TOWARDS AN ACCURATE SPECTRE GADGET SCANNER

Qi Ling*, Yi Ren†, Baris Kasikci‡, Shuwen Deng†
*University of Michigan, †Tsinghua University, ‡University of Washington

Motivation

Since the emergence of Spectre attack in 2018, a significant effort has been dedicated to countering this threat with software patches. However, these mitigation strategies typically incur substantial performance drawbacks. The key to minimizing these slowdowns lies in accurately identifying Spectre gadgets—code segments vulnerable to such attacks. **Despite ongoing research, current scanners still struggle to precisely quantify gadget security risks and avoid false positives.**

In this work, we target at developing an accurate Spectre gadget scanner, addressing all recognized Spectre-V1 variants.

Problem Analysis

Current Spectre scanners, despite advancements, often incorrectly flag non-exploitable gadgets as vulnerable, as shown in Figure 1.

```
x = user_input();
if (x < 16) {
  y = array1[x];
  z = array2[512 * y];
}
```

a) The conditional branch resolves faster, only comparing a constant, while the disclosure gadget requires slower arithmetic operations and memory loads.

```
x = user_input();
if (x < obj.size) {
  y = obj.array1[x];
  z = array2[512 * y];
}
```

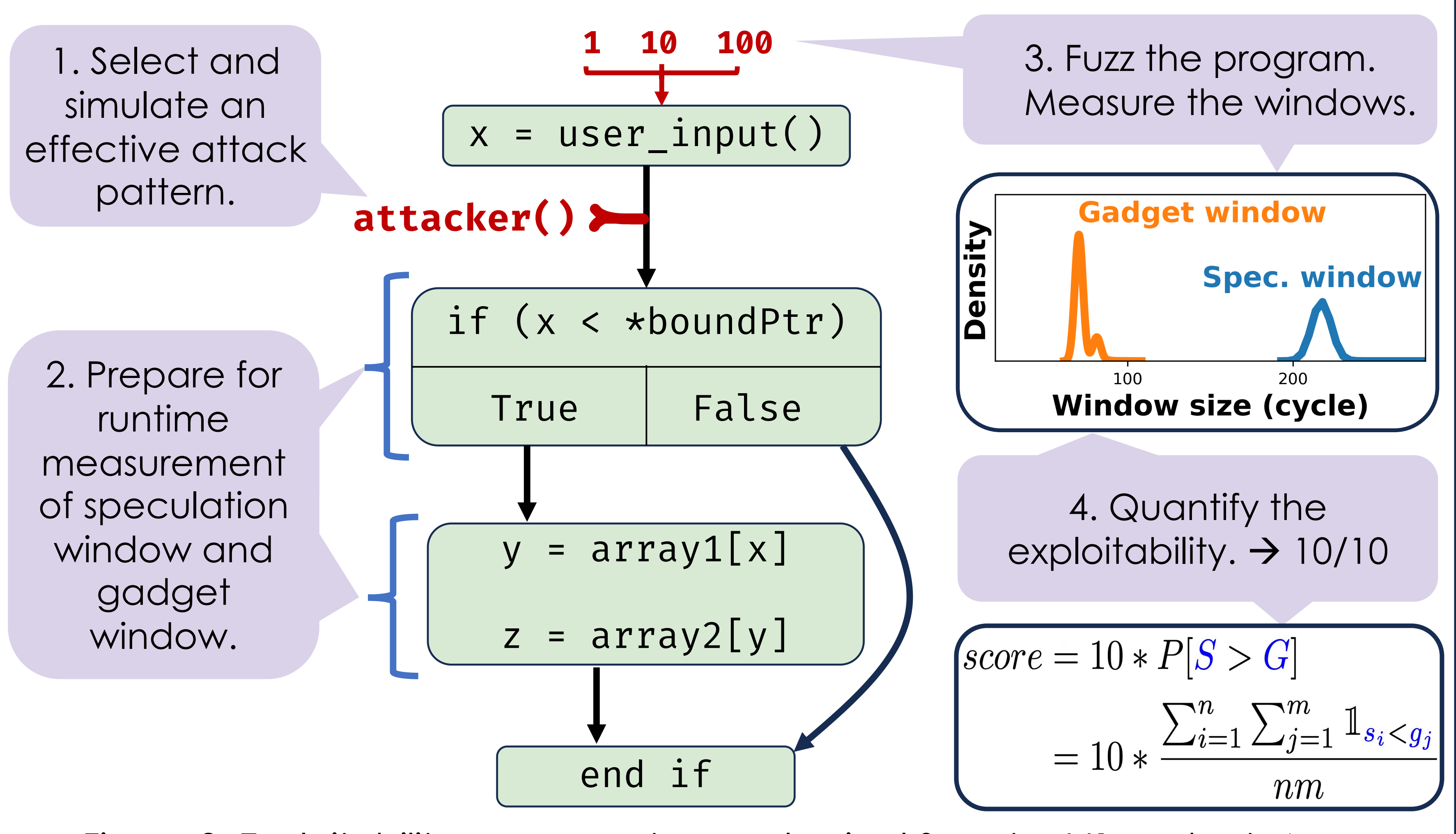
b) The conditional branch resolves faster, even if the bound value is delayed by techniques like cache eviction, as the bound value shares a cache line with the array address.

Figure 1. Gadgets incorrectly deemed vulnerable by prior scanners, though not exploitable by Spectre-V1 attacks. This results from overestimating the speculation window as the full Reorder Buffer size.

We observe that the limited exploitability of these gadgets is due to the attacker's inability to fit the disclosure gadget within the speculation window, a condition we term the 'windowing primitive'.
Our key insights include:

- **Windowing primitive constrains exploitability of gadgets.**
- **Windowing primitive depends on the runtime behavior of gadgets.**

Our Approach: Exploitability Assessment


1. Select and simulate an effective attack pattern.
1 10 100
x = user_input()
attacker()
if (x < *boundPtr)
True
False
y = array1[x]
z = array2[y]
end if
2. Prepare for runtime measurement of speculation window and gadget window.
3. Fuzz the program. Measure the windows.
Density
Gadget window
Spec. window
100
200
Window size (cycle)
4. Quantify the exploitability. → 10/10

$$score = 10 * P[S > G] = 10 * \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} \mathbb{1}_{s_i < g_j}}{nm}$$

Figure 2. Exploitability assessment on a classical Spectre-V1 gadget. A higher score signifies greater exploitability of the gadgets.

We propose exploitability assessment, which quantifies the windowing primitive at runtime, under a simulated attacker:

- In simulating an attacker, we emphasize two key characteristics of attack patterns: enhancing the coexistence of the speculation and gadget windows while preserving their isolation. We focus on an attacker capable of widening the speculation window via cache eviction, proven most effective for Spectre-V1. While this has been the sole effective strategy to date, our methodology can accommodate stronger capabilities.
- To quantify the windowing primitive, we assess the likelihood of fitting the gadget window within the speculation window during runtime. In particular, we approximate both windows through the execution times of selected instructions. Then, we compare them with a probabilistic equation, as shown in Figure 2.

Evaluation Results

Validation of Approach

```
if (*idx < *boundPtr) {
  y = array1[*idx];
  z = array2[y * 512];
}
```

Attack Pattern	Succ. Rate
flush idx	0.0%
➜ **flush boundPtr**	**99.9%**
flush boundPtr + idx	86.8%

```
if (cond) bound=*boundPtr;
else bound=16;
if (x < bound) {
  y = array1[x];
  z = array2[512 * y]; }
```

Attack Pattern	Succ. Rate
➜ **trigger if-branch**	**99.9%**
trigger else-branch	0.0%

Figure 3. Validating the effectiveness of simulated attack patterns. a) Cache eviction of selected address yields highest attack success rate. b) Triggering selected control flow yields highest attack success rate.

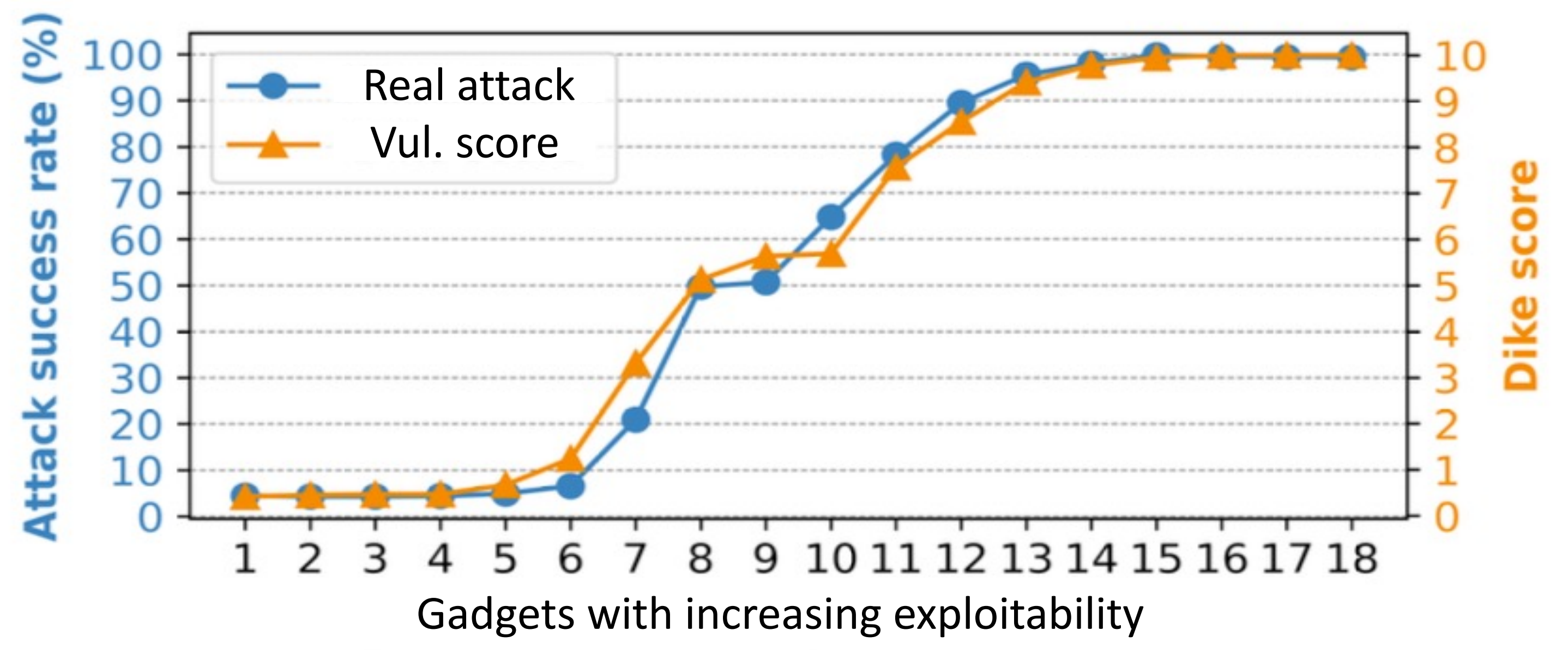


Figure 4. Validating the accuracy of vulnerability scores. We conducted real attacks on a range of gadgets, confirming that the vulnerability scores aligned closely with the actual attack success rates.

Comparison with SOTA Scanners

	Kocher's Dataset	Brotli	HTTP	JSMN	libHTP	libYAML	OpenSSL	Linux kernel
#gadgets reported by SOTA scanners	14	724	5	3	207	180	1415	1498
#FPs identified by our approach	1	80	0	3	49	16	755	17
Rate	7.1%	11.0%	0%	100%	23.7%	8.8%	53.0%	1.1%

Figure 5. Enhanced accuracy in detection. Our approach achieves an average reduction of **22.4%** in false positives compared to SpecFuzz for userspace applications and Kasper for the Linux kernel.

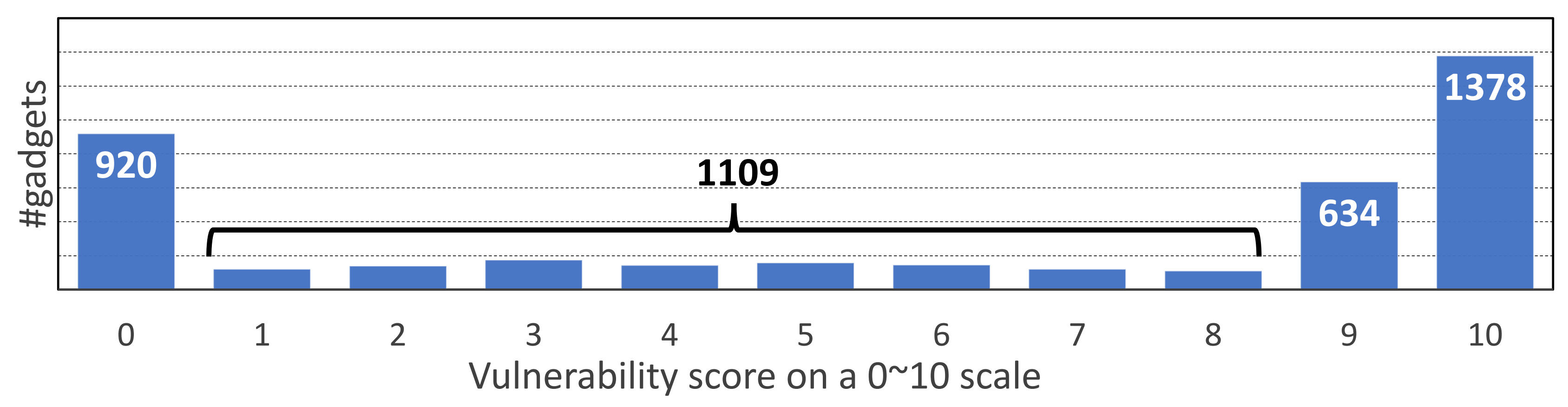
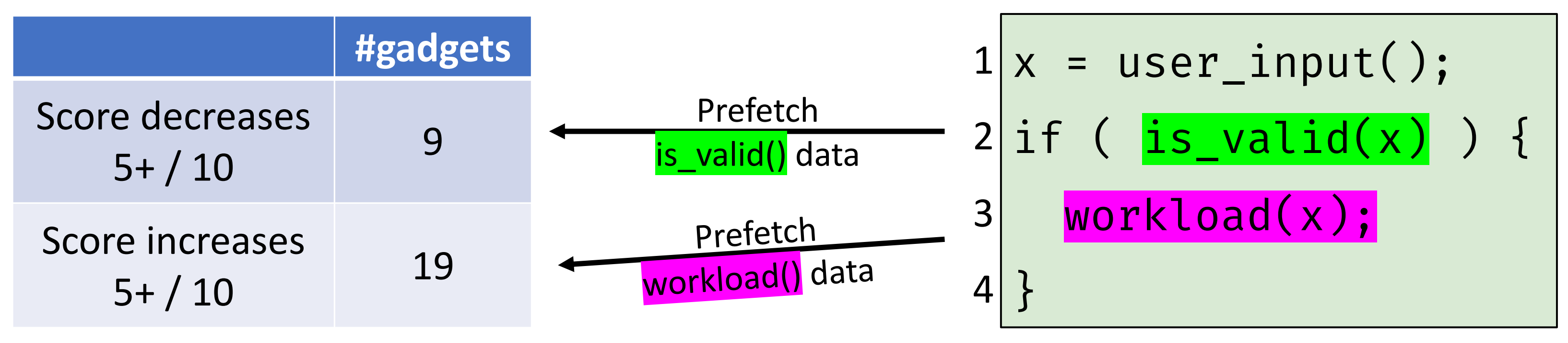


Figure 6. Quantitative evaluation results. Our approach reveals 1109 gadgets with varying exploitability levels between highly exploitable (10/10) and hardly exploitable (0/10). This enables customized defenses with minimal performance overhead.



	#gadgets
Score decreases 5+ / 10	9
Score increases 5+ / 10	19

```
x = user_input();
if ( is_valid(x) ) {
  workload(x);
}
```

Figure 7. Security impact of IP-based prefetcher. Our approach reveals that IP-based prefetcher significantly reduces the exploitability of 9 gadgets, while simultaneously increasing the exploitability of 19 others.

Conclusion

In this work, we propose exploitability assessment, an approach to model the windowing primitive of Spectre gadgets. **Our implemented approach:**

- **Reduces false positives in SOTA works by 22.4% on average,**
- **Quantifies the exploitability of gadgets with fine-grained precision,**
- **Examines the security impact of prefetching techniques.**

These findings are corroborated by comprehensive case studies, validating the effectiveness of our approach.