Build It, Break It, Fix It

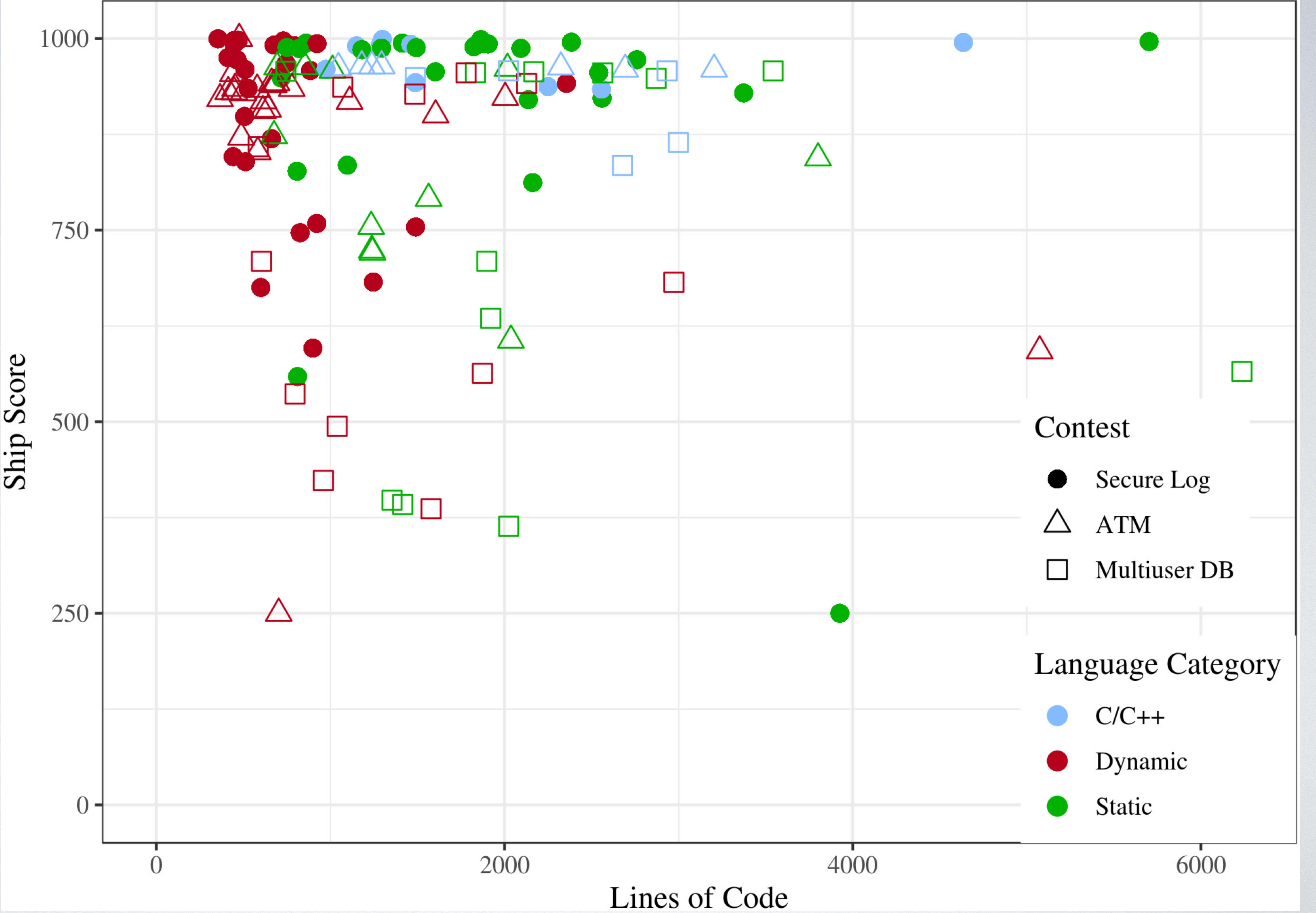
Build It, Break It, Fix It

- Long-running project (publications 2016 - 2022)
- You read a paper analyzing quantitative data from the contest
- Let's also talk about a *qualitative* study: "Understanding security mistakes developers make: Qualitative analysis from Build It, Break It, Fix It" (USENIX 2020)

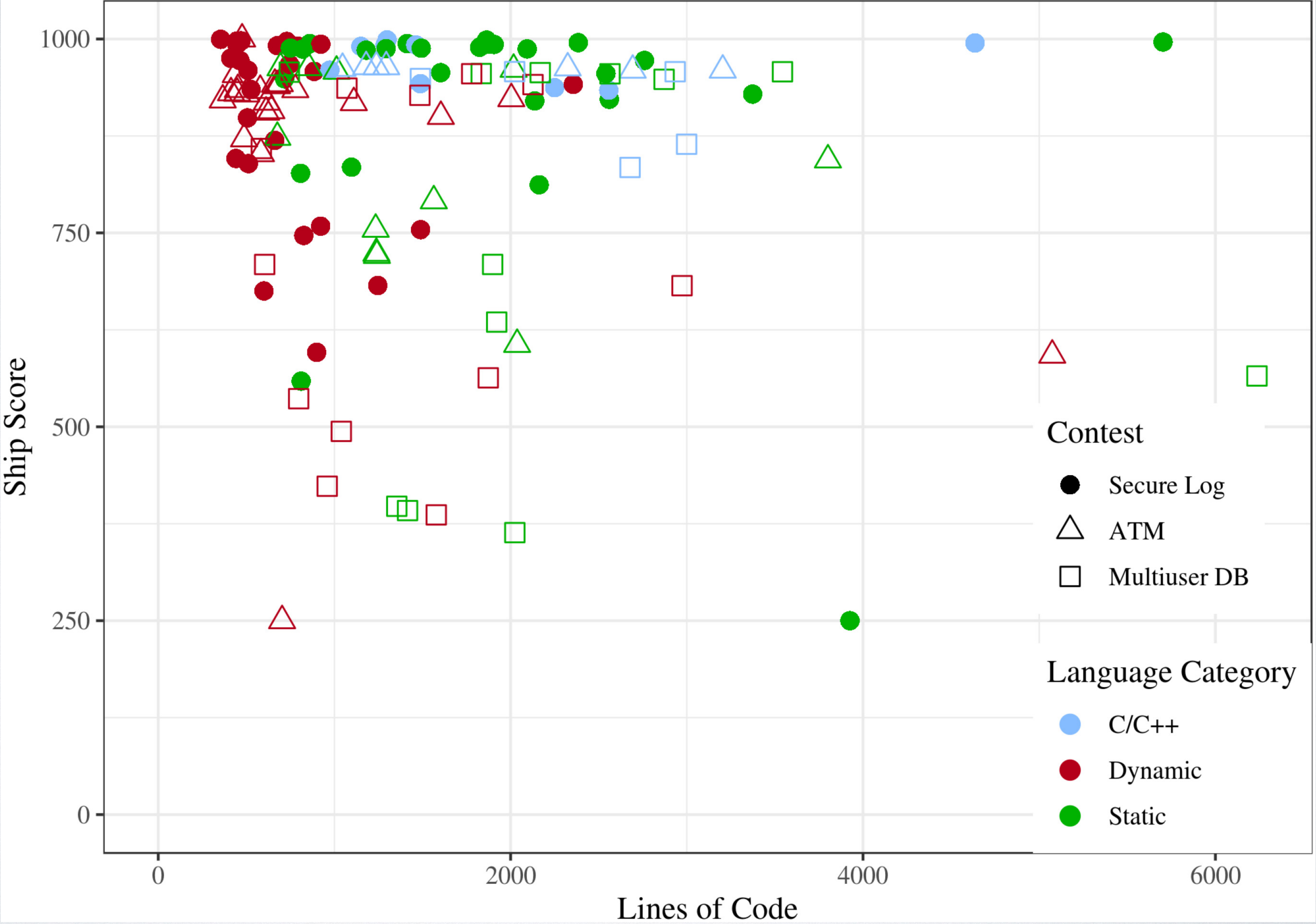
First: Focus on the Paper You Read

- Iterative design and execution of studies allowed them to do a power analysis: f^2
  - (interesting because many studies don't even try)

- What do you notice?



- Lines vs. score?



Some Key Definitions

- Integrity: data is not compromised (no improper data modifications)
- Confidentiality: only authorized users see data
- Availability: system responds to requests

Fa[illegible]

(Negative coefs. indicate

Large design space makes ATM problem hard

Table 7. Final Logistic Model Measuring Log[illegible]hood of the Discovery of a Security Bug in a Team[illegible]bmis[illegible]

Custom access control is hard

Factor	Coef.	Exp([illegible]oef)		
Secure Log	—	—	[illegible]	[illegible]
ATM	**4.639**	**103.415**	**[[illegible], 594.11]**	**<0.001***
Multiuser DB	**3.462**	**31.892**	**[7.06, 144.07]**	**<0.001***
C/C++	—	—	—	—
Statically typed	**−2.422**	**0.089**	**[0.02, 0.51]**	**0.006***
Dynamically typed	−0.9[illegible]	[illegible]	[illegible]	[illegible]
# Team members	[illegible]	[illegible]	[illegible]	[illegible]
Knowledge of C	−1.[illegible]	[illegible]	[illegible]	[illegible]
Lines of code	0.00[illegible]	[illegible]	[1, 1]	0.090

Model explains 62% of the variance!

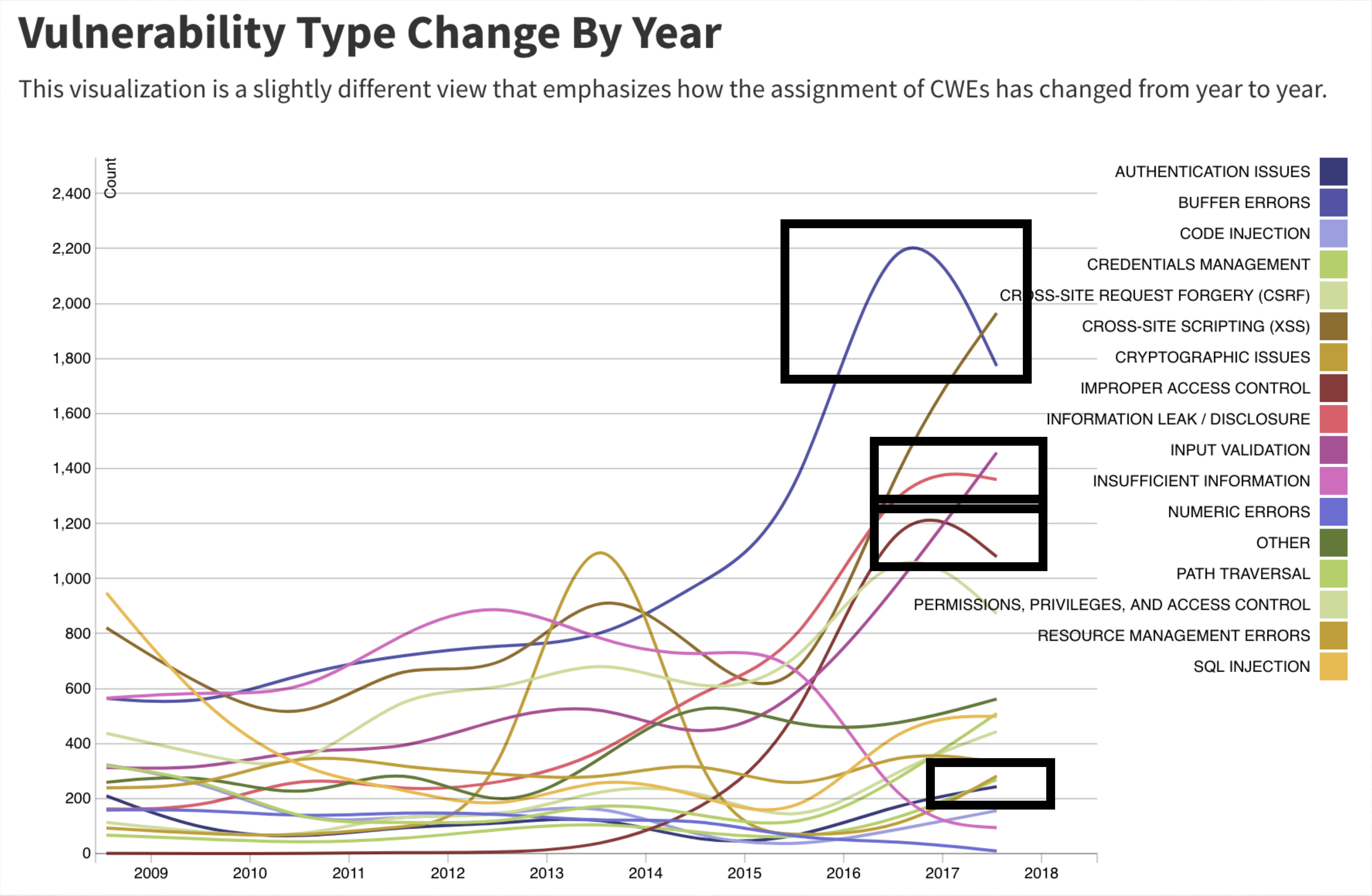
Nagelkerke $R^2 = 0.619$.

Next Up: Qualitative Study

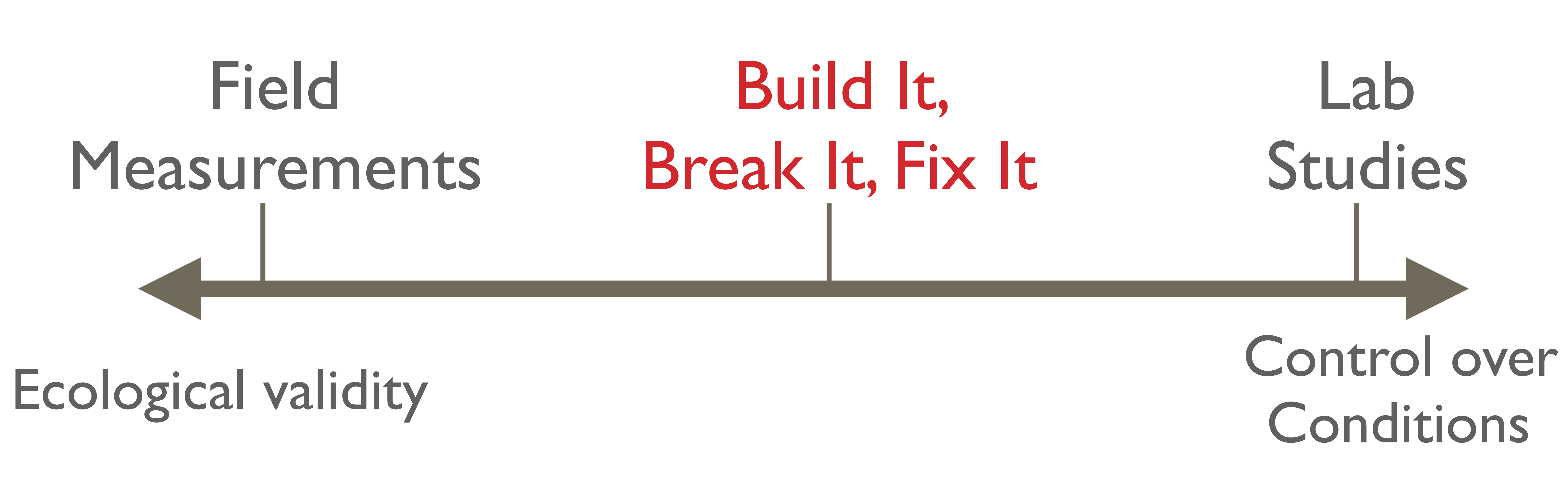
- Daniel Votipka, Kelsey Fulton, James Parker, Matthew Hou, Michelle Mazurek, and Mike Hicks. **Understanding Security Mistakes Developers Make: Qualitative Analysis from Build It, Break It, Fix It**

- Source: Dan Votipka's slides (https://www.usenix.org/system/files/sec20_slides_votipka-understanding.pdf)

"Solved" Vulnerabilities Are Still a Problem



Goal: Understand the Types, Causes, and Pervasiveness of Vulnerabilities

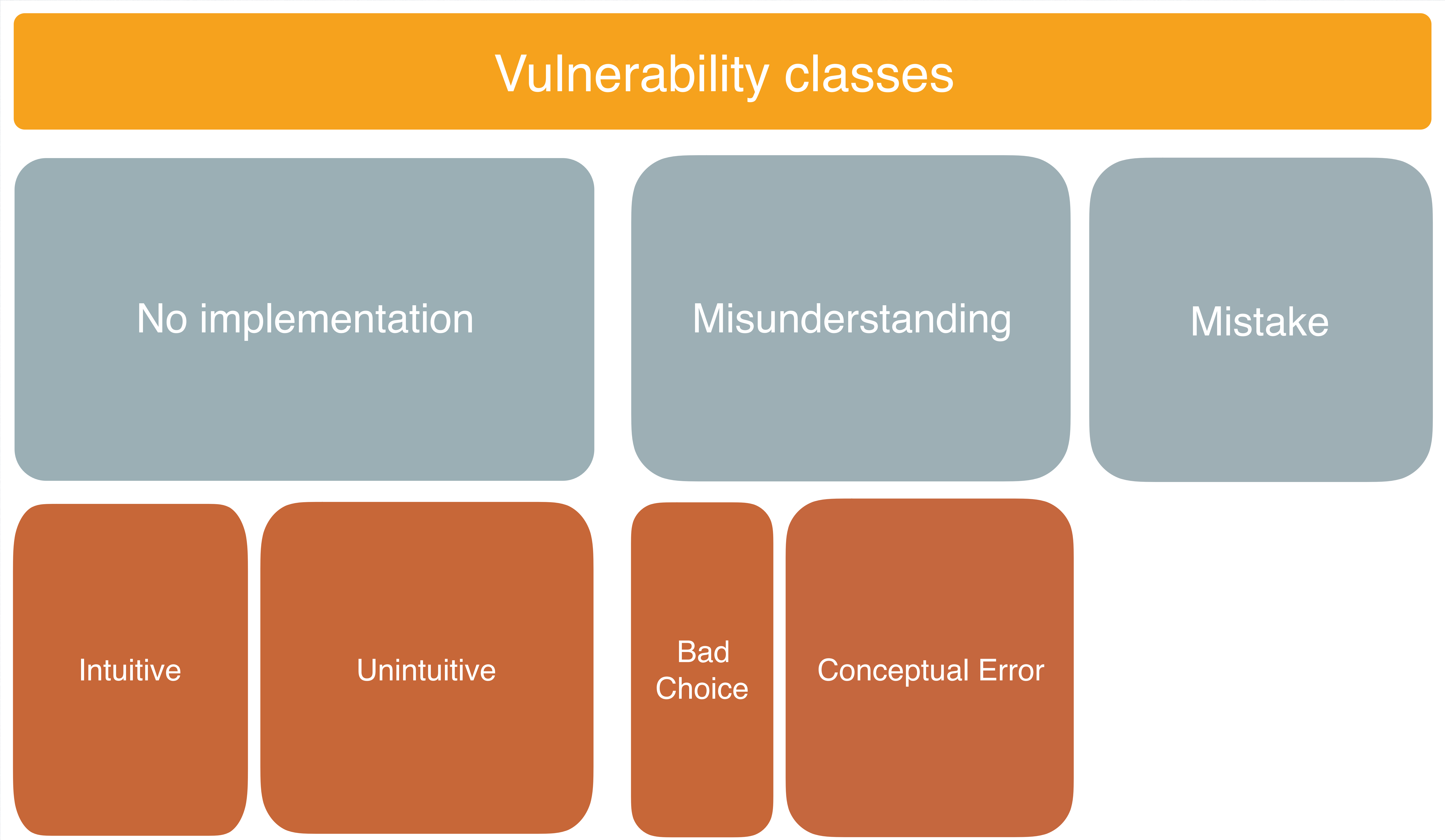


Source: Dan Votipka (https://www.usenix.org/system/files/sec20_slides_votipka-understanding.pdf)

Method

- Open coding on vulnerability data from BIBIFI
- 94 projects, 866 exploits

Classes



Source: Dan Votipka (https://www.usenix.org/system/files/sec20_slides_votipka-understanding.pdf)

Vulnerability classes

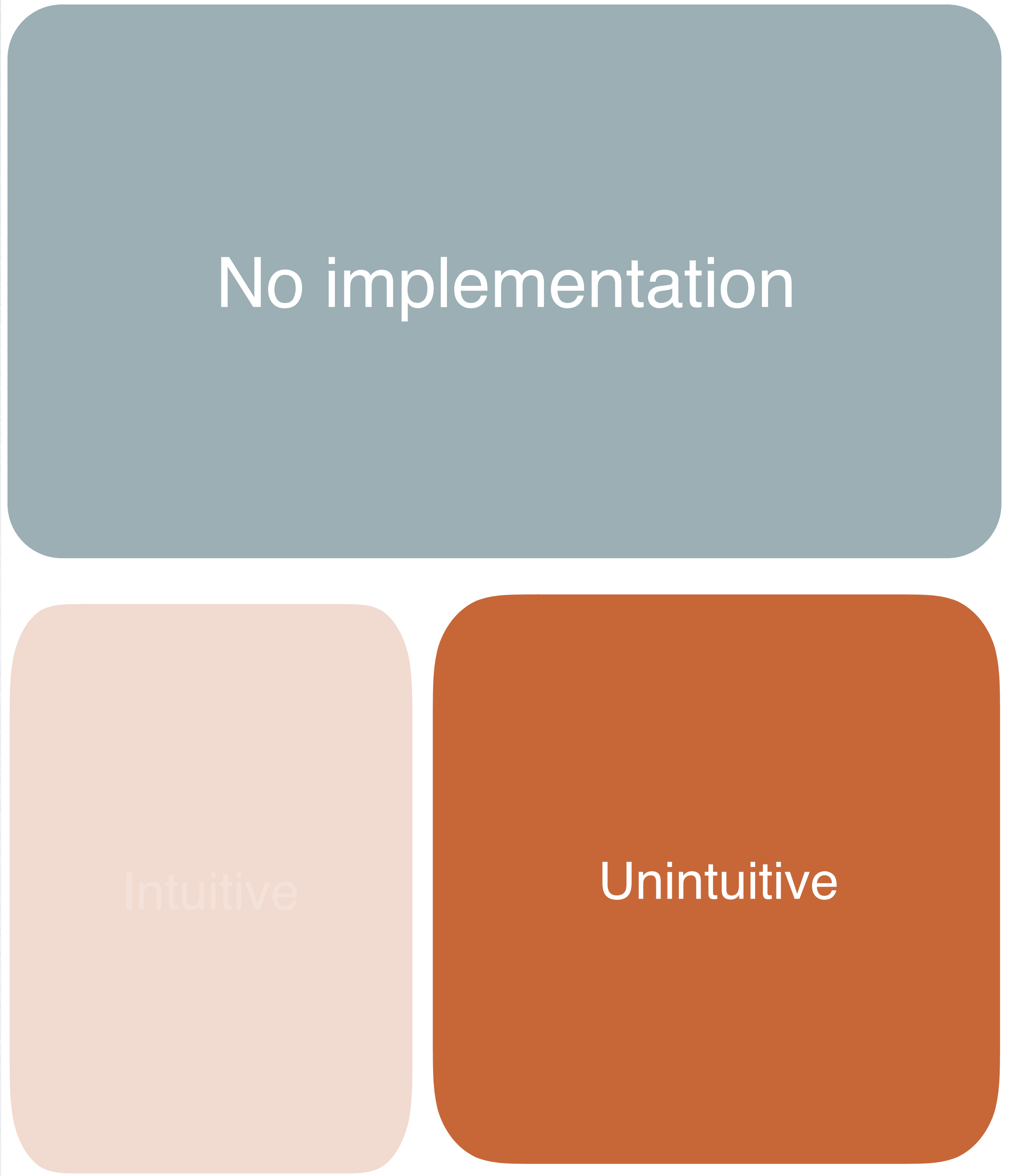
No implementation

Intuitive

- Missed something "Intuitive"
  - No encryption
  - No access control

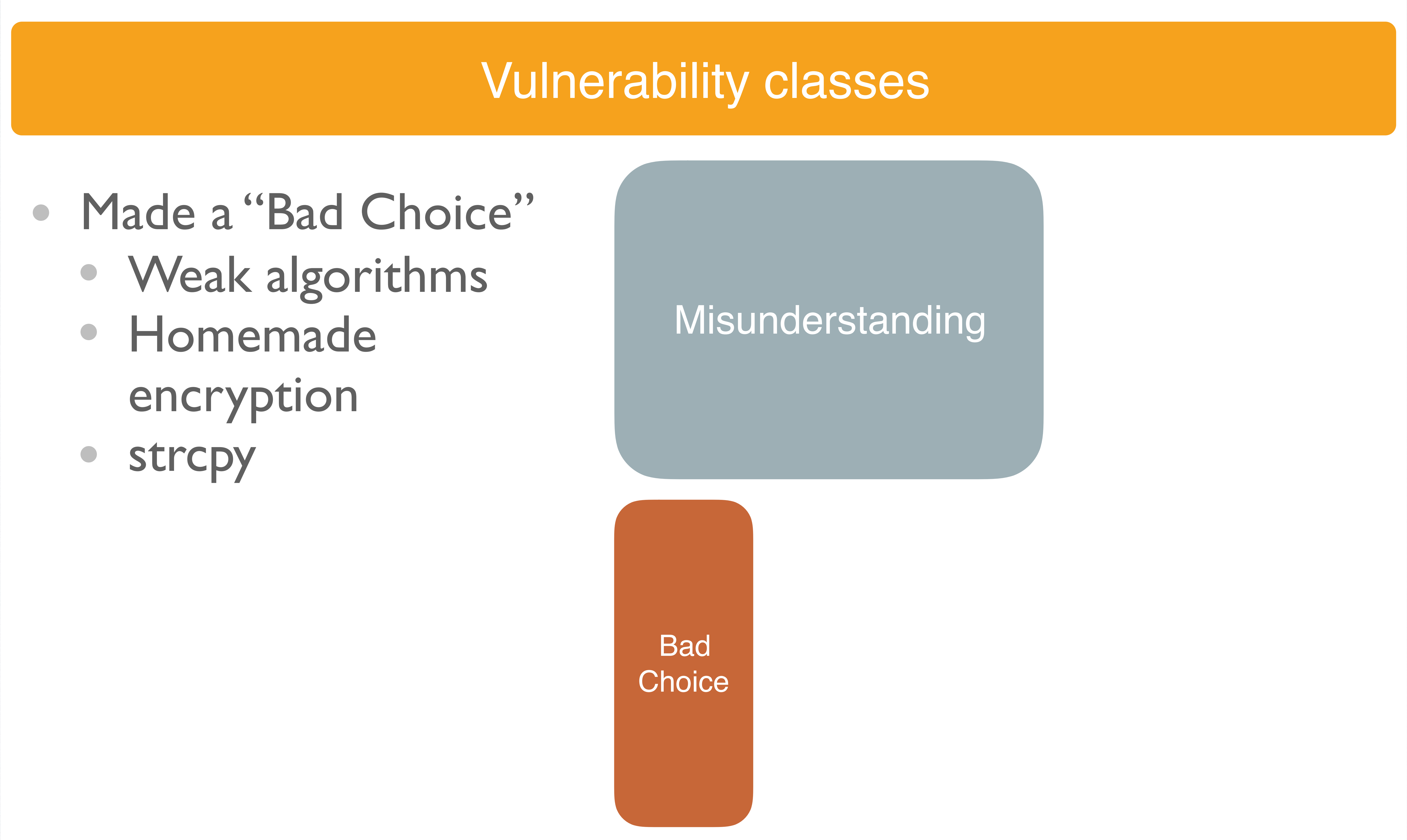
Source: Dan Votipka (https://www.usenix.org/system/files/sec20_slides_votipka-understanding.pdf)

Vulnerability classes



- Missed something "Intuitive"
  - No encryption
  - No access control
- Missed something "Unintuitive"
  - No MAC
  - Side-channel leakage
  - No replay prevention

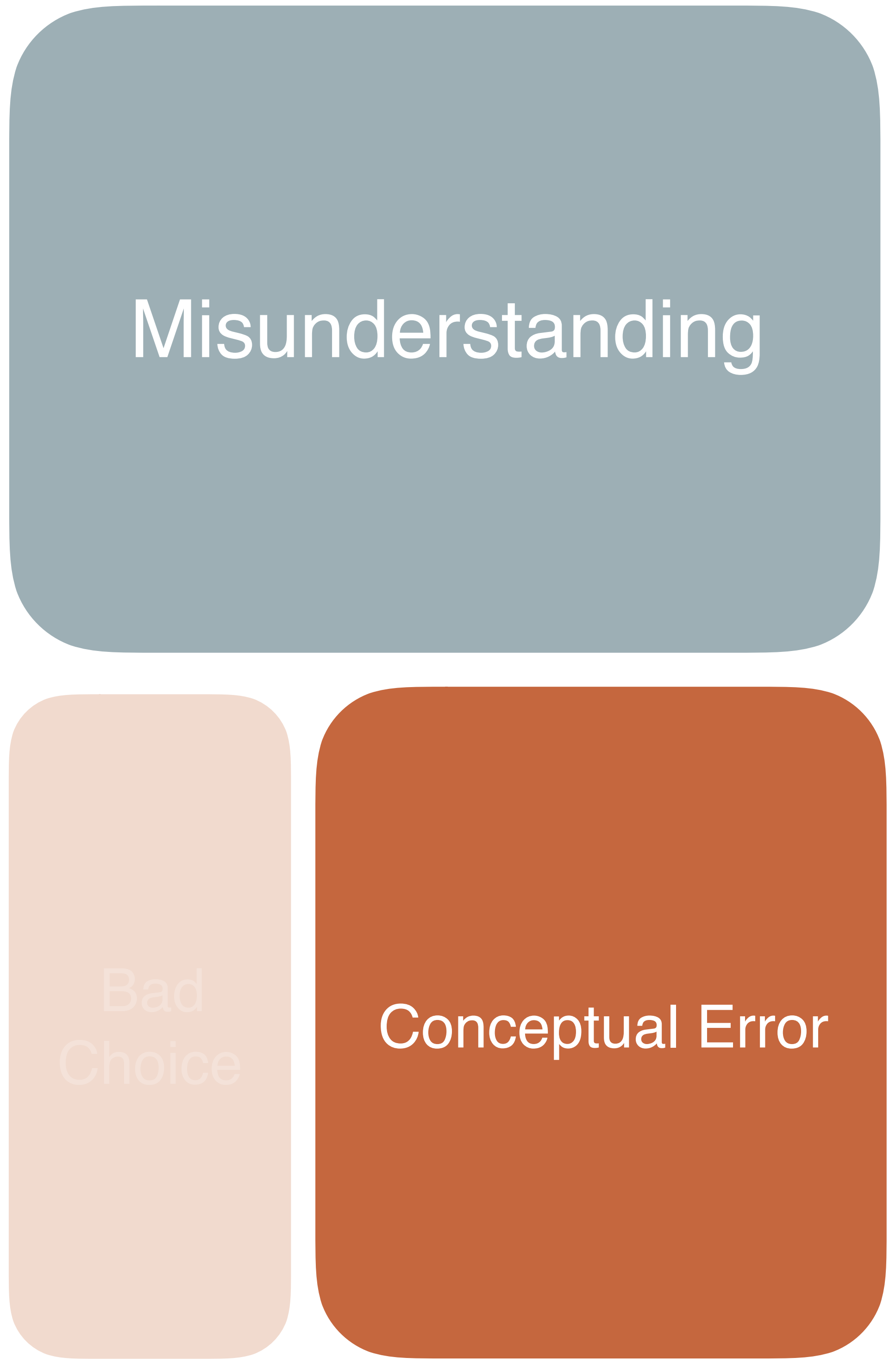
Source: Dan Votipka (https://www.usenix.org/system/files/sec20_slides_votipka-understanding.pdf)



Source: Dan Votipka (https://www.usenix.org/system/files/sec20_slides_votipka-understanding.pdf)

Vulnerability classes

- Made a "Conceptual Error"
  - Fixed value



Source: Dan Votipka (https://www.usenix.org/system/files/sec20_slides_votipka-understanding.pdf)

```
def fillercrypter(sharedkey, text):
    ...
    encryption_suite = AES.new(sharedkey,
      AES.MODE_CBC, 'This is an IV456')
    ...
```

StackOverflow answer uses a fixed initialization vector (should be randomly generated for each message)

8

From this site I have this code snippet:

```
>>> from Crypto.Cipher import AES
>>> obj = AES.new('This is a key123', AES.MODE_CBC, 'This is an IV456')
>>> message = "The answer is no"
>>> ciphertext = obj.encrypt(message)
>>> list(bytearray(ciphertext))
[214, 131, 141, 100, 33, 86, 84, 146, 170, 96, 65, 5, 224, 155, 139, 241]
```

asked 5 years, 2 months ago

viewed 2,188 times

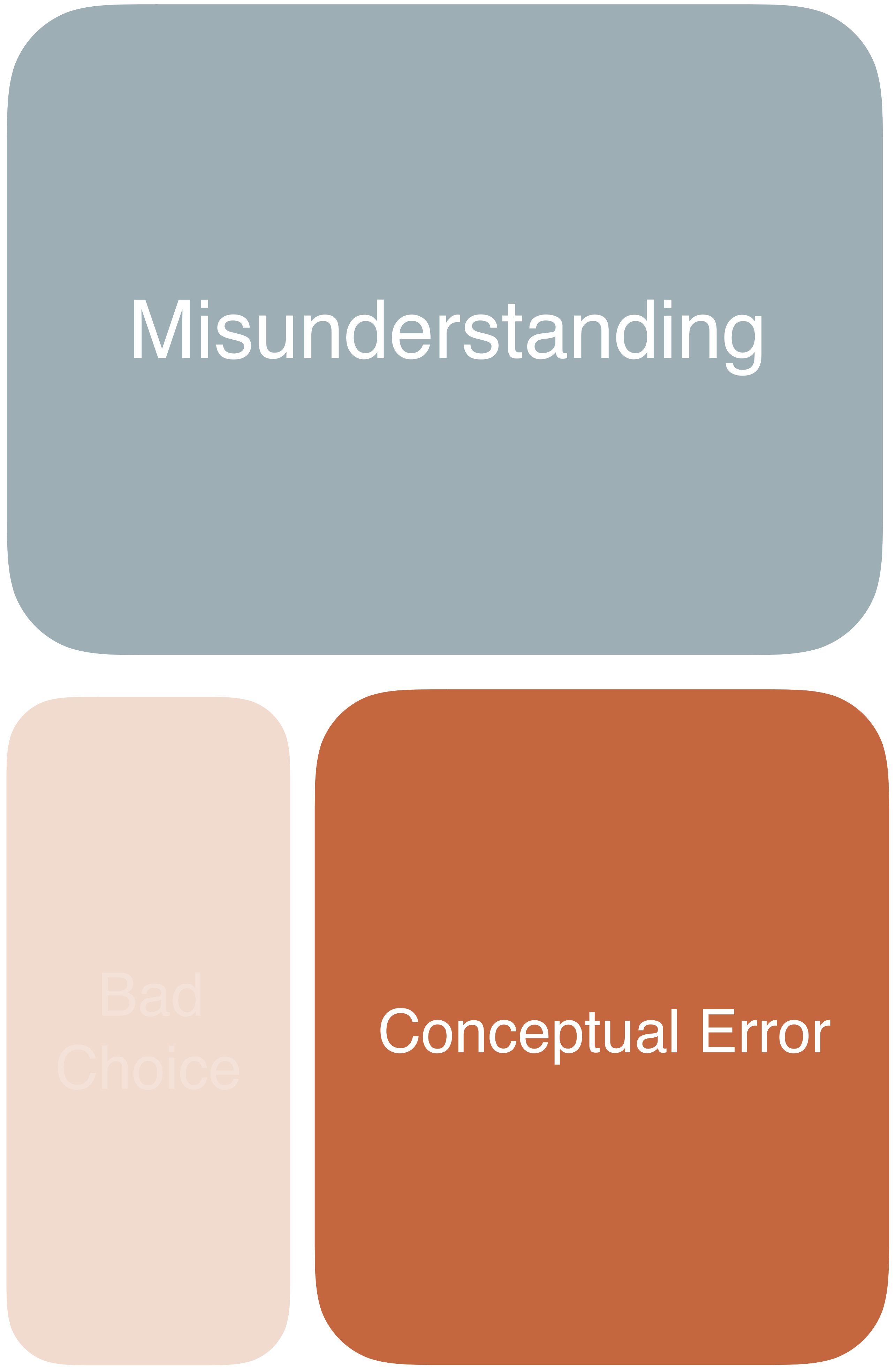
active 5 years, 2 months ago

BLOG

Source: Dan Votipka (https://www.usenix.org/system/files/sec20_slides_votipka-understanding.pdf)

Vulnerability classes

- Made a “Conceptual Error”
  - Fixed value
  - Lacking sufficient randomness
  - Disabling protections in library



Source: Dan Votipka (https://www.usenix.org/system/files/sec20_slides_votipka-understanding.pdf)

Vulnerability classes

- Made a "Mistake"
  - Control flow mistake
  - Skipped algorithmic step

Mistake

Source: Dan Votipka (https://www.usenix.org/system/files/sec20_slides_votipka-understanding.pdf)

PREVALENCE

50%
No implementation

56%
Misunderstanding

21%
Mistake

16%
Intuitive

45%
Unintuitive

21%
Bad Choice

44%
Conceptual Error

Source: Dan Votipka (https://www.usenix.org/system/files/sec20_slides_votipka-understanding.pdf)

RECOMMENDATIONS

- Simplify API design
  - Build in security primitives and focus on common use-cases
- Indicate security impact of non-default use in API Documentation
  - Explain the negative effects of turning off certain things
- Vulnerability Analysis Tools
  - More emphasis on design-level conceptual issues

Source: Dan Votipka (https://www.usenix.org/system/files/sec20_slides_votipka-understanding.pdf)

So, Now What?

- Language matters a lot
- But most vulnerabilities transcend language design!
- What are we to do?

Information Flow Control

- Assumption: there's high-security data and low-security data
- Noninterference: high-security data shouldn't affect low-security outputs

```
String@public f (String@secret password) {
   // ERROR: can't send secret data to public sink
   return password + "A"
}
```

Implicit Flows

```
int@public f (int@secret x) {
   if (x == 0)
      return 0; // ERROR: implicit leak
   else
      return 1;
}
```

Conclusion

- Languages matter for security!
  - But maybe not as much as other factors
  - Unless we expand our ideas about what languages can do!