ACSAC 2023

DefWeb: Defending User Privacy against Cache-based Website Fingerprinting Attacks with Intelligent Noise Injection

December 8^{th} ,2023

Seonghun Son, Debopriya Roy Dipta and Berk Gulmezoglu

Microarchitecture and Artificial Intelligence Security (MAIS) Laboratory

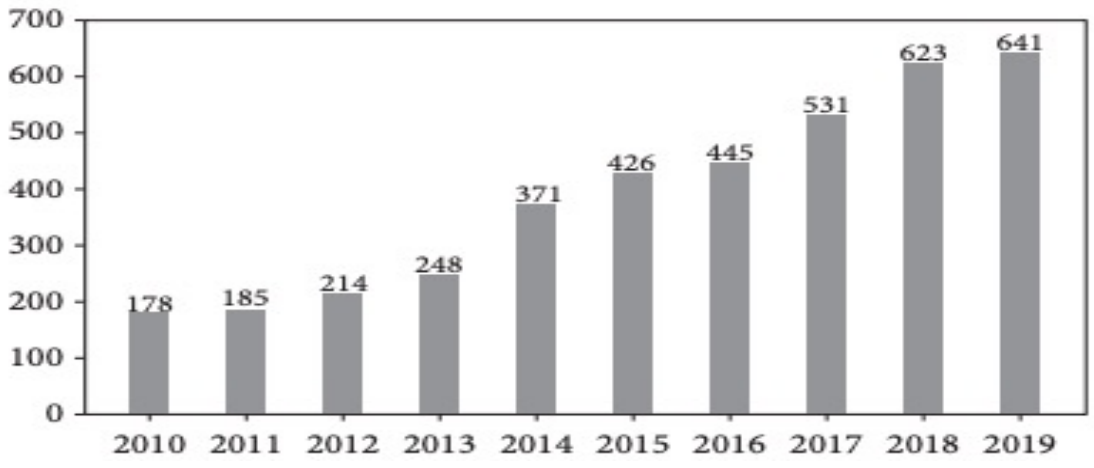
Department of Electrical and Computer Engineering,

Iowa State University, Ames, Iowa

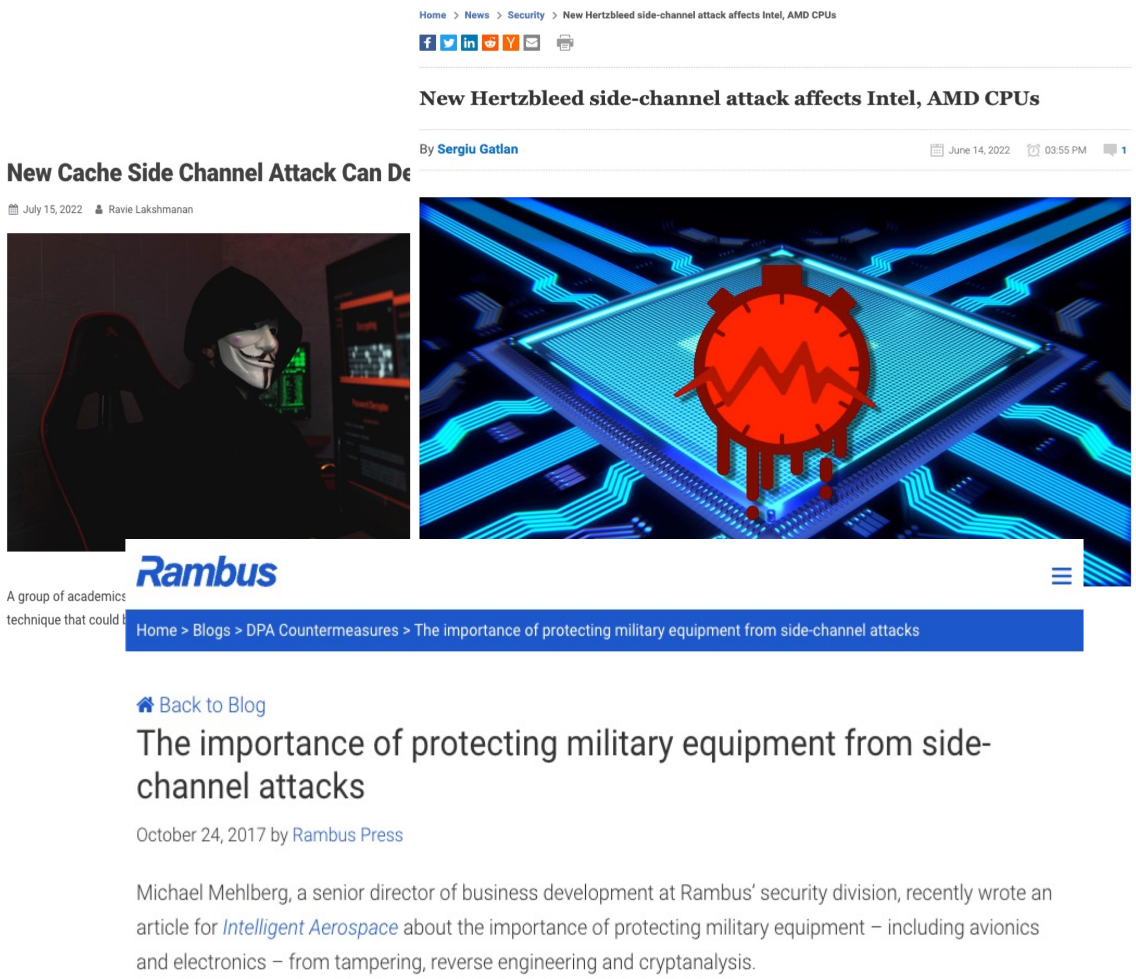
Why Side-channel attack?

- Leak sensitive information not reading key information directly (unintentional information leakage)
  - Electromagnetic
  - Acoustic
  - Micro-architectural information
    1. Cache timing
    2. Power monitoring
    3. Port contention
    4. …

- Side-channel attacks have become a serious threat nowadays.



Number of research papers on side−channel attacks[1]



1. Survey of CPU Cache-Based Side-Channel Attacks: Systematic Analysis, Security Models, and Countermeasures

Motivation

- 2015: Yossef Oren, Vasileios P. Kemerlis, Simha Sethumadhavan, Angelos D. Keromytis, "The spy in the sandbox: Practical cache attacks in javascript and their implications."

- 2019 : Anatoly Shusterman et al. "Cache Occupancy: Robust website fingerprinting through the cache occupancy channel."
    - **Cache masking technique** (cache-sweep noise) : 78.4% to 76.2%*

- 2022 : Jack Cook, Jules Drean, Jonathan Behrens, and Mengjia Yan. "There's always a bigger fish: a clarifying analysis of a machine-learning-assisted side-channel attack".
    - Introducing Loop-counting Attack
    - Randomized timer
    - **Spurious interrupt noise**: 95.7% to 62.0%*

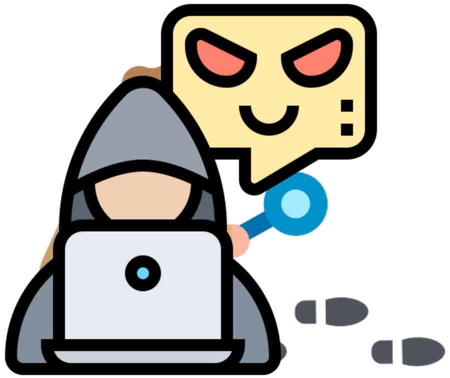
Our Objective

Obfuscate the Attacker's Deep Learning model and
effectively mitigate the Website Fingerprint attacks with **less performance overhead**

*Deep Learning Accuracy derived from Cook et al. paper

What is a Website Fingerprint Attack?

Web User / Victim

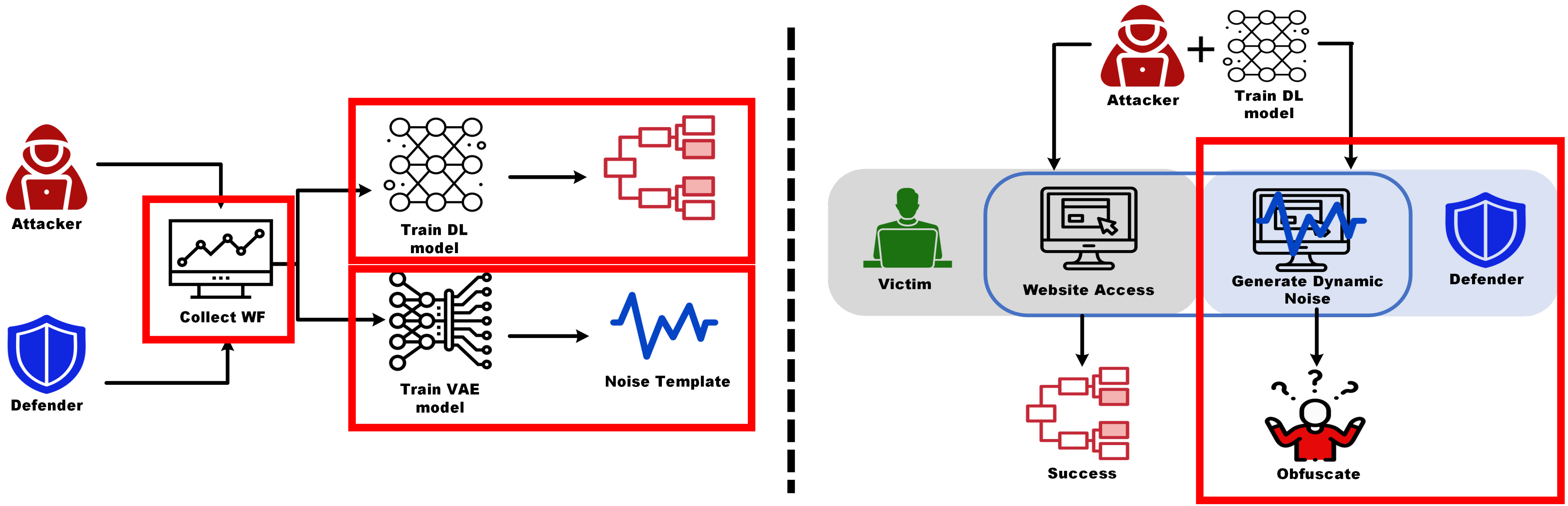


Attacker / Detective

Website Fingerprints / Fingerprints/Footprints

Website Fingerprint Attack / Criminal

DefWeb scenario

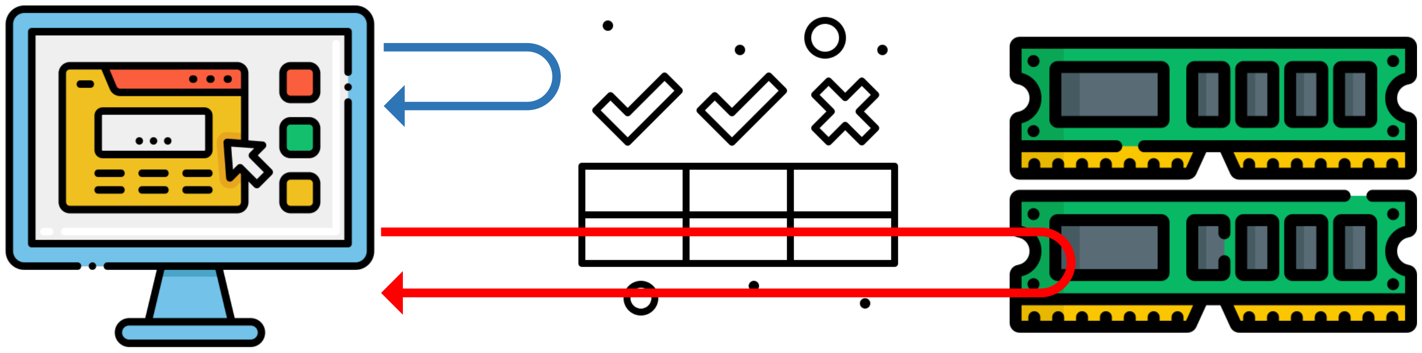


Offline Phase

Online Phase

Data Collection : Prime and Probe attack

- Anatoly Shusterman, Lachlan Kang, Yarden Haskal, Yosef Meltser, Prateek Mittal, Yossi Oren, and Yuval Yarom. 2019. Robust website fingerprinting through the cache occupancy channel. [32]
  - Implemented as JavaScript code to apply **Cache occupancy channel** in a web environment
  - Instead of measuring the timing of cache sets individually, they measure the whole cache.
  - Overcome the limitation of timer resolution in the web environment.



Website access   Cache Occupancy Channel (JS)   Memory

Algorithm 1: Website Data Collection Algorithm

```
// I Interval time between each attack
// url The website address
Input: s, w, url
Output: S_i
for i ← 1 to w do
    Run url in the browser ;
    for j ← 1 to s do
        S_i ← Prime and Probe ;
        sleep I ;
    Close the browser ;
    sleep 10s ;
```

WF Data collection

Data Collection : Prime and Probe attack

- **Hardware**
  - **Intel Tiger Lake:**
    - CPU Model: Intel(R) Core (TM) i7-1165G7 @ 2.80GHz
    - 12MB Last Level Cache
    - Ubuntu 20.04 LTS OS
  - **GPU:**
    - NVIDIA GeForce RTX 3090 GPU card

- **Software**
  - **Google Chrome:**
    - Version: 101.0.4951.64
  - **Mozilla Firefox:**
    - Version: 111.0
  - **Tor :**
    - Version: 10.5.10

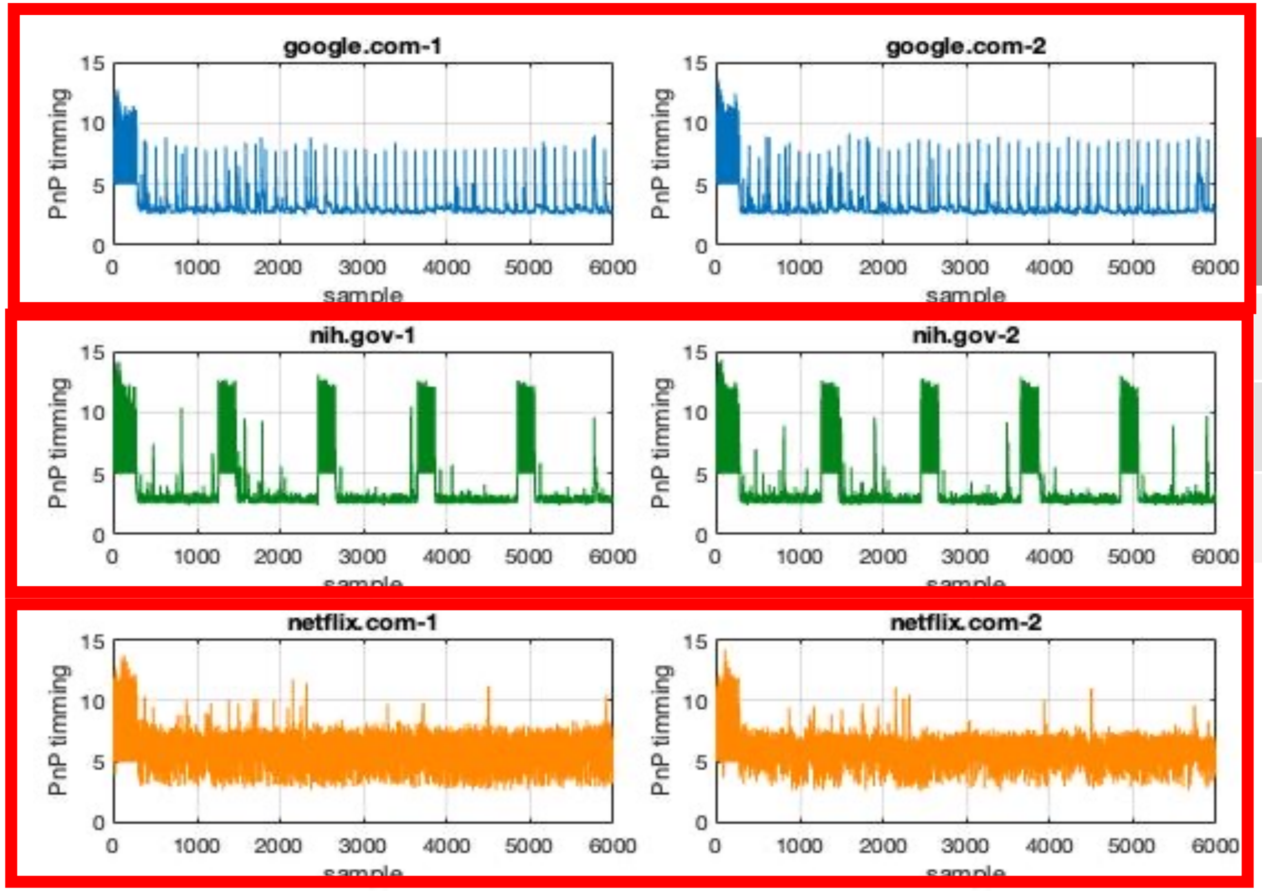
1. 360.cn	21. dailymotion.com	41. imdb.com	61. oracle.com	81. tistory.com
2. 9gag.com	22. digikala.com	42. imgur.com	62. paypal.com	82. tmall.com
3. abs-cbn.com	23. discordapp.com	43. indeed.com	63. pinterest.com	83. tribunnews.com
4. adobe.com	24. dropbox.com	44. intuit.com	64. popads.net	84. tripadvisor.com
5. airbnb.com	25. ebay.com	45. jd.com	65. qq.com	85. tumblr.com
6. aliexpress.com	26. espn.com	46. kompas.com	66. quora.com	86. twitch.tv
7. allegro	27. espncricinfo.com	47. linkedin.com	67. reddit.com	87. vimeo.com
8. amazon.com	28. etsy.com	48. liputan6.com	68. researchgate.net	88. walmart.com
9. apple.com	29. exoclick.com	49. live.com	69. scribd.com	89. weather.com
10. archive.org	30. flipkart.com	50. mail.ru	70. slideshare.net	90. weibo.com
11. baidu.com	31. force.com	51. mediafire.com	71. sohu.com	91. wellsfargo.com
12. bbc.com	32. foxnews.com	52. medium.com	72. soundcloud.com	92. wikipedia.org
13. bing.com	33. github.com	53. mozilla.org	73. spotify.com	93. yahoo.com
14. booking.com	34. globo.com	54. msn.com	74. stackexchange.com	94. yandex.ru
15. bukalapak.com	35. godaddy.com	55. naver.com	75. stackoverflow.com	95. yelp.com
16. canva.com	36. goodreads.com	56. netflix.com	76. steamcommunity.com	96. youtube.com
17. chase.com	37. google.com	57. nih.gov	77. steampowered.com	97. yy.com
18. craigslist.org	38. healthline.com	58. nordstrom.com	78. taobao.com	98. zhanqi.tv
19. csdn.net	39. hulu.com	59. office.com	79. theguardian.com	99. zillow.com
20. dailymail.co.uk	40. ikea.com	60. okezone.com	80. thesaurus.com	100. zoom.us

Alexa's top 100 website List

*Additional 50 websites are included in the open-world scenario

Data Collection : Prime and Probe attack

- Website Fingerprints
  - 100 measurements from 100 websites



Collected Website Fingerprint (Google Chrome)

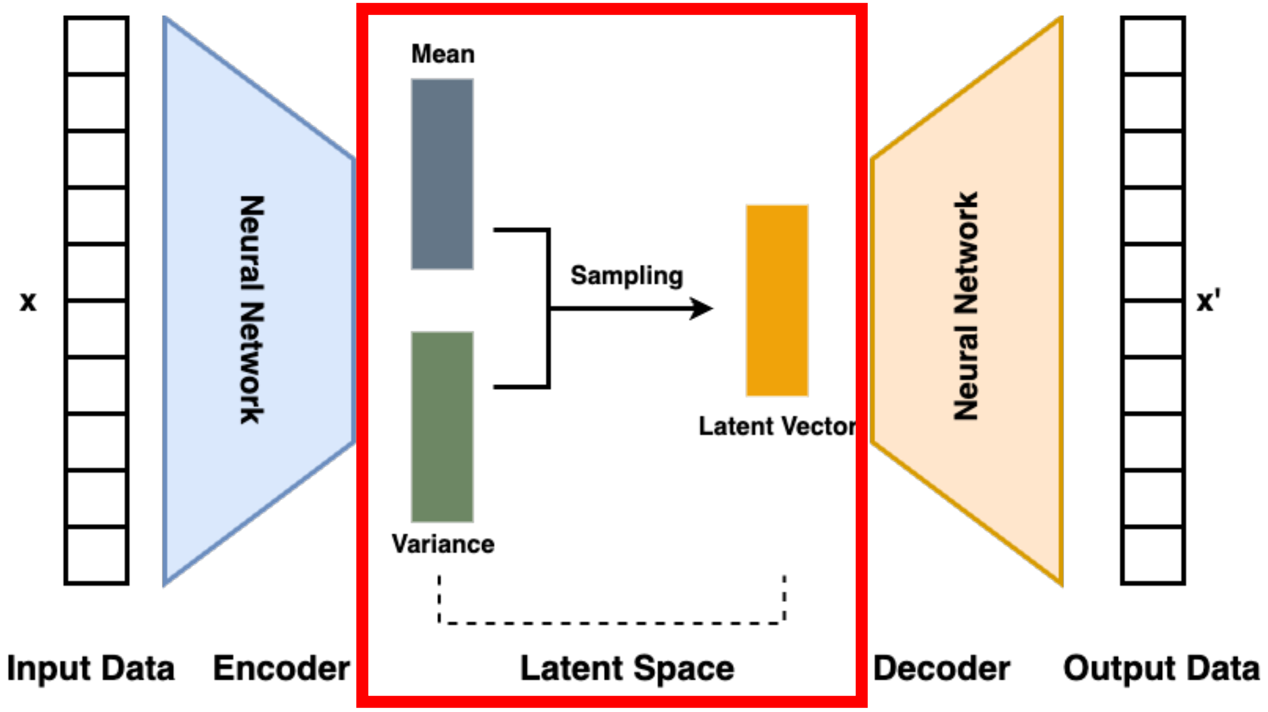
- Website Fingerprint Attack Accuracy
  - Achieve similar accuracy with previous research [4], [32]

Browser	Loop-Counting Attack [4]	Sweep-Counting Attack [32]	Our Setup	
			CNN	LSTM
Chrome	96.6% $\pm$ 0.8%	91.4% $\pm$ 1.2%	95.7% $\pm$ 0.2%	95.8% $\pm$ 0.5%
Firefox	95.3% $\pm$ 0.7%	80.0% $\pm$ 0.6%	95.7% $\pm$ 0.1%	95.5% $\pm$ 0.3%
Tor	49.8% $\pm$ 4.2%	46.7% $\pm$ 4.1%	46.2% $\pm$ 1.4%	40.9% $\pm$0.4%

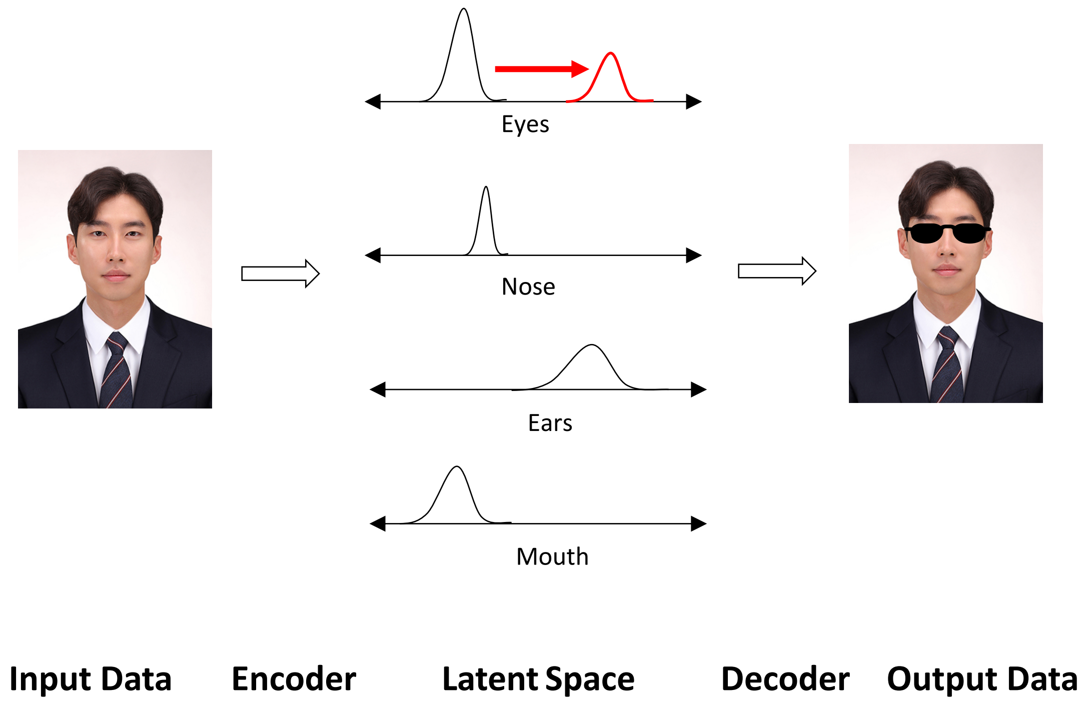
Website Fingerprint Attack Accuracy

Simulation Noise Template Generation: Variational Autoencoder

- VAE compresses meaningful features in the latent space
- Mean and variances in the latent space create normal distribution in the latent space



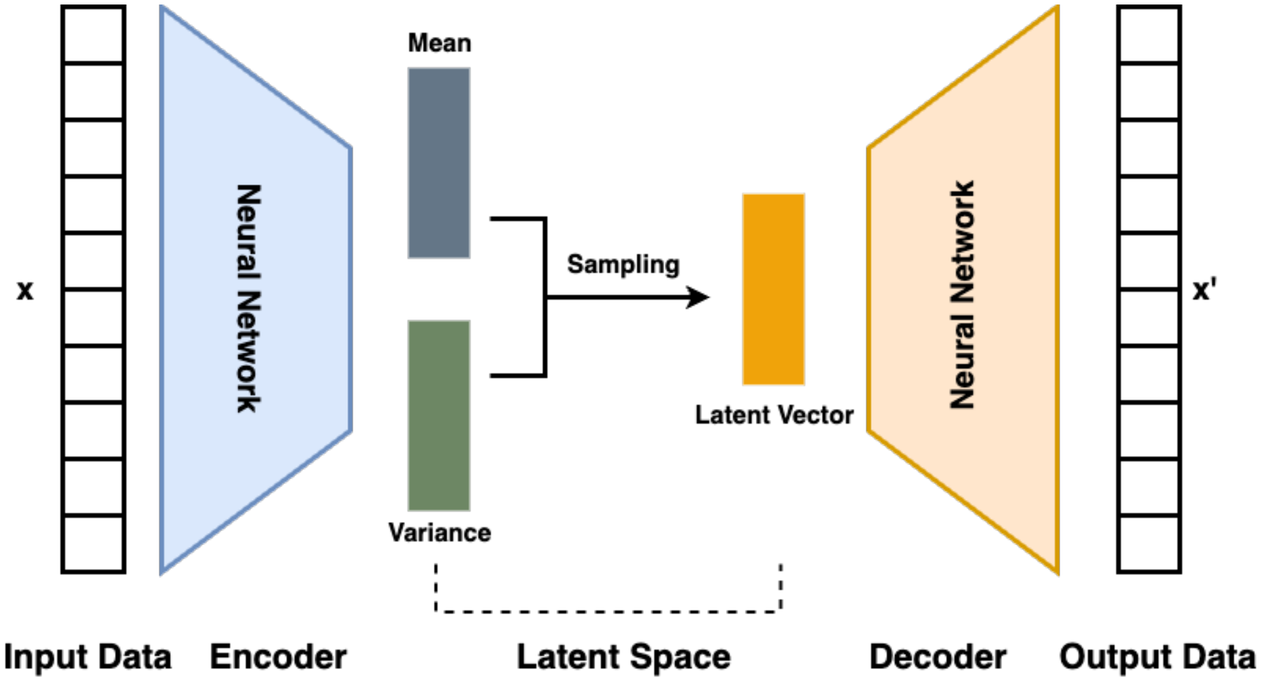
Structure of Variational Autoencoder (VAE)



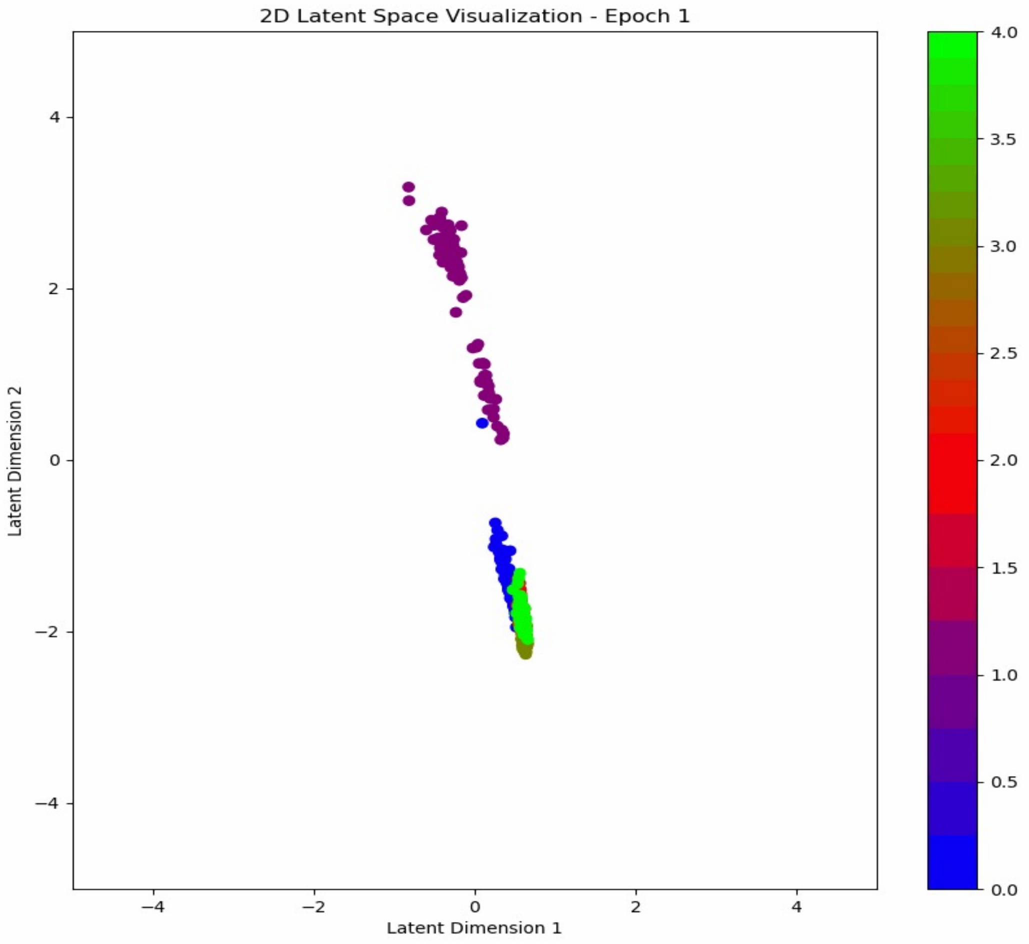
VAE Example

Simulation Noise Template Generation: Variational Autoencoder

- Each time VAE training, VAE separated the clusters distinctively



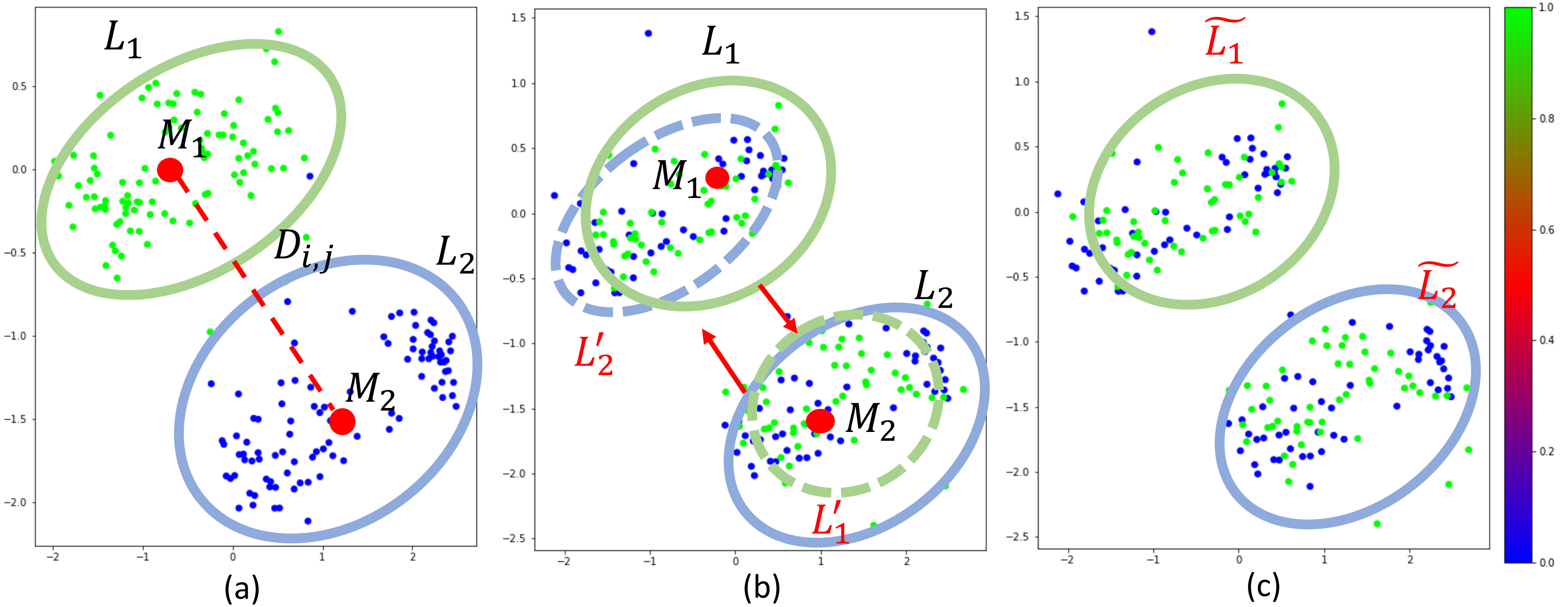
Structure of Variational Autoencoder (VAE)



Latent Space Example (W=5, D=2)

Simulation Noise Template Generation: Variational Autoencoder

- Noisy WF data creation
  - Calculate the distance (moving) between each mean of the cluster's
  - Generate one website to the others



$$L'_i = L_i + D_{i,j}$$

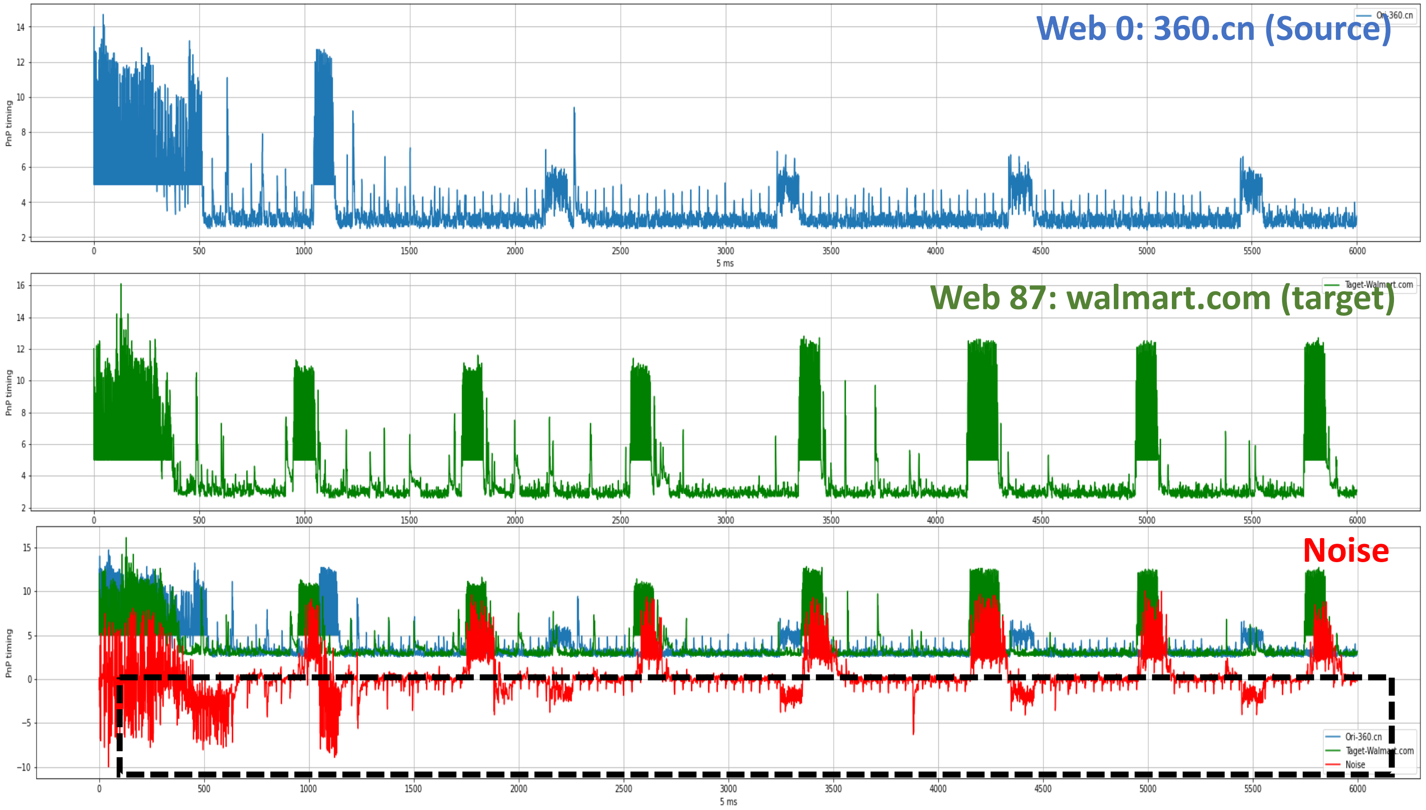
$$S' = V_D(L'_i)$$

$$\text{where } i, j = \{0, \cdots, w-1\}$$

Clusters in the latent Space (W=google.com, amazon.com, D=2)

Simulation Noise Template Generation: Variational Autoencoder

- Noise template creation
  - Noise datasets are extracted from the differences between the noisy (VAE generated) and original datasets.



Noise extraction: Web 0 to 87

$$Noise = S' - S$$

- Noise can't be the negative value.
- Zeroing out the negative noise is less impactful.

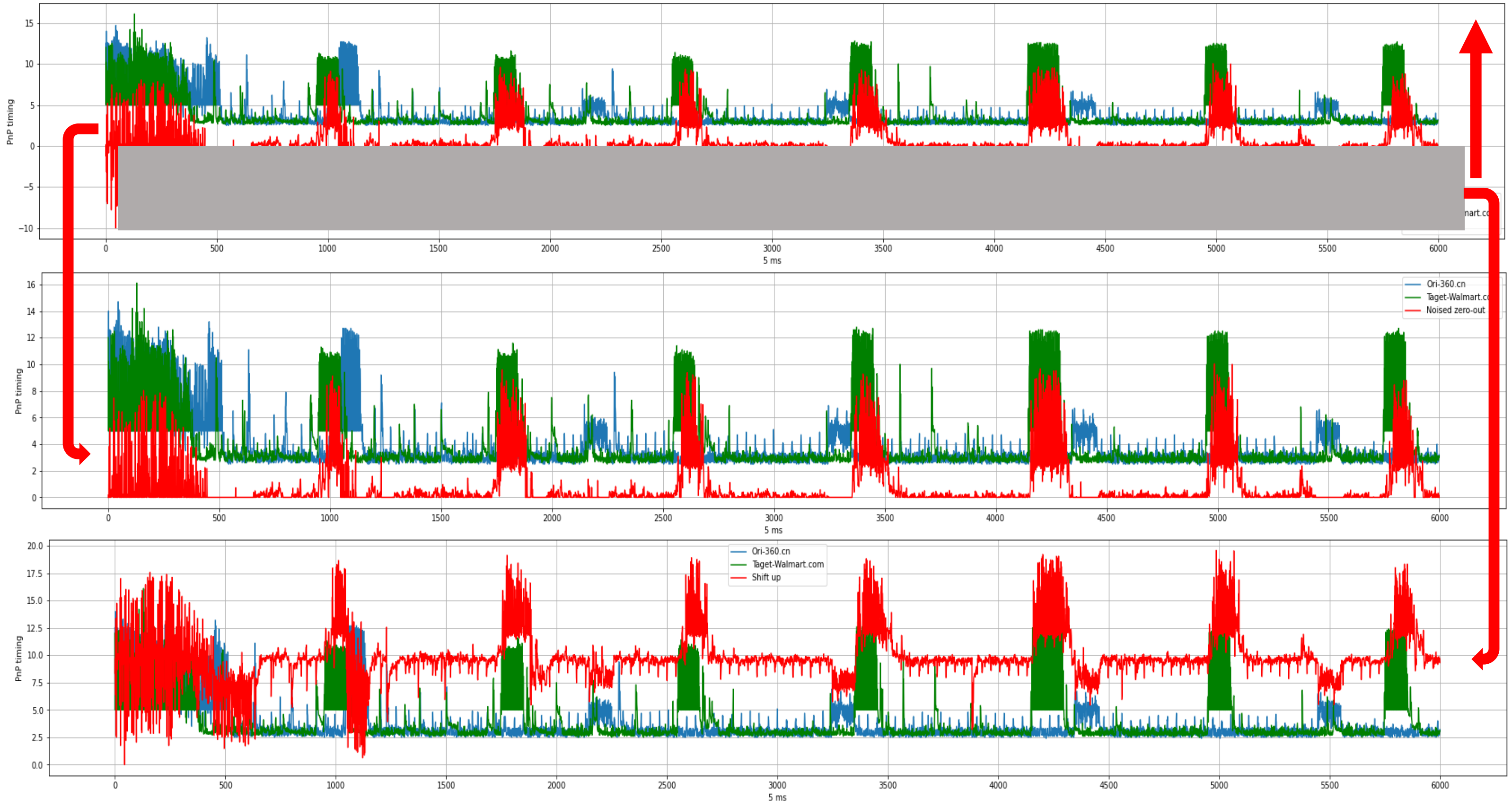
*NNV: Noise added with Negative Values
*NZO: Negative Noise Zeroed out

Browser	CNN		LSTM	
	*NNV	*NZO	*NNV	*NZO
Chrome	3.2% ± 3.0%	83.4% ±1.5%	0.8% ±0.2%	86.7% ±1.3%
Firefox	1.1% ± 0.5%	86.4% ±1.9%	0.7% ±0.1%	93.1% ±2.1%

WF attack accuracy

Simulation Noise Template Generation: Variational Autoencoder

- Noise template creation
  - Shift up *c* amount from the extracted noise



Noise extraction: Web 0 to 87

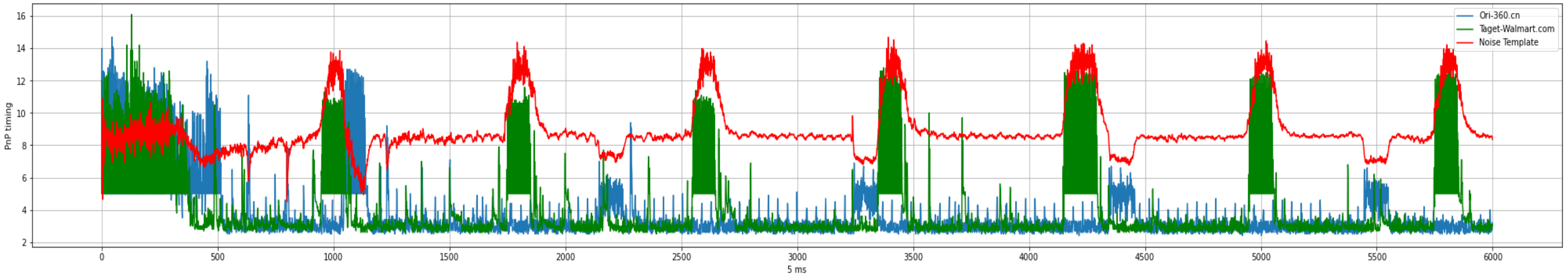
$$Noise' = Noise - \frac{\min(Noise)}{c}$$

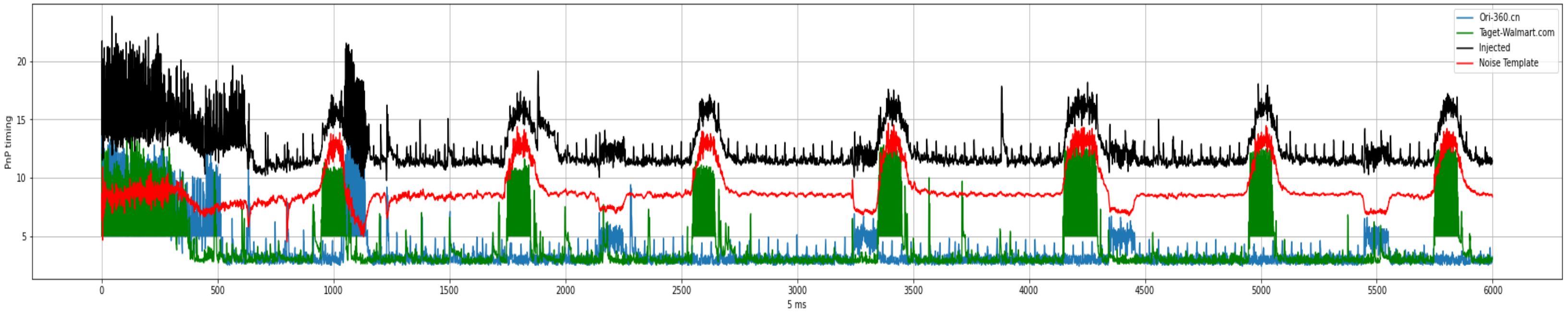
$$Noise' = \begin{cases} Noise' (Noise' > 0) \\ 0 \quad (Noise' \leq 0) \end{cases}$$

- Noise can’t be the negative value.
- Zeroing out the negative noise is less impactful.
- **Shifting up noise and zeroing out negative values keeps significant features.**

Simulation Noise Template Generation: Variational Autoencoder

- Noise template creation
  - Averaged 100 noise templates to create a generic Noise template to convert to specific website fingerprints.





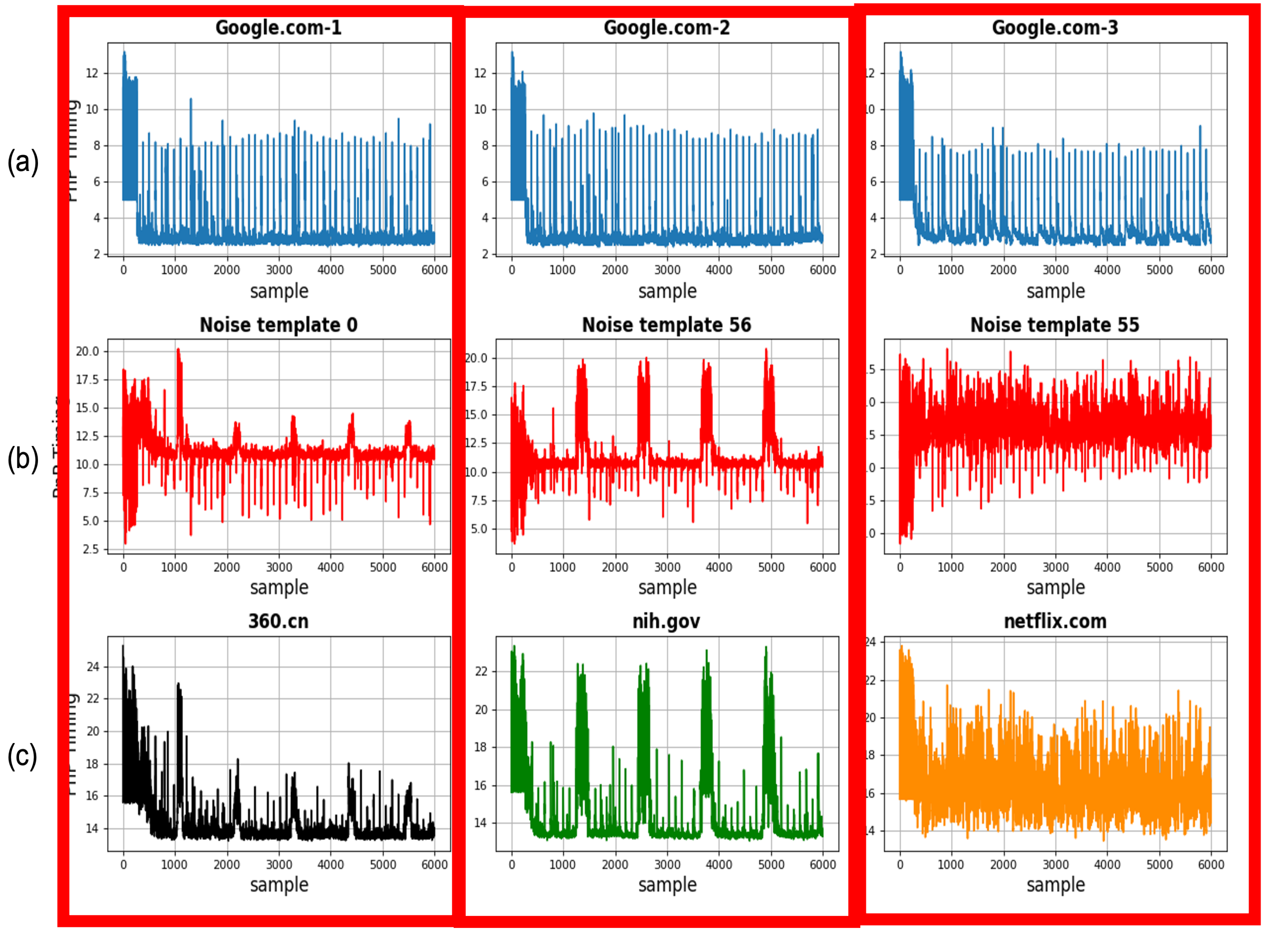
Noise extraction: Web 0 to 87

$$S_{injected} = S + Noise'$$

- Average precise noise dataset to create a generic noise template (ANT)
- Inject noise template to original WF.

Simulation Noise Template Generation: Variational Autoencoder

- Noise Template Injected Website Fingerprint



(a) Original WF, (b) Noise Template, (c) Noise-injected WF

- Dimension selection in the latent space

Test with pre-trained CNN model — Retrain with CNN model

Dim\Type	Trained CNN	With Noise	Regenerated	Noise dataset	Injected
50	53.84%	2.48%	90.55%	6.10%	5.2%
100	93.79%	3.18%	94.35%	6.30%	4.54%
200	97.39%	6.78%	93.69%	24.89%	23.35%
300	97.74%	10.10%	95.64%	36.75%	33.05%

- Shift value *c* selection

Browser	CNN			
	C = 2	C = 3	C = 4	C = 5
Chrome	3.9% ±3.0%	4.8% ± 3.2%	24.6% ± 10.6%	32.7% ± 19.7%
Firefox	1.6% ± 1.0%	1.3% ± 0.8%	5.7% ± 4.1%	16.8% ± 12.7%

Practical Noise Generation: Self-modifying Code

- Intelligent Noise creation with Self-Modifying Code (SMC)
  - Create practical noise in actual microarchitecture

Algorithm 1: Self-Modifying Code (SMC)

```
// buffer_start_pointer Pointer to the address of the
   buffer initiation position
// buffer_end_pointer Pointer to the address of the
   buffer end position
Input: N_repeat, T_sleep
define buffer_start_pointer
define buffer_end_pointer
for i ← 1 to N_repeat do
    Run SMC(buffer_start_pointer, buffer_end_pointer) ;
    usleep T_sleep ;
_asm
{
align 64
payload :
    ret
define buffer_step ← 64
align 64 [global SMC]
rdi ← buffer_start_pointer
rsi ← buffer_end_pointer
SMC :
    mov rax, 100
    mov r8, rdi
    mov r9, rdi
    mov r10, rsi
.loop :
    dec rax
    je .end
    mov rcx, 64
    mov rdi, r9
    lea rsi, [rel payload]
    rep movsb
    call r9
    lea r9, [r9 + buffer_step]
    cmp r9, r10
    jb .next
    mov r9, r8
.next :
    jmp .loop
.end :
    ret
}
```

- SMC modifies the program's executable code page by altering its own instructions while the program is being executed.
- If SMC is executed, the prefetched queue becomes invalidated since the wrong instructions are executed.
- Other types of interrupts can create noise.

Practical Noise Generation: Self-modifying Code

- Intelligent Noise creation with Self-Modifying Code (SMC)
  - Create practical noise in actual microarchitecture

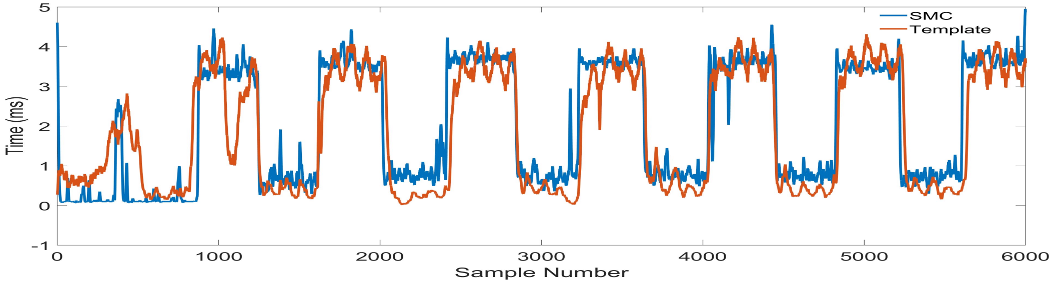
S1) Misalignment

- Mask dominant features of the original WF
- Insert distinctive features to target WF
- Change Point Detection (CPD) algorithm
- Cross-correlation coefficient
- Expansion amount of 50 samples for both sides.

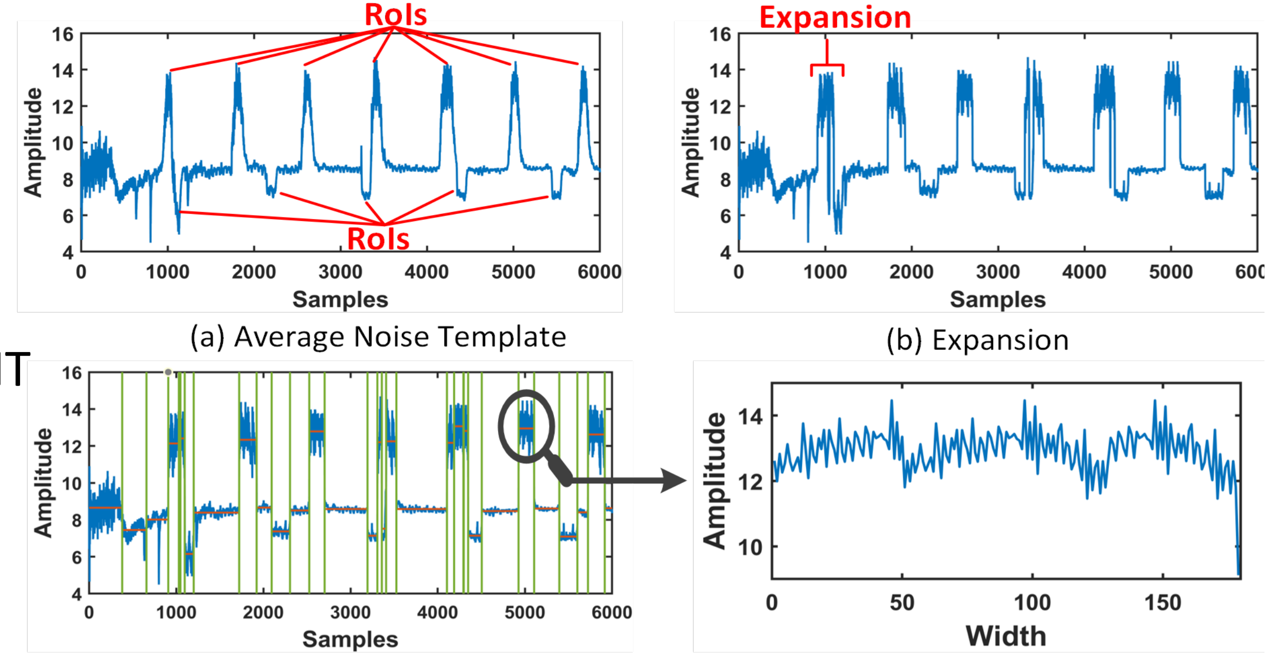
S2) Segmentation into Dynamic Noise Blocks from ANT

S3) Look-up Table Creation

S4) Practical Noise



Practical Noise example



(a) Average Noise Template
(b) Expansion
(c) Segmentation
(d) Parameter extraction

Steps of creating practical noise

DefWeb Results

- Accuracy degradation
  - The classification accuracy for 100 websites drops to **28.8%**, **29.7%**, and **5.2%** accuracy for Chrome, Firefox, and Tor, respectively.
  - The classification accuracy for 150 websites drops to 24%.

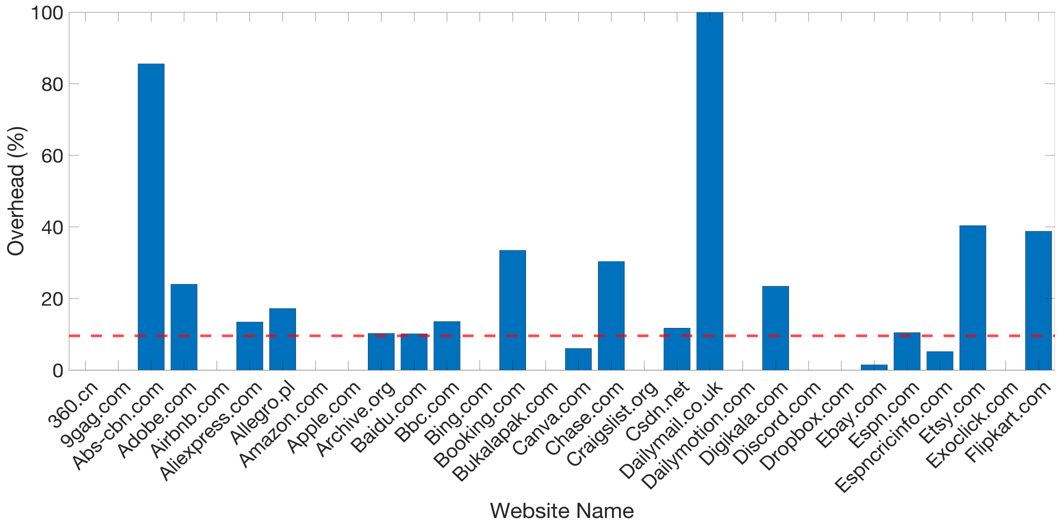
Attack	Cache-Sweep	Interrupt Injection	*DefWeb*	
			Chrome & Firefox	*Tor*
Loop-Counting Attack[4]	x1.03	x1.42	**x3.32**	**x9.2**
Sweep-Counting [32]	x1.03	x1.54	**x3.93**	

WF attack accuracy degradation

- Performance Overhead
  - We created a performance tool using *WebAPI* and *Selenium* library to measure the rendering time.
  - Compare performance overhead with "attacks only time" and "implementing defense time".
  - Better performance tool compared with Benchmarks since we directly check the overhead in a web environment

Defense technique	**Cache Shaping**	**Interrupt Injection**	***DefWeb***
Performance Overhead	51.4 – 71.8%	15.7%	**9.5%**

Performance Overhead



Performance Overhead Results

Future Work

- The effect of the SMC might be different with different microarchitectures so transferability of ***DefWeb*** can be investigated
- SMC creation in the browser environment can be used in future work.
- Source websites with **high activity** are challenging to convert to the target websites with **low activity**.

Conclusion

- ***DefWeb*** demonstrates that intelligent noise injection can decrease the attacker model's accuracy significantly compared to random noise injection methods.
- Our results show that we can achieve **1.6%** (simulation) and **29.7%** (practical) classification accuracy in **Mozilla Firefox** **4.8%** (simulation) and **28.8%** (practical) in **Google Chrome**.
- During the reviewing process, we conducted on the **Tor** browser and achieved **5.2%** accuracy in practical experiments setup.
- The performance overhead introduced by ***DefWeb*** is less than previous defense techniques while degrading the attacker's accuracy considerably.

ACSAC 2023

THANK YOU

- **Seonghun Son**
  - Ph.D. Candidate at Iowa State University
  - seonghun@iastate.edu

- The dataset and the code are made available on GitHub: https://github.com/hunie-son/DefWeb