

Lessons Learned Building Generative AI Applications

Aaron D'Amico

What are we talking about?

- Steps to making a generative AI application
- Dos and Don'ts
- Tech stack
- Example (code available on github)
- Demo?? (If we have time)

Disclaimer

- Only one man's opinion
- I'm going to make a lot of broad statements that might not apply directly to you.
- I don't love Dalmatians
- I used AI to make images in the style of a certain film and it went too far.



Steps to making a generative AI application



1. Chunk and Vectorize a dataset
2. Create and Vectorize a query
3. Build a prompt that includes:
 1. Your query
 2. And data from the dataset with similar vector to your query
4. Send to LLM

Lessons Learned: dos and don'ts

Where did the lessons come from?

3Flatline.ai

- ML/AI driven source code and binary vulnerability scanner.
- Finds bugs, write code to fixes bugs, write code to test bugs
- AI Application Security Engineer



Generative AI is a Pattern Matching Machine



- AI is not skynet. It is a pattern matching machine right now.
- To get the most out of LLM, think about what patterns you can match against for your use case.
- LLMs are great at summaries
- LLMs are great at parsing non-uniform data

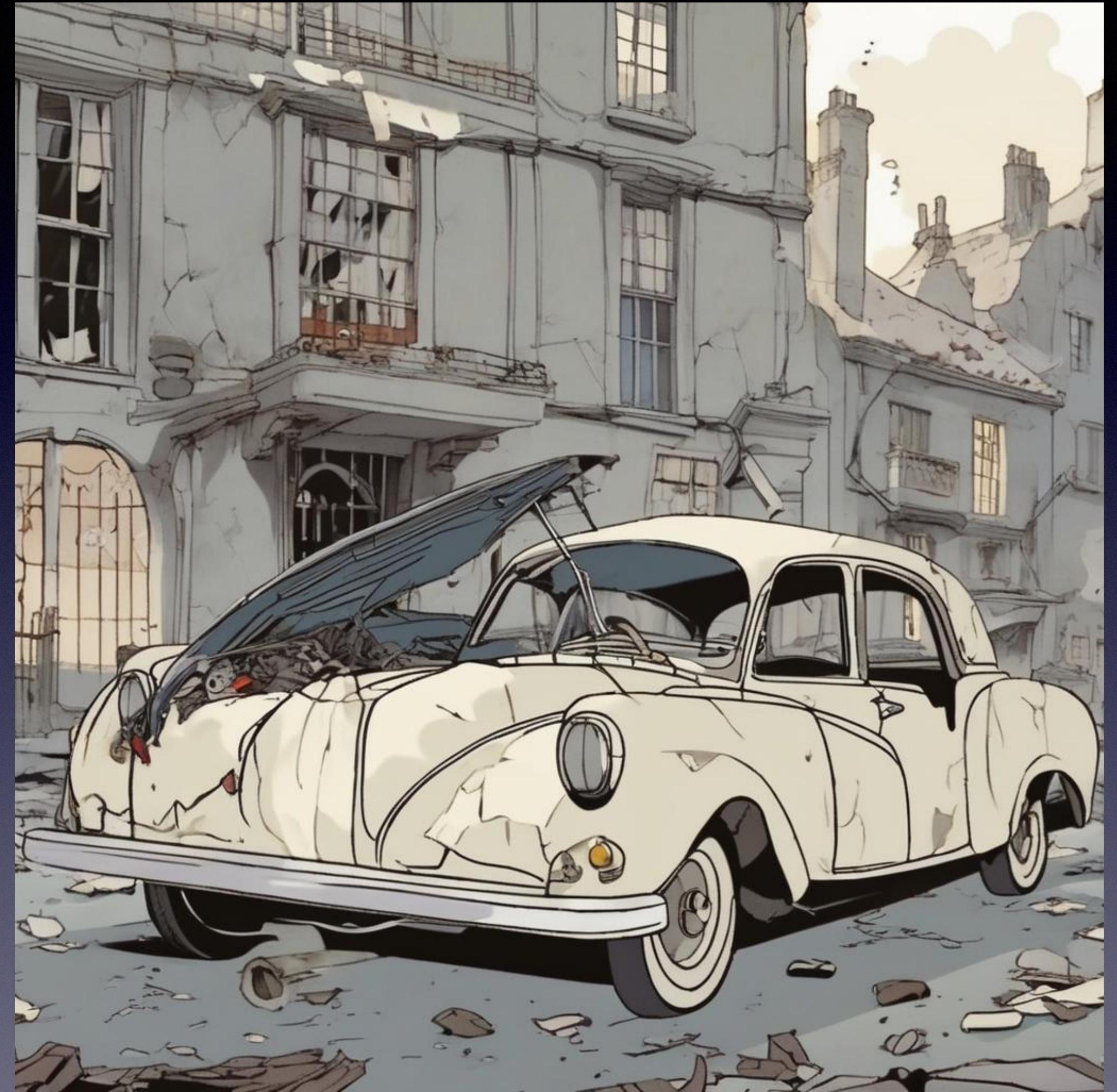
Think of LLMs as a dog



- LLMs really want to please and do not like giving no or negative answers.
- LLMs don't want to prove a negative

Don't fine tune your LLM

- Fine tuning is for the rich and those that need really well tuned formats for responses.
- Just use Retrieval-Augmented Generation.



Use the RAG approach

(Retrieval-Augmented Generation)

- Can be considered a form of fine tuning
- Reduces hallucinations
- Easy to do.
- Cheap
- Do not just send your prompt (question) to a LLM without addition data



Too large of a context window is BAD

- Be like Goldilocks
- Anything greater than 32,000 tokens and your responses get muddled.



Data Quality over Quantity

- **BE LIKE GOLDBLOCKS**
- Your dataset needs to be of a high signal to noise ratio. Preferably all signal
- AI has a garbage in garbage out problem





Don't use agents

- Super insecure
- All you need is chat completion.

Tech Stack: the how

We need

- A model
- A vector store
- Our dataset (as discussed)
- It is not a lot of code. Magic is all in the dataset

What models to use?

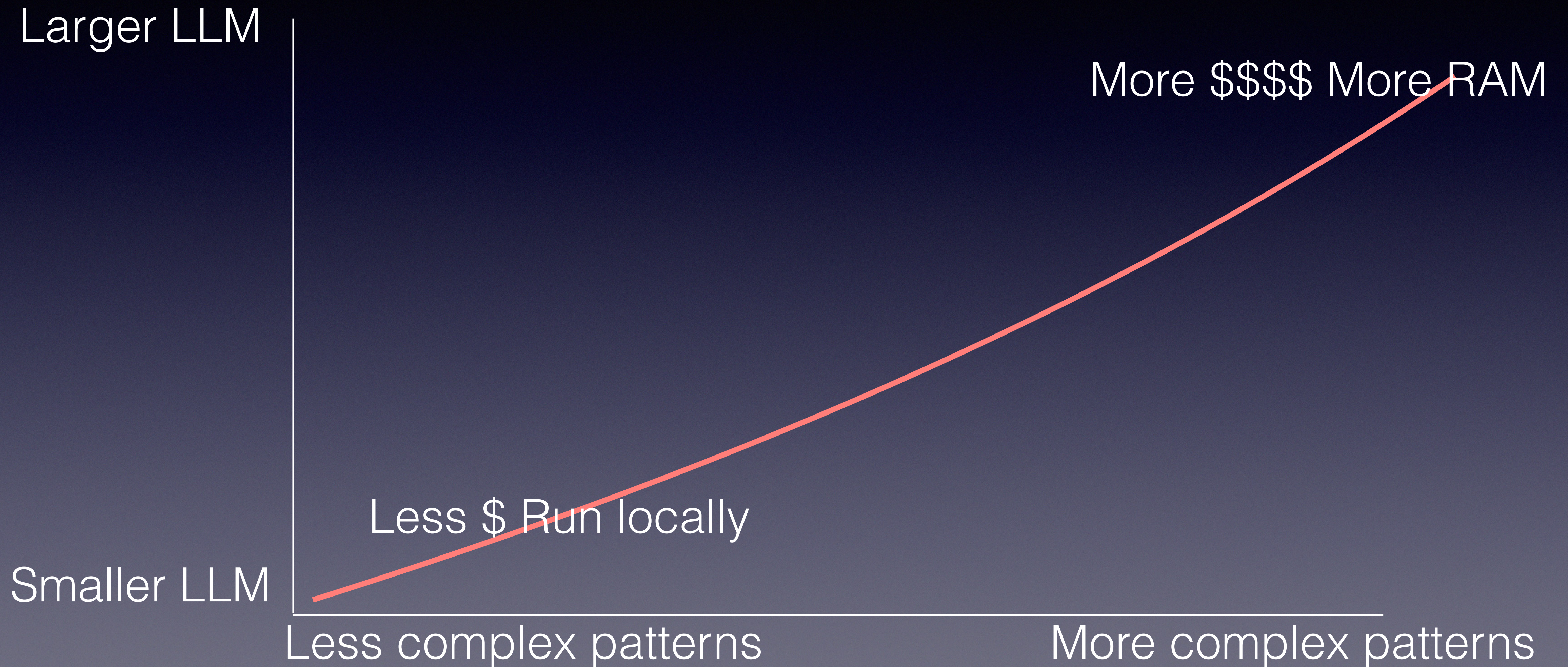
Use the right tool for the right job.

- Maybe that's OpenAI
- Maybe that's an OpenSource model that runs local on your machine

Ask yourself: How complex is
the pattern I want to find?

Now you know what model to use.

Whatever makes sense.



Don't use llamaIndex

- Everything wrong with Python
- Too much micro code. Obtuse and abstracted.
- Impossible to debug
- Just as easy to interact with the model directly



Use pgvector for your vector store

- Fast
- Easy
- Store data with the vectors in the same table



Practical Example

Used the above techniques to learn javascript exploitation

Code and this slide show will be
available on GitHub tomorrow.



Query

How would I test a type confusion bug in v8. Here is the bug report.

Vuln Class: Type Confusion

Summary:

<example type confusion bug code>

However, there is no type check on the arguments before passing them to the runtime function. This can lead to a type confusion vulnerability where an attacker can pass unexpected types of arguments and potentially execute arbitrary code.

CVSS3.1 Score: High (8.8)

WITHOUT lessons learned

To test a type confusion bug in v8, you can follow these steps:

1. Clone the v8 repository from GitHub:

```
git clone https://github.com/v8/v8.git
```

2. Build v8 with the necessary flags to enable the debugging features:

```
cd v8
```

```
./configure --debug
```

```
ninja -C out.gn/x64.debug
```

3. Create a test file, let's call it `type_confusion_test.js`, with the following content:

```
function testTypeConfusion() {  
  // Replace the following code with the code from the bug report  
  // ...  
}
```

```
testTypeConfusion();
```

4. Replace the code in the `testTypeConfusion` function with the code from the bug report. Make sure to keep the function name intact.

5. Run the test file with the v8 debugger:

```
out.gn/x64.debug/d8 --debug-brk=100 --allow-natives-syntax type_confusion_test.js
```

6. In the debugger, set a breakpoint at the beginning of the `testTypeConfusion` function:

```
breakpoint set type_confusion_test.js:1
```

7. Resume the execution of the script:

```
continue
```

8. Step into the `testTypeConfusion` function:

```
Step
```

9. Step through the code, line by line, until you reach the line where the **`NewTypeError`** runtime function is called.

10. At this point, you can try to trigger the type confusion vulnerability by passing unexpected types of arguments to the **`NewTypeError`** function. For example, you can try passing a string instead of a **`Smi`** object:

```
builder()  
  .LoadLiteral("test") // Replace Smi::FromEnum(tmp1) with a string literal  
  .StoreAccumulatorInRegister(args[0])  
  .LoadLiteral(name)  
  .StoreAccumulatorInRegister(args[1])  
  .CallRuntime(Runtime::kNewTypeError
```


WITHOUT lessons learned

To test a type confusion bug in v8, you can follow these steps:

1. Clone the v8 repository from GitHub:

```
git clone https://github.com/v8/v8.git
```

2. Build v8 with the necessary flags to enable the debugging features:

```
cd v8
```

```
./configure --debug
```

```
ninja -C out.gn/x64.debug
```

3. Create a test file, let's call it `type_confusion_test.js`, with the following content:

```
function testTypeConfusion() {
```

```
    // Replace the following code with the code from the bug report
```

```
    // ..
```

```
}
```

```
testTypeConfusion();
```

4. Replace the code in the `testTypeConfusion` function with the code from the bug report. Make sure to keep the function name intact.

5. Run the test file with the v8 debugger:

```
out.gn/x64.debug/d8 --debug-brk=100 --allow-natives-syntax type_confusion_test.js
```

6. In the debugger, set a breakpoint at the beginning of the `testTypeConfusion` function:

```
breakpoint set type_confusion_test.js:1
```

7. Resume the execution of the script:

```
continue
```

8. Step into the `testTypeConfusion` function:

```
Step
```

9. Step through the code line by line, until you reach the line where the `NewTypeError` runtime function is called.

10. At this point, you can try to trigger the type confusion vulnerability by passing unexpected types of arguments to the `NewTypeError` function. For example, you can try passing a string instead of a `Smi` object:

```
builder()
```

```
    .LoadLiteral("test") // Replace Smi::FromEnum(tmp1) with a string literal
```

```
    .StoreAccumulatorInRegister(args[0])
```

```
    .LoadLiteral(name)
```

```
    .StoreAccumulatorInRegister(args[1])
```

```
    CallRuntime(Runtime::kNewTypeError
```

Generic instructions

Responds like a
high school student
trying to BS a paper

Poor Code Quality

WITH lessons learned

To trigger the specific type confusion bug in V8, you can follow these steps:

1. Clone the V8 repository from GitHub:

```
git clone https://github.com/v8/v8.git
```

2. Build V8 with the necessary flags to enable the debug build and the required runtime function:

```
./configure --debug-code --debug-brk --shared-libs  
make -j4
```

3. Create a test script that triggers the type confusion bug. You can use the provided code snippet as a starting point:

```
function exploit() {  
  let tmp1 = 42; // Replace with the desired template value  
  let name = "test"; // Replace with the desired name value  
  
  let args = register_allocator().NewRegisterList(2);  
  builder()  
    .LoadLiteral(Smi::FromEnum(tmp1))  
    .StoreAccumulatorInRegister(args[0])  
    .LoadLiteral(name)  
    .StoreAccumulatorInRegister(args[1])  
    .CallRuntime(Runtime::kNewTypeError, args)  
    .Throw();  
}  
  
exploit();
```

4. Run the test script using the built V8 engine. You can use the following command as an example (replace `v8` with the path to your built V8 binary):

```
./v8 --allow-natives-syntax --expose_debug_asserts test_script.js
```

5. Observe the behavior of the V8 engine when running the test script. If the type confusion bug is present, you may see unexpected behavior or even a crash.

6. Analyze the crash dump or debug the V8 engine to confirm the presence of the type confusion bug.

Remember to replace the placeholder values (`tmp1` and `name`) in the test script with the desired values to trigger the type confusion bug. Additionally, ensure that you have the necessary permissions and tools to analyze and debug the V8 engine.

WITH lessons learned

To trigger the specific type confusion bug in V8, you can follow these steps:

1. Clone the V8 repository from GitHub:

```
``  
git clone https://github.com/v8/v8.git  
``
```

2. Build V8 with the necessary flags to enable the debug build and the required runtime function:

```
``  
./configure --debug-code --debug-brk --shared-libs  
make -j4  
``
```

3. Create a test script that triggers the type confusion bug. You can use the provided code snippet as a starting point:

```
``javascript  
function exploit() {  
  let tmp1 = 42; // Replace with the desired template value  
  let name = "test"; // Replace with the desired name value  
  
  let args = register_allocator().NewRegisterList(2);  
  builder()  
    .LoadLiteral(Smi::FromEnum(tmp1))  
    .StoreAccumulatorInRegister(args[0])  
    .LoadLiteral(name)  
    .StoreAccumulatorInRegister(args[1])  
    .CallRuntime(Runtime::kNewTypeError, args)  
    .Throw();  
}  
  
exploit();  
``
```

4. Run the test script using the built V8 engine. You can use the following command as an example (replace `v8` with the path to your built V8 binary):

```
``  
./v8 --allow-natives-syntax --expose_debug_asserts test_script.js  
``
```

5. Observe the behavior of the V8 engine when running the test script. If the type confusion bug is present, you may see unexpected behavior or even a crash.

6. Analyze the crash dump or debug the V8 engine to confirm the presence of the type confusion bug.

Remember to replace the placeholder values (`tmp1` and `name`) in the test script with the desired values to trigger the type confusion bug. Additionally, ensure that you have the necessary permissions and tools to analyze and debug the V8 engine.

- More detailed instructions
- Much better code sample
- Higher quality

Q&A

Sources

- <https://arxiv.org/abs/2308.16898> Transformers as Support Vector Machines
- <https://ieeexplore.ieee.org/document/788646> Support Vector Machines for Histogram-Based Image Classification
- <https://arxiv.org/abs/2307.03172> Lost in the Middle: How Language Models Use Long Contexts