SCIENCE
PASSION
TECHNOLOGY

Continuous User Behavior Monitoring using DNS Cache Timing Attacks

Hannes Weissteiner[1] Roland Czerny[1] Simone Franza[1] Stefan Gast[1]
Johanna Ullrich[2] Daniel Gruss[1]

[1]Graz University of Technology | [2]University of Vienna

NDSS 2026

› isec.tugraz.at

DNS

DNS

- Domain Name Service maps domain names to IPs

DNS

- Domain Name Service maps domain names to IPs
- Client queries DNS server to resolve domain

DNS

- Domain Name Service maps domain names to IPs
- Client queries DNS server to resolve domain
- DNS request adds latency to connections

DNS

- Domain Name Service maps domain names to IPs
- Client queries DNS server to resolve domain
- DNS request adds latency to connections
- Local DNS cache prevents unnecessary latency

DNS

- Domain Name Service maps domain names to IPs
- Client queries DNS server to resolve domain
- DNS request adds latency to connections
- Local DNS cache prevents unnecessary latency
- ⇒ DNS cache timing attacks

Motivation

Motivation

- DNS cache timing can reveal recent website visits [1]

Motivation

- DNS cache timing can reveal recent website visits [1]
- Prior work lacked eviction mechanisms

Motivation

- DNS cache timing can reveal recent website visits [1]
- Prior work lacked eviction mechanisms
- Find reliable eviction primitives

Motivation

- DNS cache timing can reveal recent website visits [1]
- Prior work lacked eviction mechanisms
- Find reliable eviction primitives
- $\Rightarrow$ Evict+Reload-style attack

Motivation

- DNS cache timing can reveal recent website visits [1]
- Prior work lacked eviction mechanisms
- Find reliable eviction primitives
- ⇒ Evict+Reload-style attack
- Concurrent work: Moav et. al. [2] focus on the router cache

Measuring the DNS Cache

Execution Contexts

We consider 3 different execution contexts:

Execution Contexts

We consider 3 different execution contexts:

- Native code execution

Execution Contexts

We consider 3 different execution contexts:

- Native code execution
- In browser using JavaScript

Execution Contexts

We consider 3 different execution contexts:

- Native code execution
- In browser using JavaScript
- In browser without JavaScript

Native Code Execution

Native Code Execution

- Most main-stream Linux distributions use `systemd-resolved`

Native Code Execution

- Most main-stream Linux distributions use `systemd-resolved`
- `resolvectl show-caches`: Privileged operation

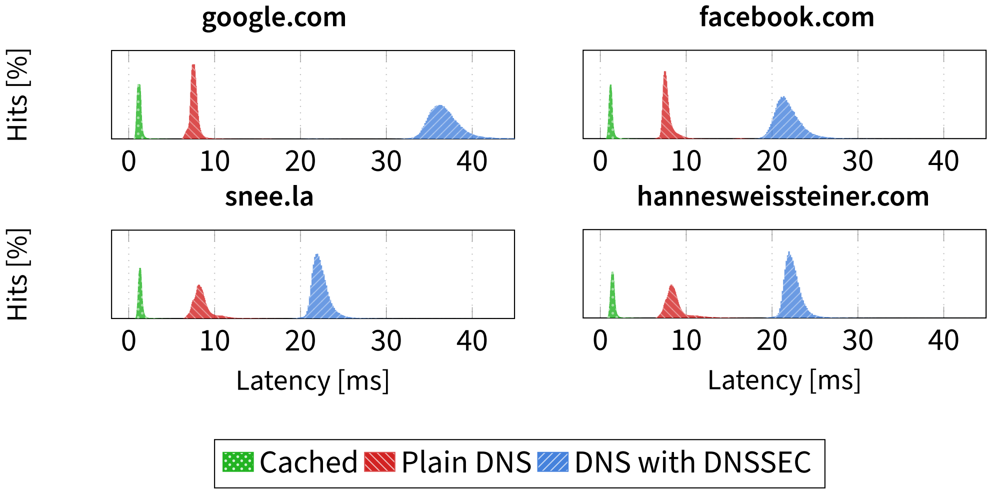
Native Code Execution

- Most main-stream Linux distributions use `systemd-resolved`
- `resolvectl show-caches`: Privileged operation
- `resolvectl query`: reports “Data from: (network | cache)”

Native Code Execution

- Most main-stream Linux distributions use `systemd-resolved`
- `resolvectl show-caches`: Privileged operation
- `resolvectl query`: reports “Data from: (network | cache)”
- Without access to `resolvectl`: Timing attack required

Native Code Execution (Linux)



Native Code Execution (Linux)

Possible from:

Native Code Execution (Linux)

Possible from:

- Virtual Machines in `libvirt`’s default configuration

Native Code Execution (Linux)

Possible from:

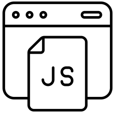
- Virtual Machines in `libvirt`'s default configuration
- Docker containers

Native Code Execution (Linux)

Possible from:

- Virtual Machines in `libvirt`'s default configuration
- Docker containers
- Application Sandboxes

JavaScript



JavaScript

- No DNS resolution API in JavaScript

JavaScript

- No DNS resolution API in JavaScript
- ⇒ We trigger DNS resolutions using `fetch`

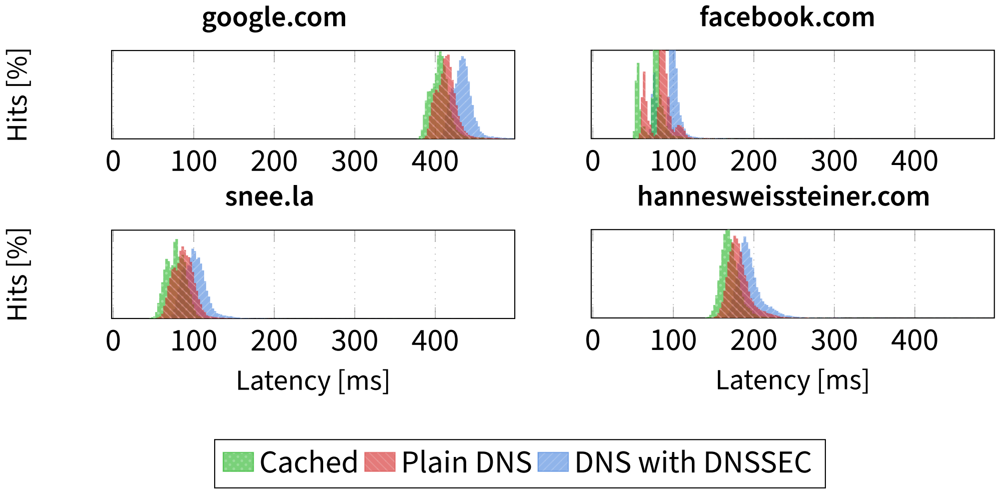
JavaScript

- No DNS resolution API in JavaScript
- ⇒ We trigger DNS resolutions using `fetch`
- `fetch` accesses the target server ⇒ Noise

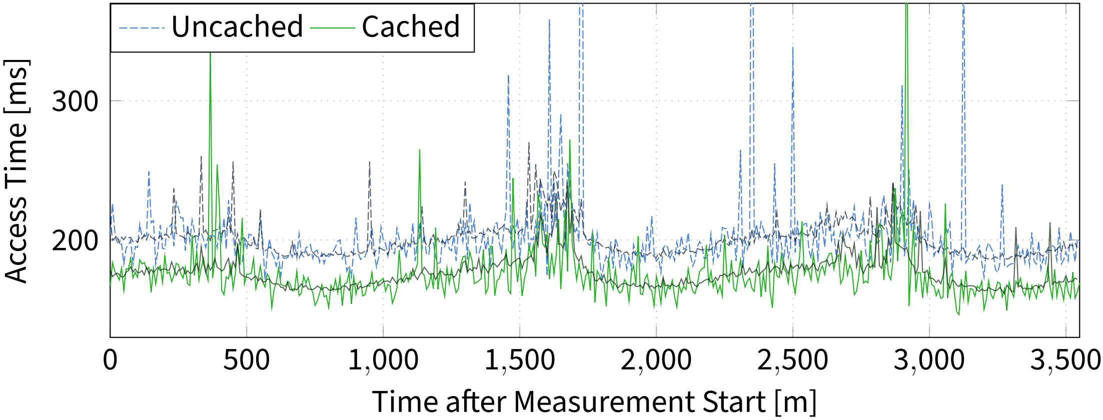
JavaScript

- No DNS resolution API in JavaScript
- ⇒ We trigger DNS resolutions using `fetch`
- `fetch` accesses the target server ⇒ Noise
- `CORS` avoids most data transfer

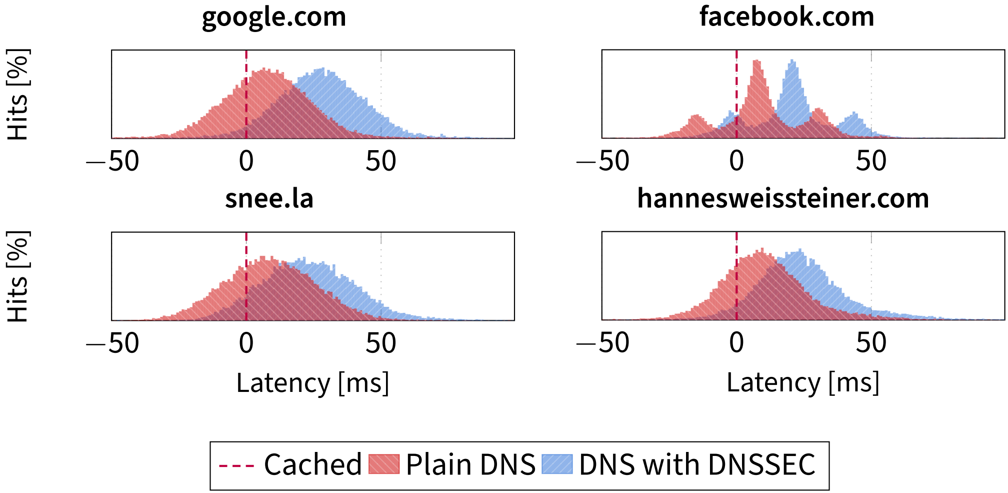
JavaScript


google.com
facebook.com
snee.la
hannesweissteiner.com
Hits [%]
Hits [%]
0
100
200
300
400
Latency [ms]
Latency [ms]
Cached
Plain DNS
DNS with DNSSEC

JavaScript


Uncached
Cached
Access Time [ms]
300
200
0
500
1,000
1,500
2,000
2,500
3,000
3,500
Time after Measurement Start [m]

JavaScript


google.com
facebook.com
snee.la
hannesweissteiner.com
Hits [%]
Hits [%]
−50
0
50
Latency [ms]
Latency [ms]
--- Cached
▨ Plain DNS
▨ DNS with DNSSEC

Scriptless

- What if JavaScript is not available?

Scriptless

- What if JavaScript is not available?
- Make the Browser load resources via HTML/CSS

Scriptless

- What if JavaScript is not available?
- Make the Browser load resources via HTML/CSS
- Time the requests using more requests

Scriptless

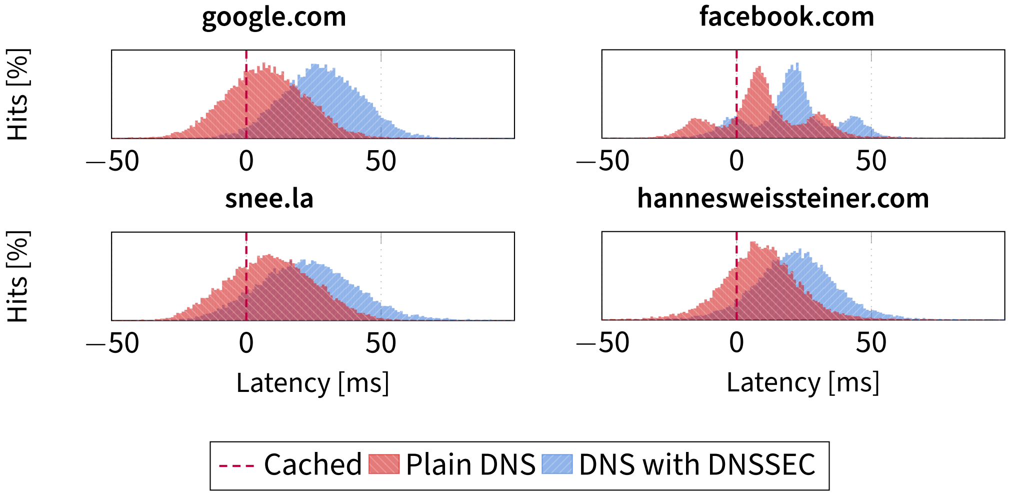
- What if JavaScript is not available?
- Make the Browser load resources via HTML/CSS
- Time the requests using more requests
- Use CSS font alternatives to serialize requests

Scriptless

```
body {
  font-family: "DMTFont";
}

@font-face {
  font-family: "DMTFont";
  src: url("https://attacker.com/measurement-start"),
       url("https://target-domain.com/random-value"),
       url("https://attacker.com/measurement-end");
}
```

Scriptless


google.com
facebook.com
snee.la
hannesweissteiner.com
Hits [%]
Hits [%]
−50
0
50
−50
0
50
−50
0
50
−50
0
50
Latency [ms]
Latency [ms]
Cached
Plain DNS
DNS with DNSSEC

	DNSSEC	Average Offset	Standard Deviation	False Negatives
Native	✓	20.663 ms	1.677 ms	0.000 %
	✗	6.701 ms	1.517 ms	0.006 %
JavaScript	✓	20.356 ms	16.871 ms	13.641 %
	✗	8.942 ms	17.487 ms	24.192 %
JS 50 Mbit/s	✓	82.363 ms	18.041 ms	1.110 %
	✗	22.902 ms	31.044 ms	13.209 %
JS 300 Mbit/s	✓	35.998 ms	35.368 ms	9.839 %
	✗	16.567 ms	34.029 ms	19.630 %
Scriptless	✓	20.744 ms	17.102 ms	12.494 %
	✗	9.395 ms	18.039 ms	23.463 %

Evicting the DNS Cache

Eviction

Eviction

- Required to achieve continuous monitoring

Eviction

- Required to achieve continuous monitoring
- Faster eviction reduces measurement blind spot

Eviction

- Required to achieve continuous monitoring
- Faster eviction reduces measurement blind spot
- We must evict all target entries

Eviction

- Required to achieve continuous monitoring
- Faster eviction reduces measurement blind spot
- We must evict all target entries
- Ideally: DNS cache is completely empty

Eviction

- Required to achieve continuous monitoring
- Faster eviction reduces measurement blind spot
- We must evict all target entries
- Ideally: DNS cache is completely empty
- We demonstrate 4 primitives

Direct Flushing

Direct Flushing

- Clear DNS cache using dedicated command

Direct Flushing

- Clear DNS cache using dedicated command
- Example: `resolvectl flush-caches`

Direct Flushing

- Clear DNS cache using dedicated command
- Example: `resolvectl flush-caches`
- ⊕ Simplest primitive

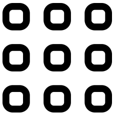
Direct Flushing

- Clear DNS cache using dedicated command
- Example: `resolvectl flush-caches`
- ➕ Simplest primitive
- ➕ Fastest primitive

Direct Flushing

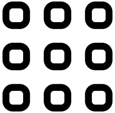
- Clear DNS cache using dedicated command
- Example: `resolvectl flush-caches`
- ⊕ Simplest primitive
- ⊕ Fastest primitive
- ⊖ Only available from unsandboxed native code

Eviction using Individual Requests

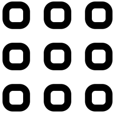


Eviction using Individual Requests

- DNS Standard defines an unbounded cache

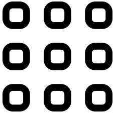


Eviction using Individual Requests



- DNS Standard defines an unbounded cache
- Real resolvers:

Eviction using Individual Requests



- DNS Standard defines an unbounded cache
- Real resolvers: Some kind of practical limit

Eviction using Individual Requests

- DNS Standard defines an unbounded cache
- Real resolvers: Some kind of practical limit
- Evict by flooding the cache with requests

Eviction using Individual Requests

- DNS Standard defines an unbounded cache
- Real resolvers: Some kind of practical limit
- Evict by flooding the cache with requests
- Requires knowledge of cache size and eviction policy

Eviction using Individual Requests

- DNS Standard defines an unbounded cache
- Real resolvers: Some kind of practical limit
- Evict by flooding the cache with requests
- Requires knowledge of cache size and eviction policy
- `systemd-resolved`: 4096 entries, evict entry with lowest TTL first

Eviction using Individual Requests

- DNS Standard defines an unbounded cache
- Real resolvers: Some kind of practical limit
- Evict by flooding the cache with requests
- Requires knowledge of cache size and eviction policy
- `systemd-resolved`: 4096 entries, evict entry with lowest TTL first
- ⇒ Evict cache using maximum-TTL entries

Cache Hole-Punching

Cache Hole-Punching

- Cache is full with long-TTL entries

Cache Hole-Punching

- Cache is full with long-TTL entries
- New genuine entries have lower TTLs

Cache Hole-Punching

- Cache is full with long-TTL entries
- New genuine entries have lower TTLs ⇒ they evict each other

Cache Hole-Punching

- Cache is full with long-TTL entries
- New genuine entries have lower TTLs ⇒ they evict each other
- We need to create space in the cache for new entries

Cache Hole-Punching

- Cache is full with long-TTL entries
- New genuine entries have lower TTLs ⇒ they evict each other
- We need to create space in the cache for new entries
- Craft a DNS reply with many short-TTL RRs

Cache Hole-Punching

- Cache is full with long-TTL entries
- New genuine entries have lower TTLs ⇒ they evict each other
- We need to create space in the cache for new entries
- Craft a DNS reply with many short-TTL RRs
- `systemd-resolved`: Evicts enough entries to fit all RR’s in the cache

Cache Hole-Punching

- Cache is full with long-TTL entries
- New genuine entries have lower TTLs ⇒ they evict each other
- We need to create space in the cache for new entries
- Craft a DNS reply with many short-TTL RRs
- `systemd-resolved`: Evicts enough entries to fit all RR’s in the cache
- Short TTLs immediately get evicted

Cache Hole-Punching

- Cache is full with long-TTL entries
- New genuine entries have lower TTLs ⇒ they evict each other
- We need to create space in the cache for new entries
- Craft a DNS reply with many short-TTL RRs
- `systemd-resolved`: Evicts enough entries to fit all RR’s in the cache
- Short TTLs immediately get evicted
- ⇒ We “punch a hole” in the cache

Eviction using Individual Requests

⊕ Most widely applicable primitive

Eviction using Individual Requests

- ⊕ Most widely applicable primitive
- ⊕ Challenging to mitigate

Eviction using Individual Requests

- ⊕ Most widely applicable primitive
- ⊕ Challenging to mitigate
- ⊖ Slow

Eviction using Individual Requests

- ⊕ Most widely applicable primitive
- ⊕ Challenging to mitigate
- ⊖ Slow
- ⊖ Hole-punching is implementation-specific

Eviction using Large Requests

- Idea: “Hole-punch” the entire cache

Eviction using Large Requests

- Idea: “Hole-punch” the entire cache
- Evict the cache using one large DNS response

Eviction using Large Requests

- Idea: “Hole-punch” the entire cache
- Evict the cache using one large DNS response
- Problem 1: DNS response’s 2 byte **length** header field

Eviction using Large Requests

- Idea: “Hole-punch” the entire cache
- Evict the cache using one large DNS response
- Problem 1: DNS response’s 2 byte **length** header field
  We can only fit 4091 RR’s in one response

Eviction using Large Requests

- Idea: “Hole-punch” the entire cache
- Evict the cache using one large DNS response
- Problem 1: DNS response’s 2 byte **length** header field
  We can only fit 4091 RR’s in one response
- Solution: Evict some entries **before**, using long-TTL single requests

Eviction using Large Requests

- Idea: “Hole-punch” the entire cache
- Evict the cache using one large DNS response
- Problem 1: DNS response’s 2 byte **length** header field

  We can only fit 4091 RR’s in one response
- Solution: Evict some entries **before**, using long-TTL single requests
- Problem 2: Public DNS server limits

Eviction using Large Requests

⊕ Few Requests necessary

Eviction using Large Requests

- ⊕ Few Requests necessary
- ⊕ Faster than individual requests

Eviction using Large Requests

- ⊕ Few Requests necessary
- ⊕ Faster than individual requests
- ⊖ Depends on configured DNS server

Eviction using Large Requests

- ⊕ Few Requests necessary
- ⊕ Faster than individual requests
- ⊖ Depends on configured DNS server
- ⊖ Not possible using most public DNS servers

Error-based Eviction

Error-based Eviction

- Receiving invalid DNS responses causes `SERVFAIL` or timeout

Error-based Eviction

- Receiving invalid DNS responses causes `SERVFAIL` or timeout
- `systemd-resolved` retries 3 times

Error-based Eviction

- Receiving invalid DNS responses causes `SERVFAIL` or timeout
- `systemd-resolved` retries 3 times
- Afterwards: Switch to fallback DNS server

Error-based Eviction

- Receiving invalid DNS responses causes `SERVFAIL` or timeout
- `systemd-resolved` retries 3 times
- Afterwards: Switch to fallback DNS server Flushes DNS cache!

Error-based Eviction

- Receiving invalid DNS responses causes `SERVFAIL` or timeout
- `systemd-resolved` retries 3 times
- Afterwards: Switch to fallback DNS server Flushes DNS cache!
- Send valid response after 3 retries to stop loop

Error-based Eviction

- Receiving invalid DNS responses causes `SERVFAIL` or timeout
- `systemd-resolved` retries 3 times
- Afterwards: Switch to fallback DNS server Flushes DNS cache!
- Send valid response after 3 retries to stop loop
- ⊕ Only 1 request necessary

Error-based Eviction

- Receiving invalid DNS responses causes `SERVFAIL` or timeout
- `systemd-resolved` retries 3 times
- Afterwards: Switch to fallback DNS server Flushes DNS cache!
- Send valid response after 3 retries to stop loop
- ⊕ Only 1 request necessary
- ⊕ Very fast

Error-based Eviction

- Receiving invalid DNS responses causes `SERVFAIL` or timeout
- `systemd-resolved` retries 3 times
- Afterwards: Switch to fallback DNS server Flushes DNS cache!
- Send valid response after 3 retries to stop loop

⊕ Only 1 request necessary

⊕ Very fast

⊖ Specific to `systemd-resolved`

Error-based Eviction

- Receiving invalid DNS responses causes `SERVFAIL` or timeout
- `systemd-resolved` retries 3 times
- Afterwards: Switch to fallback DNS server   Flushes DNS cache!
- Send valid response after 3 retries to stop loop
- ⊕ Only 1 request necessary
- ⊕ Very fast
- ⊖ Specific to `systemd-resolved`
- ⊖ Fixed in `systemd` 256+ for most DNS servers (EDE support)

Eviction Results

Primitive	Availability			Eviction
	Native	JS	HTML	Time
Direct Flushing	✓	✗	✗	10.987 ms
Many Requests	✓	✓	✓	5.109 s
Large Response	✓	✓	✓	1.387 s
Error-Based	✓	✓	✓	79.1 ms

Browser Cache Eviction

Browser Cache Eviction

- Browsers also have DNS caches

Browser Cache Eviction

- Browsers also have DNS caches
- Protected by Network State Partitioning

Browser Cache Eviction

- Browsers also have DNS caches
- Protected by Network State Partitioning
- Similar eviction to `resolved`

Browser Cache Eviction

- Browsers also have DNS caches
- Protected by Network State Partitioning
- Similar eviction to `resolved`
- Bypass using individual requests

End-to-End Attacks

Native Cross-VM

- Attacker: Python script inside VM

Native Cross-VM

- Attacker: Python script inside VM
- Victim: Host machine

Native Cross-VM

- Attacker: Python script inside VM
- Victim: Host machine
- Victim accesses random set out of 100 domains

Native Cross-VM

- Attacker: Python script inside VM
- Victim: Host machine
- Victim accesses random set out of 100 domains
- Attacker monitors, and evicts using error-based eviction

Native Cross-VM

- Attacker: Python script inside VM
- Victim: Host machine
- Victim accesses random set out of 100 domains
- Attacker monitors, and evicts using error-based eviction

True Positives 22 999	False Negatives 3 502
False Positives 240	True Negatives 314 498

(F_1 Score 92.48%)

JavaScript End-to-End

Setup:

JavaScript End-to-End

Setup:

- Attacker running in Browser

JavaScript End-to-End

Setup:

- Attacker running in Browser
- Victim accesses random set out of 10 Domains

JavaScript End-to-End

Setup:

- Attacker running in Browser
- Victim accesses random set out of 10 Domains
- Attacker monitors from JavaScript, evicts using error-based eviction

JavaScript End-to-End

Setup:

- Attacker running in Browser
- Victim accesses random set out of 10 Domains
- Attacker monitors from JavaScript, evicts using error-based eviction
- Also bypasses browser cache

Domain	DNSSEC	
	✗	✓
`amazon.com`	81.63 %	91.67 %
`pornhub.com`	85.71 %	80.77 %
`reddit.com`	86.49 %	97.78 %
`wikipedia.com`	95.24 %	91.67 %
Macro-average	78.89 %	82.86 %

Discussion

What about DNS-over-HTTPS?

Discussion

What about DNS-over-HTTPS?

- Prevents this attack in theory

Discussion

What about DNS-over-HTTPS?

- Prevents this attack in theory
- **Chrome:** Uses `/etc/resolv.conf` ⇒ `systemd-resolved`

Discussion

What about DNS-over-HTTPS?

- Prevents this attack in theory
- **Chrome:** Uses `/etc/resolv.conf` ⇒ `systemd-resolved`
- **Firefox:** DoH disabled in all but 4 countries worldwide

Discussion

What about DNS-over-HTTPS?

- Prevents this attack in theory
- **Chrome:** Uses `/etc/resolv.conf` ⇒ `systemd-resolved`
- **Firefox:** DoH disabled in all but 4 countries worldwide

Discussion

What about DNS-over-HTTPS?

- Prevents this attack in theory
- **Chrome:** Uses `/etc/resolv.conf` ⇒ `systemd-resolved`
- **Firefox:** DoH disabled in all but 4 countries worldwide

What about other operating systems?

Discussion

What about DNS-over-HTTPS?

- Prevents this attack in theory
- **Chrome:** Uses `/etc/resolv.conf` ⇒ `systemd-resolved`
- **Firefox:** DoH disabled in all but 4 countries worldwide

What about other operating systems?

- **macOS:** Possible by evicting using individual requests

Discussion

What about DNS-over-HTTPS?

- Prevents this attack in theory
- **Chrome:** Uses `/etc/resolv.conf` ⇒ `systemd-resolved`
- **Firefox:** DoH disabled in all but 4 countries worldwide

What about other operating systems?

- **macOS:** Possible by evicting using individual requests
- **Windows:** Theoretically possible, no working eviction primitive yet

Responsible Disclosure

Responsible Disclosure

- **systemd** considers this local-only, not a security issue

Responsible Disclosure

- **systemd** considers this local-only, not a security issue
- **Chromium** sees no practical way to fix this without significant drawbacks

Responsible Disclosure

- **systemd** considers this local-only, not a security issue
- **Chromium** sees no practical way to fix this without significant drawbacks
- **Firefox** also sees the responsibility mostly with the resolver

Responsible Disclosure

- **systemd** considers this local-only, not a security issue
- **Chromium** sees no practical way to fix this without significant drawbacks
- **Firefox** also sees the responsibility mostly with the resolver
- **Apple** is planning to address the issue in a future update

Conclusion

- We demonstrated an Evict+Reload attack on local DNS caches
- Our attack works native, from JavaScript or scriptless, even over VPN
- No fixes have been deployed yet, the attack is still possible today

hannes.weissteiner@tugraz.at
Social: hweissi@infosec.exchange
https://hannesweissteiner.com

Acknowledgments

This research was made possible by generous funding from:

Funded by the European Union

fssec erc

European Research Council
Established by the European Commission

SPyCoDe FWF
Der Wissenschaftsfonds.

Google

Supported in part by the European Research Council (ERC project FSSec 101076409), and the Austrian Science Fund (FWF project NeRAM 10.55776/I6054 and FWF SFB project SPyCoDe 10.55776/F85). Additional funding was provided by generous gifts from Google. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding parties.

Continuous User Behavior Monitoring using DNS Cache Timing Attacks

Hannes Weissteiner[1] Roland Czerny[1] Simone Franza[1] Stefan Gast[1]
Johanna Ullrich[2] Daniel Gruss[1]

[1]Graz University of Technology | [2]University of Vienna

NDSS 2026

› isec.tugraz.at

Bibliography

[1] Edward W Felten and Michael A Schneider. **Timing attacks on web privacy.** CCS. 2000.

[2] Gilad Moav et al. **DNS FLaRE: A Flush-Reload Attack on DNS Forwarders.** USENIX Security. 2025.