

# *Kelpie: A Black-Box, Zero Query, Mimicry Attack on ML-based Binary Function Classifiers*

Victor Matrat (1), Gabriel Sauger (1), Jean-Yves Marion (1),  
Sazzadur Rahaman (2), Vincent Tourneur (1), Muaz Ali (2).

(1) Université de Lorraine, Loria, Nancy, France;  
(2) University of Arizona, Tucson, USA

September 2025

# Supply chain attack powered by targeted adversarial evasion

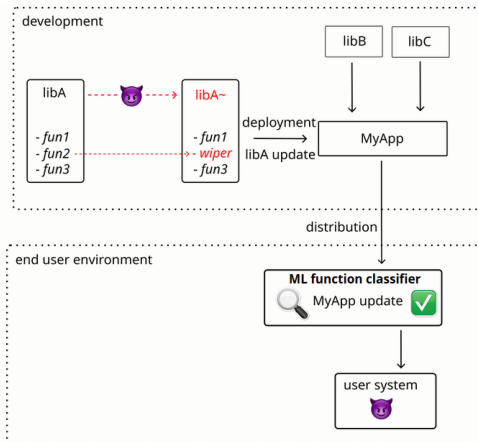


Figure: Infecting end user's environment through a supply chain attack

# A classical targeted evasion attack

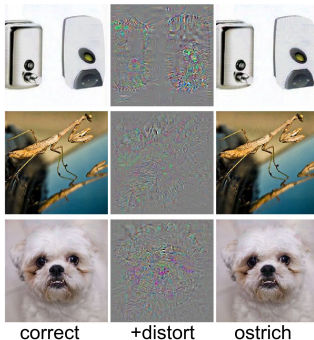
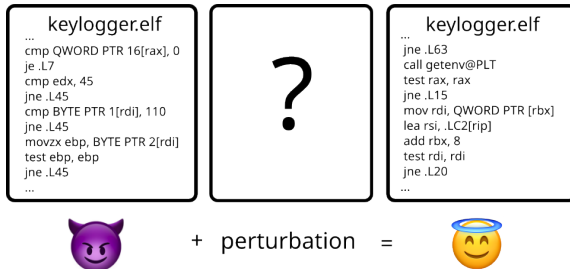


Figure: [Intriguing properties of neural networks, Szegedy et al., 2013]

Note: This is a targeted evasion attack, which is different than simple obfuscation.

# Our domain → Binary function classification



**Figure:** Modifying the payload for 'benign' classification

Perturbation is more difficult: payload functionality must be conserved, which reduces our possible actions.

# Our domain → Binary function classification

## Defenders

Classifiers using ML techniques on static analysis features:

- Graph-based features (ex CFG nodes and edges)
- Assembly code features (ex sequences of instructions)

# Evasion attack landscape

## Attacker threat model for ML evasion attacks

- White-Box
- Transfer attacks
- Black-Box (Kelpie)

# Evasion attack landscape

## Attacker threat model for ML evasion attacks

- White-Box
- Transfer attacks
- Black-Box (Kelpie)

## Black-box evasion attacks

- **Multiple queries:** building iteratively an adversarial input.
- **Zero query:** performing a one-time perturbation. (Kelpie)

# Kelpie framework

## Kelpie capabilities

Kelpie is a **black-box, zero query, targeted mimicry** attack framework:

- Aims to conceal a function (*payload*) by imitating the characteristics of a target.
- Performs perturbation at the source code and assembly level.
- No access to the classifier architecture or training dataset.

**Novel approach:** we not only want to hide the payload, but also deceive the classifier into recognizing the target.



# Kelpie Workflow

How do we perform mimicry attacks?

# Kelpie Workflow

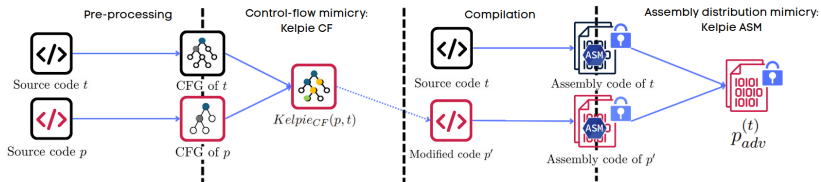
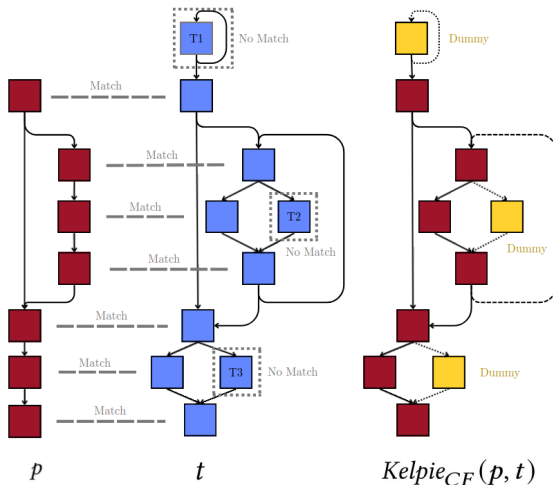


Figure: 4 steps workflow to produce the modified payload  $p_{adv}^{(t)}$

# Kelpie CF: Control-Flow mimicry



# Kelpie Workflow

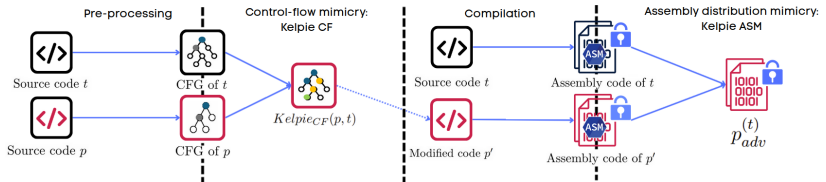


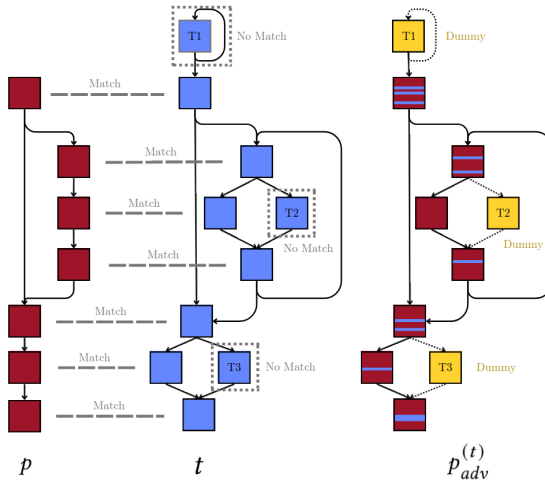
Figure: 4 steps workflow to produce the modified payload  $p_{adv}^{(t)}$

# Kelpie ASM: Assembly features mimicry

## Twofold strategy

- **Dead blocks** (added by KelpieCF): Copy and paste the instructions from the corresponding target block.
- **Alive blocks**: Perform a liveness analysis to identify safe insertions spaces, and insert instructions to imitate the operands distribution of the target.

# Kelpie ASM: Assembly features mimicry



# Kelpie Workflow

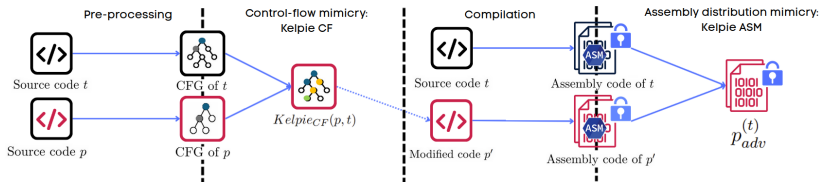


Figure: 4 steps workflow to produce the modified payload  $p_{adv}^{(t)}$

# Research Question 1

Can *Kelpie* deceive the state-of-the-art classifiers?



# RQ1 - Can *Kelpie* deceive the SotA classifiers?

## Selected models

5 top-performing classifiers identified by Marcelli *et al.*<sup>1</sup>: *Asm2vec*, *GGSNN*, *GMN*, *Trex* and *Zeek*. 3 more recent models: *JTrans*, *HermesSim*, *CLAP* (best performances and novel features).

---

<sup>1</sup>marcelli\_ how \_2022.

# RQ1 - Can *Kelpie* deceive the SotA classifiers?

## Selected models

5 top-performing classifiers identified by Marcelli *et al.*<sup>1</sup>: *Asm2vec*, *GGSNN*, *GMN*, *Trex* and *Zeek*. 3 more recent models: *JTrans*, *HermesSim*, *CLAP* (best performances and novel features).

## Function dataset

13 of the most starred publicly available C repositories from GitHub (ex: *redis*, *git*, *hashcat*, *darknet*).

---

<sup>1</sup>marcelli\_\_how\_2022.

# RQ1 - Can *Kelpie* deceit the SotA classifiers?

## Tests

- **Classification** task (New metric: Mimicry Attack Success Rate (MASR))
- **Retrieval** task

# Classification Task

A binary function classifier is represented as function  $C$ , which takes as input a pair of function and return a similarity score between 0 and 1.

$$\mathcal{F} \times \mathcal{F} \longrightarrow [0, 1]$$

$$(f, g) \longmapsto C(f, g)$$

We define a threshold  $\sigma$  through ROC curves analysis to transform the continuous score into a binary decision:

$$C(f, g) = \begin{cases} \text{True}, & C(f, g) \geq \sigma, \\ \text{False}, & C(f, g) < \sigma. \end{cases}$$

# Classification Task

For a classifier  $\mathcal{C}$ , given a payload (p), a target (t), and a modified payload ( $p_{adv}^{(t)}$ ), we have 4 possible outcomes:

| Case | $\mathcal{C}(p_{adv}^{(t)}, p)$ | $\mathcal{C}(p_{adv}^{(t)}, t)$ | Outcome    | Interpretation  |
|------|---------------------------------|---------------------------------|------------|---|
| 1    | True                            | False                           | Worst case | Modified payload still linked to payload; target rejected |
| 2    | False                           | False                           | Evasion    | Payload concealed; target rejected                        |
| 3    | True                            | True                            | Mimicry    | Payload and target both predicted similar                 |
| 4    | False                           | True                            | Ideal      | Payload concealed; target accepted                        |

$\mathcal{C}(x, y) = \text{True}$  indicates the classifier predicts  $x$  and  $y$  are similar.

# Mimicry Attack Success Rate (MASR)

Given a set of perturbed payloads  $\mathcal{P}$ , and a classifier  $\mathcal{C}$  where " $\mathcal{C}(p_{adv}^{(t)}, t) = \text{True}$ " means the classifier predicts that  $p_{adv}^{(t)}$  and  $t$  are similar, the MASR is defined as follows:

# Mimicry Attack Success Rate (MASR)

Given a set of perturbed payloads  $\mathcal{P}$ , and a classifier  $\mathcal{C}$  where " $\mathcal{C}(p_{adv}^{(t)}, t) = \text{True}$ " means the classifier predicts that  $p_{adv}^{(t)}$  and  $t$  are similar, the MASR is defined as follows:

$$MASR = \frac{|\{p_{adv}^{(t)} \mid \mathcal{C}(p_{adv}^{(t)}, t) = \text{True}\}|}{|\mathcal{P}|}$$

We want the modified payload and the target to be predicted as similar (False positive), which is a success for Mimicry.

# Mimicry Attack Success Rate (MASR)

Given a set of perturbed payloads  $\mathcal{P}$ , and a classifier  $\mathcal{C}$  where " $\mathcal{C}(p_{adv}^{(t)}, t) = \text{True}$ " means the classifier predicts that  $p_{adv}^{(t)}$  and  $t$  are similar, the MASR is defined as follows:

$$MASR = \frac{|\{p_{adv}^{(t)} \mid \mathcal{C}(p_{adv}^{(t)}, t) = \text{Similaire} \wedge \mathcal{C}(p_{adv}^{(t)}, p) = \text{Différent}\}|}{|\mathcal{P}|}$$

We also want the payload and the modified payload to be predicted as different (False Negative), which is a success for Evasion.



# Mimicry Attack Success Rate (MASR)

Given a set of perturbed payloads  $\mathcal{P}$ , and a classifier  $\mathcal{C}$  where " $\mathcal{C}(p_{adv}^{(t)}, t) = \text{True}$ " means the classifier predicts that  $p_{adv}^{(t)}$  and  $t$  are similar, the MASR is defined as follows:

$$MASR = \frac{|\{p_{adv}^{(t)} \mid \mathcal{C}(p_{adv}^{(t)}, t) = \text{True} \wedge \mathcal{C}(p_{adv}^{(t)}, p) = \text{False}\}|}{|\mathcal{P}|}$$

## Understanding the MASR score

The MASR is a score between 0 and 1, where 0.25 means that 25% of the attacks succeed.

# MASR Results (Classification Task)

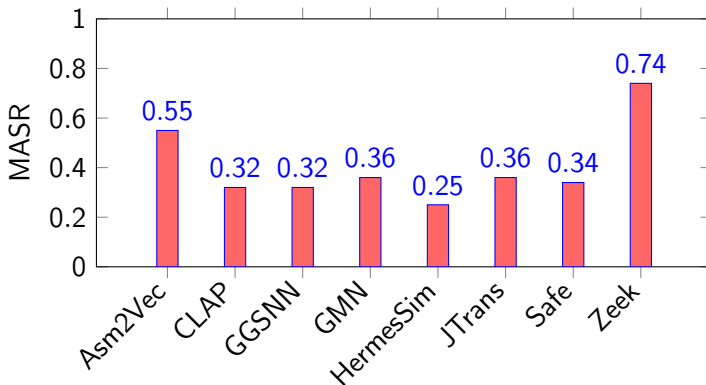


Figure: MASR in classification task after Kelpie perturbation

# RQ1 - Conclusions

## RQ1 takeaways

Our experiments show that Kelpie produces mimicry attacks which succeed in **25% to 74%** of cases. Kelpie consistently undermines the performance of state-of-the-art classifiers, exposing a **critical weakness in their design**.

## Research Question 2

Can *Kelpie* imitate a patched function while conserving a vulnerability?

## RQ2 - Can *Kelpie* imitate a patched function?

### Dataset

We use a dataset of 60 vulnerable functions from FFmpeg, Tcpcdump, and OpenSSL, with CVSS scores between 7.5 and 9.8, enabling remote access control, denial-of-service, *etc.*

**Example:** CVE-2016-10190 (FFmpeg heap buffer overflow).

## RQ2 - Can *Kelpie* imitate a patched function?

### Dataset

We use a dataset of 60 vulnerable functions from FFmpeg, Tcpdump, and OpenSSL, with CVSS scores between 7.5 and 9.8, enabling remote access control, denial-of-service, *etc.*

**Example:** CVE-2016-10190 (FFmpeg heap buffer overflow).

### Methodology:

- We choose a vulnerable function, the payload.
- We choose a patched version of the function, the target.
- We use Kelpie to produce the modified payload, imitating the patch while conserving the vulnerability.

**Note:** The patched version can be very similar to the vulnerable version of the function.

## RQ2 - Can *Kelpie* imitate a patched function?

| Model            | MASR Baseline | MASR Kelpie |
|------------------|---------------|-------------|
| <i>Asm2Vec</i>   | 0.00          | 0.26        |
| <i>CLAP</i>      | 0.00          | <b>0.24</b> |
| <i>GGNN</i>      | 0.00          | 0.28        |
| <i>GMN</i>       | 0.00          | 0.28        |
| <i>HermesSim</i> | 0.00          | 0.29        |
| <i>jTrans</i>    | 0.00          | 0.31        |
| <i>Trex</i>      | 0.00          | <b>0.24</b> |
| <i>Zeek</i>      | 0.00          | <b>0.35</b> |

Table: Kelpie MASR results in vulnerability insertion

### RQ2 takeaways

We have shown that critical vulnerabilities can be inserted **1 in 4 times**, which also demonstrate the necessity of further research into the representation of code at the function scale.

# Conclusion - Kelpie

We have presented *Kelpie*, a framework for binary function evasion attacks.

## *Kelpie* capabilities

- Black-box targeted mimicry attack
- No queries to the classifier
- Code-level perturbations (opposed to header modifications)



# Conclusion - Results

In the previous experiments, we demonstrated that:

- **RQ1:** Classifiers fail to recognize on average **40% of the mimicry attacks**.
- **RQ2:** We are able to insert a vulnerable function, deceiving the classifier into recognizing the patch **at least in 24% of the attacks**.

# Conclusion - Research perspectives

## Defender's perspective

Need of research in the field of binary function classification based on ML, to build models that are able to grasp the semantic of the code, and not only syntactic patterns.

## Attackers's perspective

Improve Kelpie performances in the mimicry task, for example being more precise in the assembly code mimicry and be able to modify sequences of instructions of the payload by semantically sequences which fit the target structure.

# References I



Ding, Steven HH, Benjamin CM Fung, and Philippe Charland. “Asm2vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization”. In: *IEEE Symposium on Security and Privacy*. 2019, pp. 472–489.






He, Haojie et al. “Code is not natural language: Unlock the power of semantics-oriented graph representation for binary code similarity detection”. In: *USENIX Security Symposium*. 2024.






Li, Yujia et al. “Graph matching networks for learning the similarity of graph structured objects”. In: *International Conference on Machine Learning (ICML)*. 2019, pp. 3835–3845.

# References II

-  Marcelli, Andrea et al. “How machine learning is solving the binary function similarity problem”. In: *USENIX Security Symposium*. 2022, pp. 2099–2116.
-  Pei, Kexin et al. “Learning Approximate Execution Semantics From Traces for Binary Function Similarity”. In: *IEEE Transactions on Software Engineering* 49.4 (2023), pp. 2776–2790.
-  Shalev, Noam and Nimrod Partush. “Binary Similarity Detection Using Machine Learning”. In: *Proceedings of the 2018 ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS '18)*. ACM, 2018, pp. 1–6.

## References III

-  Szegedy, Christian et al. “Intriguing properties of neural networks”. In: *International Conference on Learning Representations (ICLR)*. 2014.
-  Wang, Hao et al. “CLAP: Learning Transferable Binary Code Representations with Natural Language Supervision”. In: *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 2024, pp. 503–515.
-  Wang, Hao et al. “jTrans: jump-aware transformer for binary code similarity detection”. In: *ACM SIGSOFT International Symposium on Software Testing and Analysis*. 2022, pp. 1–13.