LLMs can do it better: Patching Code for Side-Channel Leakages

M. Caner Tol, Berk Sunar

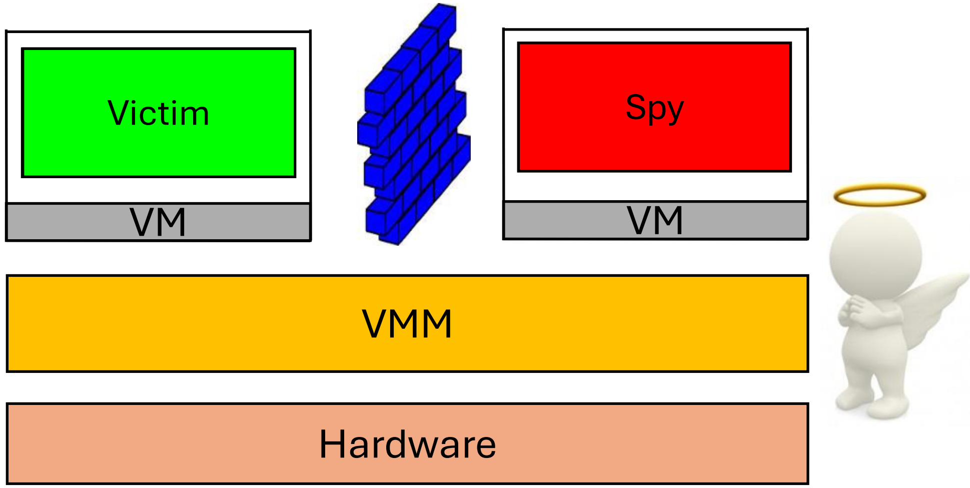
Center for Hardware Intelligence, Privacy and Security (CHIPS)

Worcester Polytechnic Institute

Real World Crypto, 2024

Microarchitectural Attacks

- Exploits uArch features in modern CPUs
- Timing side-channel leakage (early 2000’s)
- Breaks through sandboxing:
  - Virtual machines, Cloud VMs, Browsers, Mobile OSs
- Transient Execution Attacks:
  - Spectre, Meltdown, etc.
  - Side effect of speculative execution
  - E.g. allows regular users to gain root privileges on Intel and ARM machines.
- MDS Attacks RIDL, ZombieLoad, Fallout etc.
- Rowhammer fault injection
- Recent vulnerability: Downfall by Moghimi et al.



The Big Picture

- Crypto library and software developers are reluctant to issue patches e.g.
  - OpenSSL's response to Spectre/Meltdown style disclosures:
  - Similar response to Rowhammer attacks on OpenSSL primitives:

Sent: Wednesday, August 24, 2022 4:27 AM
Subject: [EXT] Re: [openssl-security] openssl - vulnerability disclosure

Thank you for this report. In general fault injection attacks are outside of our threat model - see https://www.openssl.org/policies/general/security-policy.html

OpenSSL Blog

Blog | Archives

POSTED BY OPENSSL TECHNICAL COMMITTEE , MAY 13TH, 2022 12:00 AM

Spectre and Meltdown Attacks Against OpenSSL

"Local side channel attacks, such as these, are outside the scope of our security policy, however the project generally does introduce mitigations when they are discovered. In this case, the OTC has decided that these attacks will **not** be mitigated by changes to the OpenSSL code base."

Developer’s Wishlist

From the OpenSSL Blog:

- “Maintaining code with mitigations in place would be significantly more difficult. Most potentially vulnerable code is extremely non-obvious, even to experienced security programmers.
- It would thus be quite easy to introduce new attack vectors or fix existing ones unknowingly.
- The mitigations themselves obscure the code which increases the maintenance burden.”
- “Automated verification and testing of the attacks is necessary but not sufficient. We do not have automated detection for this family of vulnerabilities and if we did, it is likely that variations would escape detection. This does not mean we won’t add automated checking for issues like this at some stage.”
- “These problems are fundamentally a bug in the *hardware*…”
- “Some kernels and compilers can provide partial mitigation. Specifically, several common compilers have introduced code generation options addressing some of these classes of vulnerability…”

Hardness

Trust

Maintainability

Automation

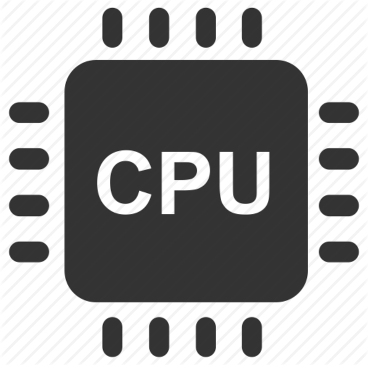
Reliability

Ownership

Toolchain Integration

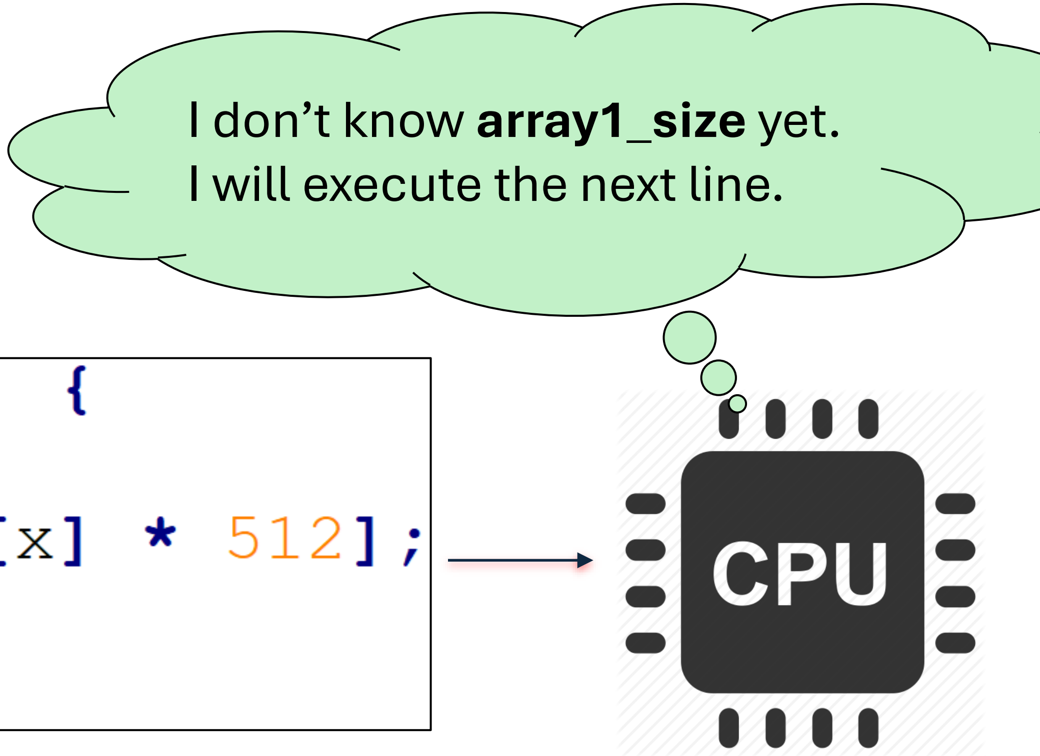
Background – Speculative Execution

```
void user_function_v01(size_t x) {
        if (x < array1_size) {
                temp &= array2[array1[x] * 512];
        }
}
```



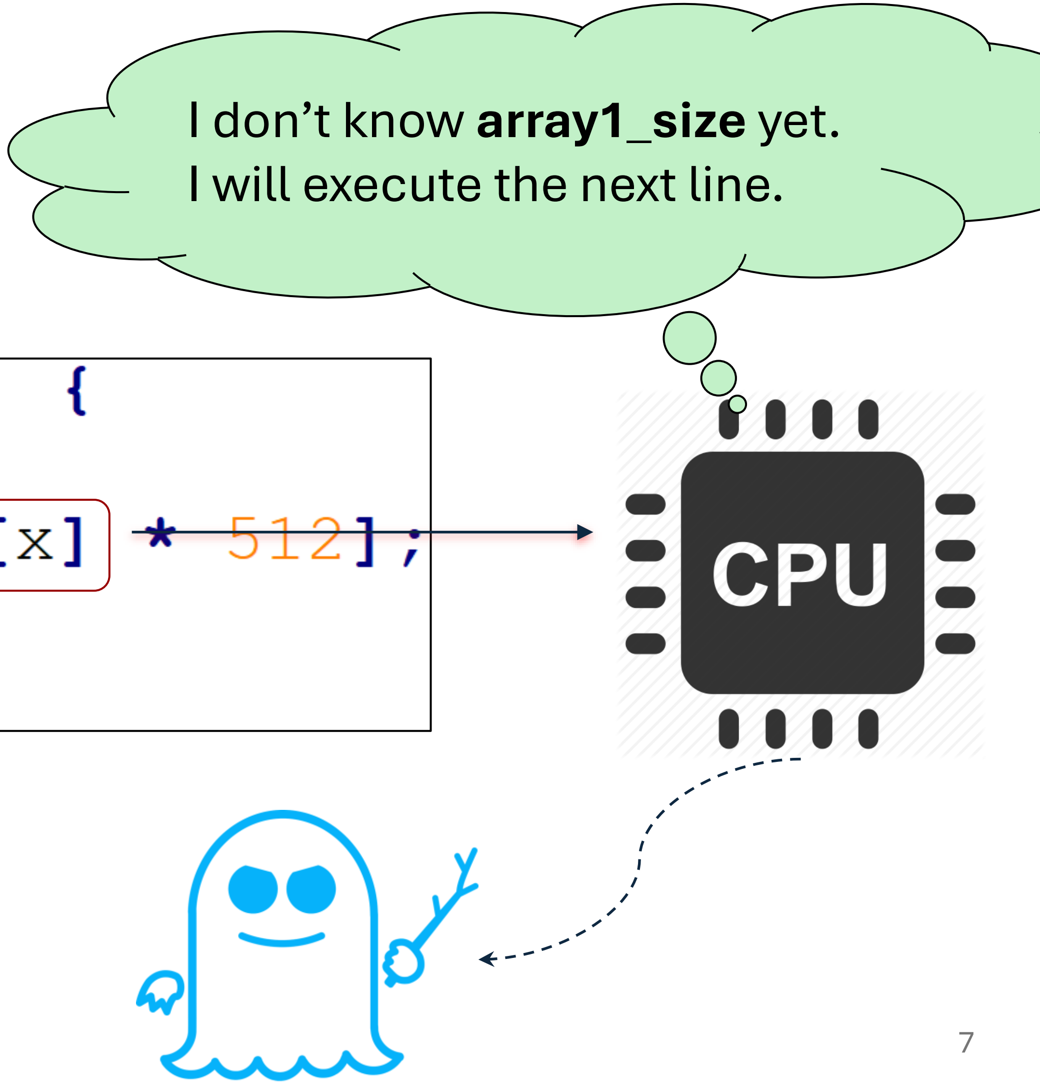
Background – Speculative Execution

```
void user_function_v01(size_t x) {
        if (x < array1_size) {
            temp &= array2[array1[x] * 512];
        }
}
```



Background – Speculative Execution

```
void user_function_v01(size_t x) {
        if (x < array1_size) {
                temp &= array2[array1[x] * 512];
        }
}
```



Our Approach

- Can we use GPT4 to rewrite vulnerable code?
  - If yes, we have a scalable tool!
  - No costly human security expert in the loop
- Goals
  - Patch non-constant time behavior by recoding
  - Patch Spectre-v1 gadgets by recoding
- For detection, we can use third-party tools
  - Microwalk for non-constant time behavior (dynamic execution)
  - Other tools for Spectre v1

Vulnerable Code Fragments

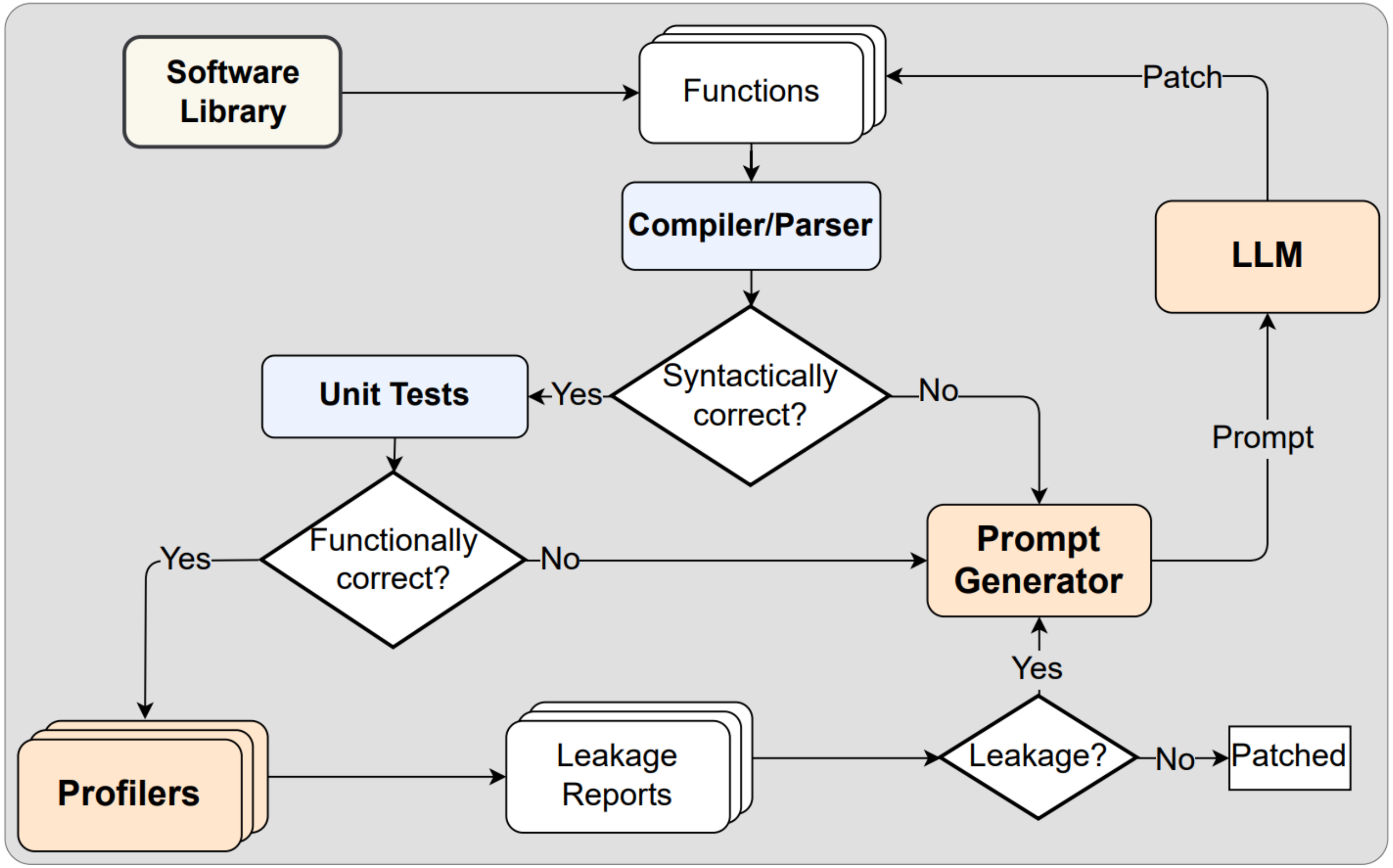
Data Dependent Equality check

```
bool equals(byte a[], size_t a_len,
            byte b[], size_t b_len) {
  for (size_t i = 0; i < a_len; i++)
    if (a[i] != b[i])  // data dependent!
      return false
  return true;
}
```

Spectre v1 example

```
void victim_function(size_t x){
 if(x < size)
  temp &= array2[array1[x] * 512];
}
```

ZeroLeak Framework



Use GPT4 for Microwalk Template Generation

- The dynamic profiler Microwalk requires a template input

- We may also use GPT4 to generate the Microwalk input template

```
User Prompt:
Implement a driver code using the following
template. Do not implement any other functions.

#include <stdint.h>
#include <stdio.h>
#include <crypto.h>

extern void RunTarget(FILE* input){
  // Read the input file and assign it to the
  // secret key
  // Initialize other variables with random data
  // Execute the primitive
  // Verify If the primitive works
}

extern void InitTarget(FILE* input){
  // Initialize library
  // If there isn't a dedicated initialization
  // function, just run the first test case for
  // the first test case file:
  // RunTarget(input);
}
```

Constant-time Patching

- In ZeroLeak, the Profiler identifies leaky C and Javascript lines reporting specifics:
  - Level of leakage
  - Line of the leaky statements
  - Type of leakage, e.g. memory reference
- We use this information to populate a Prompt Template for constant time patching
- We replace `<language>` with the programming language, (C or Javascript).
- We use `<specifics>` for instructing workarounds for the tool or language-specific compatibility issues.
  - E.g., Javascript version ES6 is not supported by Microwalk.

```
System Prompt:
You are an expert at implementing constant-time
cryptographic algorithms in <language>.
Patch the given functions according to user's
instructions. Do not give detailed explanations.
The generated code should be complete, do not omit
any part of the code. It should be able to run
without any post-processing. You can implement new
functions and integrate them with the original
function. Do not introduce new arguments to the
given function. Do not change the name of the
function. <specifics>

User Prompt:
<Option 1>
<function to patch> <array names> array is
accessed dependent on the secret in line <line>.
Patch the code such that the array access is made
input independent.
<Option 2>
<function to patch> The condition in
<if statement> is secret dependent and causes
side channel vulnerability. Patch the code such
that it does not require any conditional execution.
<Option 3>
<function to patch> The termination condition in
<loop statement> is secret dependent. Patch the
code such that loops execute the same amount of
time for every input.
<Option 4>
<crash reason> The generated code must be complete.
Generate everything even if you do not make any
changes. Try the same patch again.
```

Comparison of LLMs

Model-Version	Release Date	Publisher	Open-Sourced	Memory Leakage	Branch Leakage	Spectre-V1	Estimated Cost [USD]
GPT4-0613	06/13/2023	OpenAI	✗	**5/5**	**12/13**	**16/16**	$1.34
GPT3.5-turbo-0613	06/13/2023		✗	2/5	9/13	10/16	$0.07
text-davinci-003	10/28/2022		✗	0/5	7/13	12/16	$2.29
code-davinci-edit-001	03/15/2022		✗	0/5	8/13	5/16	$0[†]
chat-bison-001	07/10/2023	Google	✗	0/5	5/13	14/16	$0.06
codechat-bison-001	06/29/2023		✗	0/5	6/13	0/16	$0.28
code-bison-001	06/29/2023		✗	1/5	4/13	0/16	$0.04
text-bison-001	06/07/2023		✗	1/5	5/13	0/16	$0.10
LLaMA2-70B	07/18/2023	Meta	✓	1/5	8/13	3/16	$0[‡]

- Patching with different models.
- Constant-timeness, e.g. secret dependent memory access patterns, conditional branches, and varying loop sizes are tested using Microwalk. Spectre-V1 was tested using Pitchfork.
- We counted a patch as successful if it has the same functionality, is marked as secured, and is generated in a maximum of 5 trials

Spectre-v1 Results

Cases	Baseline (cc)	Inline lfence (cc)	clang SLH (cc)	clang lfence (cc)	USLH(cc) [58]	**ZeroLeak (cc)**
1	6 ✗^{p} ✗^{s} ✗^{k}	22 ✓^{p} ✓^{s} ✓^{k}	17 ✗^{p} ✓^{s}	54 ✓^{p} ✓^{s}	14 ✗^{p} ✓^{s}	**6** ✓^{p} ✓^{s} ✓^{k}
2	6 ✗^{p} ✗^{s} ✗^{k}	30 ✓^{p} ✓^{s} ✓^{k}	33 ✗^{p} ✓^{s}	56 ✓^{p} ✓^{s}	35 ✗^{p} ✓^{s}	**7** ✓^{p} ✓^{s} ✓^{k}
3	7 ✗^{p} ✗^{s} ✗^{k}	29 ✓^{p} ✓^{s} ✓^{k}	32 ✗^{p} ✓^{s}	57 ✓^{p} ✓^{s}	34 ✗^{p} ✓^{s}	**9** ✓^{p} ✓^{s} ✓^{k}
4	6 ✗^{p} ✗^{s} ✗^{k}	24 ✓^{p} ✓^{s} ✓^{k}	16 ✗^{p} ✓^{s}	54 ✓^{p} ✓^{s}	14 ✗^{p} ✓^{s}	**7** ✓^{p} ✓^{s} ✓^{k}
5	78 ✗^{p} ✗^{s} ✗^{k}	105 ✓^{p} ✗^{s} ✓^{k}	170 ✗^{p} ✓^{s*}	399 ✓^{p} ✓^{s*}	148 ✗^{p} ✓^{s*}	**88** ✓^{p} ✓^{s} ✗^{k†}
6	6 ✗^{p} ✗^{s} ✗^{k}	24 ✓^{p} ✓^{s} ✓^{k}	16 ✗^{p} ✓^{s}	58 ✓^{p} ✓^{s}	14 ✗^{p} ✓^{s}	**6** ✓^{p} ✓^{s} ✓^{k}
7	6 ✗^{p} ✗^{s} ✗^{k}	24 ✓^{p} ✓^{s} ✓^{k}	25 ✗^{p} ✓^{s}	76 ✓^{p} ✓^{s}	20 ✗^{p} ✓^{s}	**9** ✓^{p} ✓^{s} ✓^{k}
8	5 ✗^{p} ✗^{s} ✗^{k}	N/A	17 ✗^{p} ✓^{s}	42 ✓^{p} ✓^{s}	15 ✗^{p} ✓^{s}	**16** ✓^{p} ✓^{s} ✓^{k}
9	4 ✗^{p} ✗^{s} ✗^{k}	22 ✓^{p} ✓^{s} ✓^{k}	15 ✗^{p} ✓^{s}	50 ✓^{p} ✓^{s}	14 ✗^{p} ✓^{s}	**9** ✓^{p} ✓^{s} ✓^{k}
10	6 ✗^{p} ✗^{s} ✗^{k}	21 ✓^{p} ✓^{s} ✓^{k}	23 ✗^{p} ✓^{s}	66 ✓^{p} ✓^{s}	22 ✗^{p} ✓^{s}	**7** ✓^{p} ✓^{s} ✓^{k}
11gcc	14 ✗^{p} ✗^{s} ✗^{k}	35 ✓^{p} ✗^{s} ✓^{k}	65 ✗^{p} ✓^{s}	98 ✓^{p} ✓^{s}	64 ✗^{p} ✓^{s}	**17** ✓^{p} ✓^{s} ✓^{k}
11ker	15 ✗^{p} ✗^{s} ✗^{k}	35 ✓^{p} ✗^{s} ✓^{k}	69 ✗^{p} ✓^{s}	100 ✓^{p} ✓^{s}	66 ✗^{p} ✓^{s}	**20** ✓^{p} ✓^{s} ✗^{k†}
11sub	12 ✗^{p} ✗^{s} ✗^{k}	35 ✓^{p} ✗^{s} ✓^{k}	64 ✗^{p} ✓^{s}	100 ✓^{p} ✓^{s}	61 ✗^{p} ✓^{s}	**12** ✓^{p} ✓^{s} ✓^{k}
12	5 ✗^{p} ✗^{s} ✗^{k}	25 ✓^{p} ✓^{s} ✓^{k}	16 ✗^{p} ✓^{s}	55 ✓^{p} ✓^{s}	14 ✗^{p} ✓^{s}	**7** ✓^{p} ✓^{s} ✓^{k}
13	5 ✗^{p} ✗^{s} ✗^{k}	25 ✓^{p} ✓^{s} ✓^{k}	24 ✗^{p} ✓^{s}	74 ✓^{p} ✓^{s}	21 ✗^{p} ✓^{s}	**7** ✓^{p} ✓^{s} ✓^{k}
14	6 ✗^{p} ✗^{s} ✗^{k}	25 ✓^{p} ✓^{s} ✓^{k}	16 ✗^{p} ✗^{s}	54 ✓^{p} ✓^{s}	14 ✗^{p} ✗^{s}	**6** ✓^{p} ✓^{s} ✓^{k}

- Spectre v1 patch overhead comparison for GPT4
- The numbers reported are in clock cycles.
- The superscripts *p*, *s*, and *k* represent Pitchfork, Spectector, and KLEESpectre

OpenSSL Spectre v1 Example

- Found by the Clou Tool in OpenSSL
- Still unpatched
- Red: Vulnerable Spectre v1 gadget in OpenSSL
- Green: Secure patch generated by ZeroLeak (GPT4)

```
User prompt:
<function body>
if (s->shared_sigalgs == NULL
    || idx < 0
    || idx >= (int)s->shared_sigalgslen
    || s->shared_sigalgslen > INT_MAX) can be
speculatively executed when the condition inside
is wrong. Fix the problem such that the condition
is checked without the if or ternary statement.
Patch:
```

```
int SSL_get_shared_sigalgs(SSL *s, int idx, int *
     psign, int *phash, int *psignhash, unsigned
     char *rsig, unsigned char *rhash){
   const SIGALG_LOOKUP *shsigalgs;
- if (s->shared_sigalgs == NULL
-     || idx < 0
-     || idx >= (int)s->shared_sigalgslen
-     || s->shared_sigalgslen > INT_MAX)
-     return 0;
+/* If any of the conditions are true, the mask
+ * will be set to all 1s (i.e., -1).
+ * If all conditions are false,
+ * the mask will be 0. */
+ int mask = -((s->shared_sigalgs == NULL) |
+   (idx < 0) |
+   (idx >= (int)s->shared_sigalgslen) |
+   ((unsigned int)s->shared_sigalgslen >INT_MAX));
- shsigalgs = s->shared_sigalgs[idx];
- if (phash != NULL)
-   *phash = shsigalgs->hash;
+ /* Apply the mask to idx and then use the
+  * result to index the array.
+  * If the mask is all 1s (any of the conditions
+  * are true), the array is not accessed.
+  * If the mask is 0, idx is used unmodified. */
+ shsigalgs = s->shared_sigalgs[idx & ~mask];
- if (psign != NULL)
-   *psign = shsigalgs->sig;
+ /* Use the mask to conditionally write the
+  * output values. If the mask is 0, the output
+  * values are written. */
+ *phash = (shsigalgs->hash & mask)
+           | (*phash & ~mask);
+ *psign = (shsigalgs->sig & mask)
+           | (*psign & ~mask);
- if (psignhash != NULL)
-   *psignhash = shsigalgs->sigandhash;
+ *psignhash = (shsigalgs->sigandhash & mask)
+                       | (*psignhash & ~mask);
- if (rsig != NULL)
-   *rsig = (unsigned char)
-             (shsigalgs->sigalg & 0xff);
+ *rsig = ((unsigned char)
+             (shsigalgs->sigalg & 0xff)
+                & mask) | (*rsig & ~mask);
- if (rhash != NULL)
-   *rhash = (unsigned char)((shsigalgs->sigalg
-                     >> 8) & 0xff);
+ *rhash = ((unsigned char)((shsigalgs->sigalg
+   >> 8) & 0xff) & mask) | (*rhash & ~mask);
- return (int)s->shared_sigalgslen;
+ return (s->shared_sigalgslen & ~mask)
+                        | (0 & mask);
 }
```

Further Scrutinizing the Patch

- We observed that LLM-generated patches for Spectre-v1 gadgets tend to use similar methods, such as index-masking which is commonly used in large commercial products.
- GPT4 generates the patch in the 3rd iteration.
- Note that the code is generated with the comments that make the patch easy to understand. *(Maintainability)*
- After careful review, we see that the if condition is eliminated, and the check logic is accumulated on the `mask` variable.
- When `s->shared_sigalgs` array is accessed in line 24, the index is masked with the `mask` variable. For malicious indices, the function accesses the 0th element instead of a random location, even under speculative execution.
- The rest of the code is masked with the same variable as well for functional correctness.

Conclusions

- Using LLMs, we can patch large repos for just cents/vulnerability
- No need for training!
- Comments included
- We can even query LLM for additional explanations
- Large variability between models
  - Can be improved by further refining prompts
- No human intervention required
- Just scratched the surface
  - Need to further study shortcomings of LLMs

Thank you!

Questions?