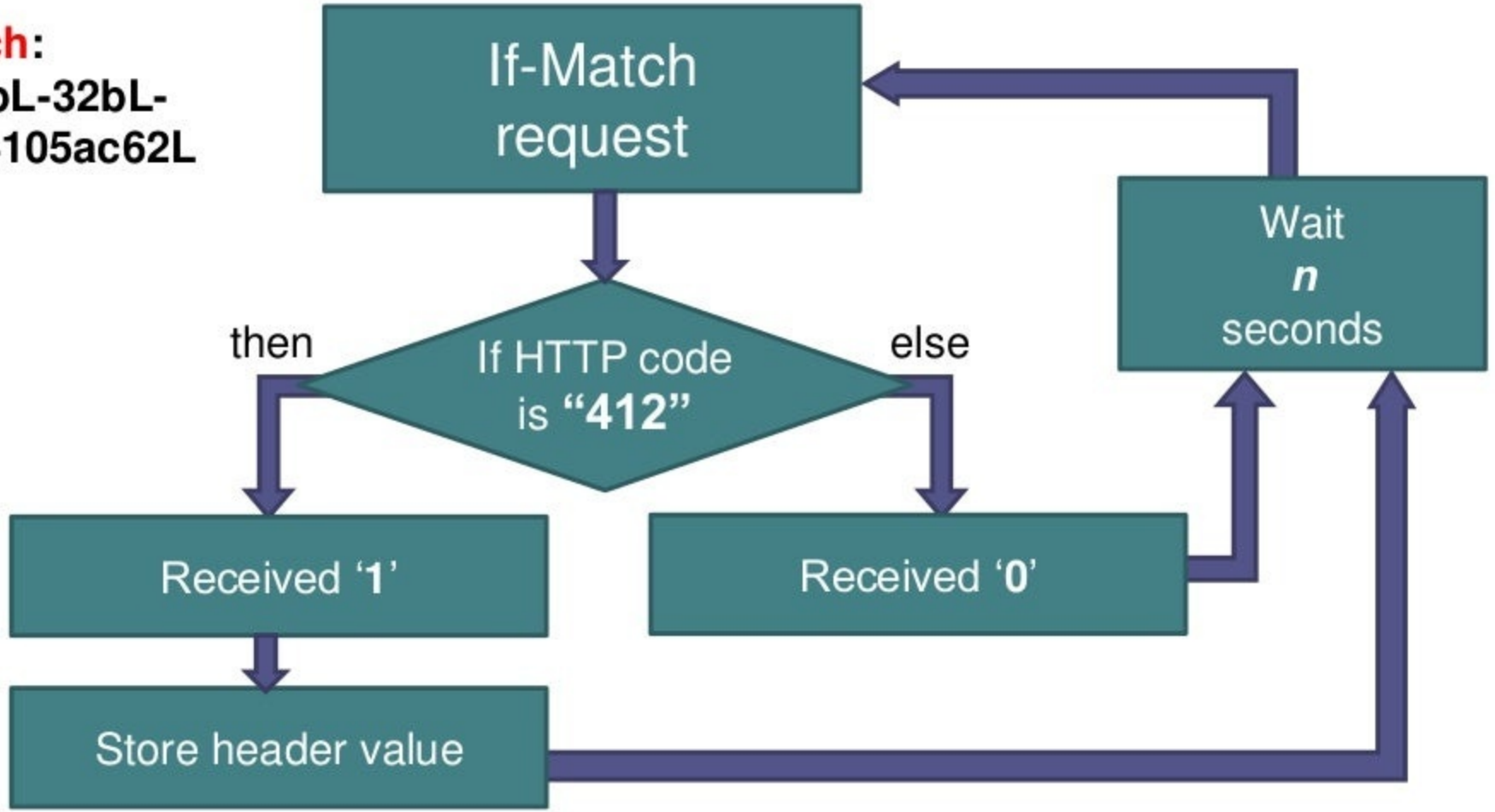
Covert channel based on If-None-Match header



ETag based Channel

Covert channel based If-Match header

If-Match:
120c7bL-32bL-4f86d4105ac62L



Software Implementation

In tons of possible ways we focused on

- Python – Socket library
- C++ – Boost ASIO library
- C – simple C socket library

We chose C due to its highest performance (among these ways) and decent stability

First threat model was chosen because of its minimal requirements

Issues

Some problems we solved during implementation

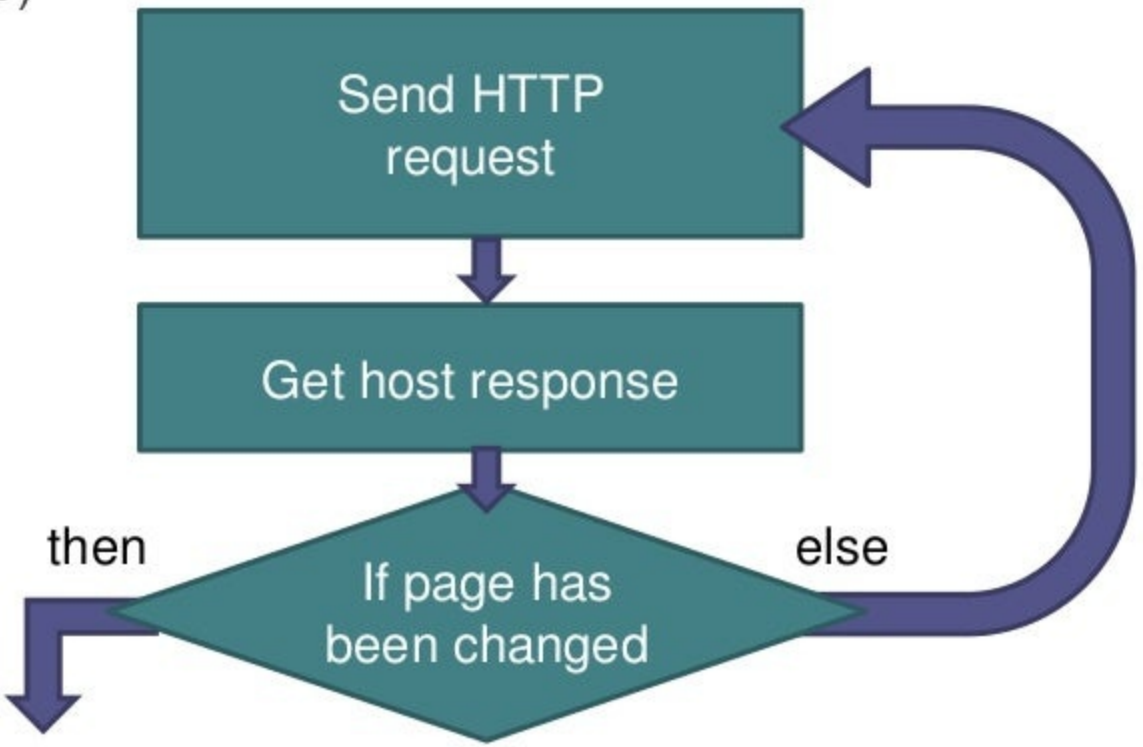
Issue	Solution
Server-client synchronization	Special synchronizing function
Different time of requests	Dynamic sleep time
Lateness after sleep	“Active” sleep
High CPU load with “active sleep”	“Dynamic” and “active” sleep combination

Issues

Necessity of synchronization “read” (web client) and “write” (host) services

Solution

Synchronizing function that does requests at a maximum speed (without sleep)

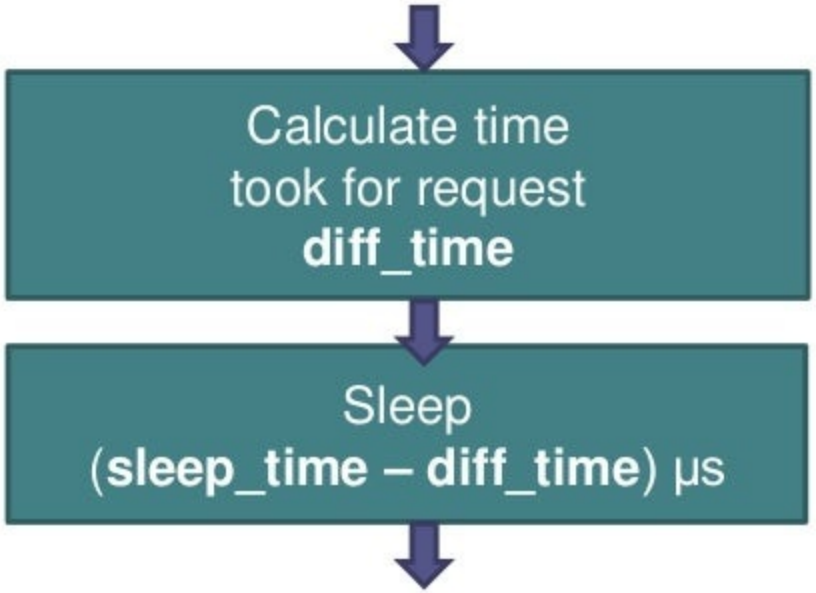


Issues

Different time of requests can break services synchronization

Solution

Dynamic sleep time equals to **sleep_time** – **diff_time**

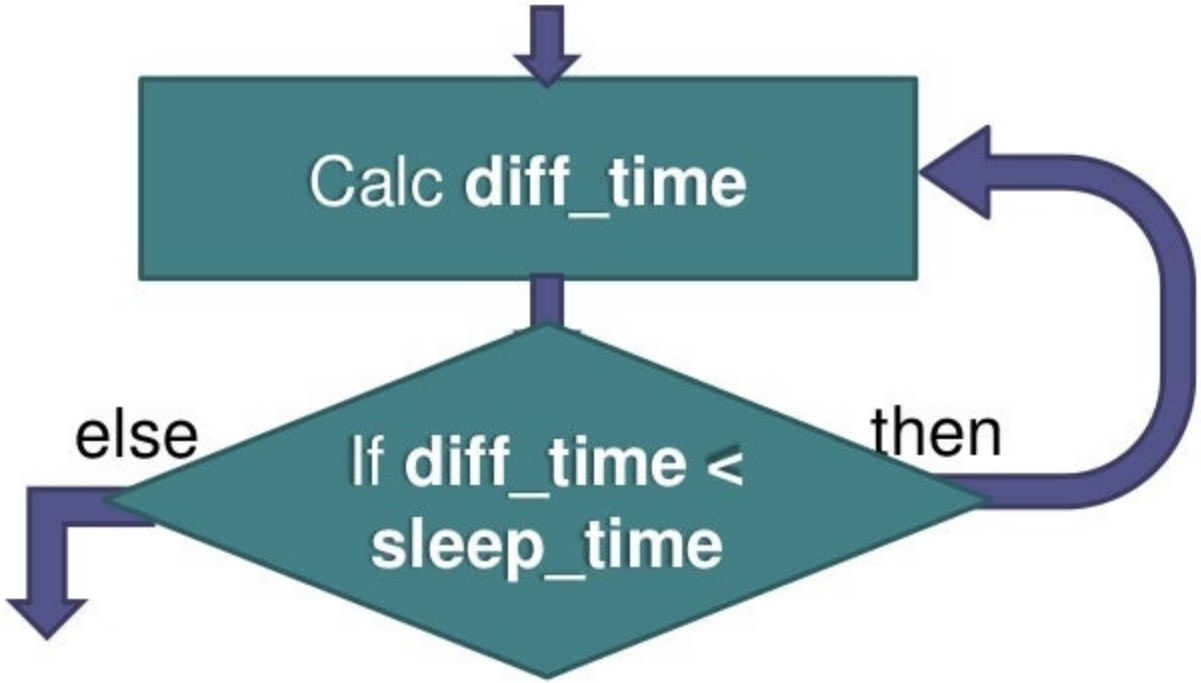


Issues

Inaccurate sleep - after sleep (*usleep*() is used) the program can awake with 10-200μs lateness

Solution:

Use “active sleep” - calculation time difference between last request and current moment while it is less than *sleep_time*

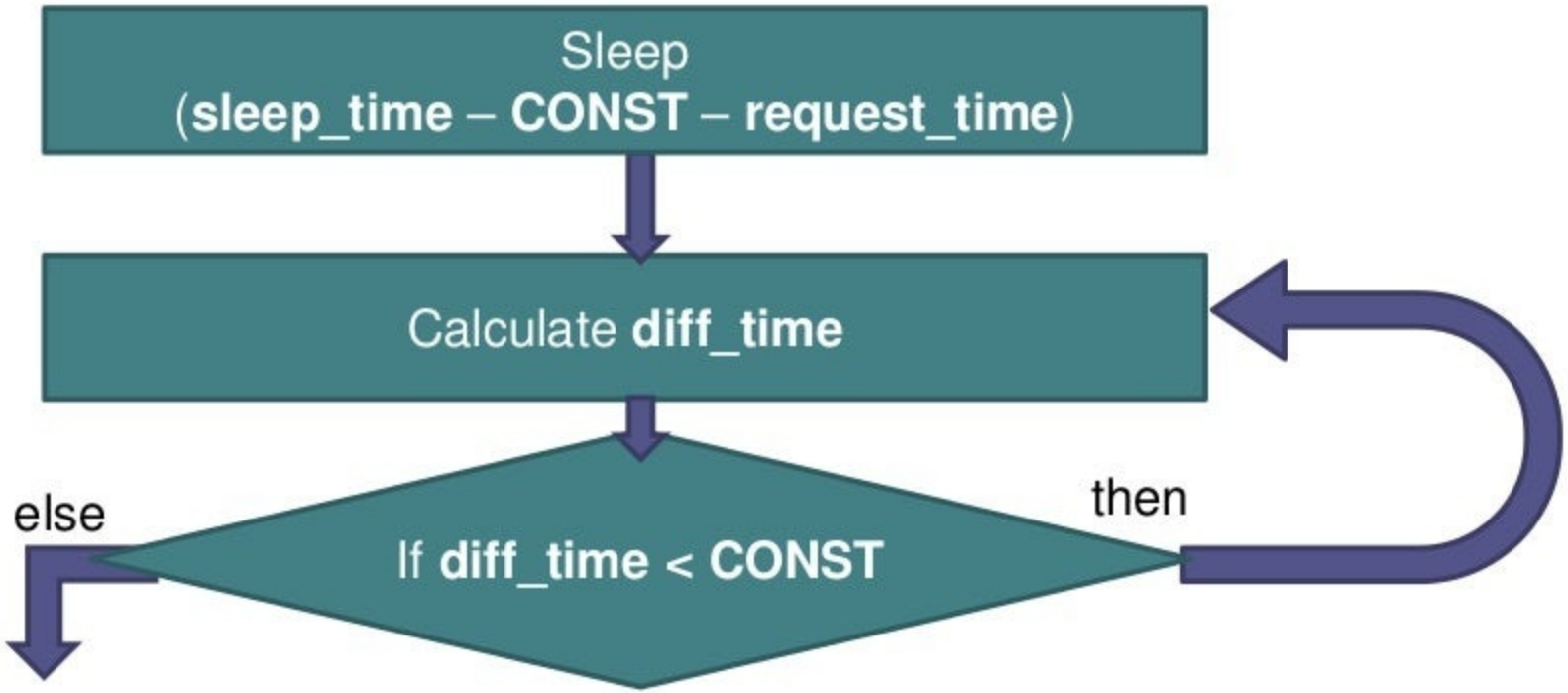


Issues

High CPU load with “active sleep”

Solution

Combine “active” and “dynamic” sleep



where CONST is constant about 1000 µs (or less depending on PC performance)

Experiment 1

Sleep time	Min start sequence	Avg sequence	Max sequence	Speed	Accuracy
1 second	3200 bits	8848 bits	19712 bits	1bit/s	99,82%
2 seconds	3400 bits	10145 bits	22143 bits	0.5 bit/s	99,87%

- C-based implementation in the first threat model
- Min start sequence – minimum number of bits passed from the beginning of a conversation till the first mistake
- Avg and Max sequence – number of bits passed without any mistakes in a row in average and at best
- Accuracy – percent of correctly transmitted bits

Experiment 2

Sleep time	Min *start* sequence	Avg sequence	Max sequence	Speed	Accuracy
1 second	3200 bits	8848 bits	19712 bits	1bit/s	99,82%
0.5 seconds	2400 bits	8142 bits	18123 bits	2 bit/s	99,5%

- C-based implementation in the first threat model
- ETag contains *mtime* (last modified time with microsecond accuracy), so theoretical channel capacity is bigger than its practically possible one.
- Maximum practical speed of the covert channels is about 1 bit per $(2L+T)$ seconds, where L is HTTP latency between u_2 and s_1 and T is a time that is needed for auxiliary operations

Google Drive API Anonymity Channel

Most of the cloud services for file hosting like Dropbox, Google Drive and others allow users to operate with files' ETags and other cache-control headers

So it is possible to implement ETag based covert timing channel in the first threat model: there are channel processes Server($attacker_1$) and Zombie ($attacker_2$) on different hosts and fully trusted web server https://drive.google.com/drive/ with some file hosted on it. The only requirement for that is file should be accessible for writing by $attacker_1$ and for reading by $attacker_2$

Google Drive API Anonymity Channel

Covert channel's logic is the same as before:

- $attacker_1$ sends a request to Google Drive API
  POST https://www.googleapis.com/drive/v2/files/fileId/touch
  to modify file's last access time (and hence ETag)
- $attacker_2$ sends a request to Google Drive API
  GET https://www.googleapis.com/drive/v2/files/fileId
  to get file's metadata (including ETag)

This channel has property that provides anonymity for communications between Server and Zombie

Experiment 3

Google Drive API anonymity covert channel based on ETag header

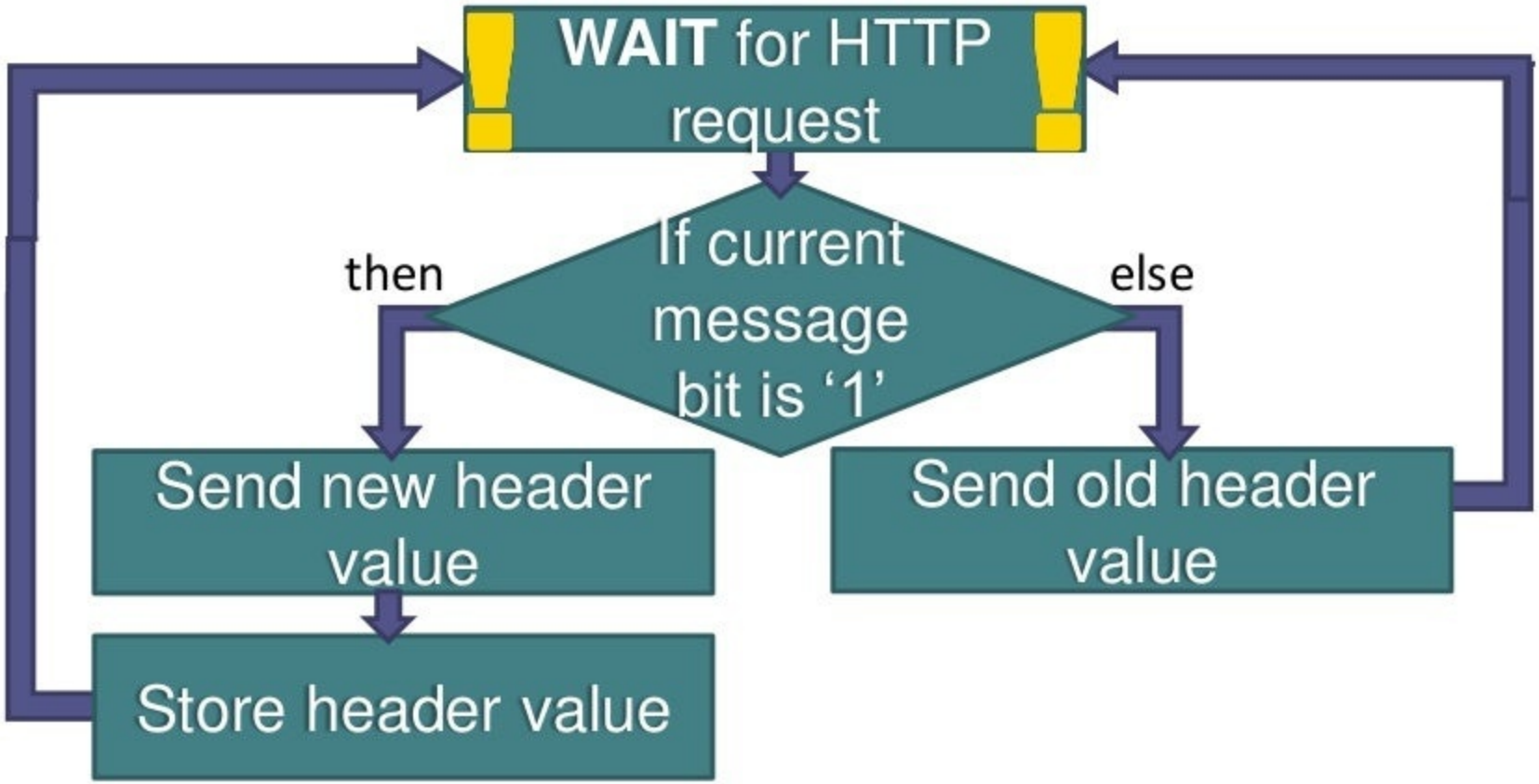
Message length	256 bit	512 bit	1024 bit	2048 bit	4096 bit
Accuracy	99.87%	99.84%	99.8%	99.8%	99.78%
Average throughput	2.92 bit/s	2.9 bit/s	2.88 bit/s	2.88 bit/s	2.86 bit/s

Advantages in the First Threat Model

- Anonymity
- Does not modify common HTTP request structure
- Does not require web-server modifications
- Any read-only activity on web page that is used by the channel do not break its work
- Information flow looks like something refreshes a web page every *n* seconds
- Covert channels based on If-* headers can work even if Last-Modified or Etag are disabled

Second Threat Model

In the second threat model we can avoid necessity of client-server synchronization by waiting for the request and responding directly



Experiment 4

C-based client, Apache + PHP-based server

Header	Network	Average HTTP ping	Speed
ETag	Local host	0.55 ms	986 bit/s
	«Digital Ocean» DC LAN	1.63 ms	845.65 bit/s
	LAN	6.9 ms	295.69 bit/s
	Internet	113.2 ms	13.09 bit/s

Experiment 5

C-based client, Flask + Python-based server

Header	Network	Average HTTP ping	Speed
ETag	Local host	0.55 ms	981 bit/s
	«Digital Ocean» DC LAN	1.63 ms	865.83 bit/s
	LAN	6.9 ms	293.9 bit/s
	Internet	103.2 ms	14.39 bit/s

Advantages in Second Threat Model

- Does not modify common HTTP request structure
- Information flow looks like something refreshes a web page every *n* seconds
- Higher throughput
- Reliability
- Simplicity
- This approach is applicable for implementation of covert channels based on HTTP cache headers in browsers

Covert Channels in Browsers

Issues

- Lack of any “sleep” function
- Low accuracy of existing time management functions
- Difficulties with synchronization of covert channel’s server and client

So implementation of the used model is pointless, but it is possible to implement covert channels in these restrictions using controlled web server in the second threat model

Experiment 6

Implementation of ETag-based covert channel in browser (client on JavaScript)

Header	Server	Average HTTP ping	Throughput
Last-Modified	0.045 ms	70 ms	1 bit/s
Last-Modified	18 ms	68 ms	1 bit/s
ETag	Python	66 ms	11.51 bit/s
ETag	PHP	72 ms	10.8 bit/s

Covert Channels in BeEF

"BeEF allows the professional penetration tester to assess the actual security posture of a target environment by using client-side attack vectors."

The main idea was proposed in Kenton Born's paper "Browser-based covert data exfiltration" [2] and is being used in BeEF [3]

To investigate covert timing channels in browsers we implemented server-to-client DNS and ETag Tunnels using AJAX and then added them to BeEF

ETag-based timing channel in BeEF

Issue	Solution
Server-client synchronization	Client does special request to begin conversation
End of message determination	Client receive some special HTTP code in response, e.g. 404 – Not Found or 403 - Forbidden
Single client communication only	Open a session that stores transferring bit number for each client

ETag-based timing channel in BeEF

ETag Tunnel in BeEF consists s of classic two parts

- extension on Ruby, that implements server side logic via couple of web pages mounted to BeEF webserver
- module on JavaScript, that is responsible for receiving information from C&C BeEF server at zombie

Sources

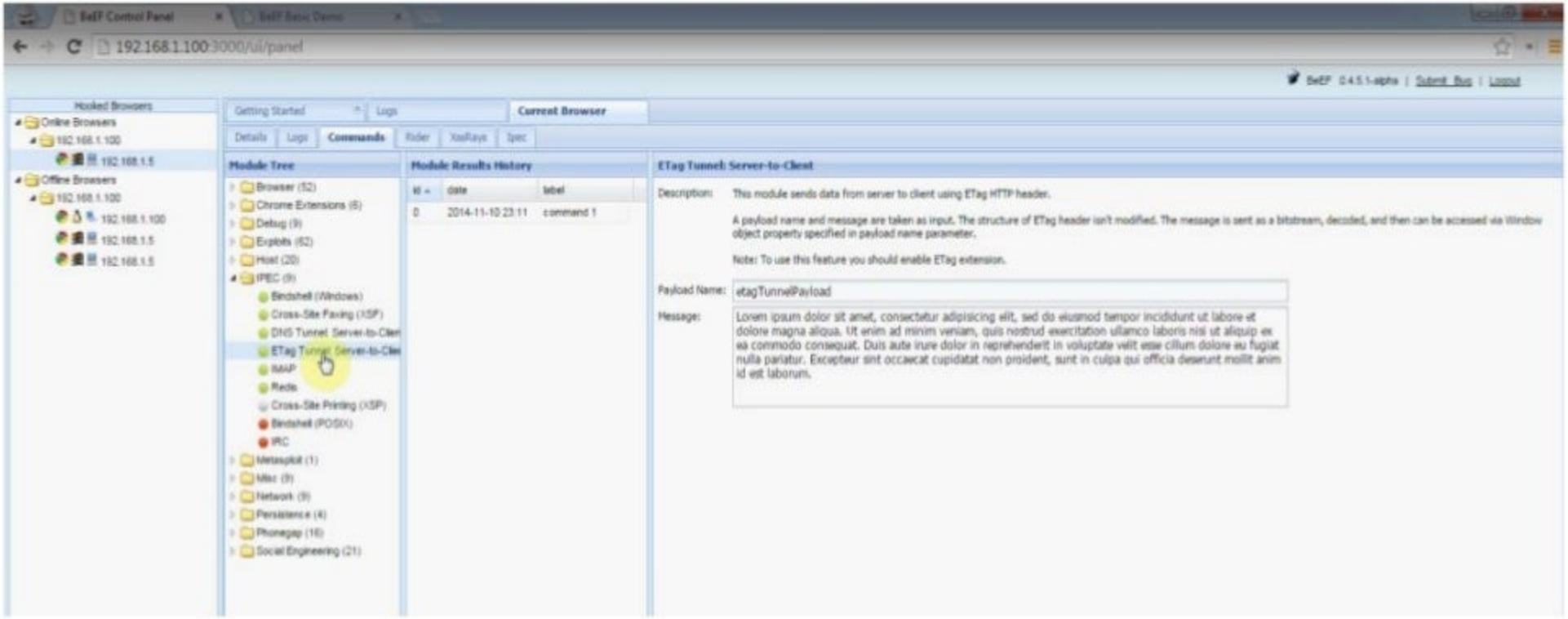
- https://github.com/beefproject/beef/tree/master/modules/ipec/etag_client
- https://github.com/beefproject/beef/tree/master/extensions/etag

Experiment 7

Implementation of ETag-based covert channel in browser (client on JavaScript)

Network	Average ping	Average HTTP ping	256 bit	1024 bit
Local host	0.045 ms	0.6 ms	10.11 bit/s	9.9 bit/s
Local network	18 ms	19.8 ms	10.3 bit/s	9.78 bit/s
Internet	176 ms	360.9 ms	5.09 bit/s	4.97 bit/s

Proof of Concept



http://youtu.be/W2qWA7XUzGQ
https://github.com/beefproject/beef

Bibliography

1. Johnson D., Yuan Bo; Lutz P., Brown E. Covert channels in the HTTP network protocol: Channel characterization and detecting man-in-the-middle attacks. URL: https://ritdml.rit.edu/handle/1850/14797

2. Kenton Born. «Browser-based covert data exfiltration». URL: http://arxiv.org/ftp/arxiv/papers/1004/1004.4357.pdf

3. W. Alcorn, C. Frichot, M. Orru. «The Browser Hacker's Handbook». URL: http://eu.wiley.com/WileyCDA/WileyTitle/productCd-1118662091.html

Denis Kolegov
dnkolegov@gmail.com
@dnkolegov

Oleg Broslavsky
ovbroslavsky@gmail.com
@yalegko

Nikita Oleksov
neoleksov@gmail.com
@neoleksov