PAVUDI:
Patch-based vulnerability discovery using Machine Learning

Tom Ganz¹, Erik Imgrund¹, Martin Härterich¹, Konrad Rieck²
AISec ’23

¹ SAP Security Research
² TU Berlin
Introduction
Vulnerability Discovery

- Classical Static Vulnerability Detection
  - Manually crafted rules
  - Often high false positive rate
  - For example
    - Flawfinder, CPPCheck
    - Coverity, Clang Analyzer

- Definition of a Vulnerability Detector

  A method for static vulnerability discovery is a decision function \( f: x \mapsto P(\text{vuln}|x) \) that maps a piece of code \( x \) to its probability of being vulnerable.
Learning-based Vulnerability Discovery

- Learning-based Static Vulnerability Detection
  - Learns rules
  - Requires dataset
  - Adjustable threshold
  - Representation learning

- Definition of a Learning-based Vulnerability Detector

A static **learning-based vulnerability discovery method** is a parametrized hypothesis function $f_\theta : x \rightarrow P(vuln|x)$ that extracts a representation $x$ and maps it to a probability of being vulnerable.
Problem Setting

- Apply vulnerability detector on each patch (CI/CD)
- Problems with patches:
  - Context-sensitive changes
  - Non-coherent changes
  - Evolution of Software

- Example: Heartbleed Bug
- Commit introducing the bug:
  - Touches 12 Files
  - 5 Header Files
  - In 2 different packages

```c
if (hbtype == TLS1_HB_REQUEST)
{
    unsigned char *buffer, *bp;
    int r;

    /* Allocate memory for the response, size is 1 byte
    * message type, plus 2 bytes payload length, plus
    * payload, plus padding
    */
    buffer = OPENSSL_malloc(1 + 2 + payload + padding);
    bp = buffer;

    /* Enter response type, length and copy payload */
    *bp++ = TLS1_HB_RESPONSE;
    s2n(payload, bp);
    memcpy(bp, pl, payload);
```
Naive Solution

Use Existing Learning-based Discovery Methods:
- Feed them Inputs with Patch Context
- Problem: Feature Space explodes

Better Idea:
- Identify security relevant Paths
- Only consider those intersecting Changes
Methodology
Representation

1. Obtain composite code graph
2. Insert call edges
3. Insert interprocedural data flow
4. Perform value-set analysis
5. Create security-relevant slices
Causal Graph Neural Network

- Graph separated into Artifacts and causal Subgraph
- Separation learned by network
- Prediction only on causal Subgraph
Training dataset

- Previous datasets contain only vulnerability-fixing patches
- We try to find vulnerability-introducing patches
  - Very difficult to collect
- Instead: Find patches that touch vulnerable code
  - From vulnerability-fixing patches, go back in time
  - Patches on same methods are vulnerable
  - Patches on other methods are assumed to be clean
Experiments
Research Questions

- **RQ1** How do other strategies compare to PAVUDI?
- **RQ2** How does the size of a commit affect the performance?
- **RQ3** How does PAVUDI behave after training and deployment?
- **RQ4** How do the individual components of PAVUDI contribute to the detection capability?
Model Baselines

- Learning-based Graph Vulnerability Detectors
  - DeepWuKong
  - ReVeal
  - Devign
  - BGNN4VD
- Learning-based Token Vulnerability Detectors
  - SySeVR
  - VulDeePecker
- Heuristics-based Vulnerability Detector
  - VUDDY

Not Applicable to Patches!
Application Strategies

Apply Models to Fragments of the Patch and aggregate prediction score

- Max
- Mean
- Probability
- Isotonic
- Commit
RQ1 How do other strategies compare to PAVUDI?

**FFmpeg**

![Graph for FFmpeg]

**QEMU**

![Graph for QEMU]

Legend:
- ▼ TaintGraph
- ▲ DeepWukong Proba
- ○ DeepWukong Max
- ● DeepWukong Isotonic
- ✗ DeepWukong Mean
- ▼ Vuddy Commit
- ● Vuddy Max
RQ1 How do other strategies compare to PAVUDI?

**FFmpeg**

**QEMU**
RQ1 How do other strategies compare to PAVUDI?

**FFmpeg**

**QEMU**
RQ2 How does the size of a commit affect the performance?

**FFmpeg**

![Graph for FFmpeg](image1)

**QEMU**

![Graph for QEMU](image2)

Legend:
- TaintGraph
- ReVeal
- Vuddy
- VulDeepecker
- Devign
- BGNN4VD
- DeepWukong
- SySeVR
RQ3 How does PAVUDI behave after training and deployment?

**FFmpeg**

- Devign
- BGNN4VD
- ReVeal
- TaintGraph
- Deepwukong
- VulDeepecker

**QEMU**

- Devign
- BGNN4VD
- ReVeal
- TaintGraph
- Deepwukong
- SySEVR

![Graphs showing F1-Score over Days since Training for FFmpeg and QEMU](image-url)
How do individual components of PAVUDI contribute to its capabilities?
Conclusion

Not Attending: Martin Härterich, Konrad Rieck
Conclusion

- Patches are the atomic unit of modern software development
- Existing vulnerability detectors are badly suited to patches
- Identified five previously undisclosed bugs
- We introduce a patch-based vulnerability discovery (PAVUDI)
  - With a new interprocedural code representation
  - An explainable graph neural network
- Our solution
  - has more than 50% increased detection performance
  - is twice as robust against concept drift
- Public Implementation: https://github.com/SAP-samples/security-research-taintgraphs
Thank you.

Corresponding Author:

Tom Ganz
tom.ganz@sap.com