Misleading Worm Signature Generators Using Deliberate Noise Injection

Roberto Perdisci[+^], David Dagon[^], Wenke Lee[^],
Prahlad Fogla[^], Monirul Sharif[^]

[^]Georgia Institute of Technology, Atlanta, GA, USA

[+]University of Cagliari, ITALY

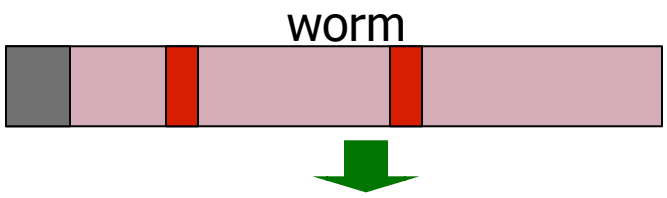
Presented by: Roberto Perdisci

Outline

- Introduction
- Syntactic-based Signature Generators
- “Traffic-based” flow classifiers
- Noise Injection Attack
- Case study: Misleading POLYGRAPH
- Experimental Results
- Conclusion

Introduction

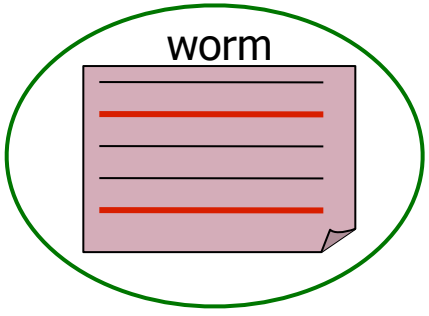
- Automatic signature generation look for invariants in polymorphic worms
  - Syntactic-based:



Packets/flows inspection

GET .* HTTP/1.1 .* Host: .* Host: .* \xFF\xBF .*

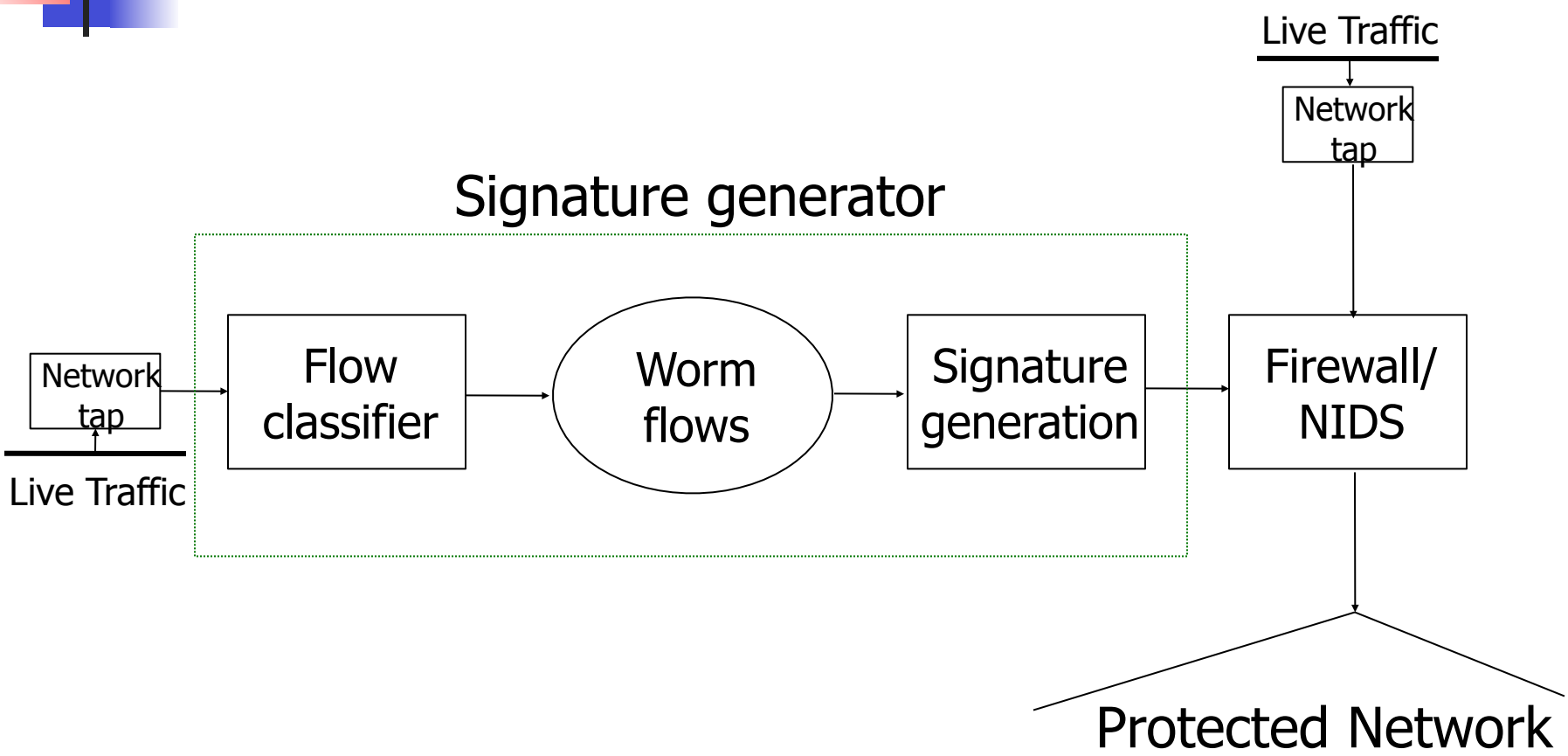
  - Semantic-based:



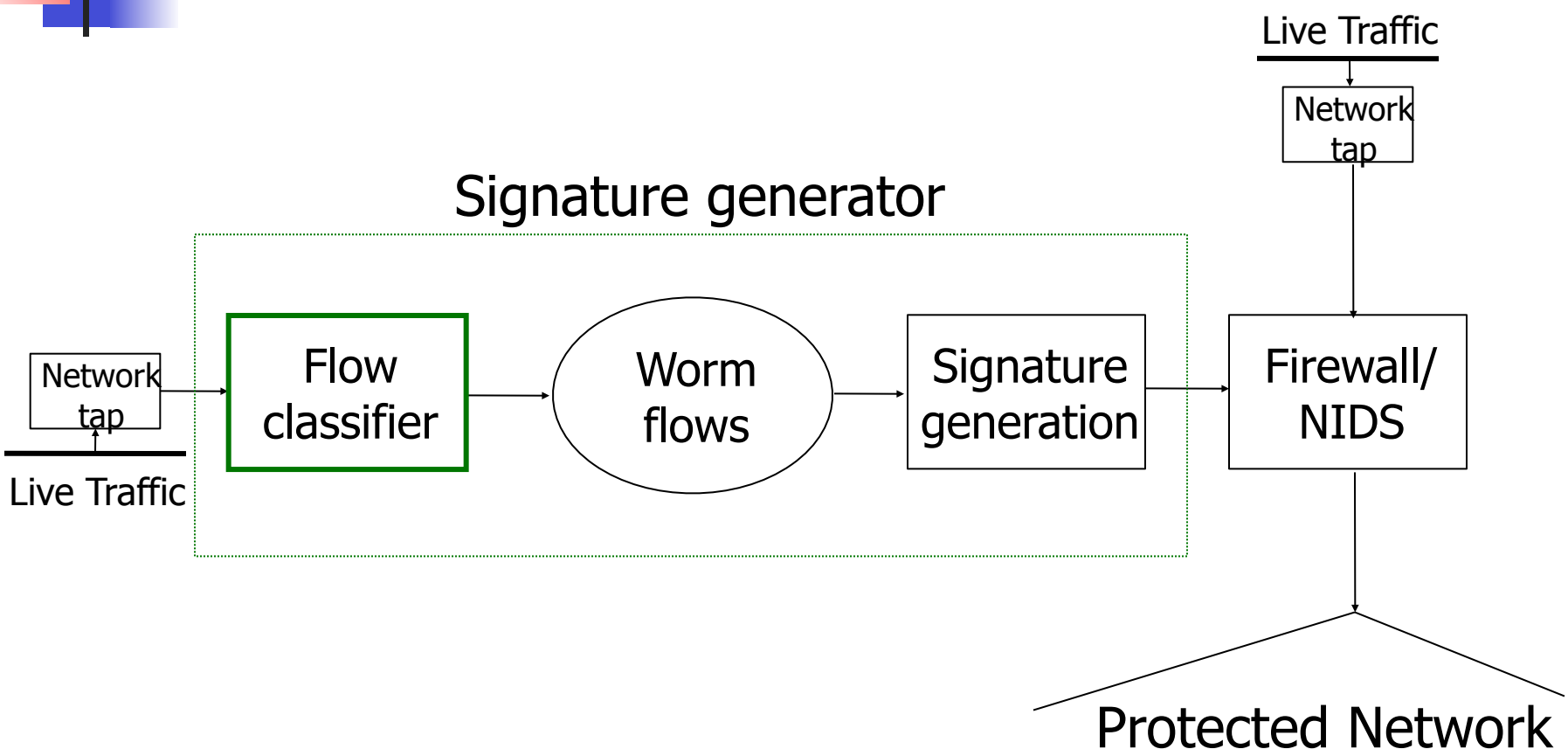
Execution of malicious code in a "protected" environement

- Our contribution: **Syntactic-based** signature generators are **vulnerable to Noise Injection Attack**

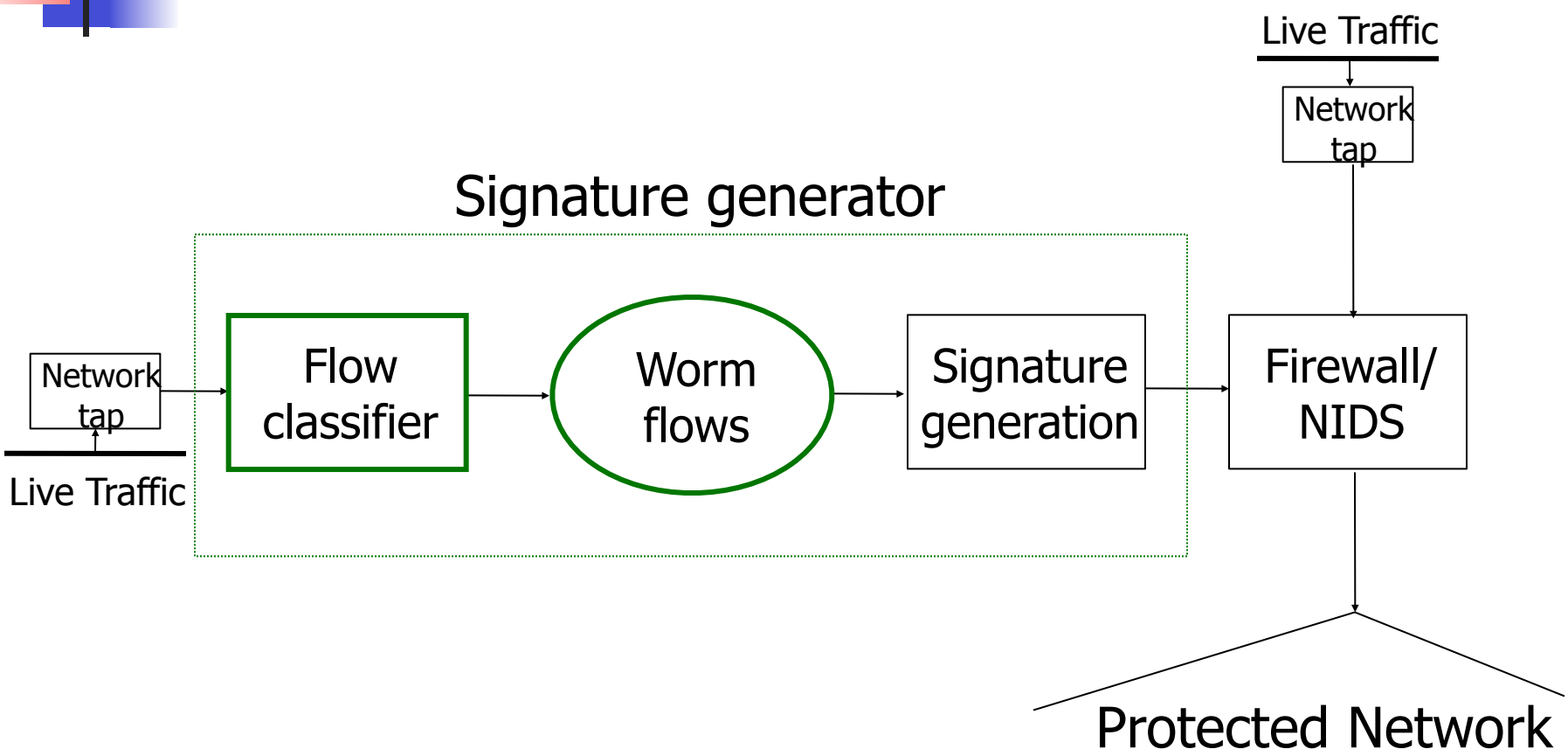
Syntactic-based Signature Generators



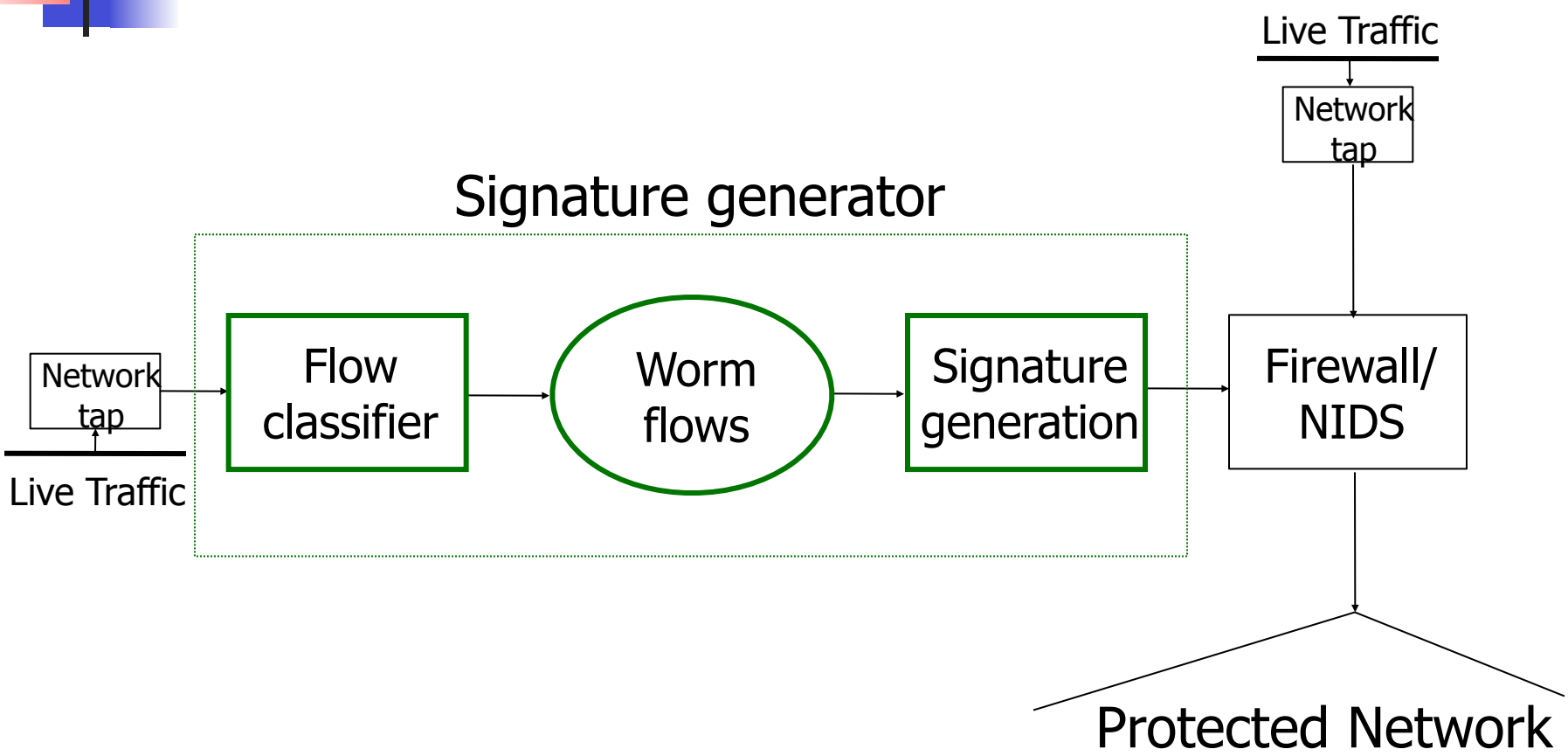
Syntactic-based Signature Generators



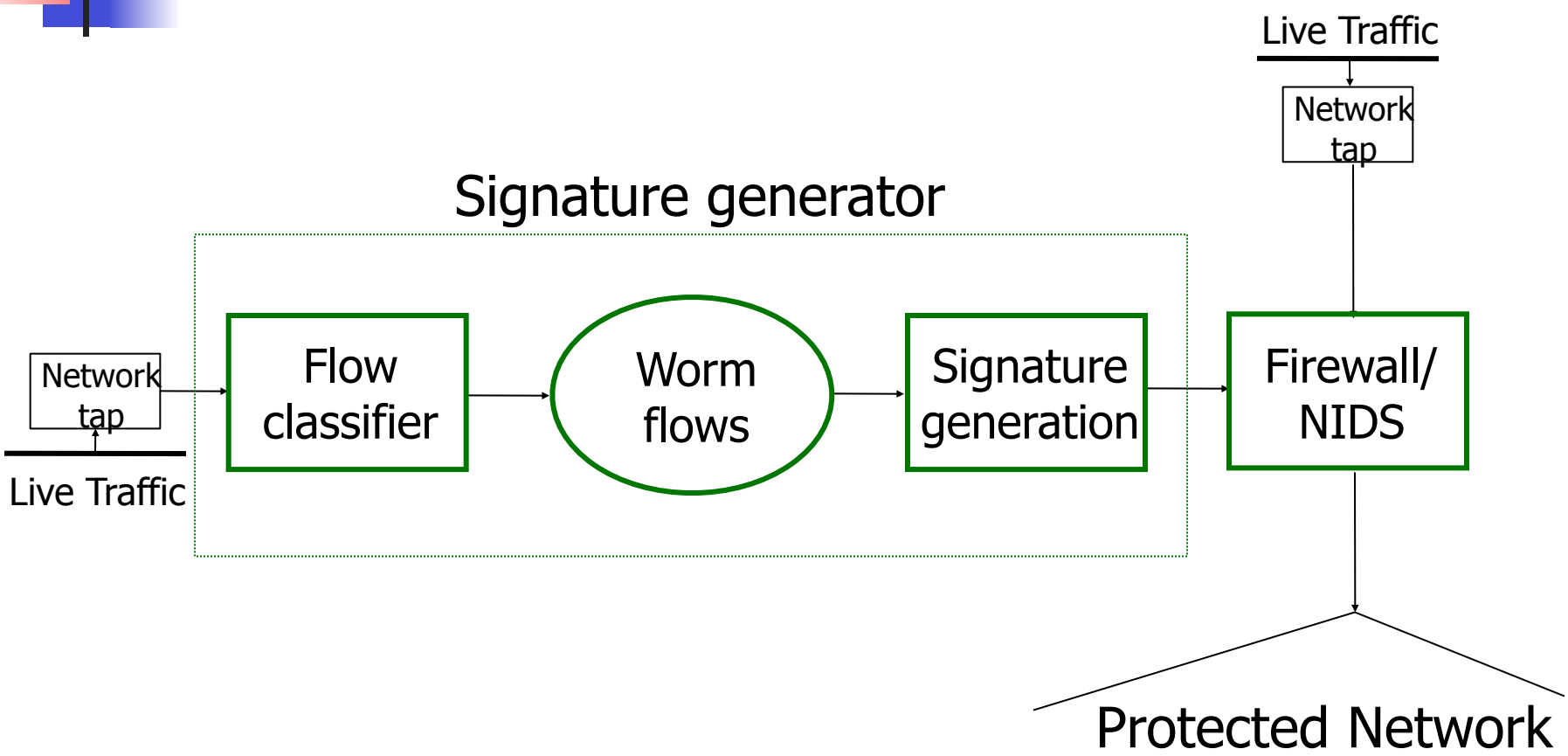
Syntactic-based Signature Generators



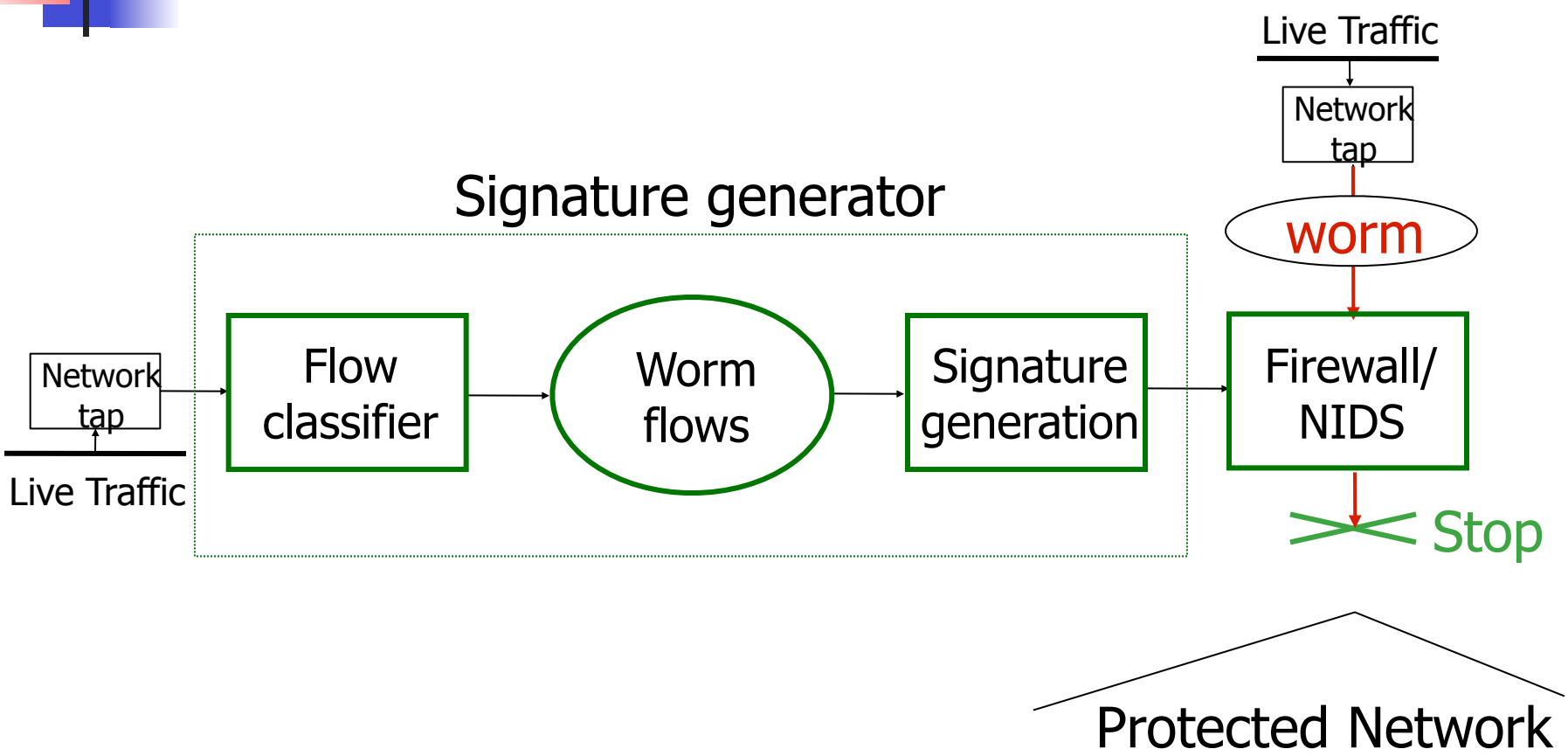
Syntactic-based Signature Generators



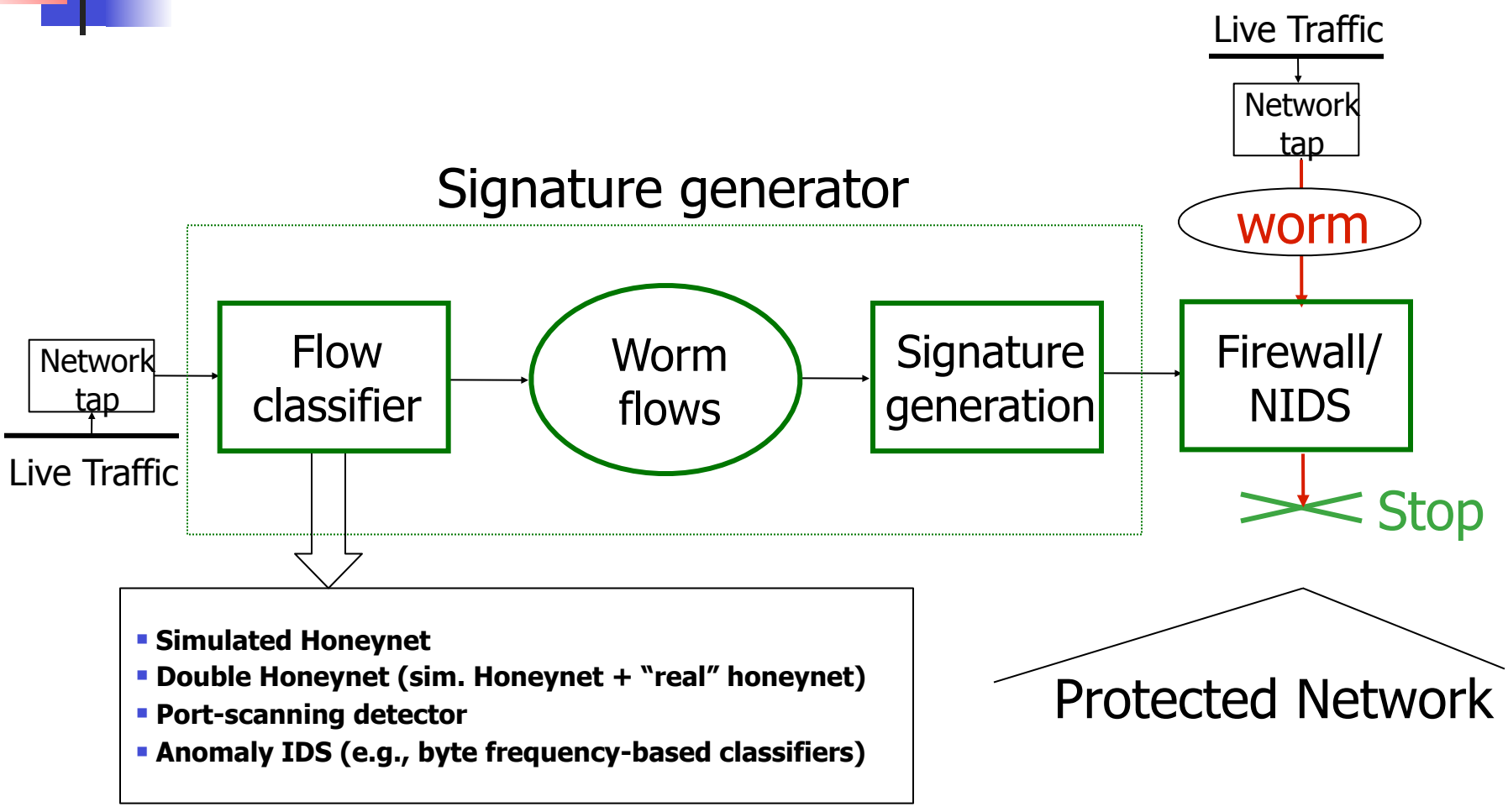
Syntactic-based Signature Generators



Syntactic-based Signature Generators



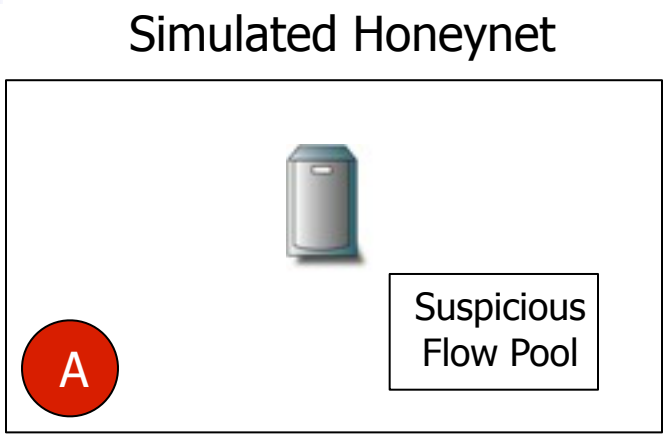
Syntactic-based Signature Generators



- **Simulated Honeynet**
- **Double Honeynet (sim. Honeynet + "real" honeynet)**
- **Port-scanning detector**
- **Anomaly IDS (e.g., byte frequency-based classifiers)**

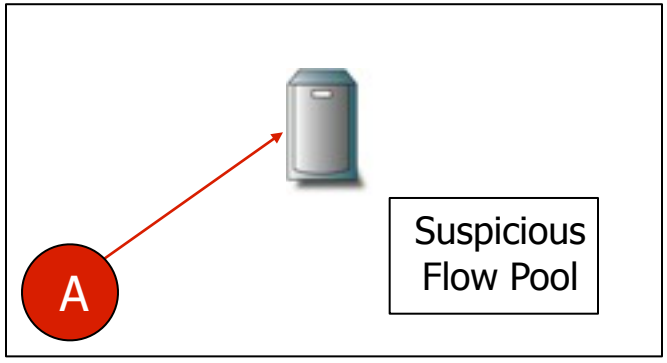
Traffic-based flow classifiers

Traffic-based flow classifiers



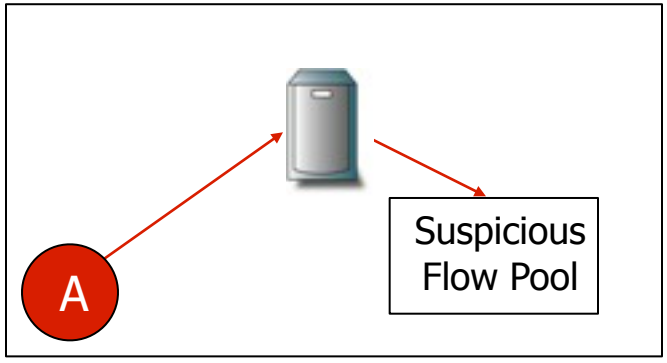
Traffic-based flow classifiers

Simulated Honeynet



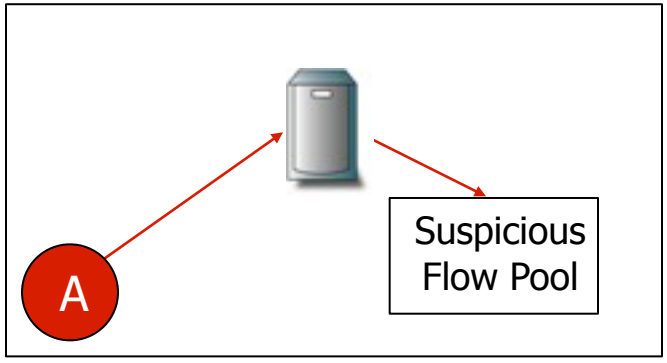
Traffic-based flow classifiers

Simulated Honeynet

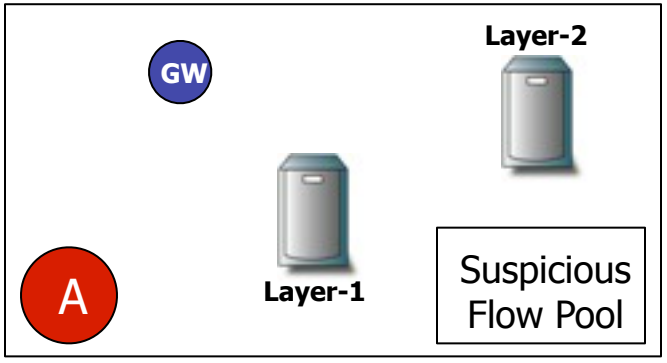


Traffic-based flow classifiers

Simulated Honeynet

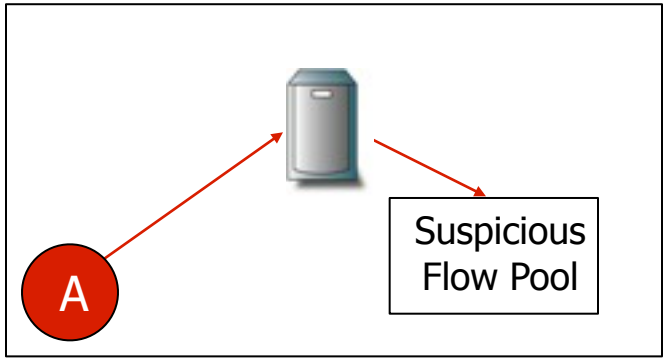


Double Honeynet

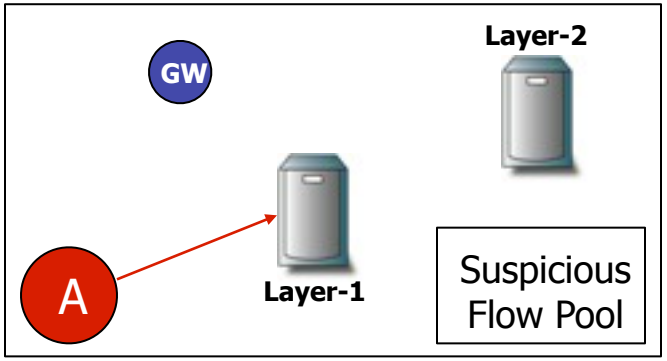


Traffic-based flow classifiers

Simulated Honeynet

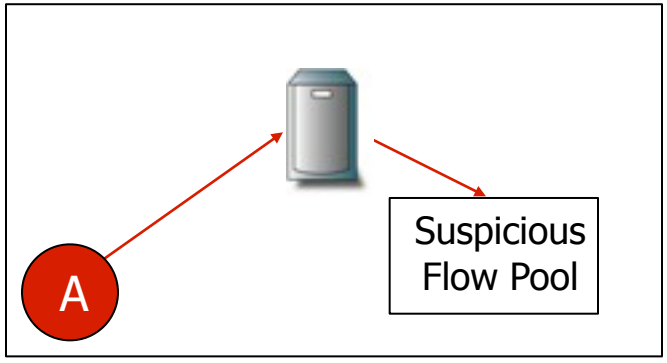


Double Honeynet

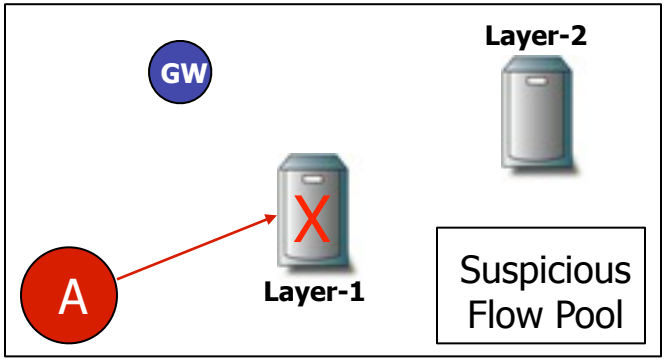


Traffic-based flow classifiers

Simulated Honeynet

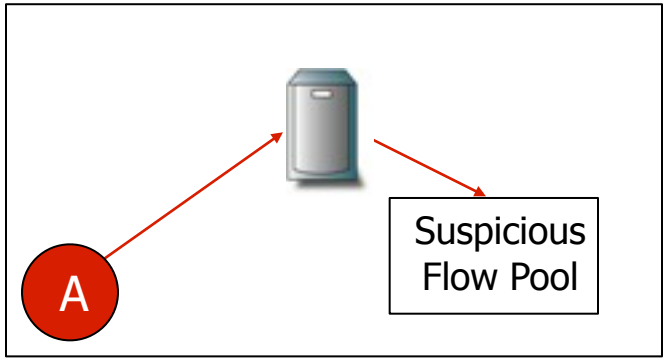


Double Honeynet

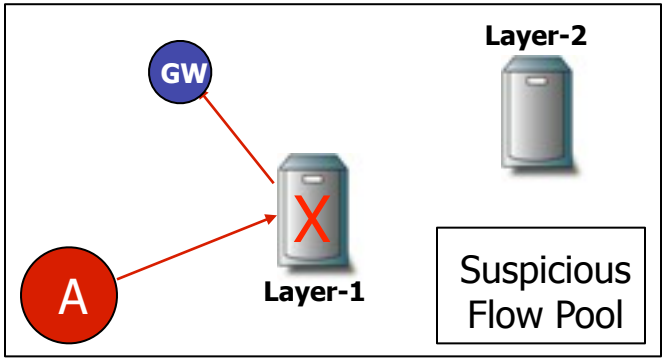


Traffic-based flow classifiers

Simulated Honeynet

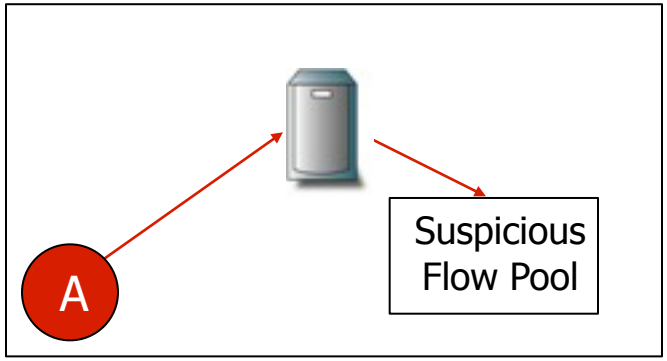


Double Honeynet

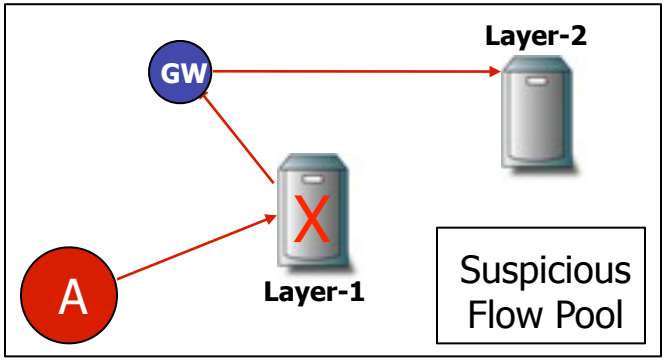


Traffic-based flow classifiers

Simulated Honeynet

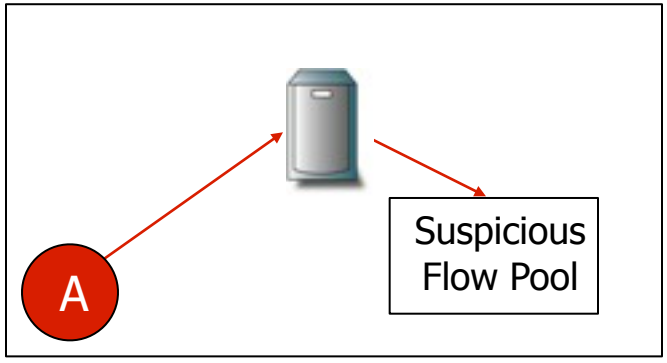


Double Honeynet

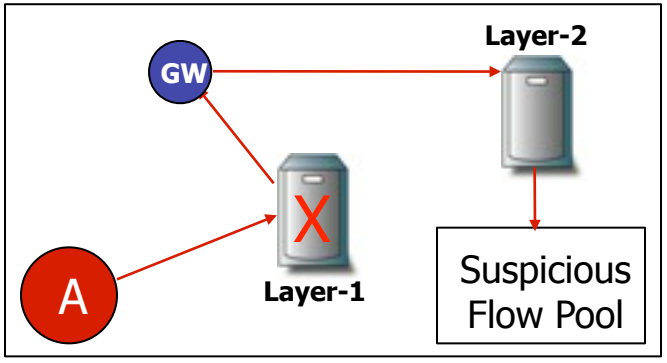


Traffic-based flow classifiers

Simulated Honeynet

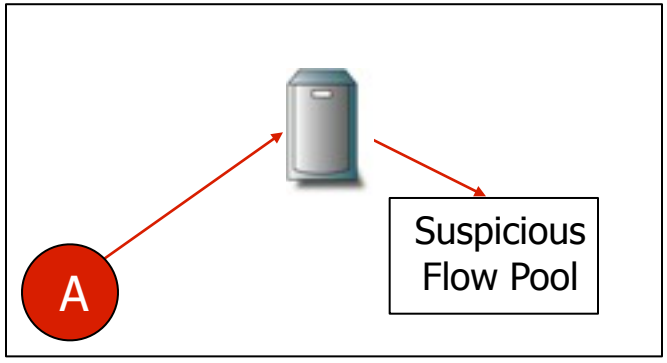


Double Honeynet

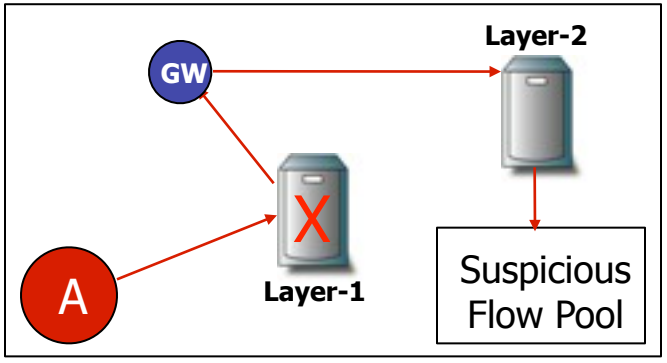


Traffic-based flow classifiers

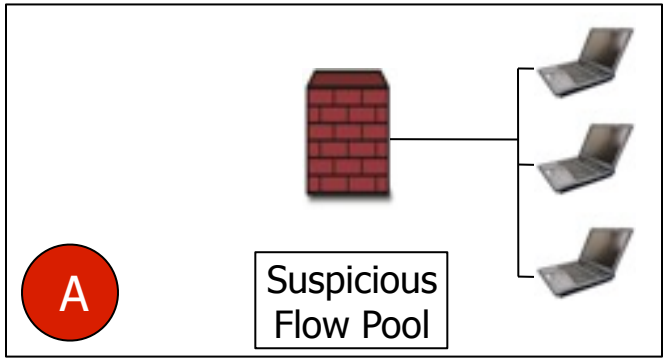
Simulated Honeynet



Double Honeynet

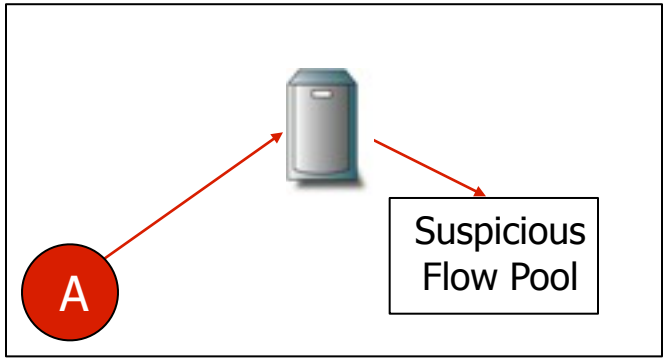


Port-scanning detection

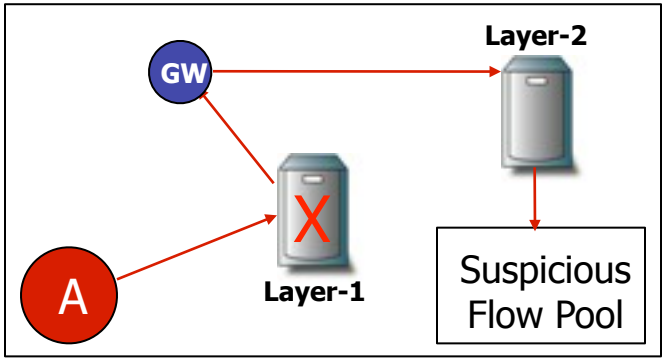


Traffic-based flow classifiers

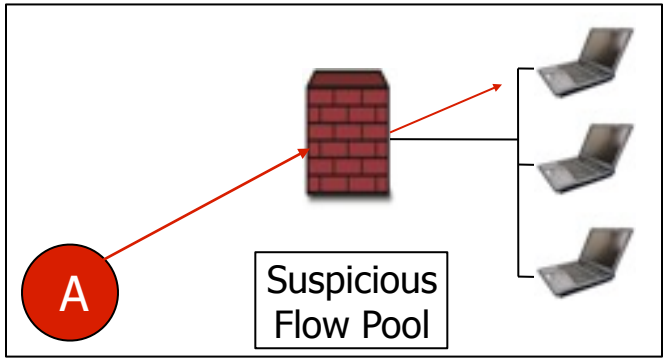
Simulated Honeynet



Double Honeynet

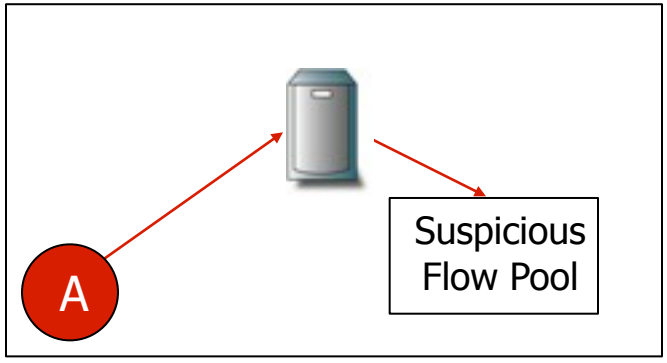


Port-scanning detection

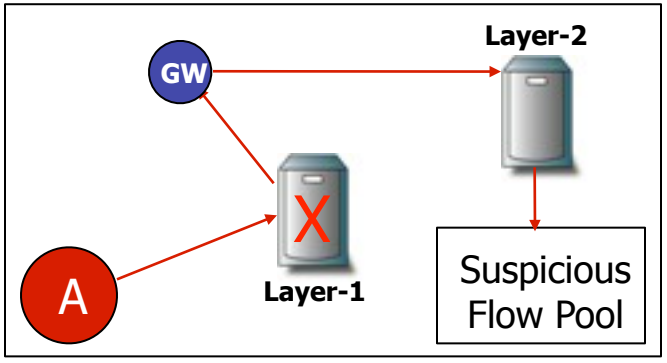


Traffic-based flow classifiers

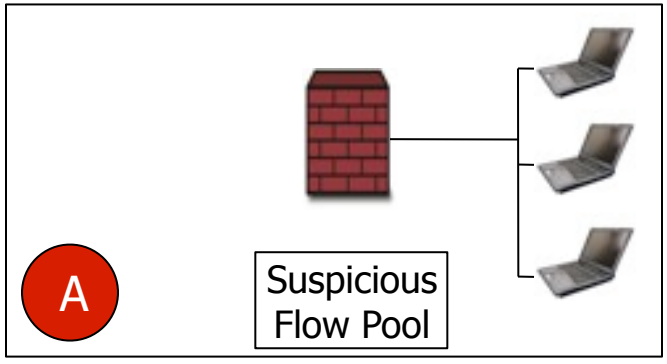
Simulated Honeynet



Double Honeynet

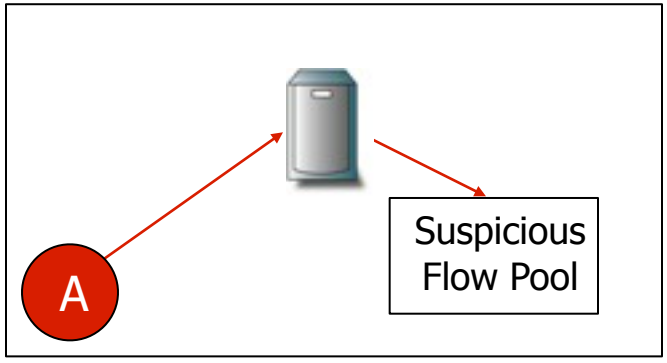


Port-scanning detection

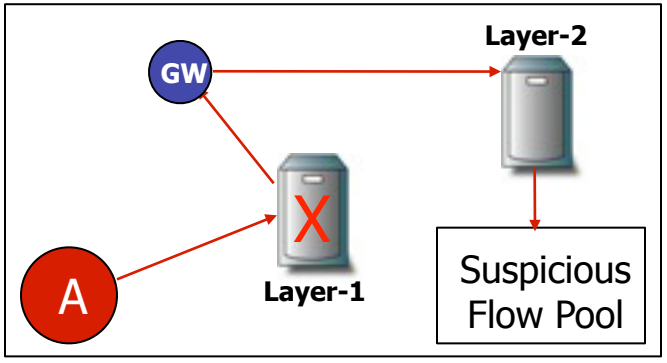


Traffic-based flow classifiers

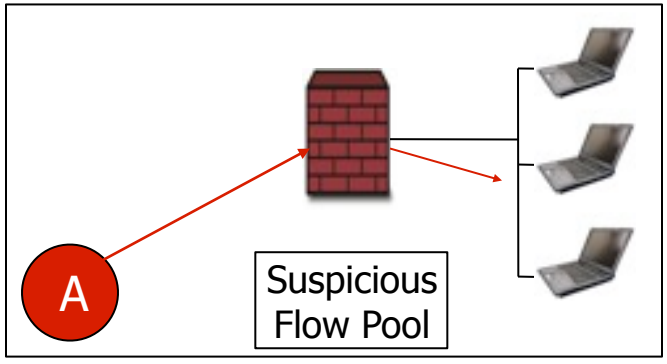
Simulated Honeynet



Double Honeynet

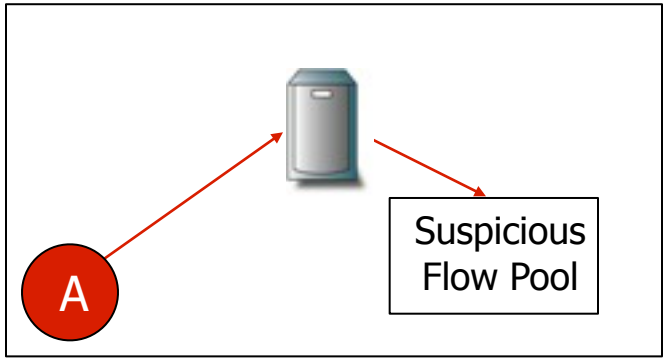


Port-scanning detection

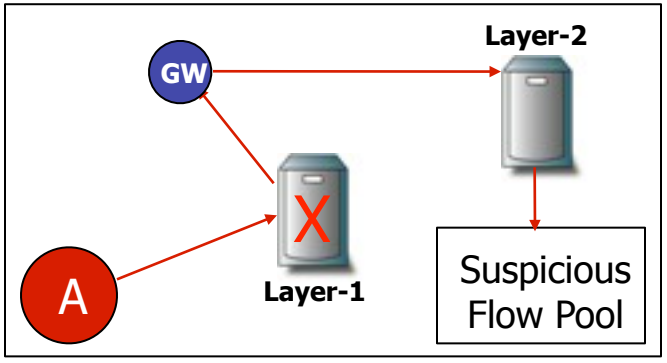


Traffic-based flow classifiers

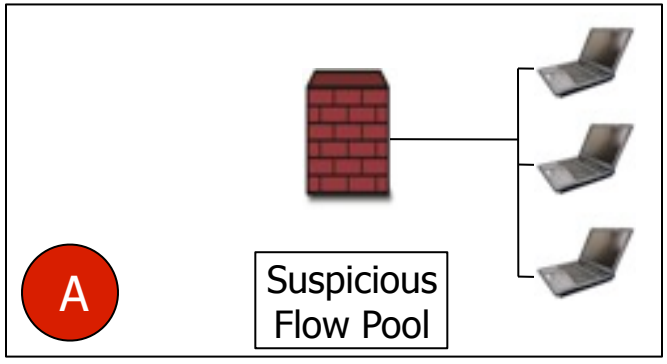
Simulated Honeynet



Double Honeynet

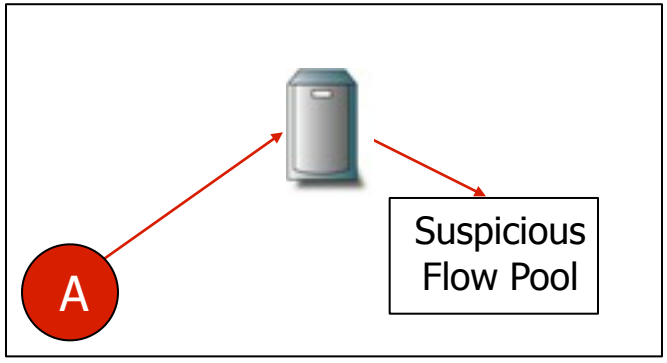


Port-scanning detection

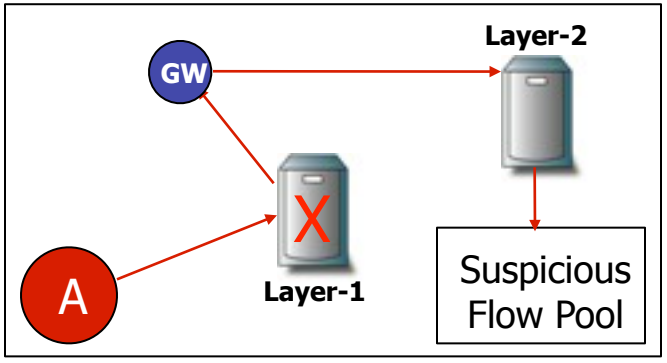


Traffic-based flow classifiers

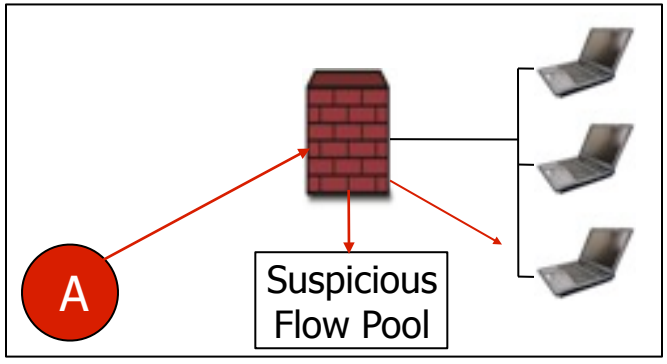
Simulated Honeynet



Double Honeynet

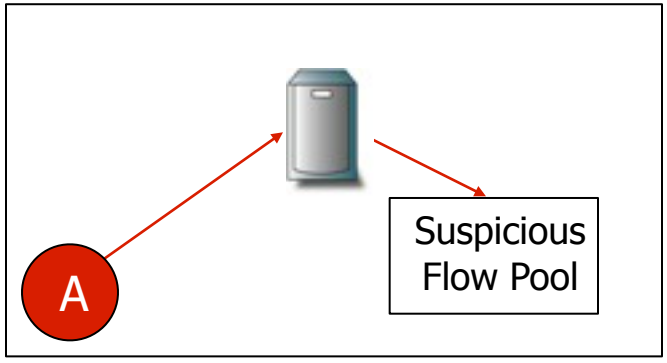


Port-scanning detection

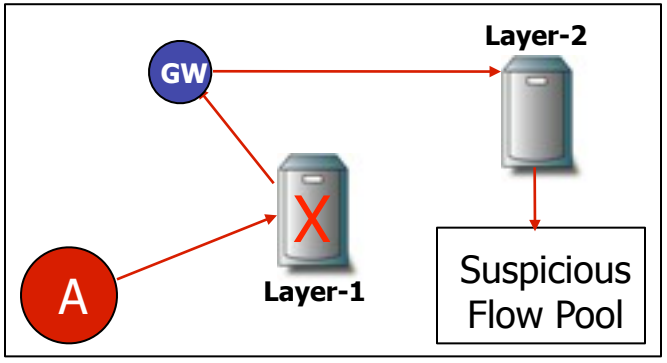


Traffic-based flow classifiers

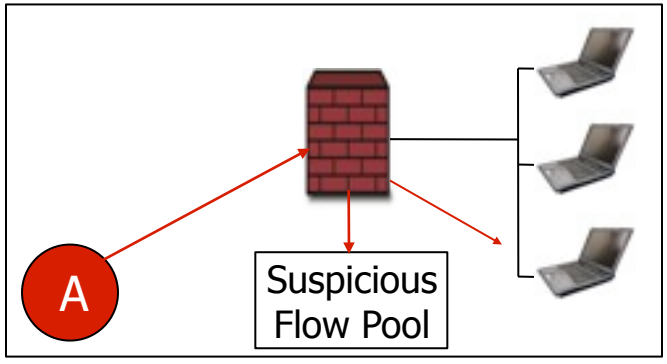
Simulated Honeynet



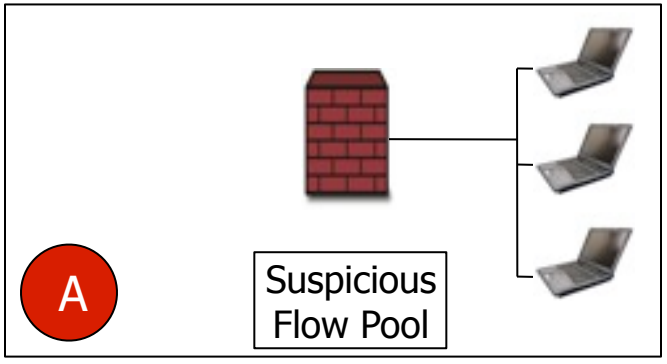
Double Honeynet



Port-scanning detection

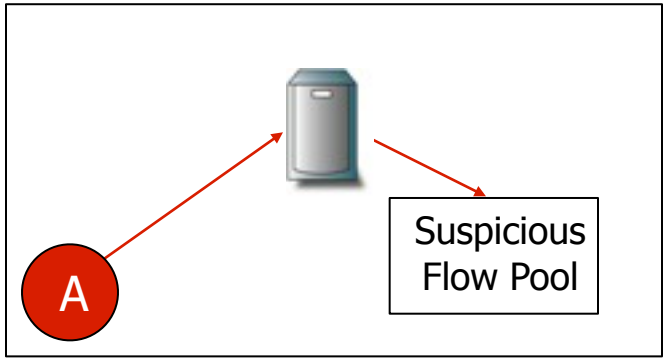


Anomaly IDS

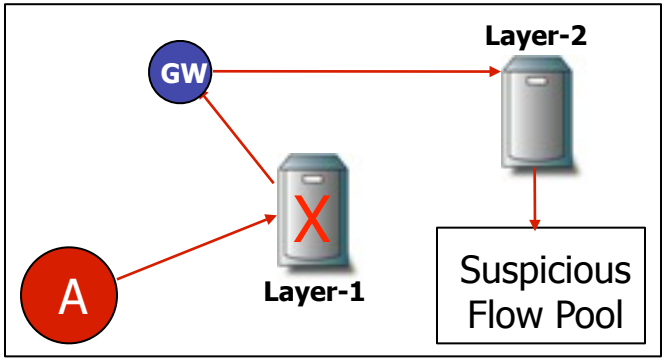


Traffic-based flow classifiers

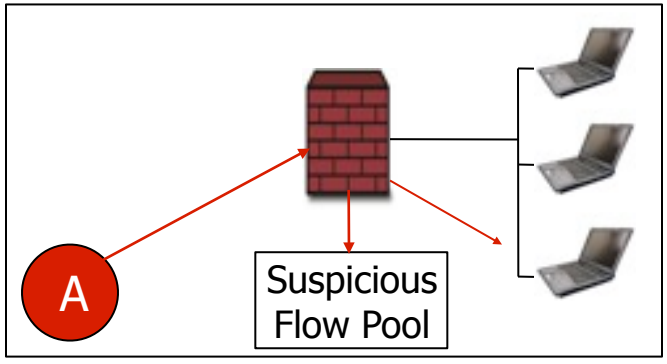
Simulated Honeynet



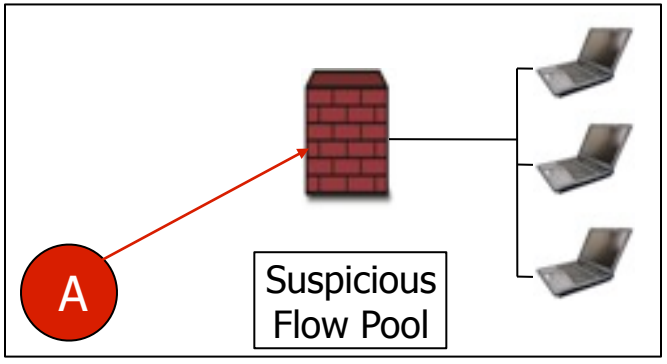
Double Honeynet



Port-scanning detection

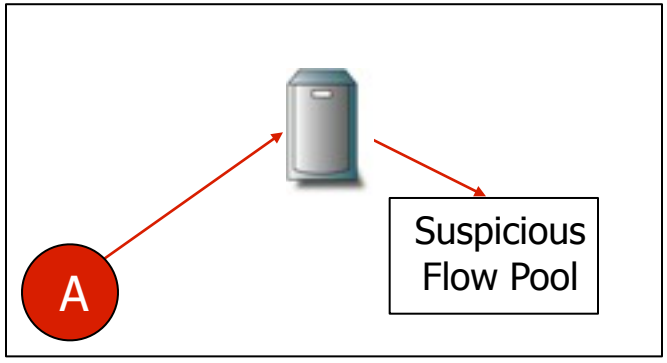


Anomaly IDS

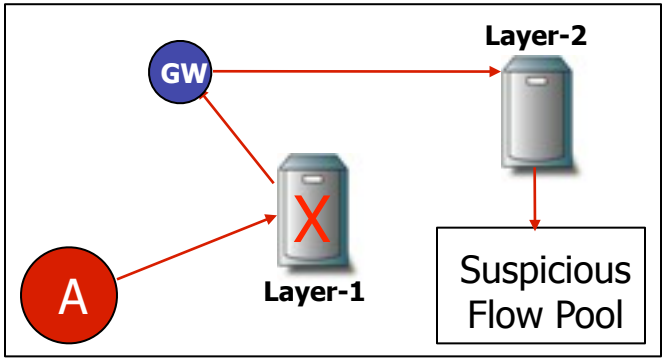


Traffic-based flow classifiers

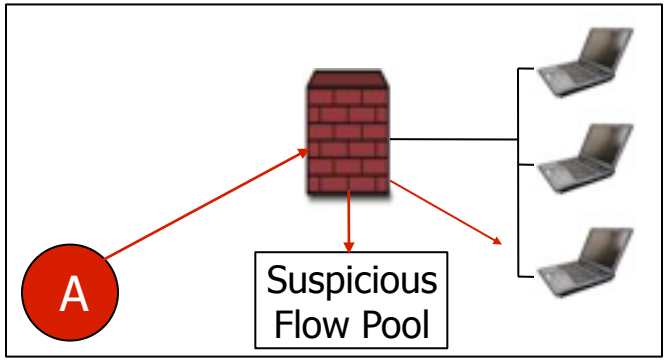
Simulated Honeynet



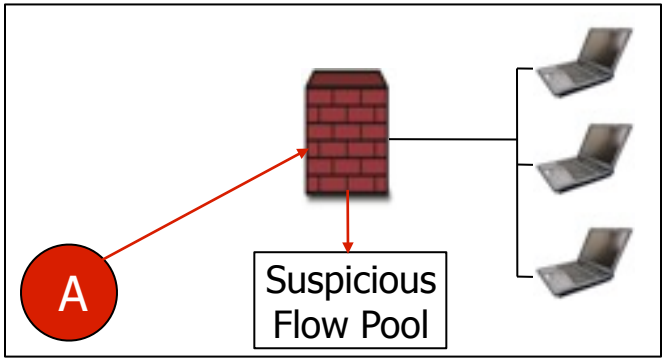
Double Honeynet



Port-scanning detection

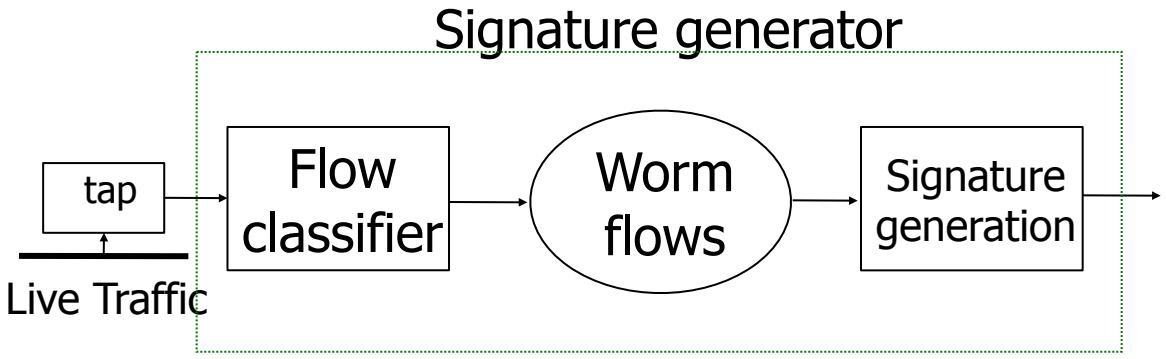


Anomaly IDS



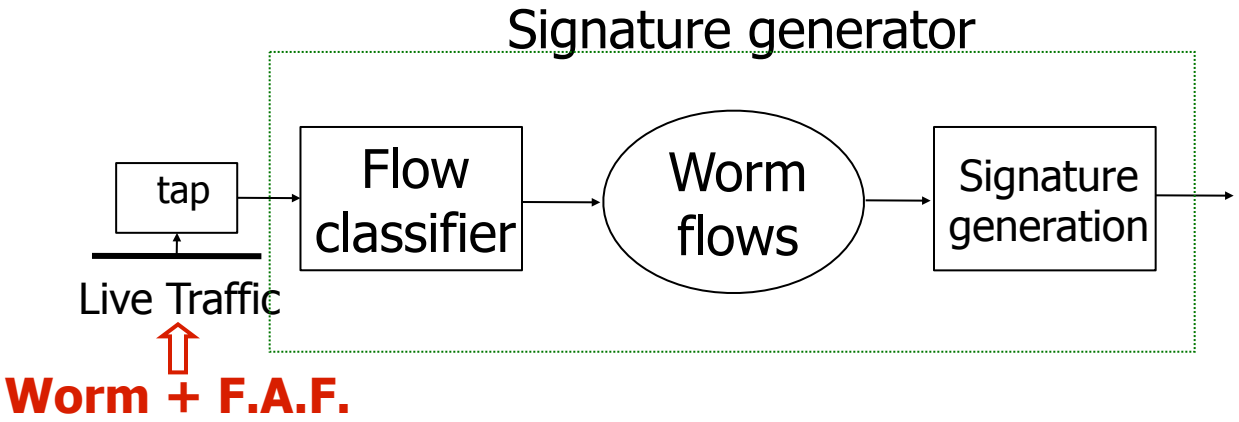
Noise Injection Attack

- Noise Injection Attack "poisons" the suspicious flow pool dataset with fake anomalous flows



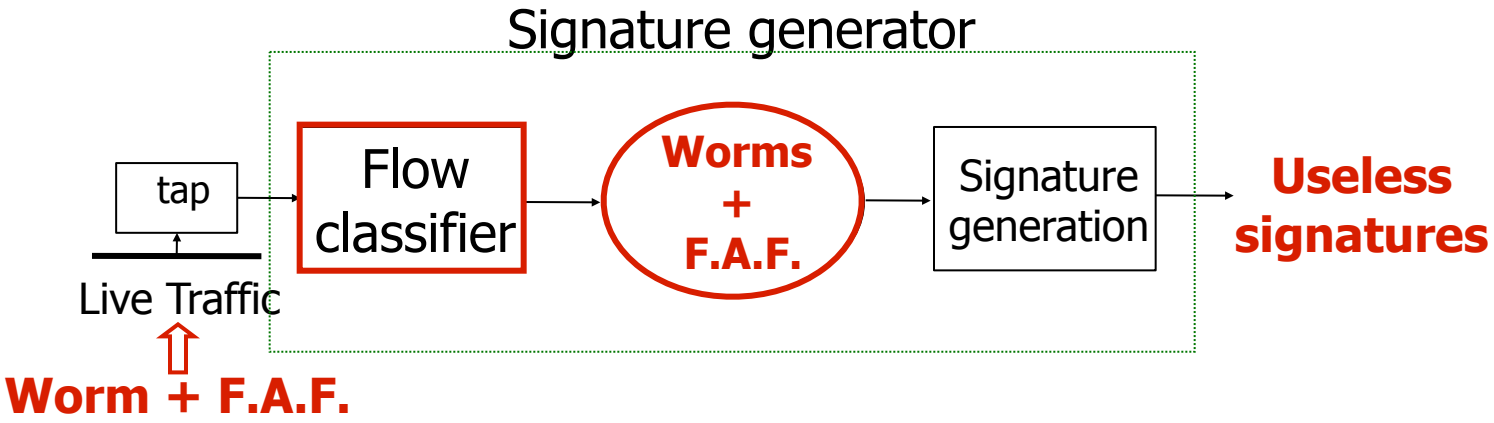
Noise Injection Attack

- Noise Injection Attack "poisons" the suspicious flow pool dataset with fake anomalous flows



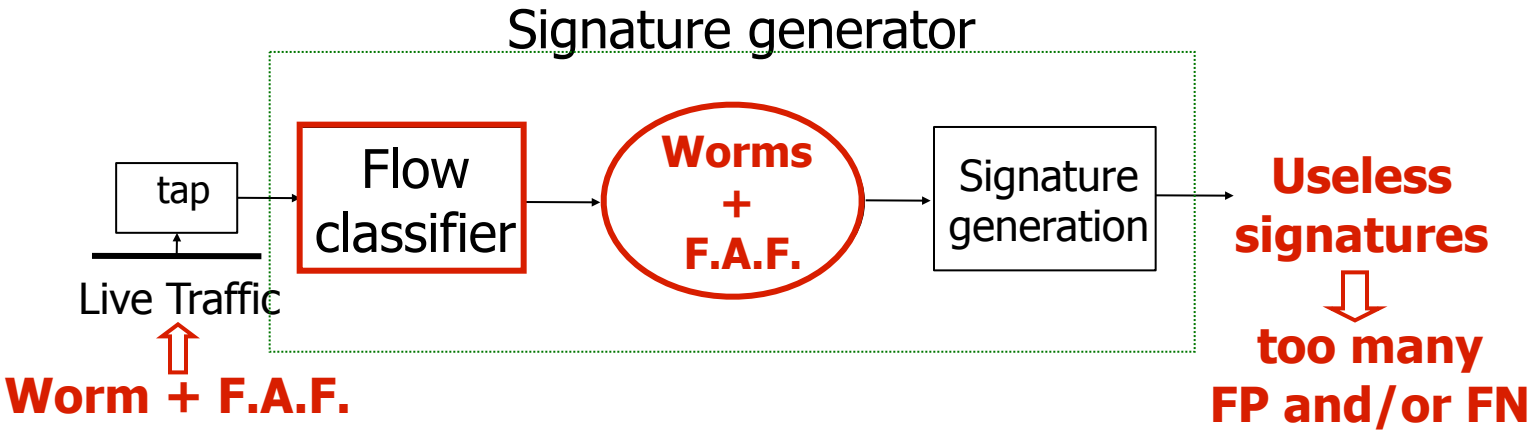
Noise Injection Attack

- Noise Injection Attack "poisons" the suspicious flow pool dataset with fake anomalous flows



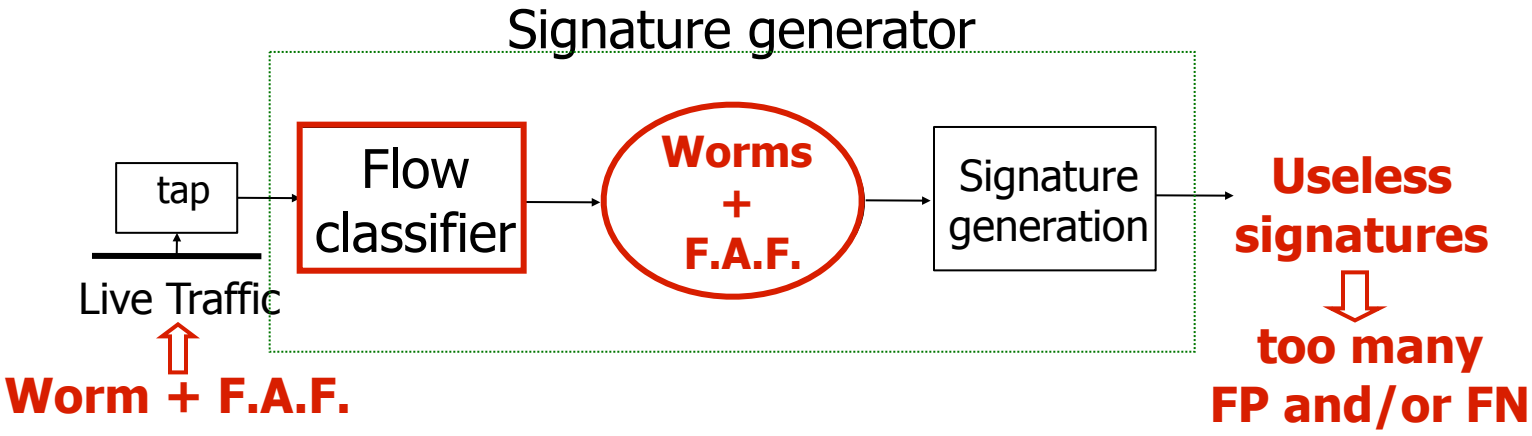
Noise Injection Attack

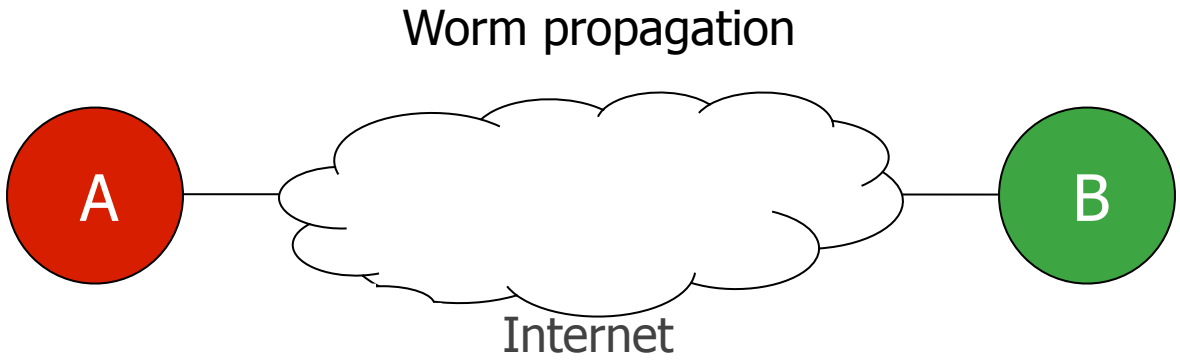
- Noise Injection Attack "poisons" the suspicious flow pool dataset with fake anomalous flows



Noise Injection Attack

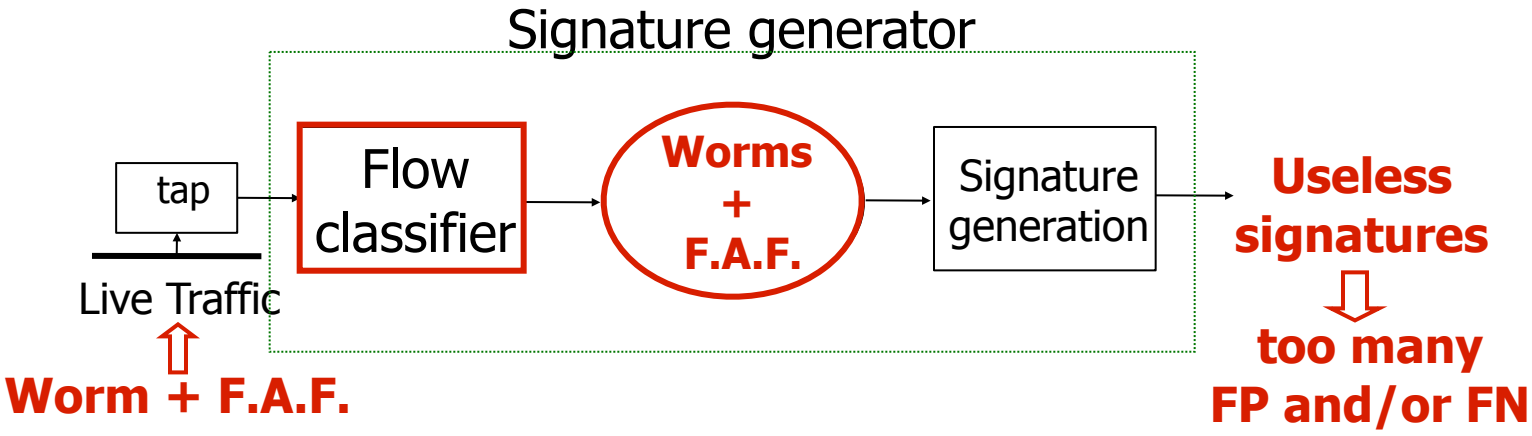
- Noise Injection Attack "poisons" the suspicious flow pool dataset with fake anomalous flows

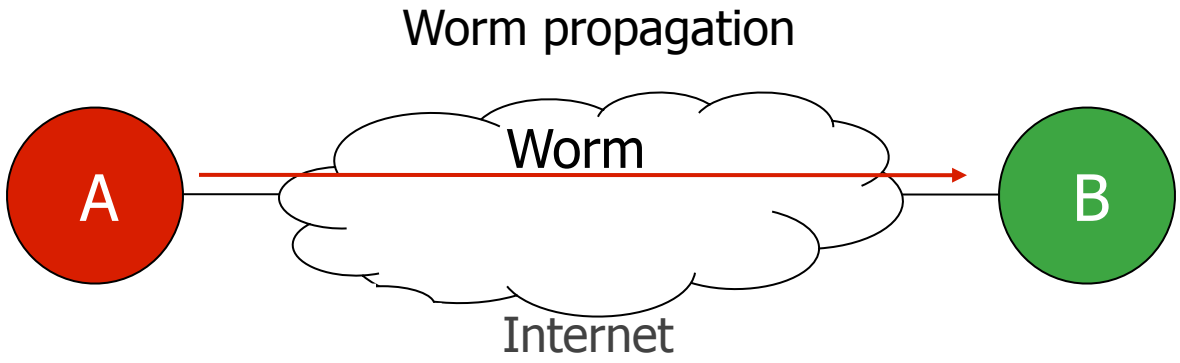




Noise Injection Attack

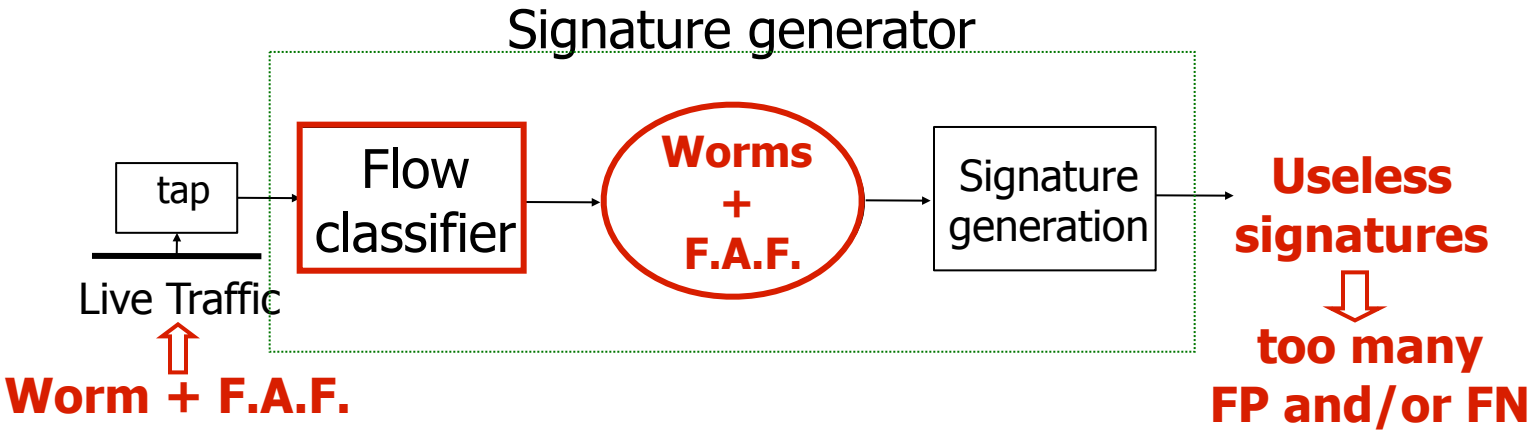
- Noise Injection Attack "poisons" the suspicious flow pool dataset with fake anomalous flows

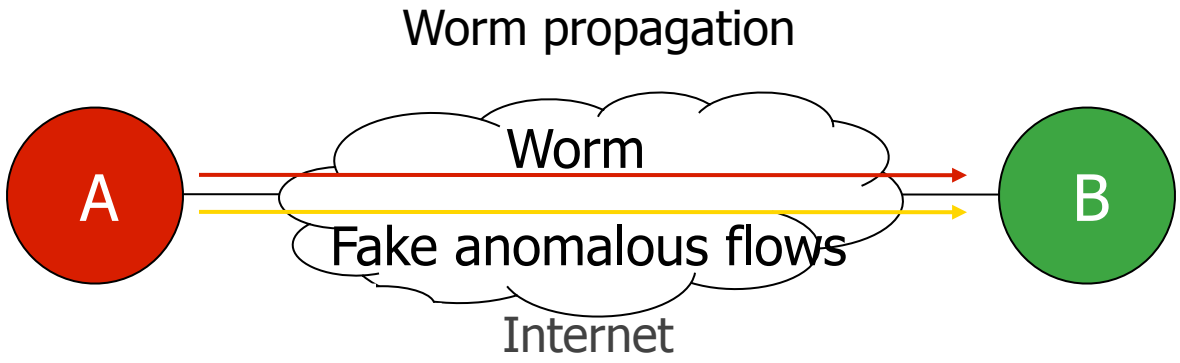




Noise Injection Attack

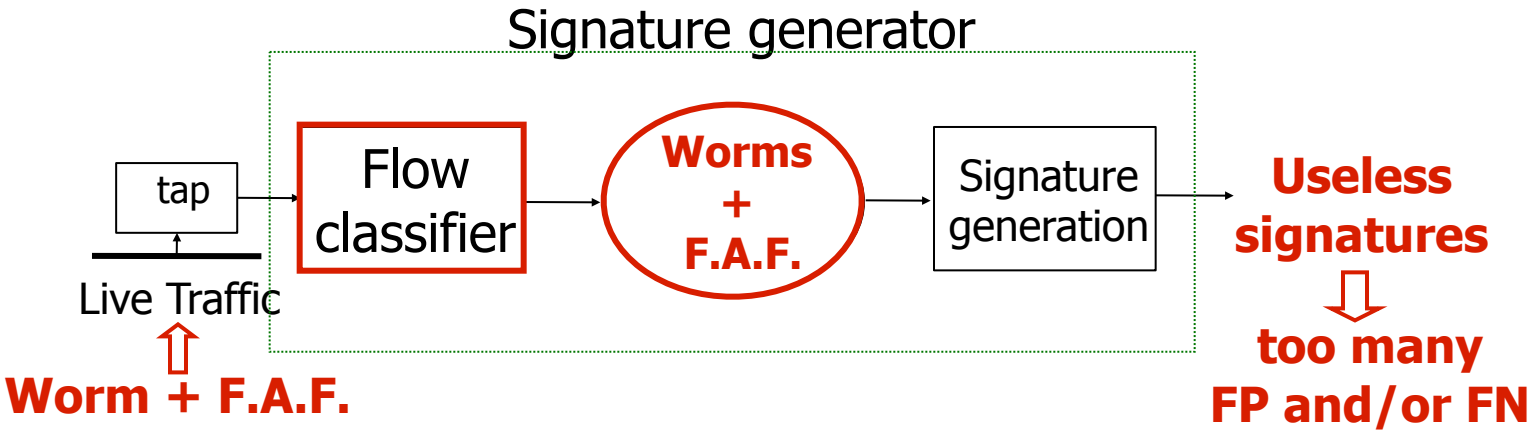
- Noise Injection Attack "poisons" the suspicious flow pool dataset with fake anomalous flows

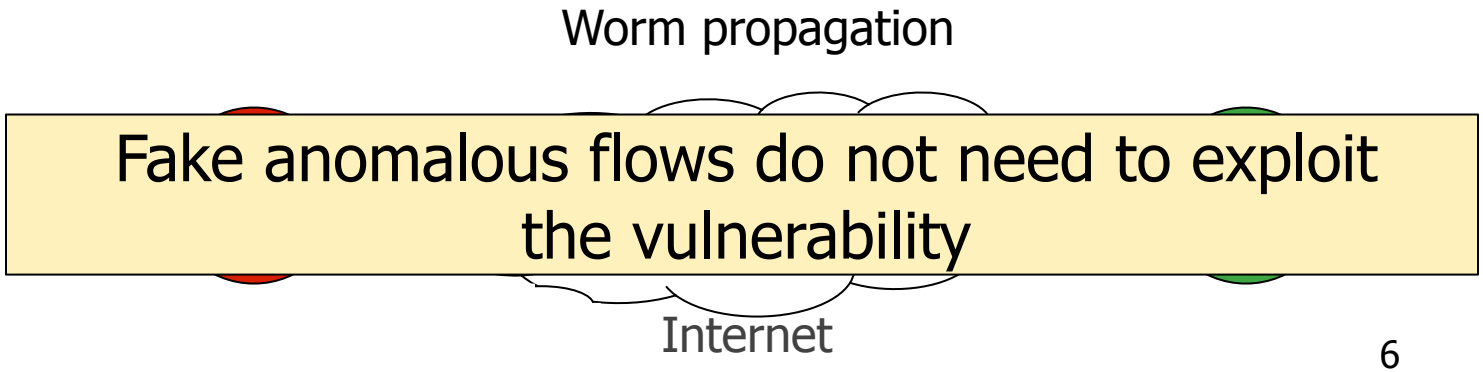




Noise Injection Attack

- Noise Injection Attack "poisons" the suspicious flow pool dataset with fake anomalous flows

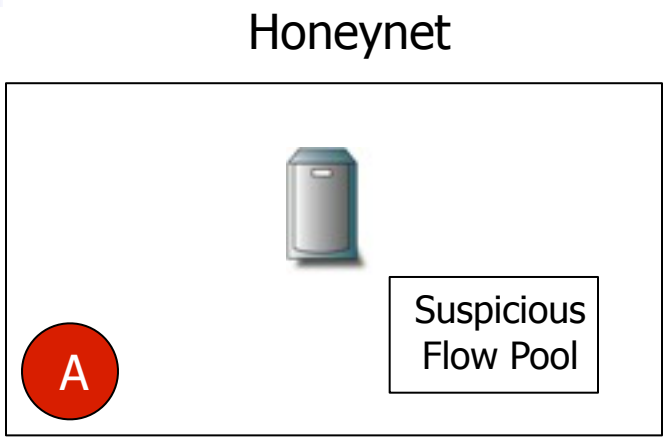




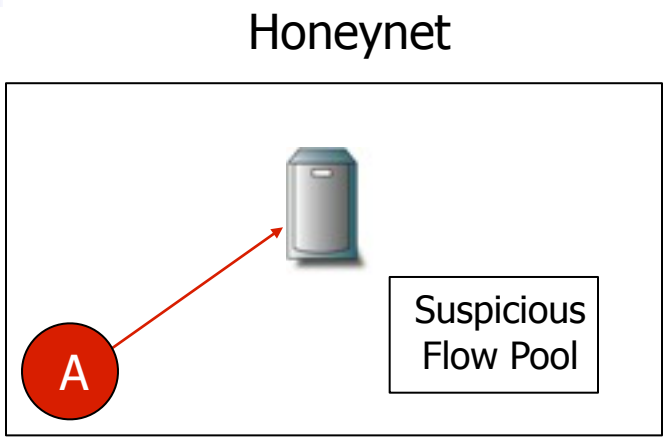
Fake anomalous flows do not need to exploit the vulnerability

Noise Injection Attack

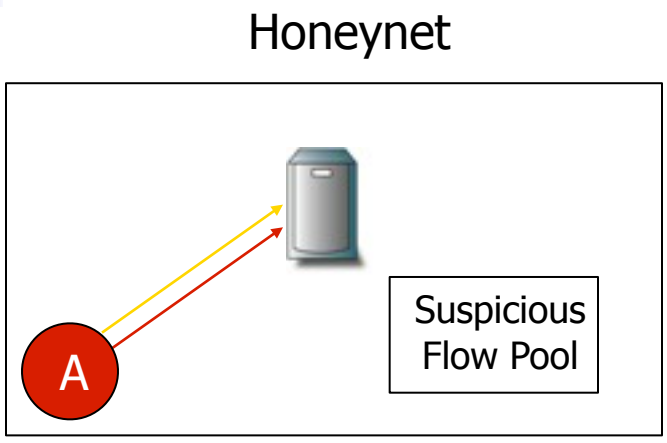
Noise Injection Attack



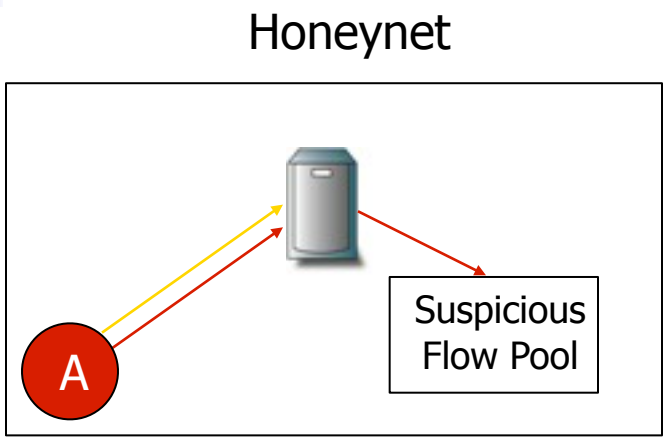
Noise Injection Attack



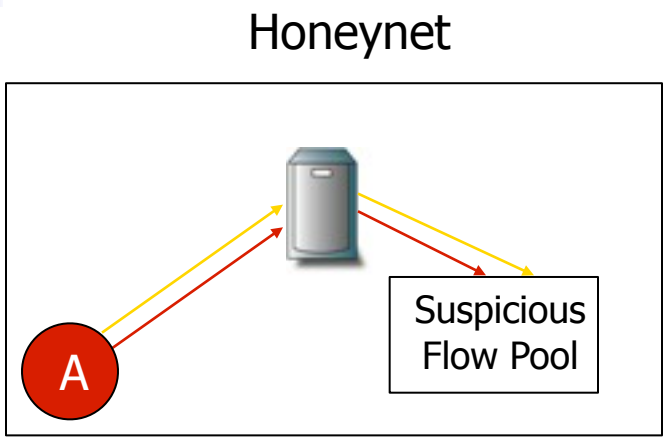
Noise Injection Attack



Noise Injection Attack

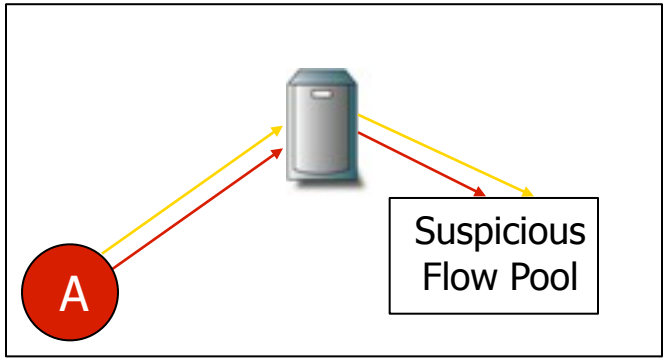


Noise Injection Attack

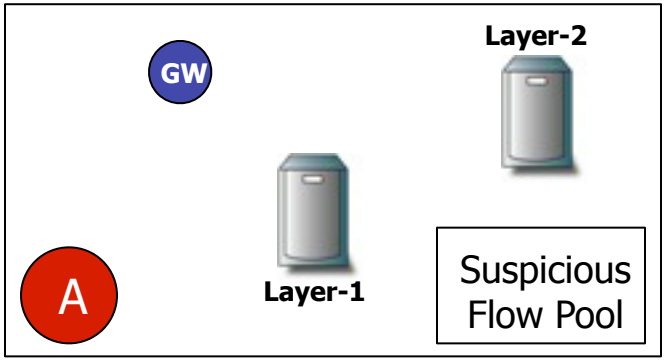


Noise Injection Attack

Honeynet

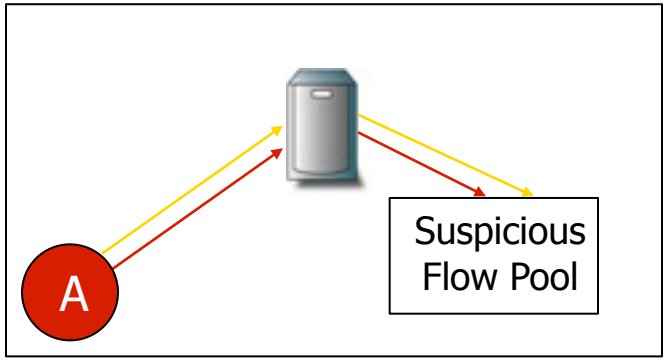


Double Honeynet

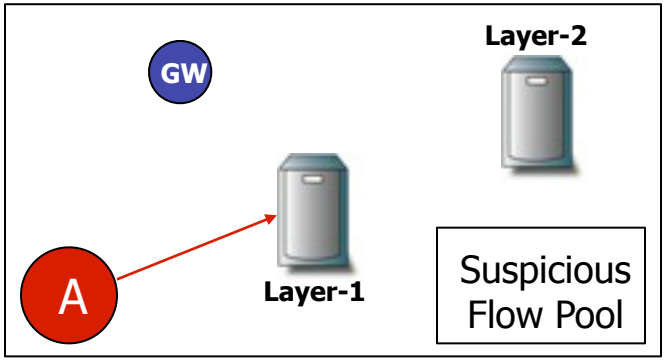


Noise Injection Attack

Honeynet

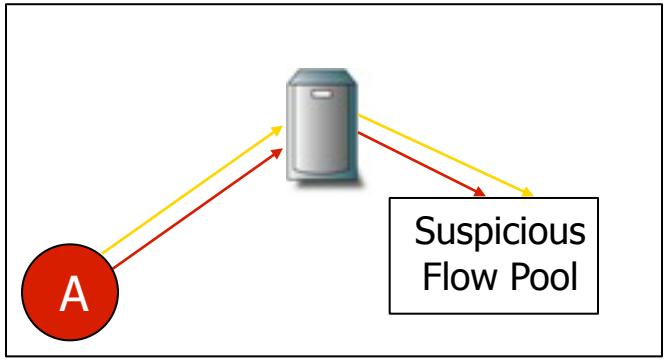


Double Honeynet

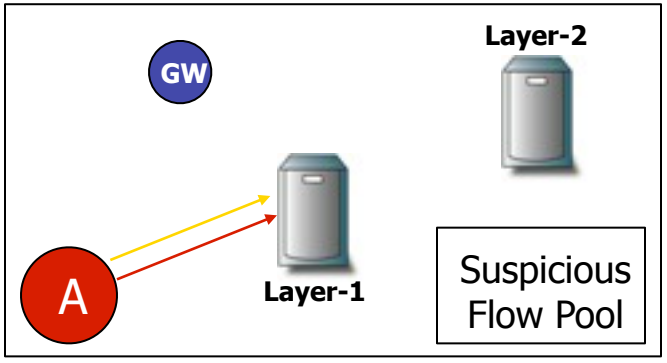


Noise Injection Attack

Honeynet

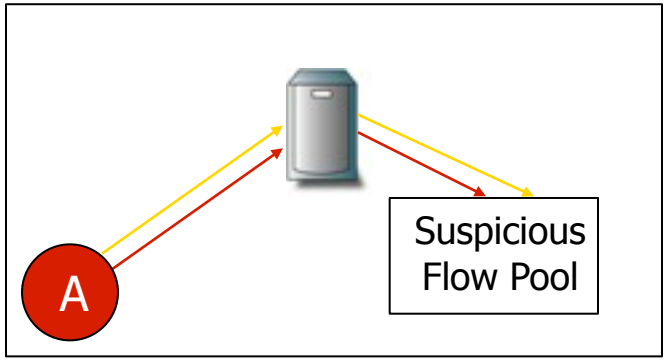


Double Honeynet

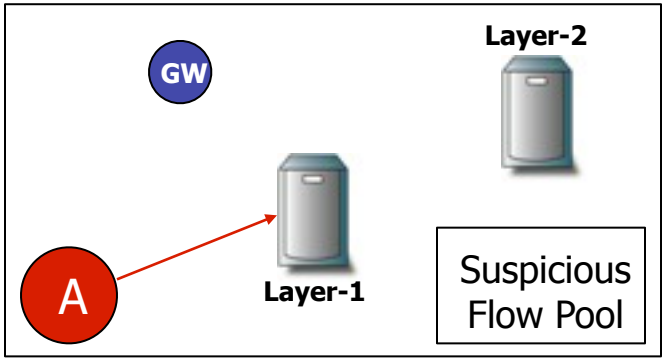


Noise Injection Attack

Honeynet

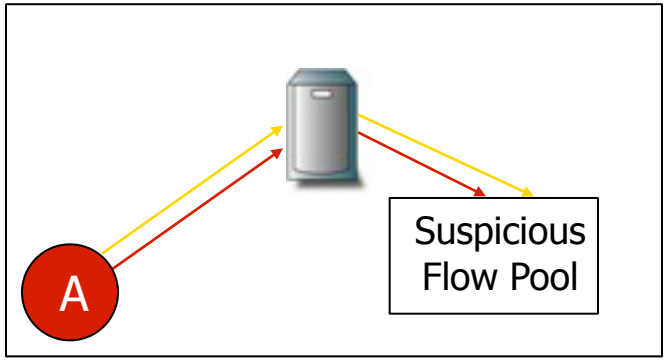


Double Honeynet

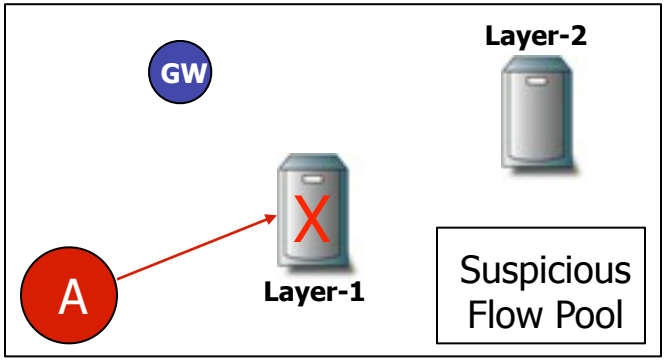


Noise Injection Attack

Honeynet

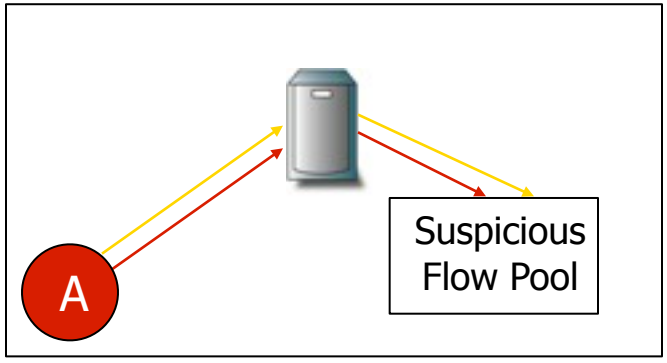


Double Honeynet

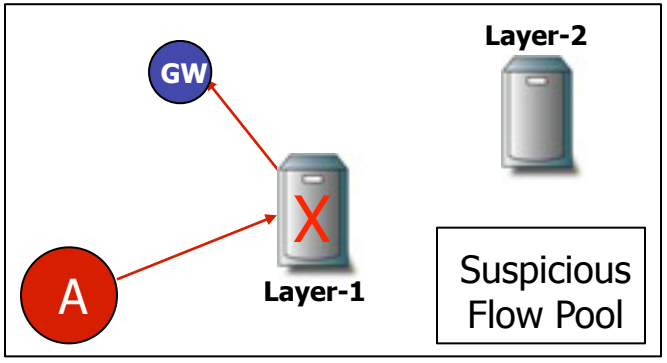


Noise Injection Attack

Honeynet

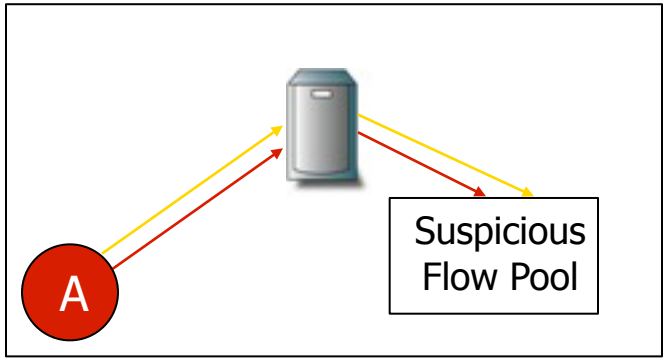


Double Honeynet

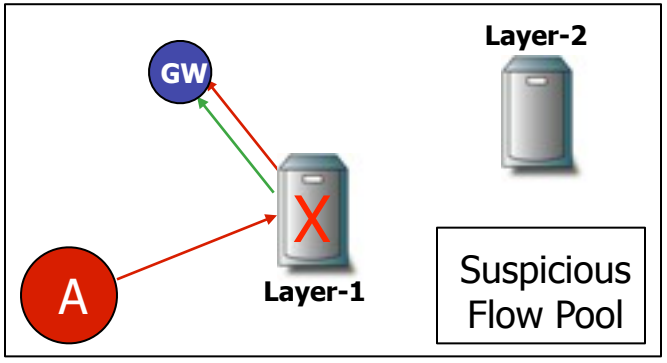


Noise Injection Attack

Honeynet

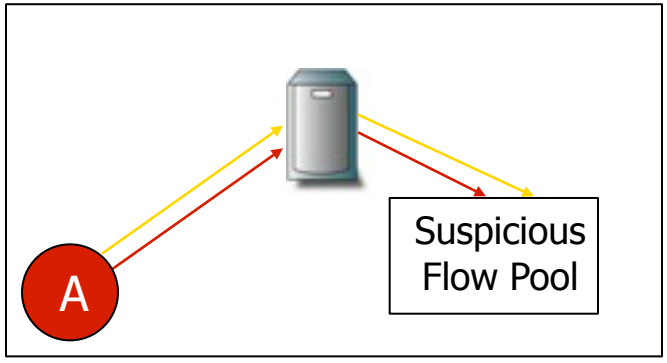


Double Honeynet

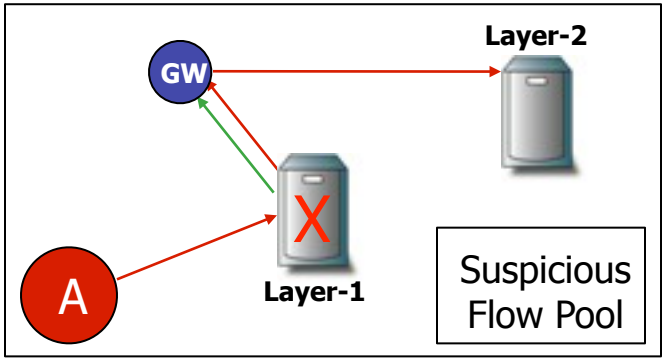


Noise Injection Attack

Honeynet

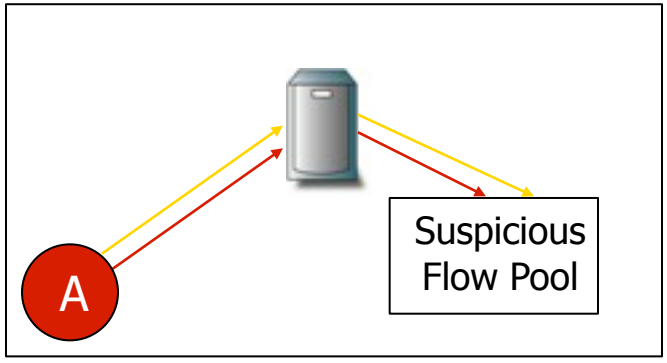


Double Honeynet

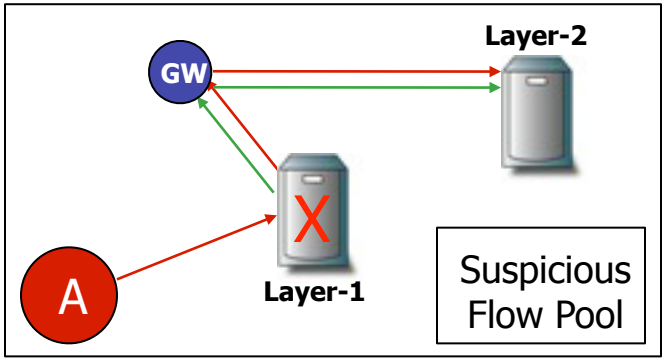


Noise Injection Attack

Honeynet

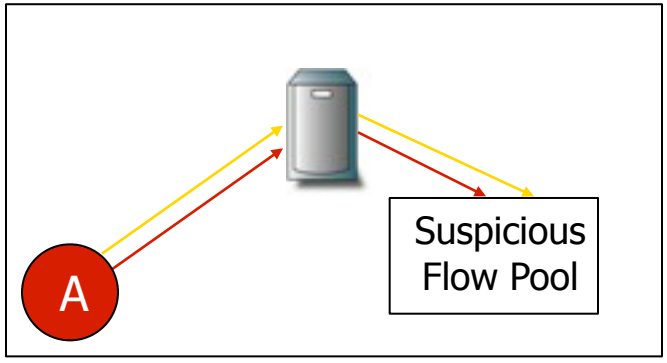


Double Honeynet

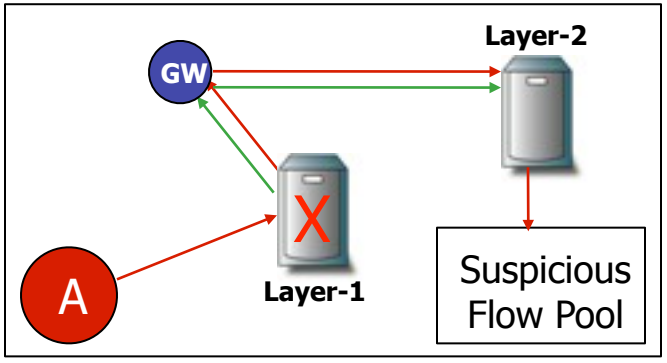


Noise Injection Attack

Honeynet

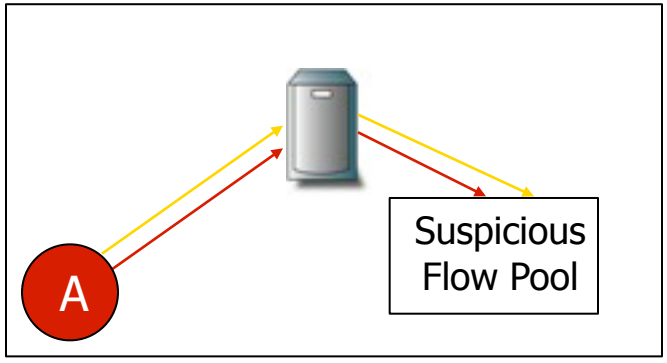


Double Honeynet

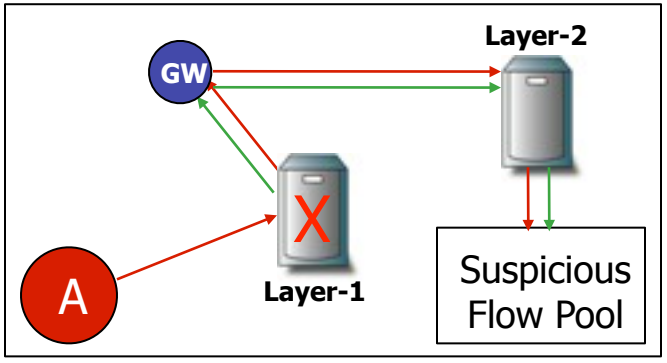


Noise Injection Attack

Honeynet

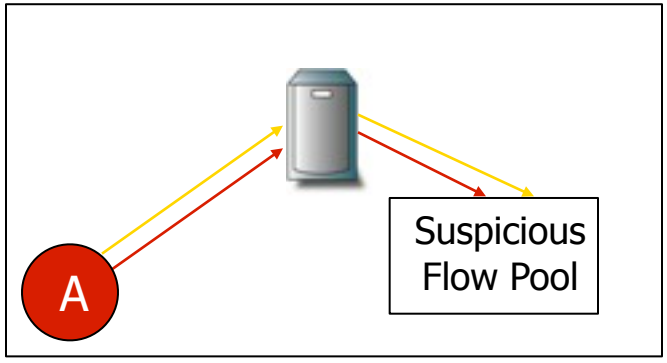


Double Honeynet

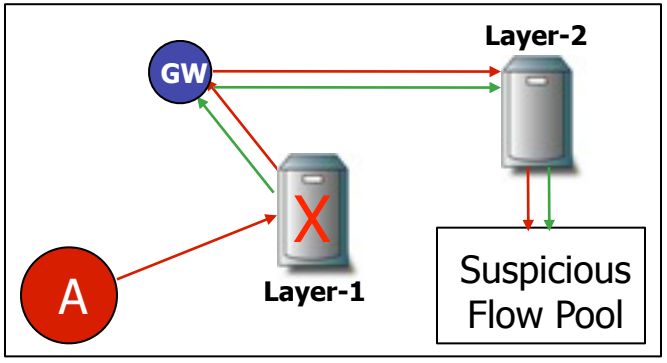


Noise Injection Attack

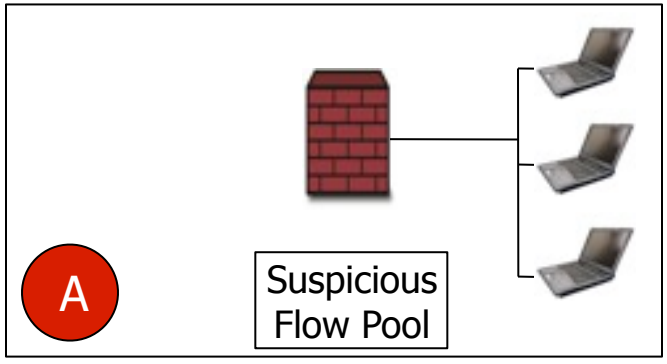
Honeynet



Double Honeynet

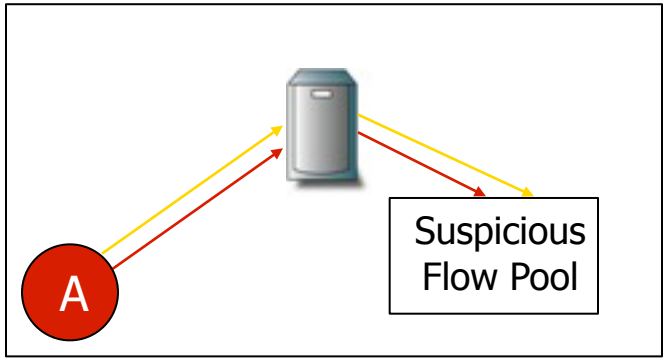


Port-scanning detection

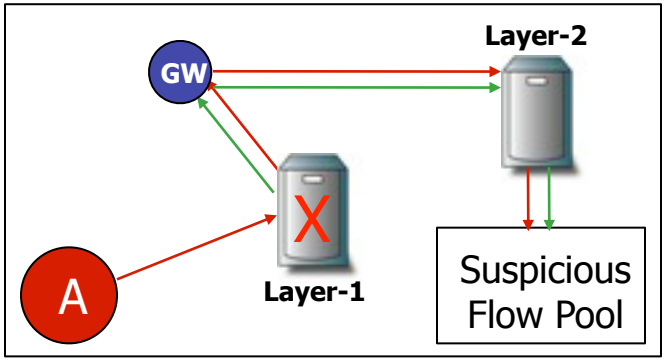


Noise Injection Attack

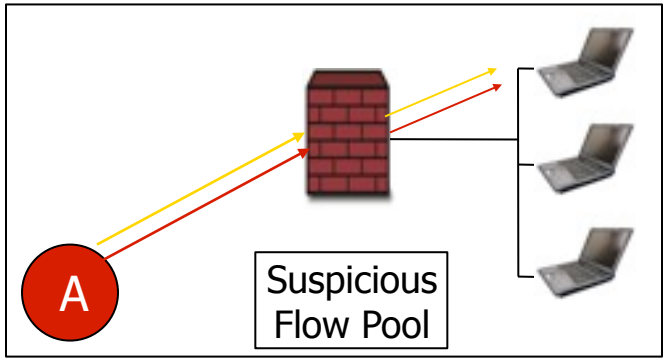
Honeynet



Double Honeynet

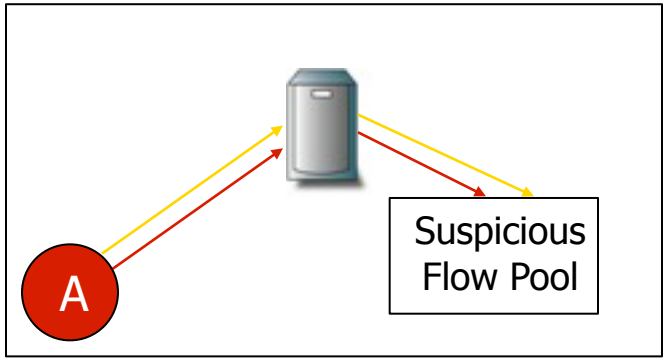


Port-scanning detection

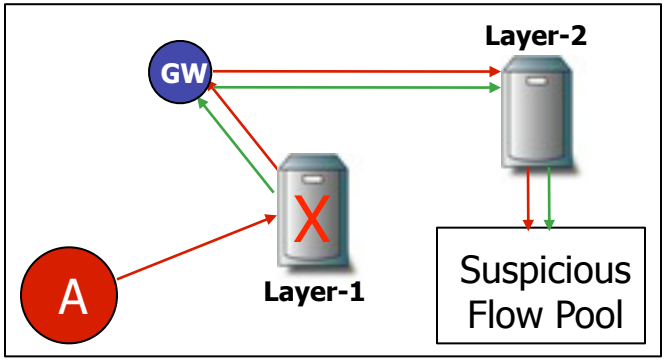


Noise Injection Attack

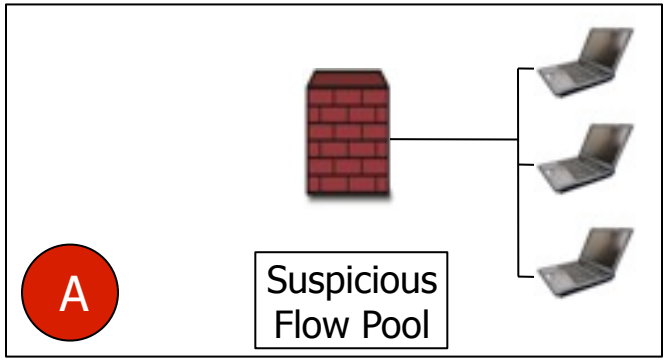
Honeynet



Double Honeynet

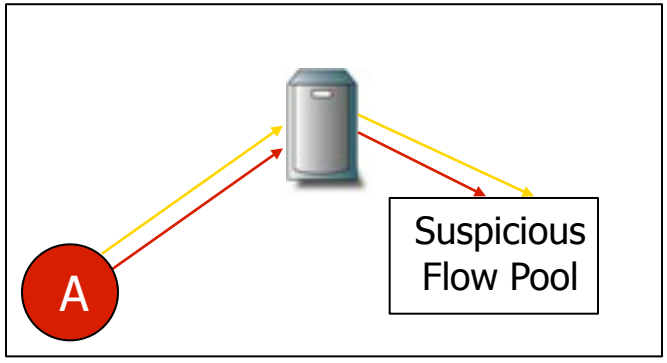


Port-scanning detection

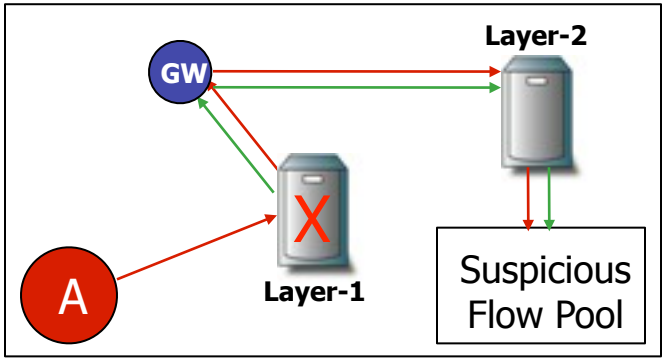


Noise Injection Attack

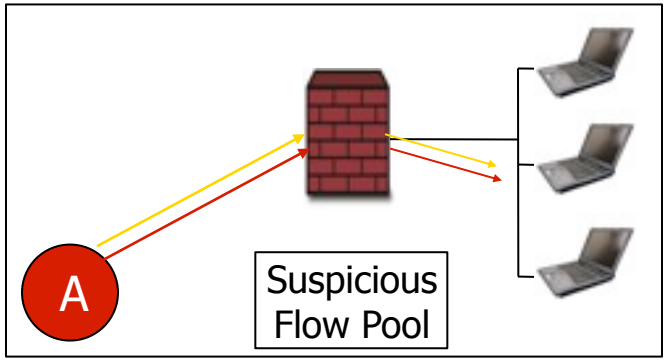
Honeynet



Double Honeynet

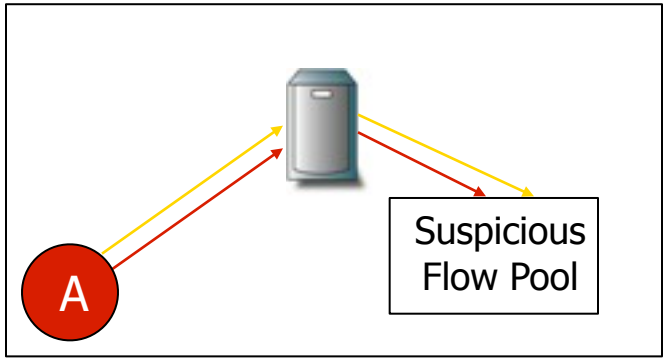


Port-scanning detection

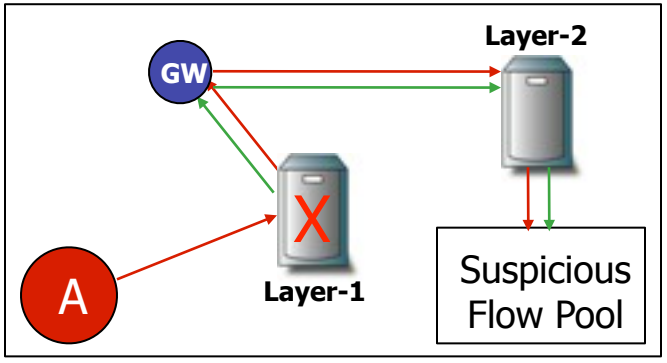


Noise Injection Attack

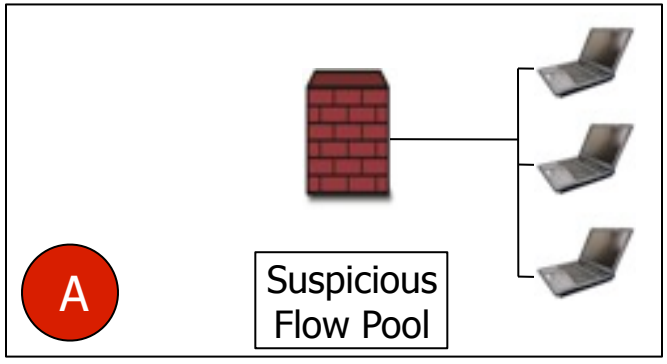
Honeynet



Double Honeynet

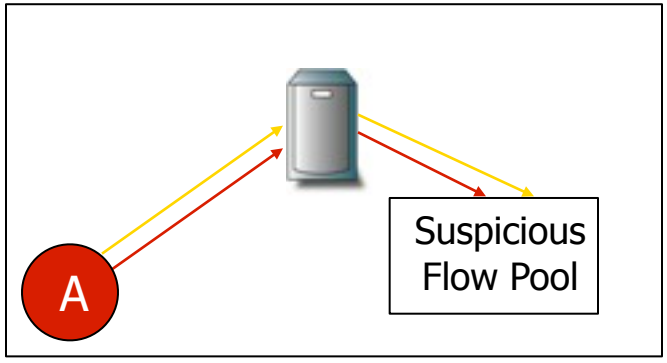


Port-scanning detection

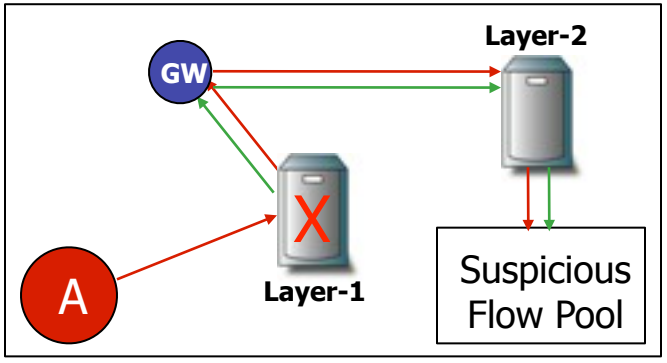


Noise Injection Attack

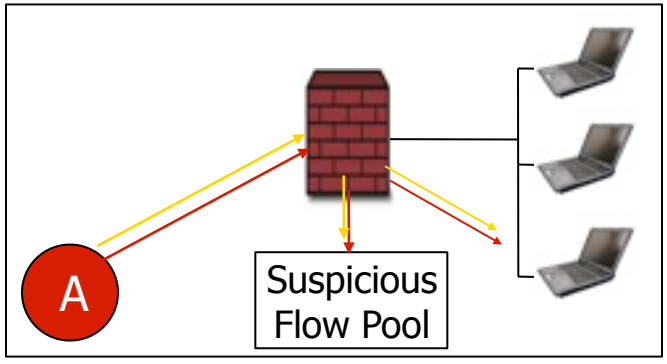
Honeynet



Double Honeynet

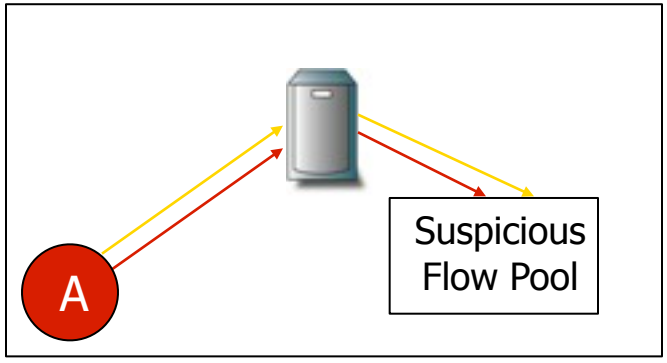


Port-scanning detection

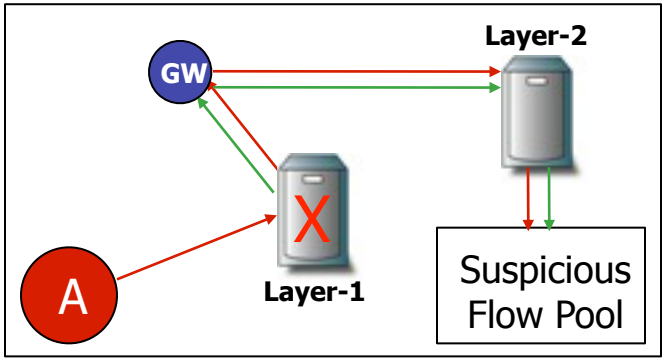


Noise Injection Attack

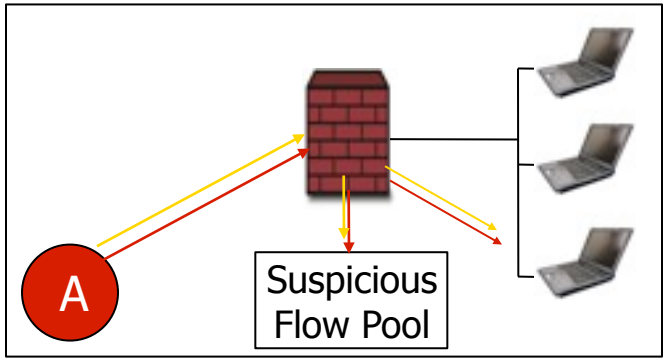
Honeynet



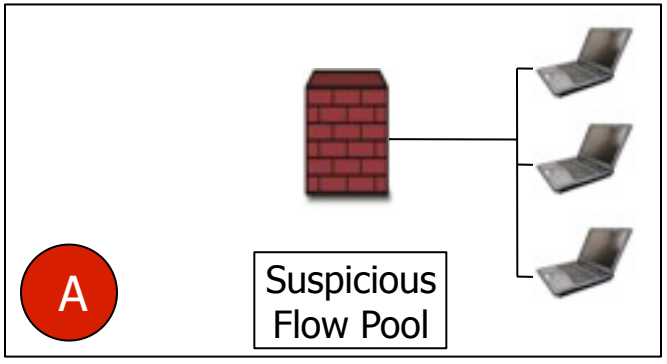
Double Honeynet



Port-scanning detection

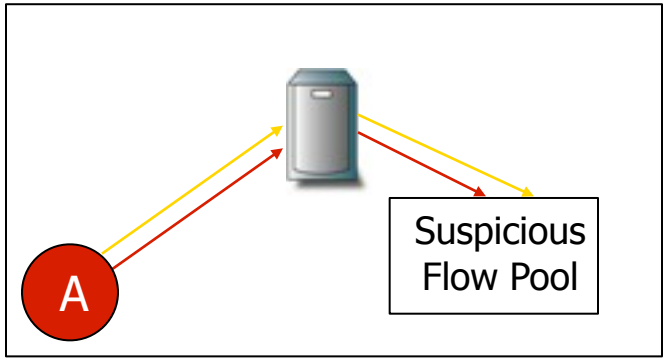


Anomaly IDS

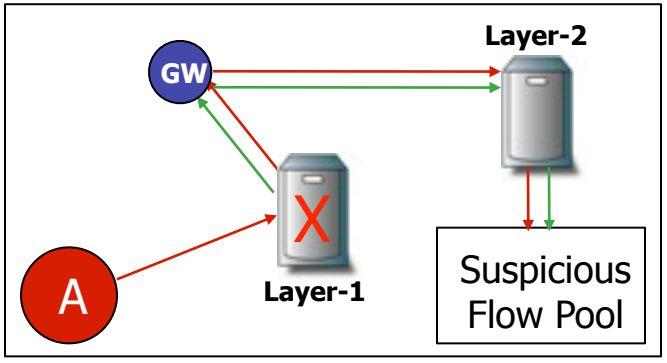


Noise Injection Attack

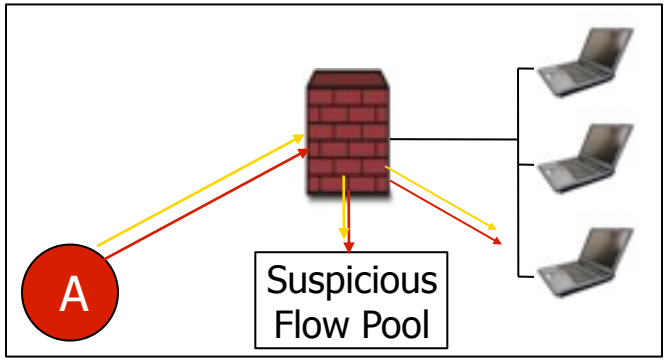
Honeynet



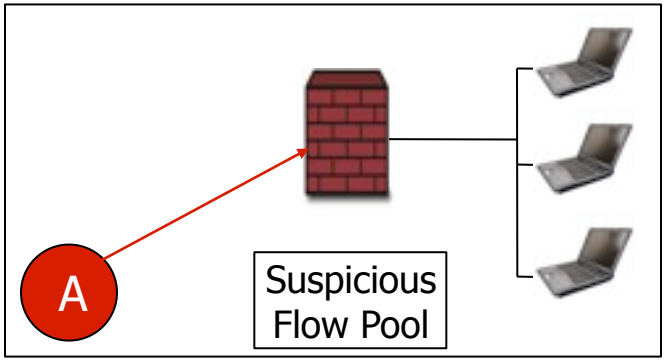
Double Honeynet



Port-scanning detection

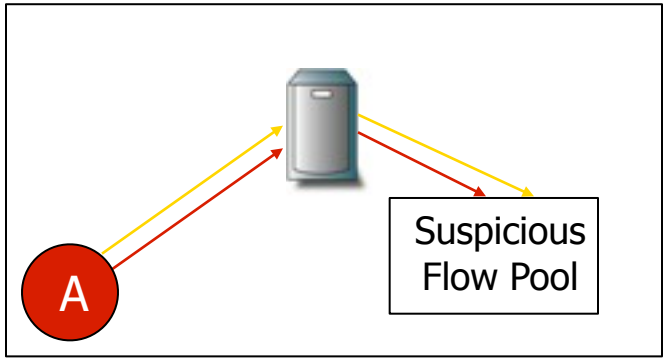


Anomaly IDS

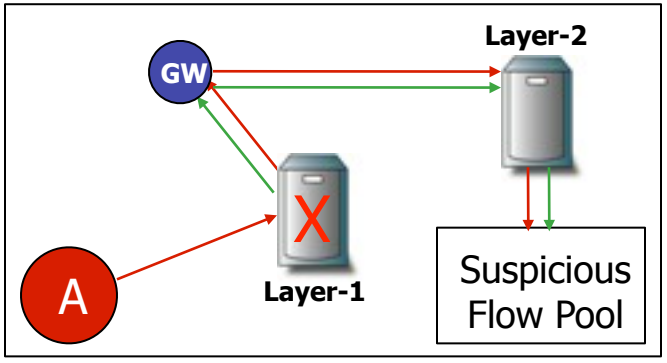


Noise Injection Attack

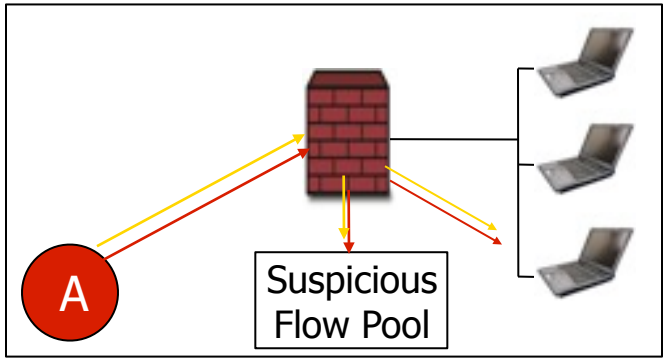
Honeynet



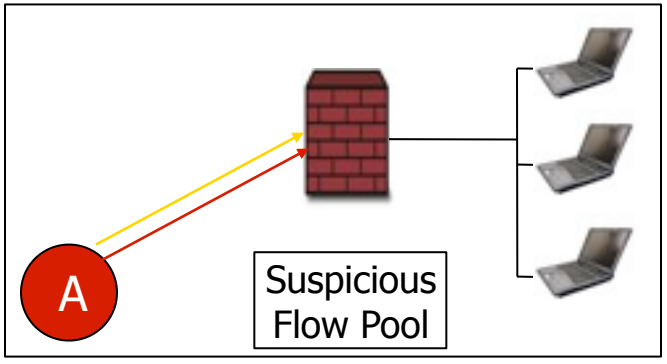
Double Honeynet



Port-scanning detection

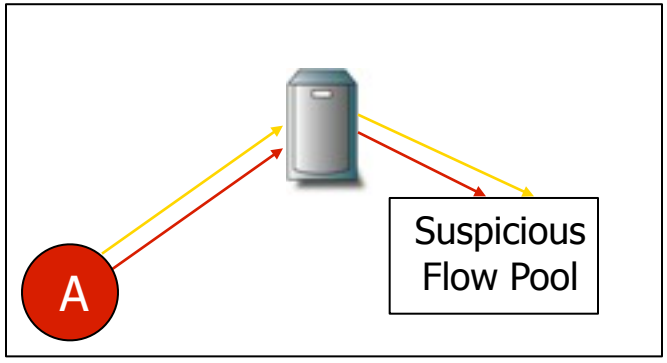


Anomaly IDS

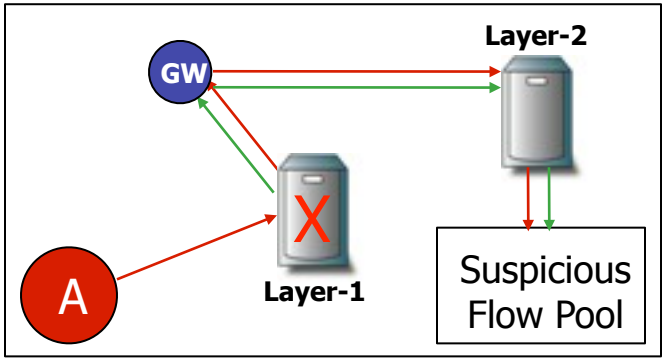


Noise Injection Attack

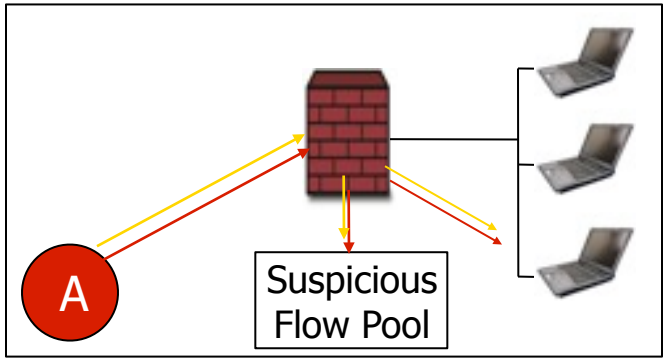
Honeynet



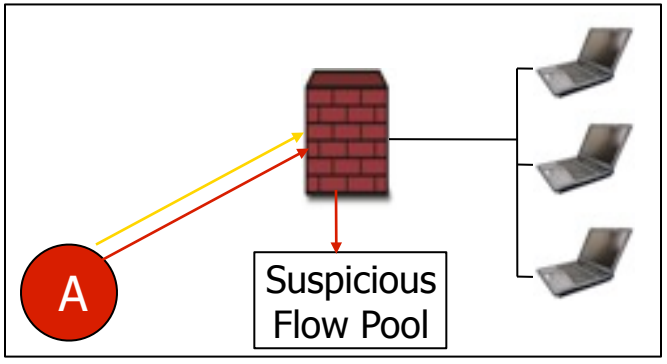
Double Honeynet



Port-scanning detection

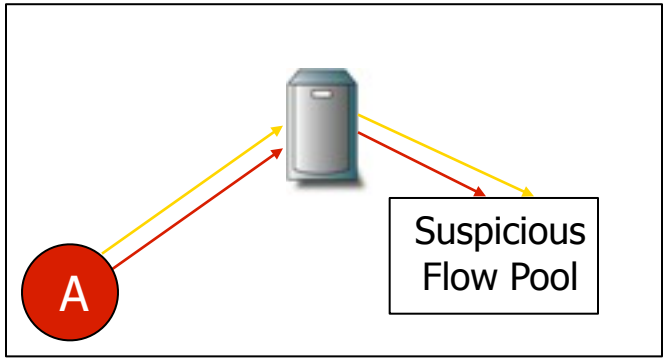


Anomaly IDS

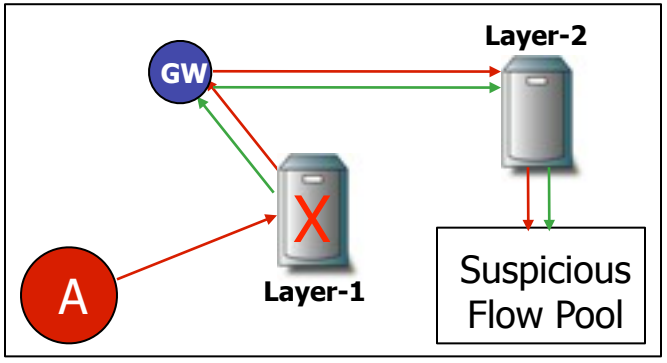


Noise Injection Attack

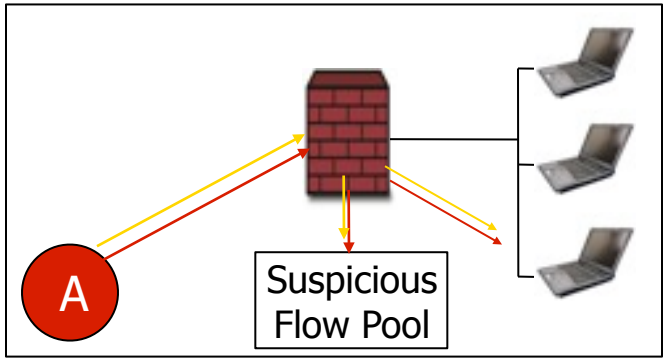
Honeynet



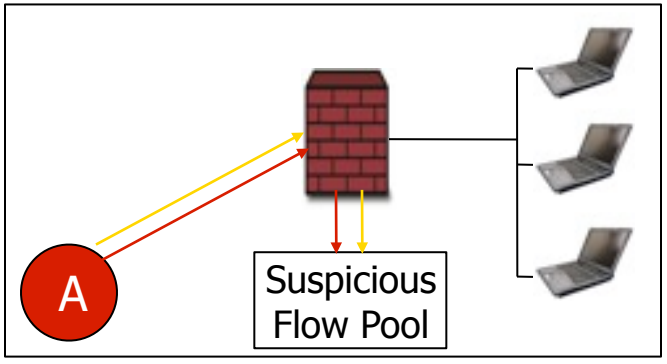
Double Honeynet



Port-scanning detection

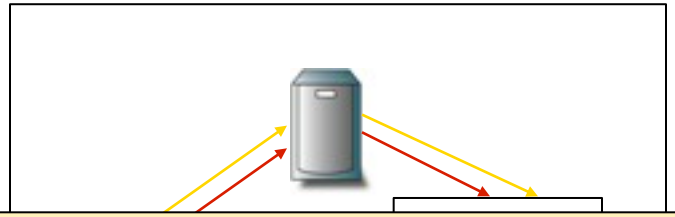


Anomaly IDS

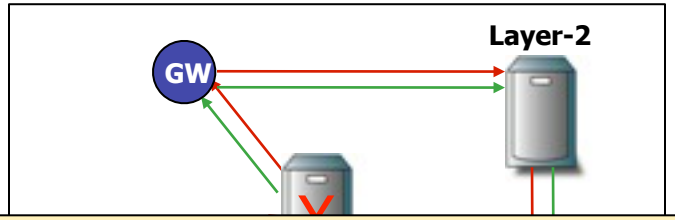


Noise Injection Attack

Honeynet

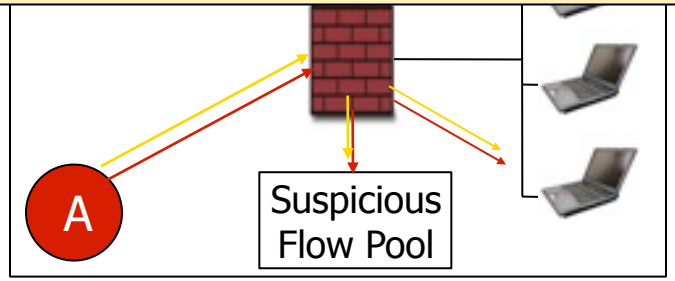


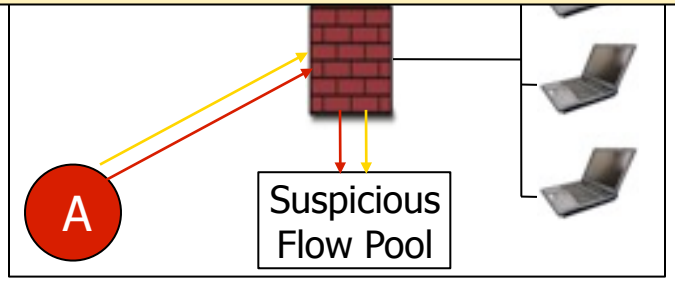
Double Honeynet



Note:

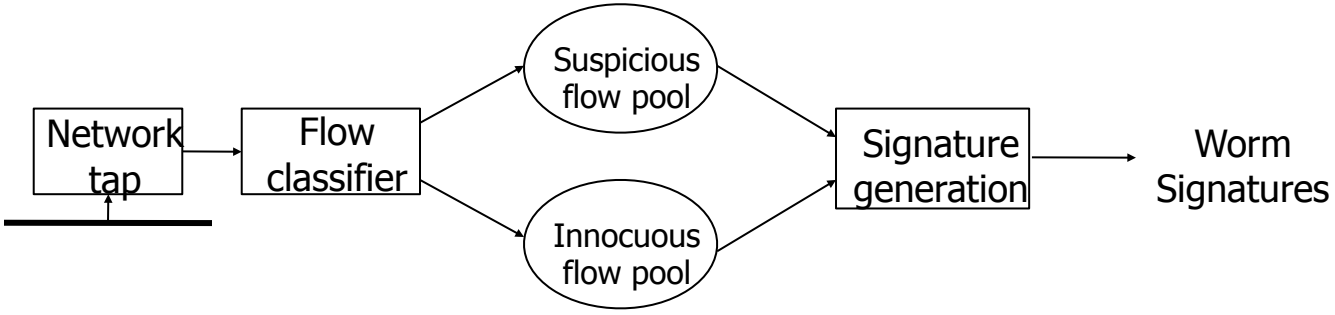
1. The worm could send **many fake anomalou flow** (roughly) at the same time
2. The worm variant and **all** its **fake anomalous flows** will be stored **into the suspicious flow pool**





Case study: POLYGRAPH

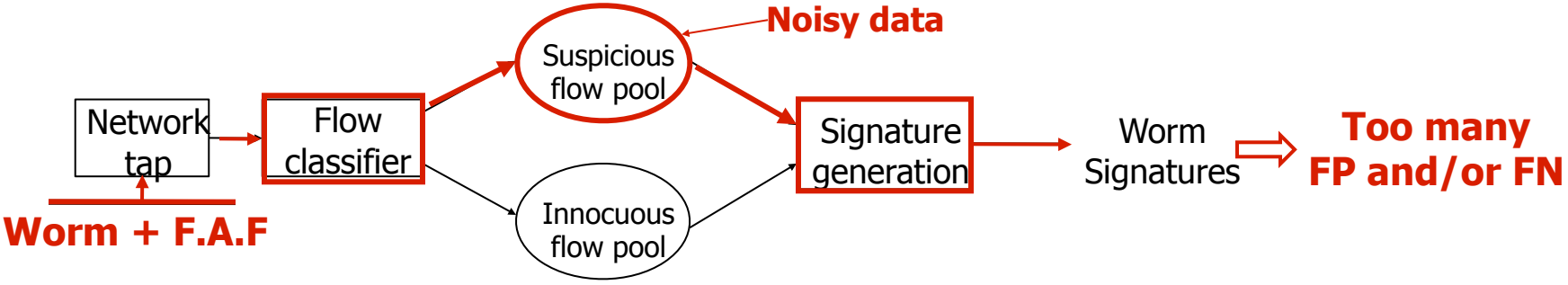
- Signature generation for polymorphic worms



- The flow classification technique is not specified
- However, the authors assume that the flow classifier is not perfect and that innocuous flows (noise) could be stored into the suspicious pool
- **POLYGRAPH seems to be resiliant to noise** into the suspicious pool (up to 80%)
- This is **not true if** the **noise** is **deliberately well-**

Case study: POLYGRAPH

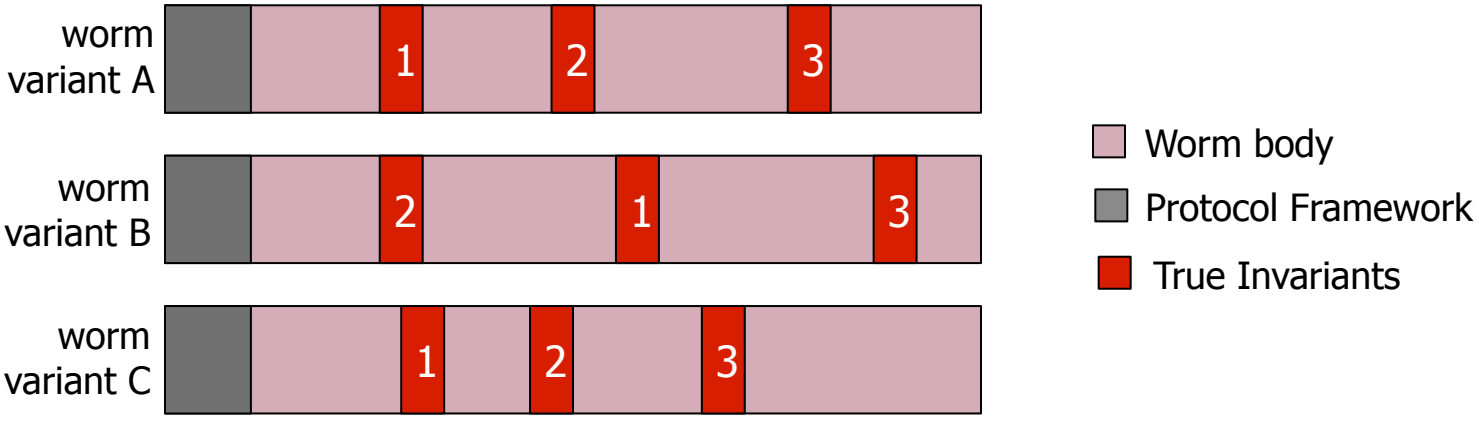
- Signature generation for polymorphic worms



- The flow classification technique is not specified
- However, the authors assume that the flow classifier is not perfect and that innocuous flows (noise) could be stored into the suspicious pool
- **POLYGRAPH seems to be resiliant to noise** into the suspicious pool (up to 80%)
- This is **not true if** the **noise** is **deliberately well-**

Case study: POLYGRAPH

- Polygraph generates 3 different types of signatures:
  - Conjunction, Token-subsequence, Bayes



Conjunction Signature

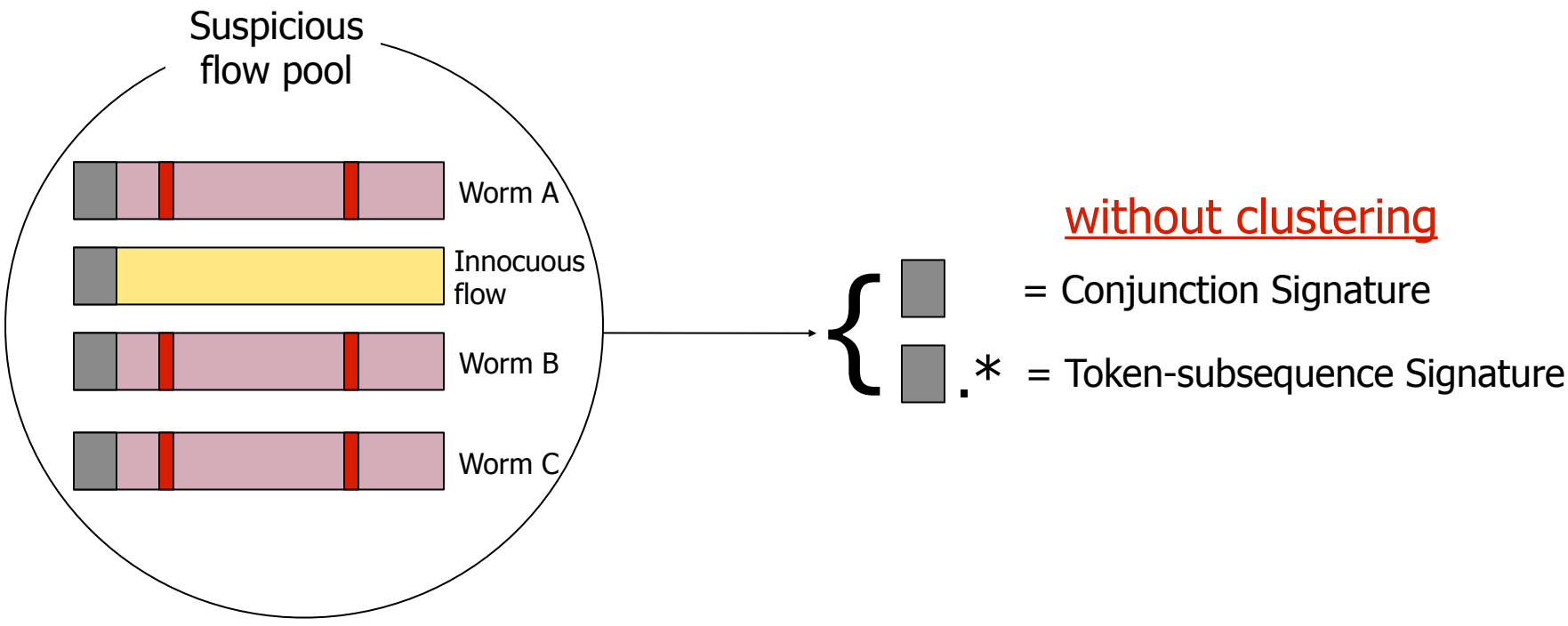
{PF, TI-1, TI-2, TI-3}

Token-subsequence Signature

PF .* TI-1 .* TI-3

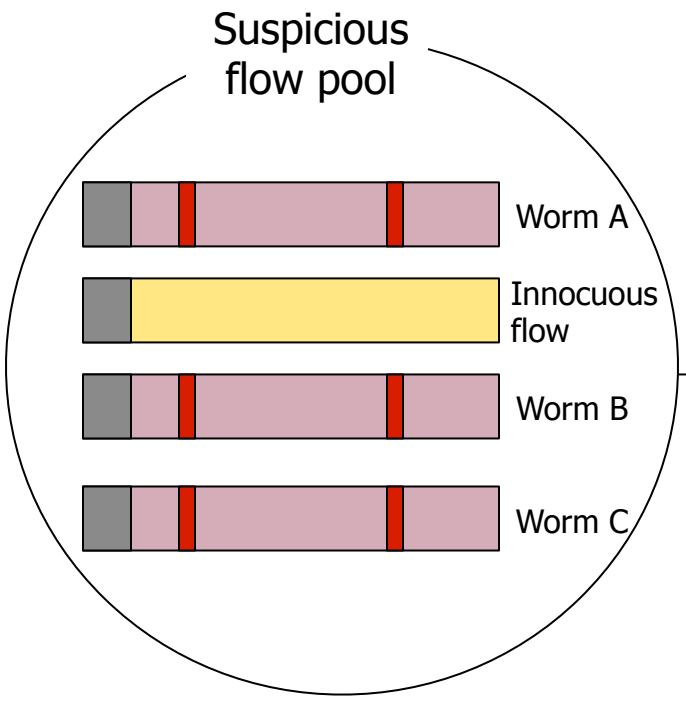
Case study: POLYGRAPH

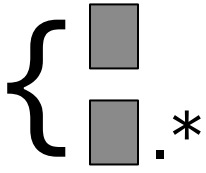
- Conjunction and Token-subsequence signatures are not resiliant to noise in the Suspicious flow pool



Case study: POLYGRAPH

- Conjunction and Token-subsequence signatures are not resiliant to noise in the Suspicious flow pool





= Conjunction Signature

= Token-subsequence Signature

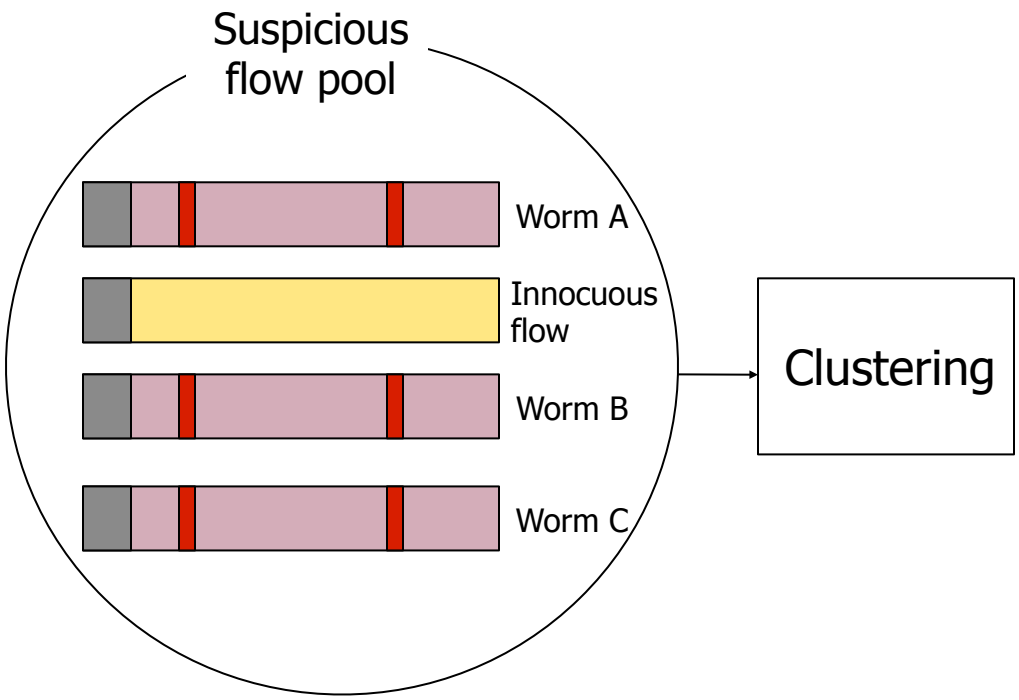
without clustering

⇩

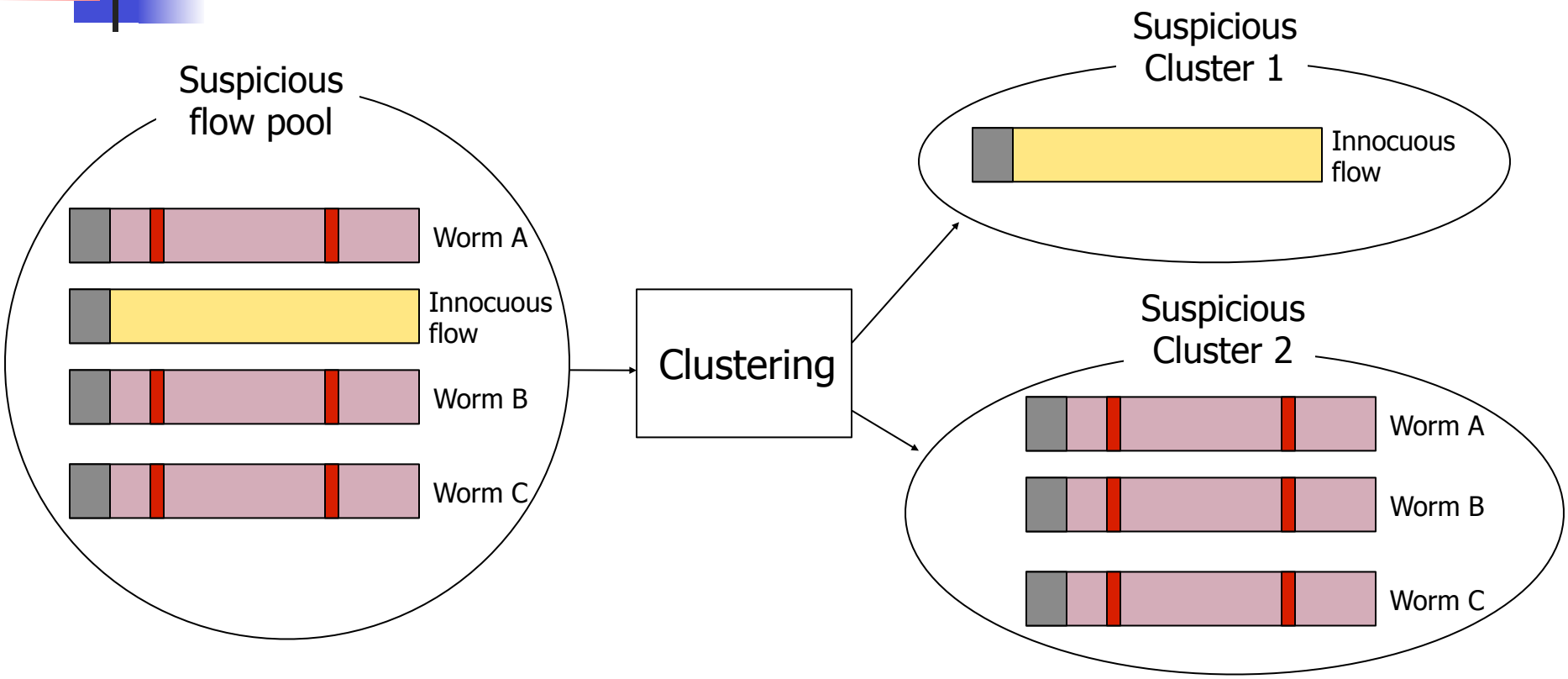
too many FP

The signatures will be **disregarded**

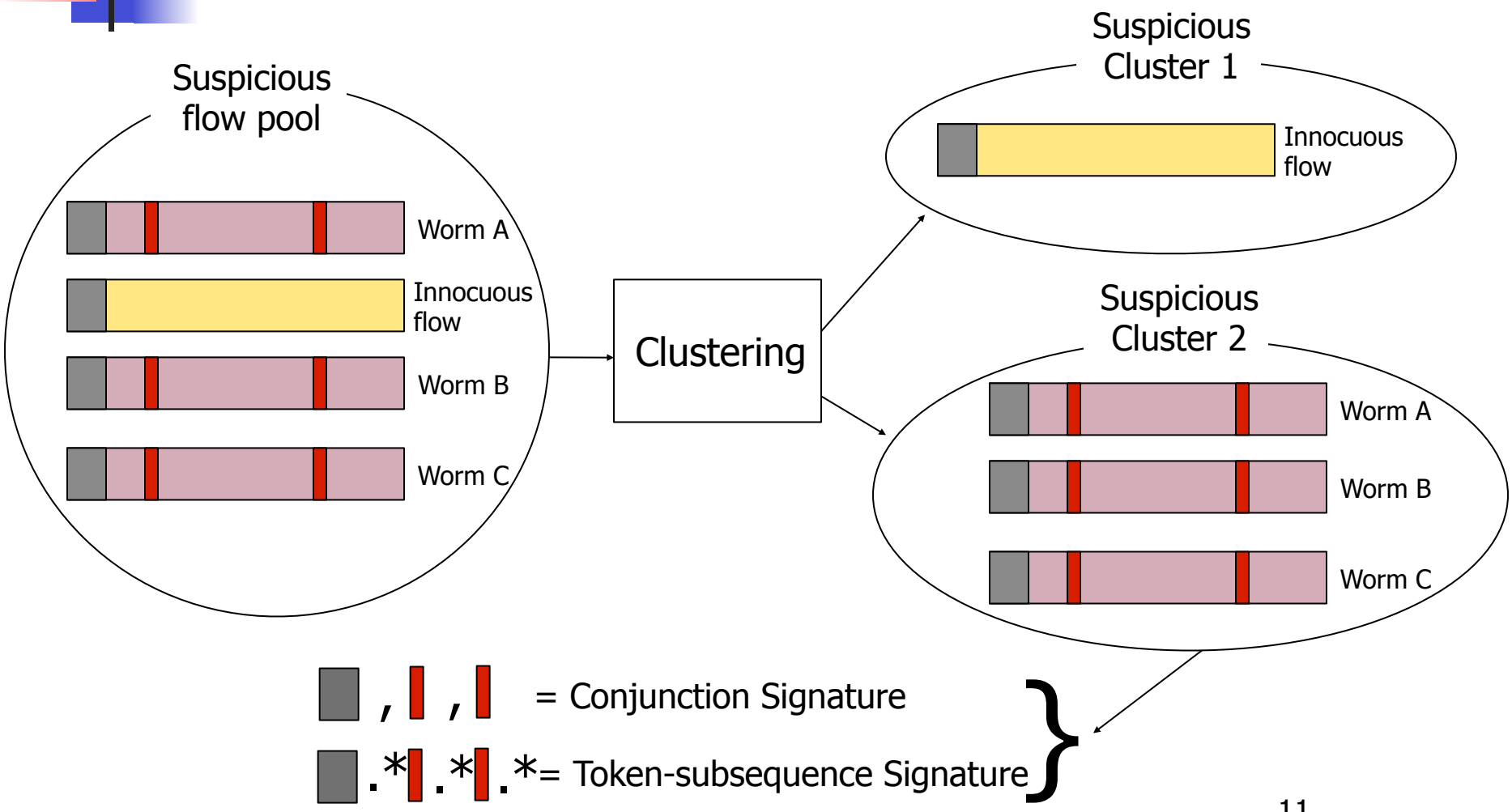
Hierarchical Clustering



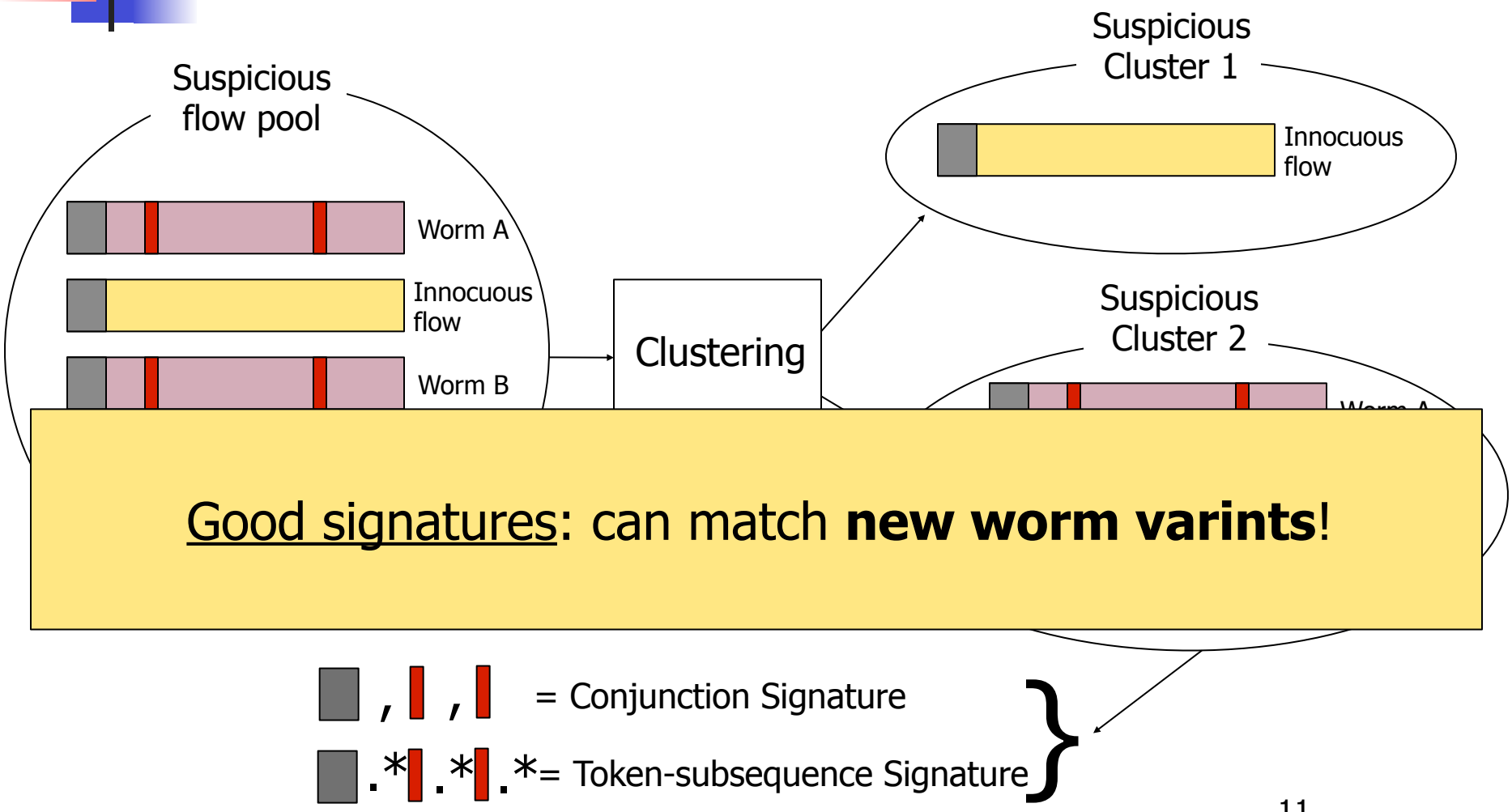
Hierarchical Clustering



Hierarchical Clustering



Hierarchical Clustering

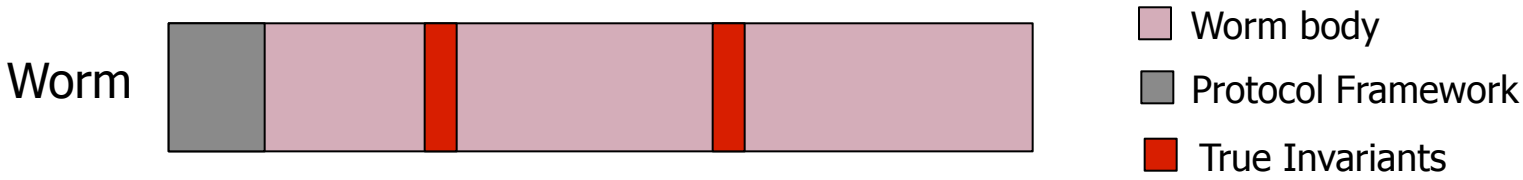


Misleading Conjunction and Token-Subsequence Signatures

- Objective: Mislead the Hierarchical Clustering algorithm so that signature will produce False Negatives

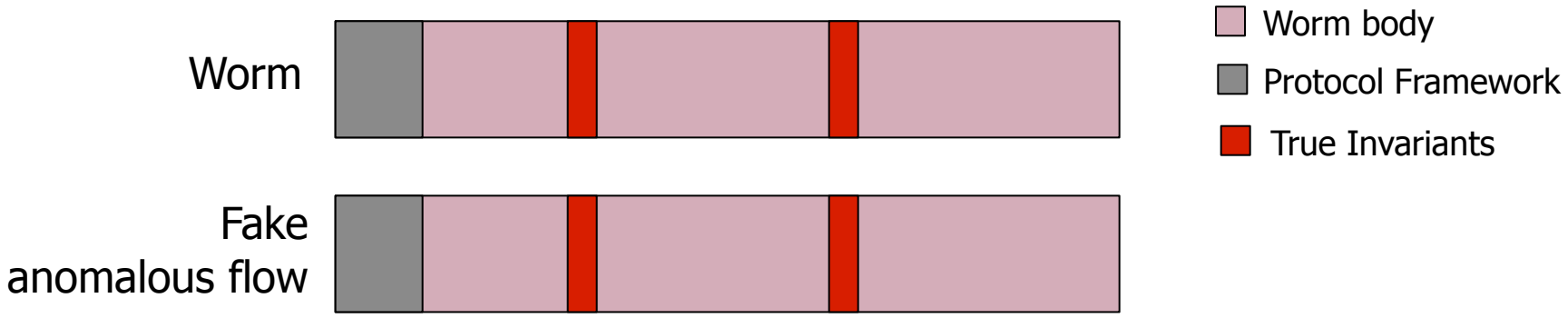
Misleading Conjunction and Token-Subsequence Signatures

- Objective: Mislead the Hierarchical Clustering algorithm so that signature will produce False Negatives



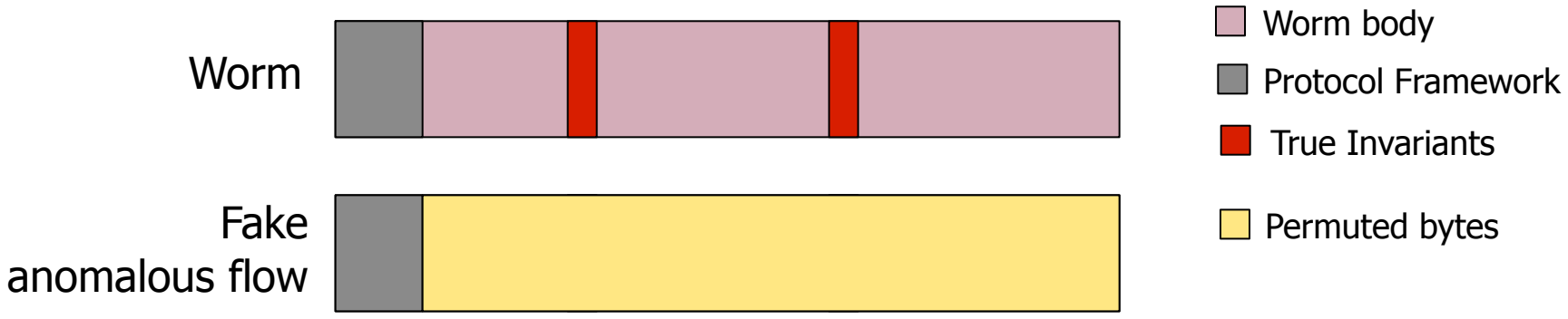
Misleading Conjunction and Token-Subsequence Signatures

- Objective: Mislead the Hierarchical Clustering algorithm so that signature will produce False Negatives



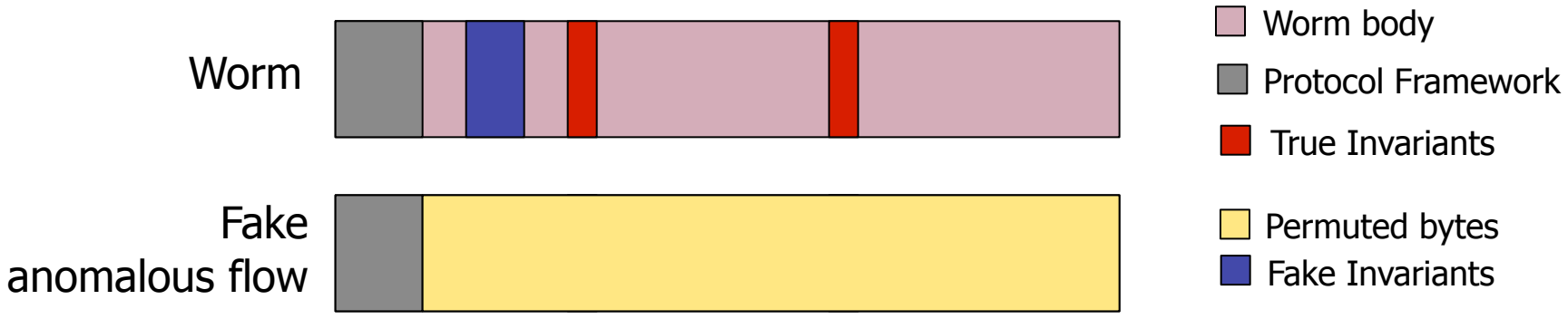
Misleading Conjunction and Token-Subsequence Signatures

- Objective: Mislead the Hierarchical Clustering algorithm so that signature will produce False Negatives



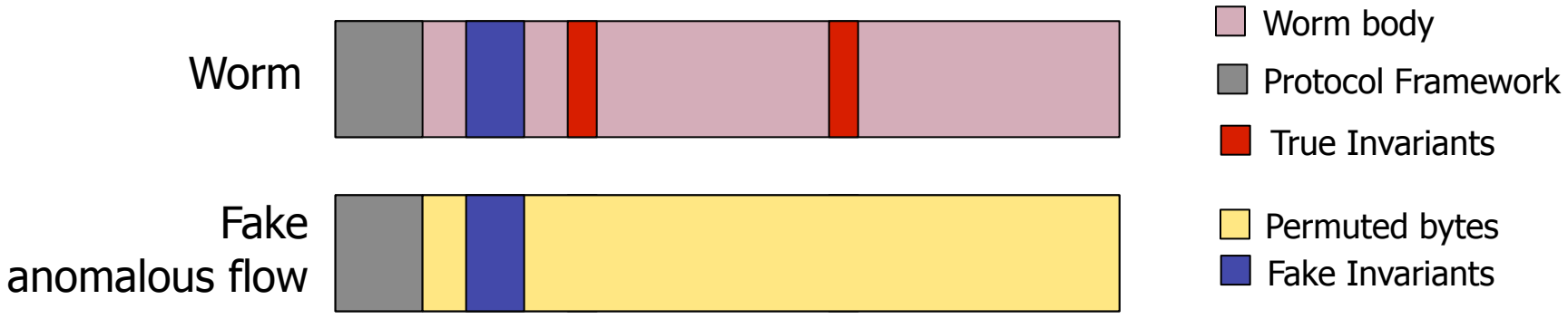
Misleading Conjunction and Token-Subsequence Signatures

- Objective: Mislead the Hierarchical Clustering algorithm so that signature will produce False Negatives



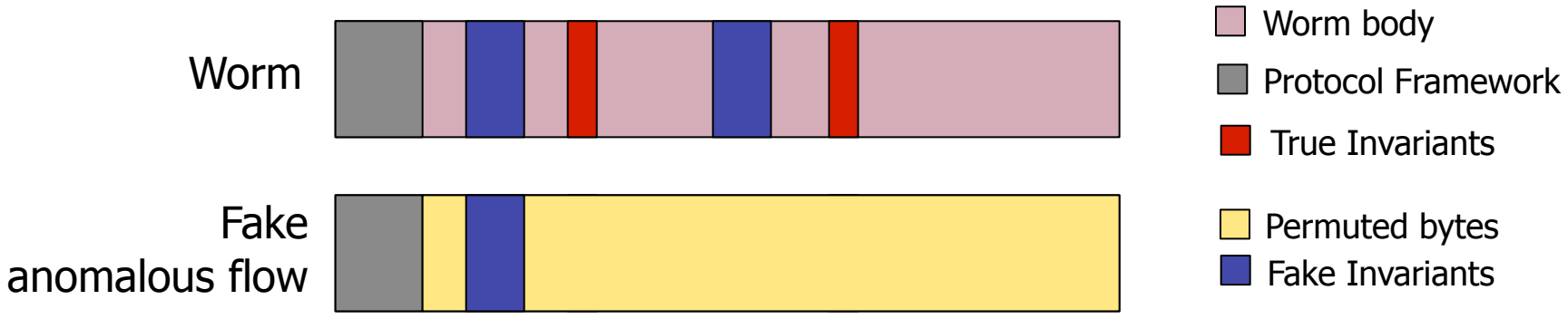
Misleading Conjunction and Token-Subsequence Signatures

- Objective: Mislead the Hierarchical Clustering algorithm so that signature will produce False Negatives



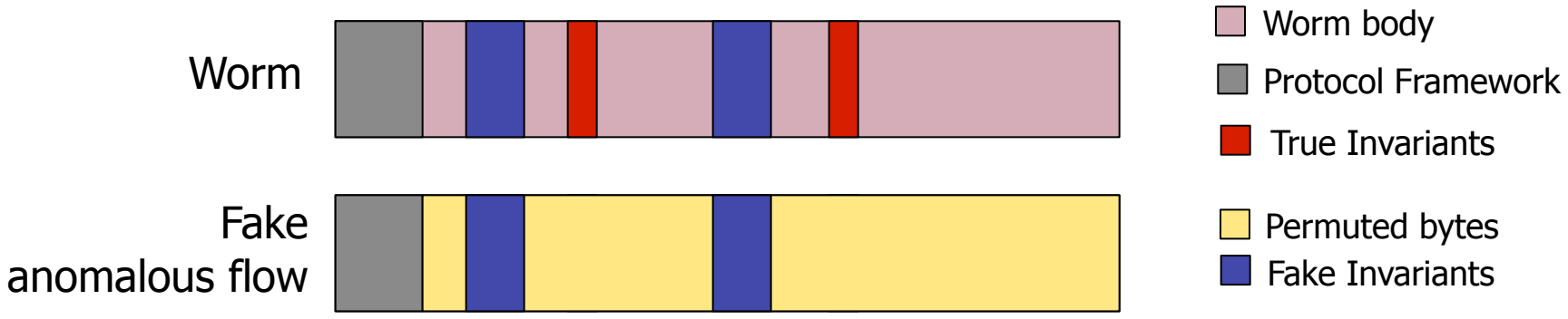
Misleading Conjunction and Token-Subsequence Signatures

- Objective: Mislead the Hierarchical Clustering algorithm so that signature will produce False Negatives



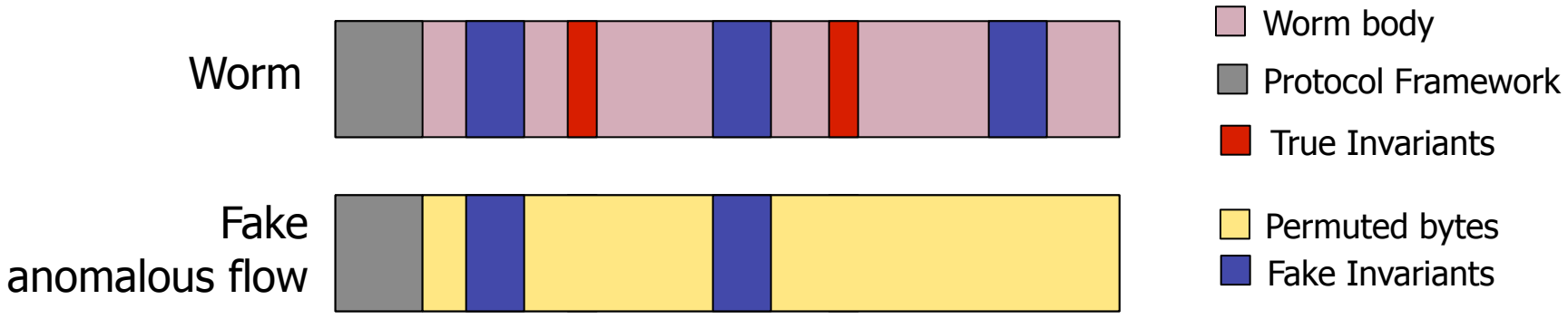
Misleading Conjunction and Token-Subsequence Signatures

- Objective: Mislead the Hierarchical Clustering algorithm so that signature will produce False Negatives



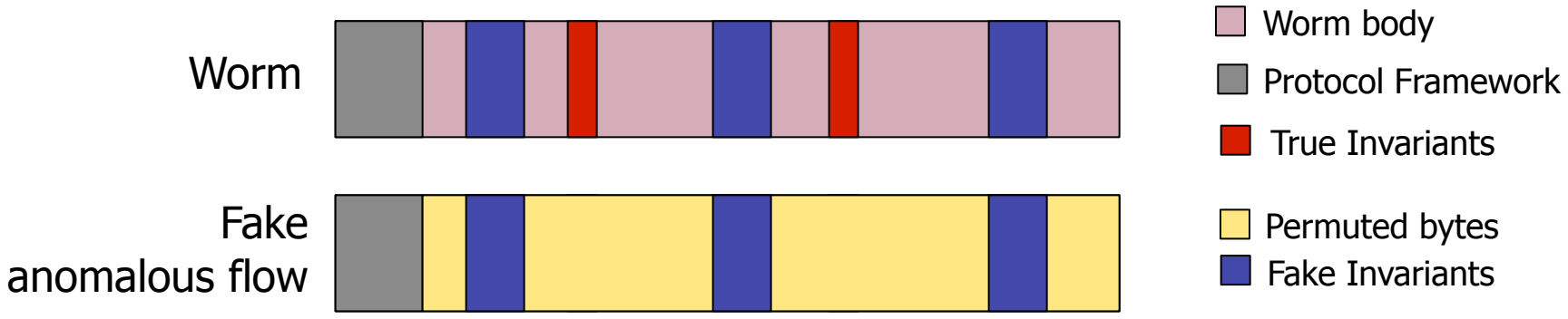
Misleading Conjunction and Token-Subsequence Signatures

- Objective: Mislead the Hierarchical Clustering algorithm so that signature will produce False Negatives



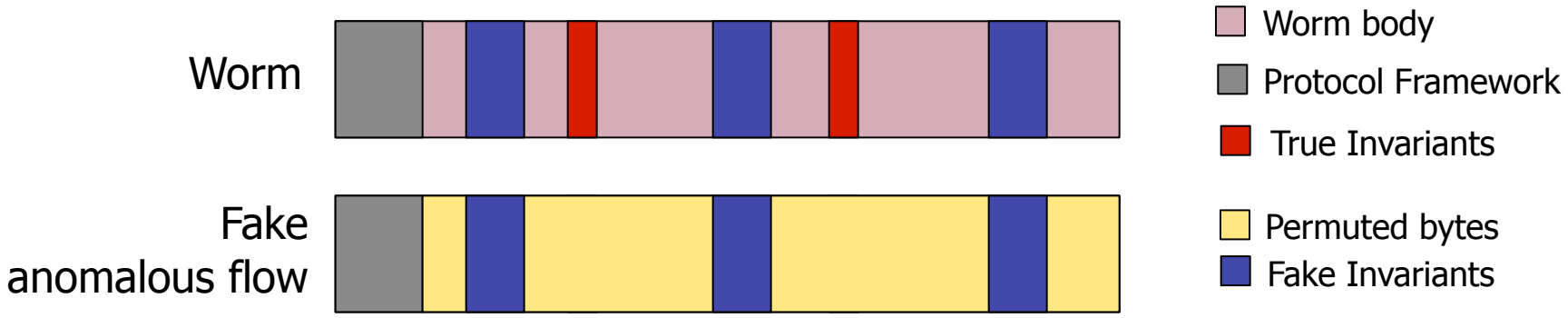
Misleading Conjunction and Token-Subsequence Signatures

- Objective: Mislead the Hierarchical Clustering algorithm so that signature will produce False Negatives



Misleading Conjunction and Token-Subsequence Signatures

- Objective: Mislead the Hierarchical Clustering algorithm so that signature will produce False Negatives



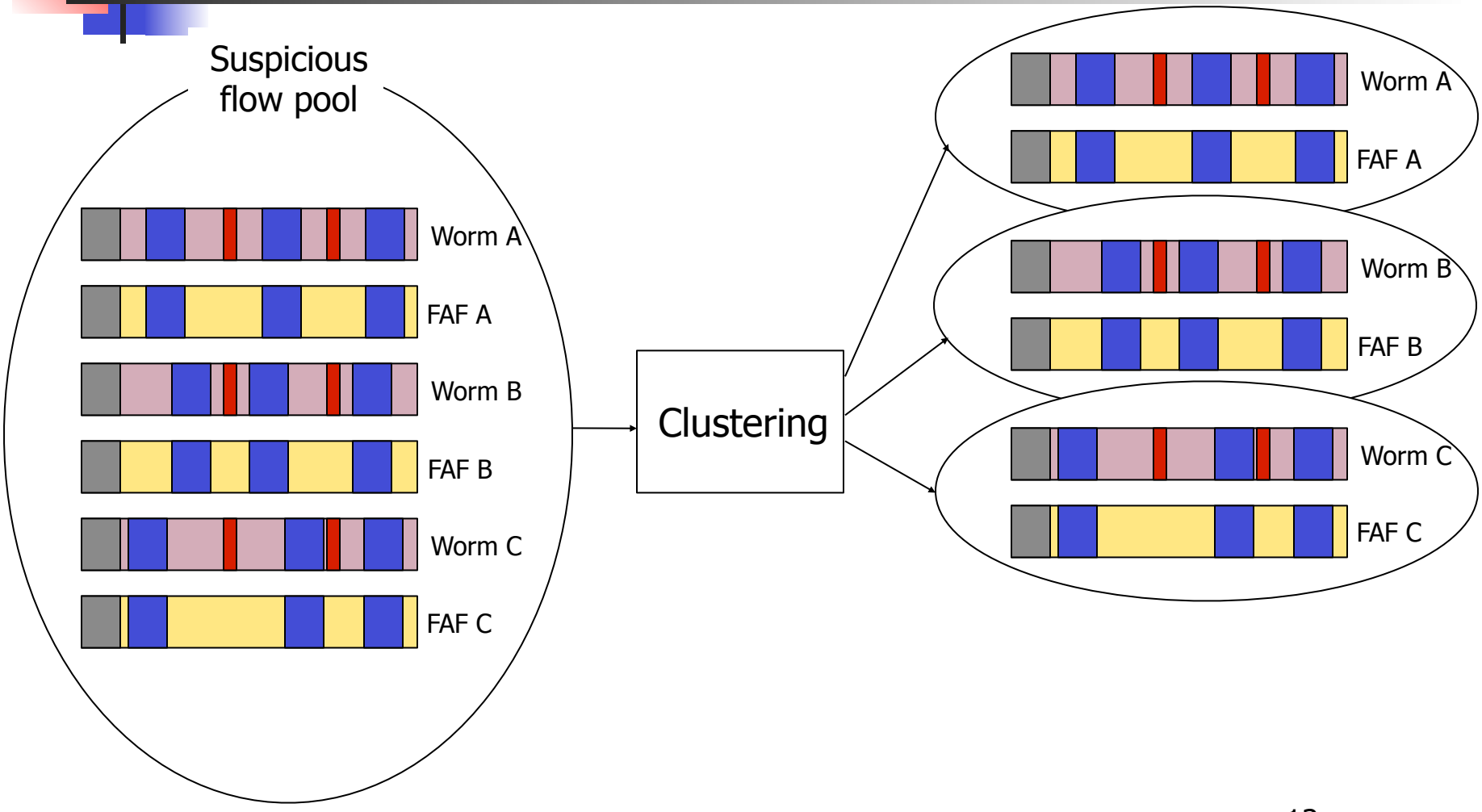
P(FI | innocuous flow) < P(TI | innocuous flow)

=

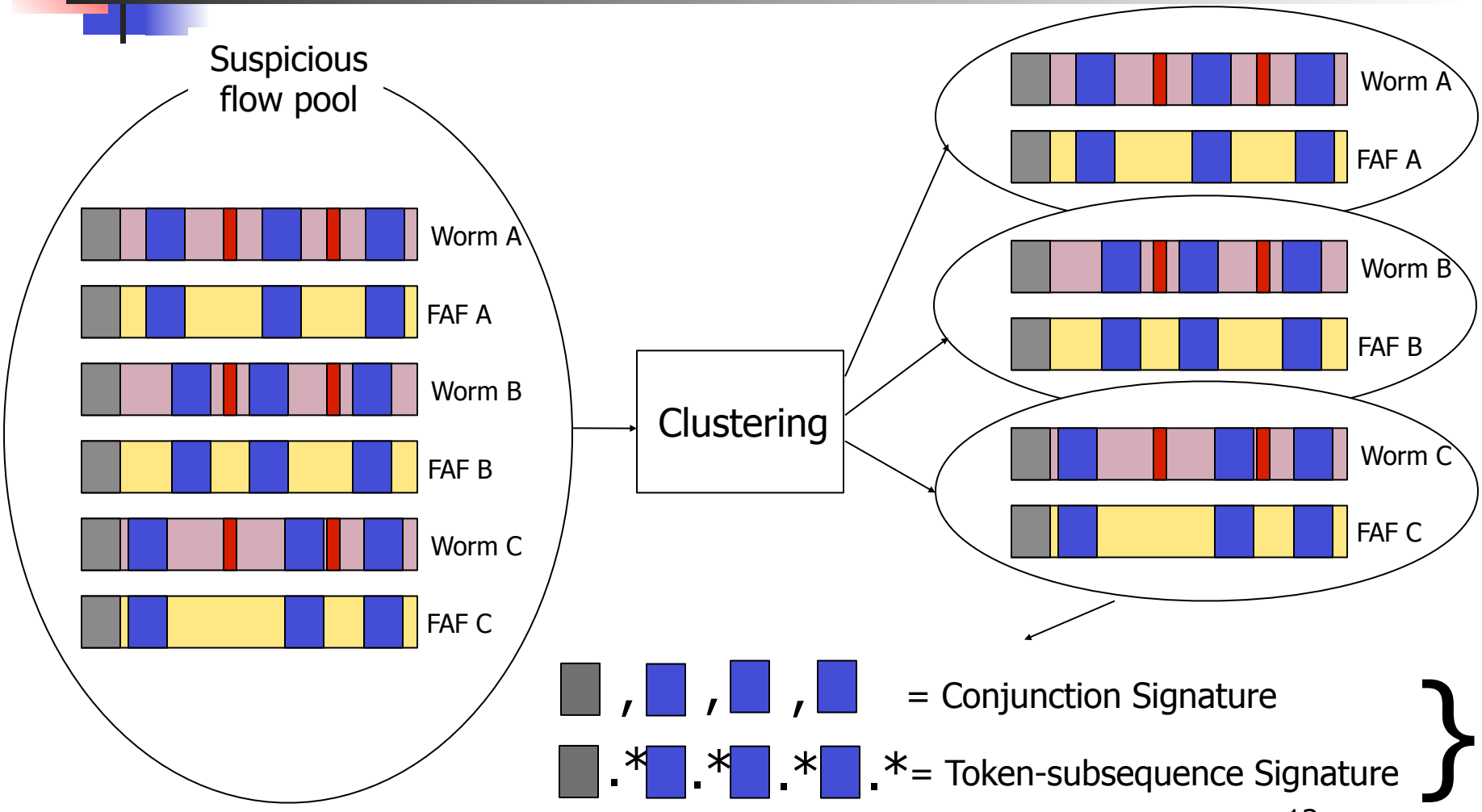
P(false positive | sig(FI)) < P(false positive | sig(TI))

Hierarchical Clustering

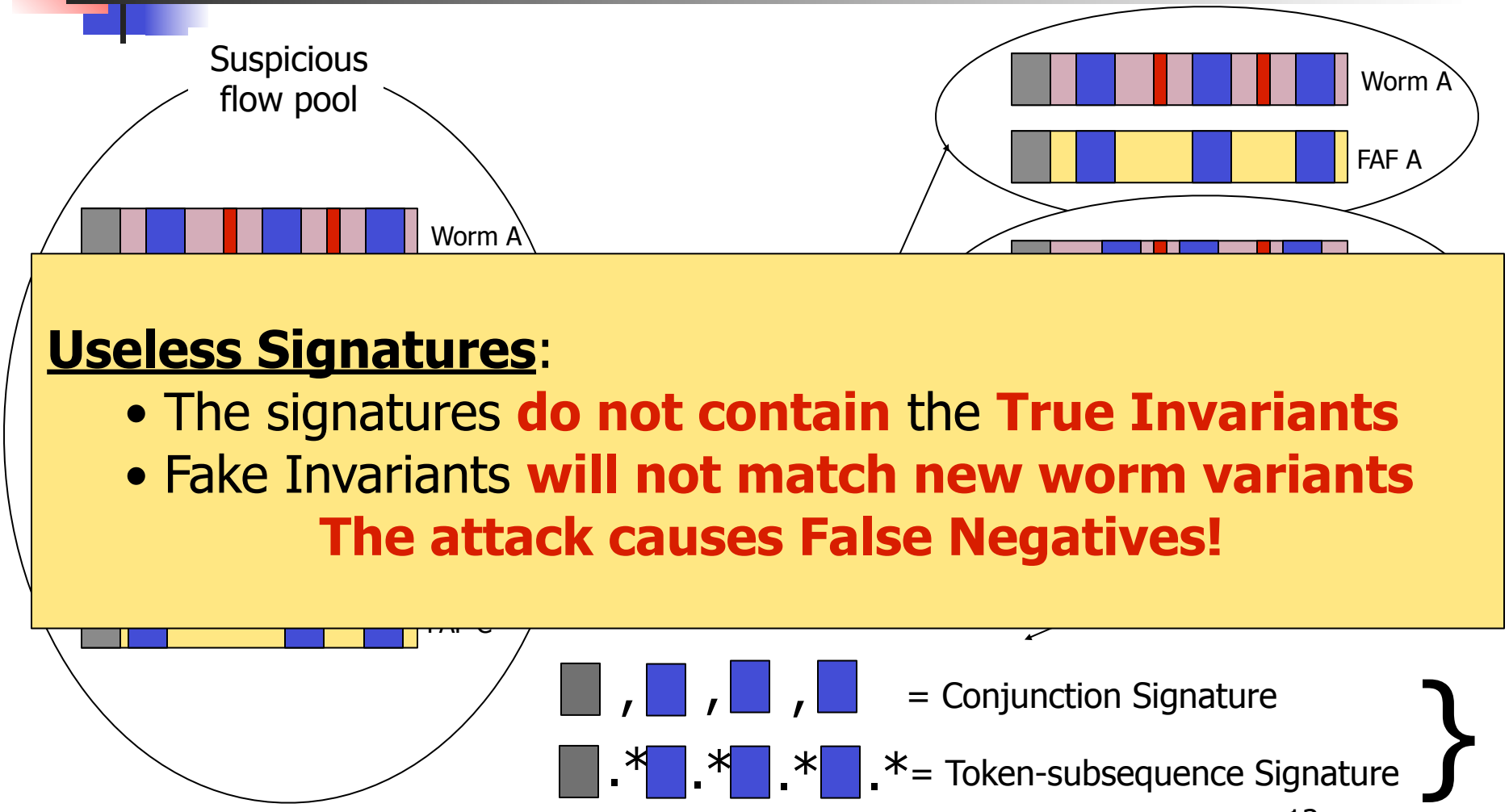
Hierarchical Clustering



Hierarchical Clustering



Hierarchical Clustering



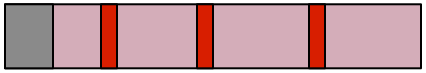
<u>Useless Signatures</u>:

- The signatures **do not contain** the **True Invariants**
- Fake Invariants **will not match new worm variants**

The attack causes False Negatives!

Case study: POLYGRAH

- Bayes signatures: All the tokens common to at least K out of the total number of suspicious flows N are extracted
  - For each token t_j
    - $P_{sf} = P(t_j \mid \text{Suspicious Flow})$
    - $P_{if} = P(t_j \mid \text{Innocuous Flow})$
    - $\lambda_j = \log(P_{sf} / P_{if})$



$$\{<PF, \lambda_{PF}>, <TI\text{-}1, \lambda_{TI\text{-}1}>, <TI\text{-}2, \lambda_{TI\text{-}2}><TI\text{-}3, \lambda_{TI\text{-}3}>\}$$

Case study: POLYGRAH

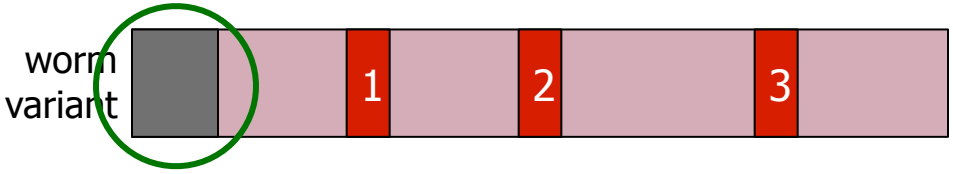
- Bayes signatures: All the tokens common to at least K out of the total number of suspicious flows N are extracted
  - For each token t_j
    - $P_{sf} = P(t_j \mid \text{Suspicious Flow})$
    - $P_{if} = P(t_j \mid \text{Innocuous Flow})$
    - $\lambda_j = \log(P_{sf} / P_{if})$

$$\{<PF, \lambda_{PF}>, <TI\text{-}1, \lambda_{TI\text{-}1}>, <TI\text{-}2, \lambda_{TI\text{-}2}>< TI\text{-}3, \lambda_{TI\text{-}3}>\}$$

Case study: POLYGRAH

- Bayes signatures: All the tokens common to at least K out of the total number of suspicious flows N are extracted
  - For each token t_j
    - $P_{sf} = P(t_j \mid \text{Suspicious Flow})$
    - $P_{if} = P(t_j \mid \text{Innocuous Flow})$
    - $\lambda_j = \log(P_{sf} / P_{if})$

$$\{<PF, \lambda_{PF}>, <TI\text{-}1, \lambda_{TI\text{-}1}>, <TI\text{-}2, \lambda_{TI\text{-}2}><TI\text{-}3, \lambda_{TI\text{-}3}>\}$$



Case study: POLYGRAH

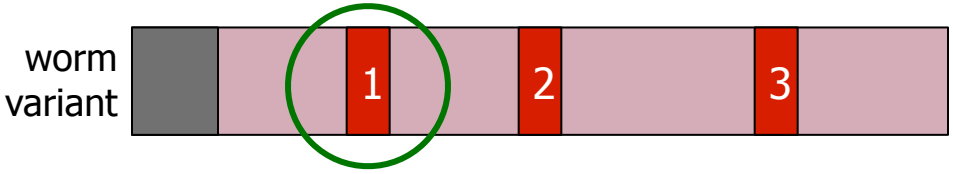
- Bayes signatures: All the tokens common to at least K out of the total number of suspicious flows N are extracted
  - For each token t_j
    - $P_{sf} = P(t_j \mid \text{Suspicious Flow})$
    - $P_{if} = P(t_j \mid \text{Innocuous Flow})$
    - $\lambda_j = \log(P_{sf} / P_{if})$

$$\{<PF, \lambda_{PF}>, <TI\text{-}1, \lambda_{TI\text{-}1}>, <TI\text{-}2, \lambda_{TI\text{-}2}>< TI\text{-}3, \lambda_{TI\text{-}3}>\}$$

Case study: POLYGRAH

- Bayes signatures: All the tokens common to at least K out of the total number of suspicious flows N are extracted
  - For each token t_j
    - $P_{sf} = P(t_j \mid \text{Suspicious Flow})$
    - $P_{if} = P(t_j \mid \text{Innocuous Flow})$
    - $\lambda_j = \log(P_{sf} / P_{if})$

$$\{<PF, \lambda_{PF}>, <TI\text{-}1, \lambda_{TI\text{-}1}>, <TI\text{-}2, \lambda_{TI\text{-}2}><TI\text{-}3, \lambda_{TI\text{-}3}>\}$$



Case study: POLYGRAH

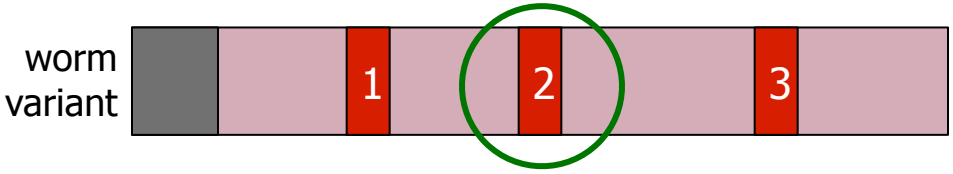
- Bayes signatures: All the tokens common to at least K out of the total number of suspicious flows N are extracted
  - For each token t_j
    - $P_{sf} = P(t_j \mid \text{Suspicious Flow})$
    - $P_{if} = P(t_j \mid \text{Innocuous Flow})$
    - $\lambda_j = \log(P_{sf} / P_{if})$

$$\{<PF, \lambda_{PF}>, <TI\text{-}1, \lambda_{TI\text{-}1}>, <TI\text{-}2, \lambda_{TI\text{-}2}>< TI\text{-}3, \lambda_{TI\text{-}3}>\}$$

Case study: POLYGRAH

- Bayes signatures: All the tokens common to at least K out of the total number of suspicious flows N are extracted
  - For each token t_j
    - $P_{sf} = P(t_j \mid \text{Suspicious Flow})$
    - $P_{if} = P(t_j \mid \text{Innocuous Flow})$
    - $\lambda_j = \log(P_{sf} / P_{if})$

$$\{<PF, \lambda_{PF}>, <TI\text{-}1, \lambda_{TI\text{-}1}>, <TI\text{-}2, \lambda_{TI\text{-}2}><TI\text{-}3, \lambda_{TI\text{-}3}>\}$$



Case study: POLYGRAH

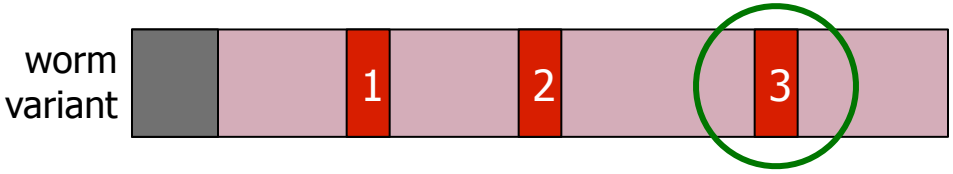
- Bayes signatures: All the tokens common to at least K out of the total number of suspicious flows N are extracted
  - For each token t_j
    - $P_{sf} = P(t_j \mid \text{Suspicious Flow})$
    - $P_{if} = P(t_j \mid \text{Innocuous Flow})$
    - $\lambda_j = \log(P_{sf} / P_{if})$

$$\{<PF, \lambda_{PF}>, <TI\text{-}1, \lambda_{TI\text{-}1}>, <TI\text{-}2, \lambda_{TI\text{-}2}>< TI\text{-}3, \lambda_{TI\text{-}3}>\}$$

Case study: POLYGRAH

- Bayes signatures: All the tokens common to at least K out of the total number of suspicious flows N are extracted
  - For each token t_j
    - $P_{sf} = P(t_j \mid \text{Suspicious Flow})$
    - $P_{if} = P(t_j \mid \text{Innocuous Flow})$
    - $\lambda_j = \log(P_{sf} / P_{if})$

$$\{<PF, \lambda_{PF}>, <TI\text{-}1, \lambda_{TI\text{-}1}>, <TI\text{-}2, \lambda_{TI\text{-}2}><\underline{TI\text{-}3, \lambda_{TI\text{-}3}}>\}$$



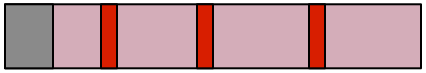
Case study: POLYGRAH

- Bayes signatures: All the tokens common to at least K out of the total number of suspicious flows N are extracted
  - For each token t_j
    - $P_{sf} = P(t_j \mid \text{Suspicious Flow})$
    - $P_{if} = P(t_j \mid \text{Innocuous Flow})$
    - $\lambda_j = \log(P_{sf} / P_{if})$

$$\{<PF, \lambda_{PF}>, <TI\text{-}1, \lambda_{TI\text{-}1}>, <TI\text{-}2, \lambda_{TI\text{-}2}>< TI\text{-}3, \lambda_{TI\text{-}3}>\}$$

Case study: POLYGRAH

- Bayes signatures: All the tokens common to at least K out of the total number of suspicious flows N are extracted
  - For each token t_j
    - $P_{sf} = P(t_j \mid \text{Suspicious Flow})$
    - $P_{if} = P(t_j \mid \text{Innocuous Flow})$
    - $\lambda_j = \log(P_{sf} / P_{if})$



$$\{<PF, \lambda_{PF}>, <TI\text{-}1, \lambda_{TI\text{-}1}>, <TI\text{-}2, \lambda_{TI\text{-}2}>< TI\text{-}3, \lambda_{TI\text{-}3}>\}$$

If $\Lambda = \Sigma_k \lambda_k > \theta$

⇩

The flow is a worm!

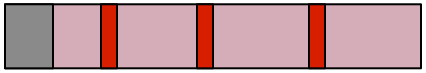
Case study: POLYGRAH

- Bayes signatures: All the tokens common to at least K out of the total number of suspicious flows N are extracted
  - For each token t_j
    - $P_{sf} = P(t_j \mid \text{Suspicious Flow})$
    - $P_{if} = P(t_j \mid \text{Innocuous Flow})$
    - $\lambda_j = \log(P_{sf} / P_{if})$

$$\{<PF, \lambda_{PF}>, <TI\text{-}1, \lambda_{TI\text{-}1}>, <TI\text{-}2, \lambda_{TI\text{-}2}>< TI\text{-}3, \lambda_{TI\text{-}3}>\}$$

Case study: POLYGRAH

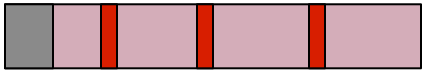
- Bayes signatures: All the tokens common to at least K out of the total number of suspicious flows N are extracted
  - For each token t_j
    - $P_{sf} = P(t_j \mid \text{Suspicious Flow})$
    - $P_{if} = P(t_j \mid \text{Innocuous Flow})$
    - $\lambda_j = \log(P_{sf} / P_{if})$



$$\{<PF, \lambda_{PF}>, <TI\text{-}1, \lambda_{TI\text{-}1}>, <TI\text{-}2, \lambda_{TI\text{-}2}><TI\text{-}3, \lambda_{TI\text{-}3}>\}$$

Case study: POLYGRAH

- Bayes signatures: All the tokens common to at least K out of the total number of suspicious flows N are extracted
  - For each token t_j
    - $P_{sf} = P(t_j \mid \text{Suspicious Flow})$
    - $P_{if} = P(t_j \mid \text{Innocuous Flow})$
    - $\lambda_j = \log(P_{sf} / P_{if})$



Worm

$$\{<PF, \lambda_{PF}>, <TI\text{-}1, \lambda_{TI\text{-}1}>, <TI\text{-}2, \lambda_{TI\text{-}2}>< TI\text{-}3, \lambda_{TI\text{-}3}>\}$$

Innocuous flow

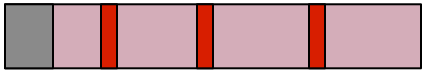
Case study: POLYGRAH

- Bayes signatures: All the tokens common to at least K out of the total number of suspicious flows N are extracted
  - For each token t_j
    - $P_{sf} = P(t_j \mid \text{Suspicious Flow})$
    - $P_{if} = P(t_j \mid \text{Innocuous Flow})$
    - $\lambda_j = \log(P_{sf} / P_{if})$

$$\{<PF, \lambda_{PF}>, <TI\text{-}1, \lambda_{TI\text{-}1}>, <TI\text{-}2, \lambda_{TI\text{-}2}><TI\text{-}3, \lambda_{TI\text{-}3}>\}$$

Case study: POLYGRAH

- Bayes signatures: All the tokens common to at least K out of the total number of suspicious flows N are extracted
  - For each token t_j
    - $P_{sf} = P(t_j \mid \text{Suspicious Flow})$
    - $P_{if} = P(t_j \mid \text{Innocuous Flow})$
    - $\lambda_j = \log(P_{sf} / P_{if})$



Worm

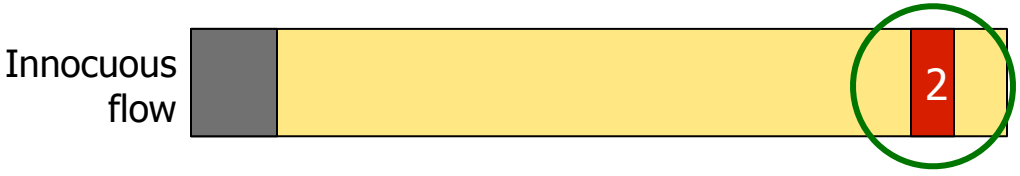
$$\{<PF, \lambda_{PF}>, <TI\text{-}1, \lambda_{TI\text{-}1}>, <TI\text{-}2, \lambda_{TI\text{-}2}>< TI\text{-}3, \lambda_{TI\text{-}3}>\}$$

Innocuous flow

Case study: POLYGRAH

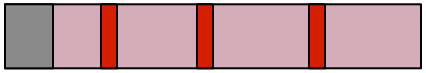
- Bayes signatures: All the tokens common to at least K out of the total number of suspicious flows N are extracted
  - For each token t_j
    - $P_{sf} = P(t_j \mid \text{Suspicious Flow})$
    - $P_{if} = P(t_j \mid \text{Innocuous Flow})$
    - $\lambda_j = \log(P_{sf} / P_{if})$

$$\{<PF, \lambda_{PF}>, <TI\text{-}1, \lambda_{TI\text{-}1}>, <TI\text{-}2, \lambda_{TI\text{-}2}><TI\text{-}3, \lambda_{TI\text{-}3}>\}$$



Case study: POLYGRAH

- Bayes signatures: All the tokens common to at least K out of the total number of suspicious flows N are extracted
  - For each token t_j
    - $P_{sf} = P(t_j \mid \text{Suspicious Flow})$
    - $P_{if} = P(t_j \mid \text{Innocuous Flow})$
    - $\lambda_j = \log(P_{sf} / P_{if})$



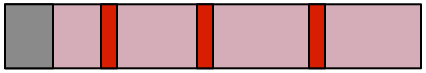
Worm

$$\{<PF, \lambda_{PF}>, <TI\text{-}1, \lambda_{TI\text{-}1}>, <TI\text{-}2, \lambda_{TI\text{-}2}>< TI\text{-}3, \lambda_{TI\text{-}3}>\}$$

Innocuous flow

Case study: POLYGRAH

- Bayes signatures: All the tokens common to at least K out of the total number of suspicious flows N are extracted
  - For each token t_j
    - $P_{sf} = P(t_j \mid \text{Suspicious Flow})$
    - $P_{if} = P(t_j \mid \text{Innocuous Flow})$
    - $\lambda_j = \log(P_{sf} / P_{if})$



Worm

$$\{<PF, \lambda_{PF}>, <TI\text{-}1, \lambda_{TI\text{-}1}>, <TI\text{-}2, \lambda_{TI\text{-}2}>< TI\text{-}3, \lambda_{TI\text{-}3}>\}$$

Innocuous flow

If $\Lambda = \Sigma_k \lambda_k < \theta$

$\Downarrow$

The flow is Innocuous!

Misleading Bayes Signatures

- Consider a string ν of length n that is present in the Innocuous pool with probability
  - $0.05 < P(\nu \mid \text{Innocuous Flow}) < 0.20$
- If all the fake anomalous flows contain ν, the string will be present in 50% of the suspicious flows
- Thus, the extracted signature will contain ν and the related score λ_ν will be
  - $\log(0.5/0.20) < \lambda_\nu < \log(0.5/0.05)$
- This means that an innocuous flow containing ν will receive a total score $\Lambda >= \lambda_\nu$

Misleading Bayes Signatures

- E.g., ν = "Pragma: no-cache"
  - Suppose P(ν | Innocuous Flow) = 0.094
  - λ_ν = log(0.5/0.094)
- Suppose signature is
  - <GET,λ_1>; < HTTP/1.1,λ_2>; … ; < Pragma: no-cache,λ_ν>;
- Given an Innocuous flow:
  - GET .* HTTP/1.1 .* Pragma: no-cache .*
  - $\Lambda = \Sigma_k \lambda_k + \lambda_\nu >= \lambda_\nu$

Misleading Bayes Signatures

- E.g., ν = "Pragma: no-cache"
  - Suppose P(ν | Innocuous Flow) = 0.094
  - λ_ν = log(0.5/0.094)
- Suppose signature is
  - <GET,λ_1>; < HTTP/1.1,λ_2>; … ; < Pragma: no-cache,λ_ν>;
- Given an Innocuous flow:
  - $\overset{\lambda_1}{\boxed{\text{GET}}}$.* $\overset{\lambda_2}{\boxed{\text{HTTP/1.1}}}$.* Pragma: no-cache .*
  - $\Lambda = \boxed{\Sigma_k \lambda_k} + \lambda_\nu >= \lambda_\nu$

Misleading Bayes Signatures

- E.g., ν = "Pragma: no-cache"
  - Suppose P(ν | Innocuous Flow) = 0.094
  - λ_ν = log(0.5/0.094)
- Suppose signature is
  - <GET,λ_1>; < HTTP/1.1,λ_2>; … ; < Pragma: no-cache,λ_ν>;
- Given an Innocuous flow:
  - $\overset{\lambda_1}{\boxed{\text{GET}}}$.* $\overset{\lambda_2}{\boxed{\text{HTTP/1.1}}}$.* $\overset{\lambda_\nu}{\boxed{\text{Pragma: no-cache}}}$.*
  - $\Lambda = \boxed{\Sigma_k \lambda_k} + \boxed{\lambda_\nu} >= \lambda_\nu$

Misleading Bayes Signatures

- E.g., ν = "Pragma: no-cache"
  - Suppose $P(\nu \mid \text{Innocuous Flow}) = 0.094$
  - $\lambda_\nu = \log(0.5/0.094)$
- Suppose signature is
  - $\langle \text{GET}, \lambda_1 \rangle$; $\langle \text{HTTP/1.1}, \lambda_2 \rangle$; … ; $\langle \text{Pragma: no-cache}, \lambda_\nu \rangle$;
- Given an Innocuous flow:
  - $\overset{\lambda_1}{\boxed{\text{GET}}}$.* $\overset{\lambda_2}{\boxed{\text{HTTP/1.1}}}$.* $\overset{\lambda_\nu}{\boxed{\text{Pragma: no-cache}}}$.*
  - $\Lambda = \boxed{\Sigma_k \lambda_k} + \boxed{\lambda_\nu} >= \lambda_\nu$ ⟹ <u>if $\Lambda < \theta$</u> ⟹ **<u>No false positives</u>**

"Score multiplier" effect

- Spliting ν into all the substrings of length m<n
  - E.g., "Pragma: no", "ragma: no-", "agma: no-c", etc.
  - If !(m<<n) then $p(\nu_{i,i+m} \mid IF) \sim= p(\nu \mid IF), \forall i$
- Injecting all the substring $\nu_{i,i+m}$ of ν into the fake anomalous flows, they will be all considered as tokens in the signature (50% occurrence freq.)
  - $\lambda_{\nu_{i,i+m}} \sim= \lambda_\nu$
- An innocuous flow that contains ν contains also all the $\nu_{i,i+m}$ tokens
- A **score multiplier effect** is obtained for the innocuous flows which contain ν
  - GET .* HTTP/1.1 .* Pragma: no-cache .*
  - $\Lambda = \Sigma_k \lambda_k + \Sigma_i \lambda_{\nu i} >> \lambda_\nu$

"Score multiplier" effect

- Spliting ν into all the substrings of length m<n
  - E.g., "Pragma: no", "ragma: no-", "agma: no-c", etc.
  - If !(m<<n) then $p(\nu_{i,i+m} \mid IF) \sim= p(\nu \mid IF), \forall i$
- Injecting all the substring $\nu_{i,i+m}$ of ν into the fake anomalous flows, they will be all considered as tokens in the signature (50% occurrence freq.)
  - $\lambda_{\nu_{i,i+m}} \sim= \lambda_\nu$
- An innocuous flow that contains ν contains also all the $\nu_{i,i+m}$ tokens
- A **score multiplier effect** is obtained for the innocuous flows which contain ν
  - GET .* HTTP/1.1 .* Pragma: no-cache .*
  - $\Lambda = \Sigma_k \lambda_k + \Sigma_i \lambda_{\nu i} >> \lambda_\nu$

"Score multiplier" effect

- Spliting ν into all the substrings of length m<n
  - E.g., "Pragma: no", "ragma: no-", "agma: no-c", etc.
  - If !(m<<n) then $p(\nu_{i,i+m} \mid IF) \sim= p(\nu \mid IF), \forall i$
- Injecting all the substring $\nu_{i,i+m}$ of ν into the fake anomalous flows, they will be all considered as tokens in the signature (50% occurrence freq.)
  - $\lambda_{\nu_{i,i+m}} \sim= \lambda_\nu$
- An innocuous flow that contains ν contains also all the $\nu_{i,i+m}$ tokens
- A **score multiplier effect** is obtained for the innocuous flows which contain ν
  - GET .* HTTP/1.1 .* Pragma: no-cache .*
  - $\Lambda = \Sigma_k \lambda_k + \Sigma_i \lambda_{\nu i} >> \lambda_\nu$

"Score multiplier" effect

- Spliting ν into all the substrings of length m<n
  - E.g., "Pragma: no", "ragma: no-", "agma: no-c", etc.
  - If !(m<<n) then $p(\nu_{i,i+m} \mid IF) \sim= p(\nu \mid IF), \forall i$
- Injecting all the substring $\nu_{i,i+m}$ of ν into the fake anomalous flows, they will be all considered as tokens in the signature (50% occurrence freq.)
  - $\lambda_{\nu i,i+m} \sim= \lambda_{\nu}$
- An innocuous flow that contains ν contains also all the $\nu_{i,i+m}$ tokens
- A **score multiplier effect** is obtained for the innocuous flows which contain ν
  - GET .* HTTP/1.1 .* Pragma: no-cache .*
  - $\Lambda = \Sigma_k \lambda_k + \Sigma_i \lambda_{\nu i} >> \lambda_{\nu}$

"Score multiplier" effect

- Spliting ν into all the substrings of length m<n
  - E.g., "Pragma: no", "ragma: no-", "agma: no-c", etc.
  - If !(m<<n) then $p(\nu_{i,i+m} \mid IF) \sim= p(\nu \mid IF), \forall i$
- Injecting all the substring $\nu_{i,i+m}$ of ν into the fake anomalous flows, they will be all considered as tokens in the signature (50% occurrence freq.)
  - $\lambda_{\nu_{i,i+m}} \sim= \lambda_\nu$
- An innocuous flow that contains ν contains also all the $\nu_{i,i+m}$ tokens
- A **score multiplier effect** is obtained for the innocuous flows which contain ν
  - GET .* HTTP/1.1 .* Pragma: no-cache .*
  - $\Lambda = \Sigma_k \lambda_k + \Sigma_i \lambda_{\nu i} >> \lambda_\nu$

"Score multiplier" effect

- Spliting ν into all the substrings of length m<n
  - E.g., "Pragma: no", "ragma: no-", "agma: no-c", etc.
  - If !(m<<n) then $p(\nu_{i,i+m} \mid IF) \sim= p(\nu \mid IF), \forall i$
- Injecting all the substring $\nu_{i,i+m}$ of ν into the fake anomalous flows, they will be all considered as tokens in the signature (50% occurrence freq.)
  - $\lambda_{\nu_{i,i+m}} \sim= \lambda_\nu$
- An innocuous flow that contains ν contains also all the $\nu_{i,i+m}$ tokens
- A **score multiplier effect** is obtained for the innocuous flows which contain ν
  - GET .* HTTP/1.1 .* Pragma: no-cache .*
  - $\Lambda = \Sigma_k \lambda_k + \Sigma_i \lambda_{\nu i} >> \lambda_\nu$

"Score multiplier" effect

- Spliting ν into all the substrings of length m<n
  - E.g., "Pragma: no", "ragma: no-", "agma: no-c", etc.
  - If !(m<<n) then $p(\nu_{i,i+m} \mid IF) \sim= p(\nu \mid IF), \forall i$
- Injecting all the substring $\nu_{i,i+m}$ of ν into the fake anomalous flows, they will be all considered as tokens in the signature (50% occurrence freq.)
  - $\lambda_{\nu_{i,i+m}} \sim= \lambda_\nu$
- An innocuous flow that contains ν contains also all the $\nu_{i,i+m}$ tokens
- A **score multiplier effect** is obtained for the innocuous flows which contain ν
  - GET .* HTTP/1.1 .* Pragma: no-cache .*
  - $\Lambda = \Sigma_k \lambda_k + \Sigma_i \lambda_{\nu i} >> \lambda_\nu$

"Score multiplier" effect

- Spliting ν into all the substrings of length m<n
  - E.g., "Pragma: no", "ragma: no-", "agma: no-c", etc.
  - If !(m<<n) then $p(\nu_{i,i+m} \mid IF) \sim= p(\nu \mid IF), \forall i$
- Injecting all the substring $\nu_{i,i+m}$ of ν into the fake anomalous flows, they will be all considered as tokens in the signature (50% occurrence freq.)
  - $\lambda_{\nu_{i,i+m}} \sim= \lambda_\nu$
- An innocuous flow that contains ν contains also all the $\nu_{i,i+m}$ tokens
- A **score multiplier effect** is obtained for the innocuous flows which contain ν
  - GET .* HTTP/1.1 .* Pragma: no-cache .*
  - $\Lambda = \Sigma_k \lambda_k + \Sigma_i \lambda_{\nu i} >> \lambda_\nu$

"Score multiplier" effect

- Spliting ν into all the substrings of length m<n
  - E.g., "Pragma: no", "ragma: no-", "agma: no-c", etc.
  - If !(m<<n) then $p(\nu_{i,i+m} \mid IF) \sim= p(\nu \mid IF), \forall i$
- Injecting all the substring $\nu_{i,i+m}$ of ν into the fake anomalous flows, they will be all considered as tokens in the signature (50% occurrence freq.)
  - $\lambda_{\nu_{i,i+m}} \sim= \lambda_\nu$
- An innocuous flow that contains ν contains also all the $\nu_{i,i+m}$ tokens
- A **score multiplier effect** is obtained for the innocuous flows which contain ν
  - GET .* HTTP/1.1 .* Pragma: no-cache .*
  - $\Lambda = \Sigma_k \lambda_k + \Sigma_i \lambda_{\nu i} >> \lambda_\nu$

"Score multiplier" effect

- Spliting ν into all the substrings of length m<n
  - E.g., "Pragma: no", "ragma: no-", "agma: no-c", etc.
  - If !(m<<n) then $p(\nu_{i,i+m} \mid IF) \sim= p(\nu \mid IF), \forall i$
- Injecting all the substring $\nu_{i,i+m}$ of ν into the fake anomalous flows, they will be all considered as tokens in the signature (50% occurrence freq.)
  - $\lambda_{\nu_{i,i+m}} \sim= \lambda_\nu$
- An innocuous flow that contains ν contains also all the $\nu_{i,i+m}$ tokens
- A **score multiplier effect** is obtained for the innocuous flows which contain ν
  - GET .* HTTP/1.1 .* Pragma: no-cache .*
  - $\Lambda = \Sigma_k \lambda_k + \Sigma_i \lambda_{\nu i} >> \lambda_\nu$

"Score multiplier" effect

- Spliting ν into all the substrings of length m<n
  - E.g., "Pragma: no", "ragma: no-", "agma: no-c", etc.
  - If !(m<<n) then $p(\nu_{i,i+m} \mid IF) \sim= p(\nu \mid IF), \forall i$
- Injecting all the substring $\nu_{i,i+m}$ of ν into the fake anomalous flows, they will be all considered as tokens in the signature (50% occurrence freq.)
  - $\lambda_{\nu_{i,i+m}} \sim= \lambda_\nu$
- An innocuous flow that contains ν contains also all the $\nu_{i,i+m}$ tokens
- A **score multiplier effect** is obtained for the innocuous flows which contain ν
  - GET .* HTTP/1.1 .* Pragma: no-cache .*
  - $\Lambda = \Sigma_k \lambda_k + \Sigma_i \lambda_{\nu i} >> \lambda_\nu$

"Score multiplier" effect

- Spliting ν into all the substrings of length m<n
  - E.g., "Pragma: no", "ragma: no-", "agma: no-c", etc.
  - If !(m<<n) then $p(\nu_{i,i+m} \mid IF) \sim= p(\nu \mid IF), \forall i$
- Injecting all the substring $\nu_{i,i+m}$ of ν into the fake anomalous flows, they will be all considered as tokens in the signature (50% occurrence freq.)
  - $\lambda_{\nu_{i,i+m}} \sim= \lambda_\nu$
- An innocuous flow that contains ν contains also all the $\nu_{i,i+m}$ tokens
- A **score multiplier effect** is obtained for the innocuous flows which contain ν
  - GET .* HTTP/1.1 .* Pragma: no-cache .*
  - $\Lambda = \Sigma_k \lambda_k + \Sigma_i \lambda_{\nu i} >> \lambda_\nu$

"Score multiplier" effect

- Spliting ν into all the substrings of length m<n
  - E.g., "Pragma: no", "ragma: no-", "agma: no-c", etc.
  - If !(m<<n) then $p(\nu_{i,i+m} \mid IF) \sim= p(\nu \mid IF), \forall i$
- Injecting all the substring $\nu_{i,i+m}$ of ν into the fake anomalous flows, they will be all considered as tokens in the signature (50% occurrence freq.)
  - $\lambda_{\nu_{i,i+m}} \sim= \lambda_\nu$
- An innocuous flow that contains ν contains also all the $\nu_{i,i+m}$ tokens
- A **score multiplier effect** is obtained for the innocuous flows which contain ν
  - GET .* HTTP/1.1 .* Pragma: no-cache .*
  - $\Lambda = \Sigma_k \lambda_k + \Sigma_i \lambda_{\nu i} >> \lambda_\nu$

"Score multiplier" effect

- Spliting ν into all the substrings of length m<n
  - E.g., "Pragma: no", "ragma: no-", "agma: no-c", etc.
  - If !(m<<n) then $p(\nu_{i,i+m} \mid IF) \sim= p(\nu \mid IF), \forall i$
- Injecting all the substring $\nu_{i,i+m}$ of ν into the fake anomalous flows, they will be all considered as tokens in the signature (50% occurrence freq.)
  - $\lambda_{\nu_{i,i+m}} \sim= \lambda_\nu$
- An innocuous flow that contains ν contains also all the $\nu_{i,i+m}$ tokens
- A **score multiplier effect** is obtained for the innocuous flows which contain ν
  - GET .* HTTP/1.1 .* Pragma: no-cache .*
  - $\Lambda = \Sigma_k \lambda_k + \Sigma_i \lambda_{\nu i} >> \lambda_\nu$

"Score multiplier" effect

- Spliting ν into all the substrings of length m<n
  - E.g., "Pragma: no", "ragma: no-", "agma: no-c", etc.
  - If !(m<<n) then $p(\nu_{i,i+m} \mid IF) \sim= p(\nu \mid IF), \forall i$
- Injecting all the substring $\nu_{i,i+m}$ of ν into the fake anomalous flows, they will be all considered as tokens in the signature (50% occurrence freq.)
  - $\lambda_{\nu_{i,i+m}} \sim= \lambda_\nu$
- An innocuous flow that contains ν contains also all the $\nu_{i,i+m}$ tokens
- A **score multiplier effect** is obtained for the innocuous flows which contain ν
  - GET .* HTTP/1.1 .* Pragma: no-cache .*
  - $\Lambda = \Sigma_k \lambda_k + \Sigma_i \lambda_{\nu i} >> \lambda_\nu$

"Score multiplier" effect

- Spliting ν into all the substrings of length m<n
  - E.g., "Pragma: no", "ragma: no-", "agma: no-c", etc.
  - If !(m<<n) then $p(\nu_{i,i+m} \mid IF) \sim= p(\nu \mid IF), \forall i$
- Injecting all the substring $\nu_{i,i+m}$ of ν into the fake anomalous flows, they will be all considered as tokens in the signature (50% occurrence freq.)
  - $\lambda_{\nu_{i,i+m}} \sim= \lambda_\nu$
- An innocuous flow that contains ν contains also all the $\nu_{i,i+m}$ tokens
- A **score multiplier effect** is obtained for the innocuous flows which contain ν
  - GET .* HTTP/1.1 .* Pragma: no-cache .*
  - $\Lambda = \Sigma_k \lambda_k + \Sigma_i \lambda_{\nu i} >> \lambda_\nu$ → If $\Lambda > \theta$ → **False positive**

Crafting the Noise

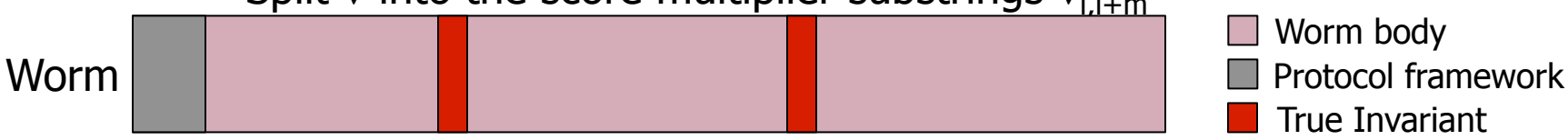
- Find candidate strings ν to produce score multiplier effect
  - Analyze normal traffic samples
  - Look for all the strings ν of length between l_1 and l_2 whereby

    $0.05 < p(\nu \mid \text{“normal” flow}) < 0.20$
  - Split ν into the score multiplier substrings $\nu_{i,i+m}$

Crafting the Noise

- Find candidate strings ν to produce score multiplier effect
  - Analyze normal traffic samples
  - Look for all the strings ν of length between l_1 and l_2 whereby

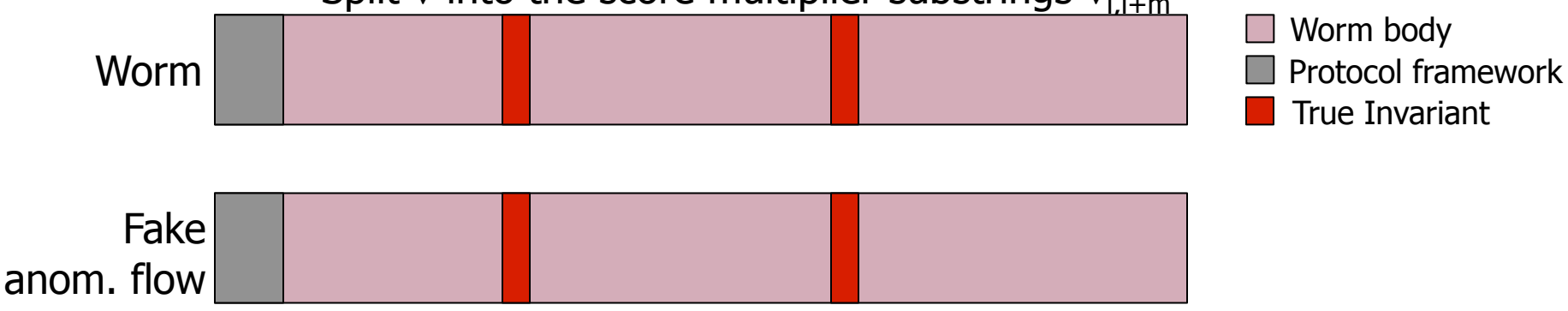
    $0.05 < p(\nu \mid \text{“normal” flow}) < 0.20$
  - Split ν into the score multiplier substrings $\nu_{i,i+m}$



Crafting the Noise

- Find candidate strings ν to produce score multiplier effect
  - Analyze normal traffic samples
  - Look for all the strings ν of length between l_1 and l_2 whereby

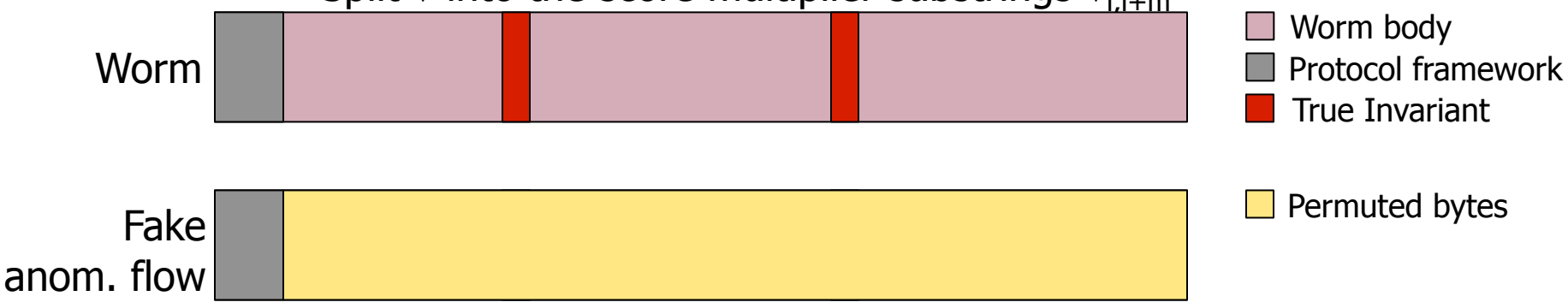
    $0.05 < p(\nu \mid \text{“normal” flow}) < 0.20$
  - Split ν into the score multiplier substrings $\nu_{i,i+m}$



Crafting the Noise

- Find candidate strings ν to produce score multiplier effect
  - Analyze normal traffic samples
  - Look for all the strings ν of length between l_1 and l_2 whereby

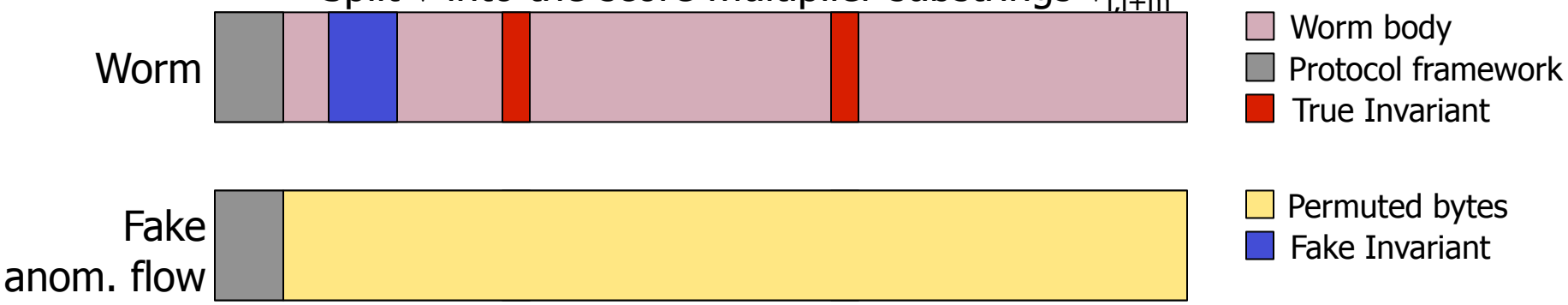
    $0.05 < p(\nu \mid \text{“normal” flow}) < 0.20$
  - Split ν into the score multiplier substrings $\nu_{i,i+m}$



Crafting the Noise

- Find candidate strings ν to produce score multiplier effect
  - Analyze normal traffic samples
  - Look for all the strings ν of length between l_1 and l_2 whereby

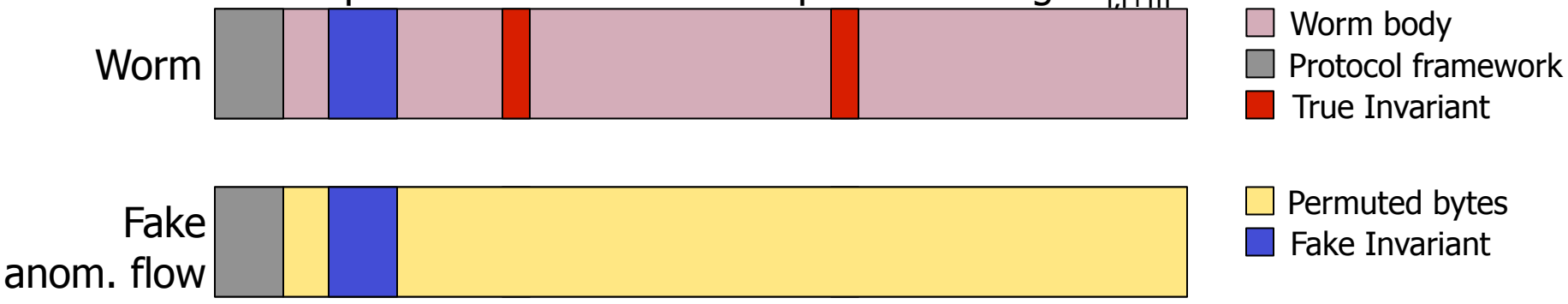
    $0.05 < p(\nu \mid \text{“normal” flow}) < 0.20$
  - Split ν into the score multiplier substrings $\nu_{i,i+m}$



Crafting the Noise

- Find candidate strings ν to produce score multiplier effect
  - Analyze normal traffic samples
  - Look for all the strings ν of length between l_1 and l_2 whereby

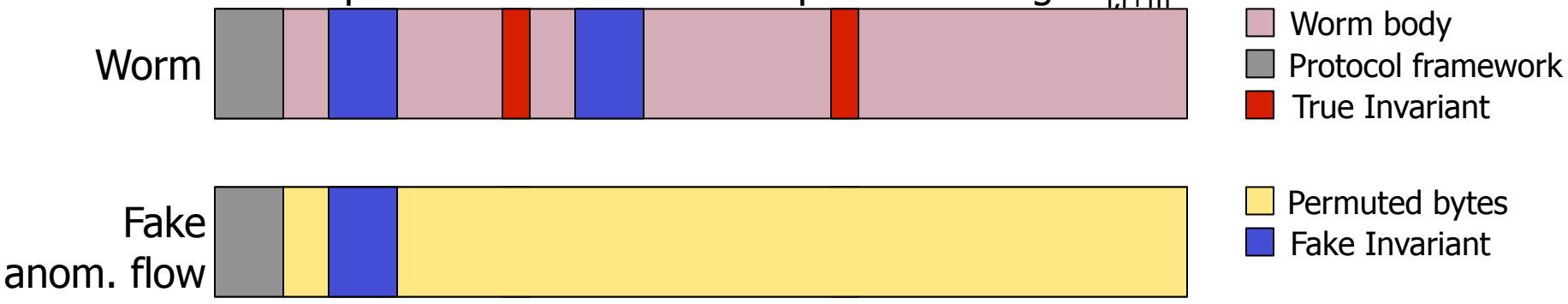
    $0.05 < p(\nu \mid \text{“normal” flow}) < 0.20$
  - Split ν into the score multiplier substrings $\nu_{i,i+m}$



Crafting the Noise

- Find candidate strings ν to produce score multiplier effect
  - Analyze normal traffic samples
  - Look for all the strings ν of length between l_1 and l_2 whereby

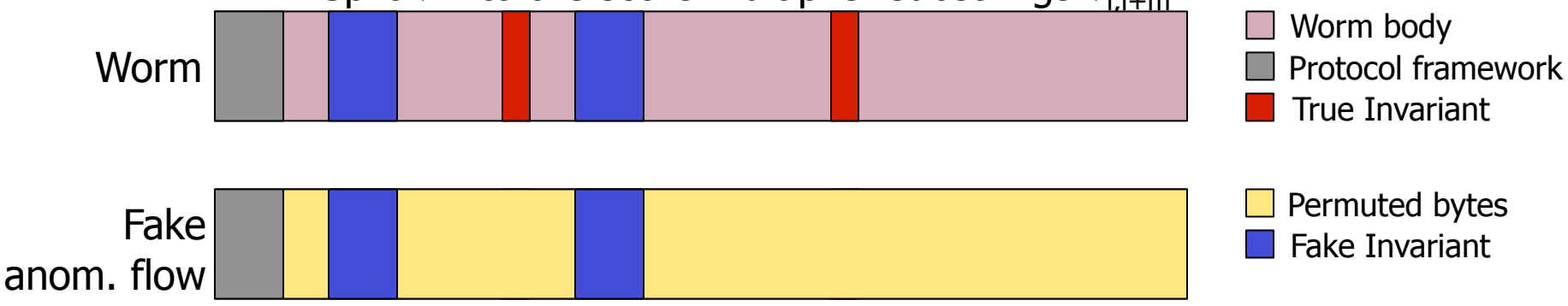
    $0.05 < p(\nu \mid \text{“normal” flow}) < 0.20$
  - Split ν into the score multiplier substrings $\nu_{i,i+m}$



Crafting the Noise

- Find candidate strings ν to produce score multiplier effect
  - Analyze normal traffic samples
  - Look for all the strings ν of length between l_1 and l_2 whereby

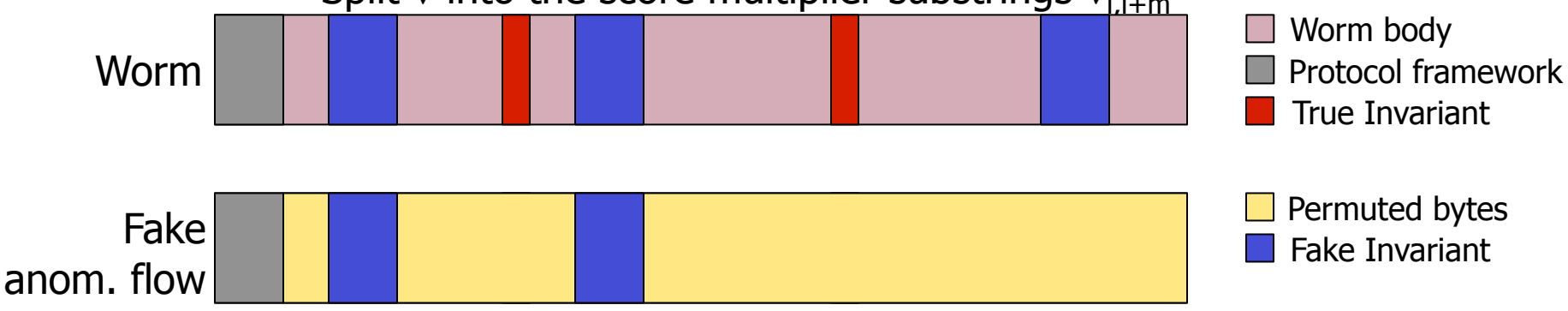
    $0.05 < p(\nu \mid \text{“normal” flow}) < 0.20$
  - Split ν into the score multiplier substrings $\nu_{i,i+m}$



Crafting the Noise

- Find candidate strings ν to produce score multiplier effect
  - Analyze normal traffic samples
  - Look for all the strings ν of length between l_1 and l_2 whereby

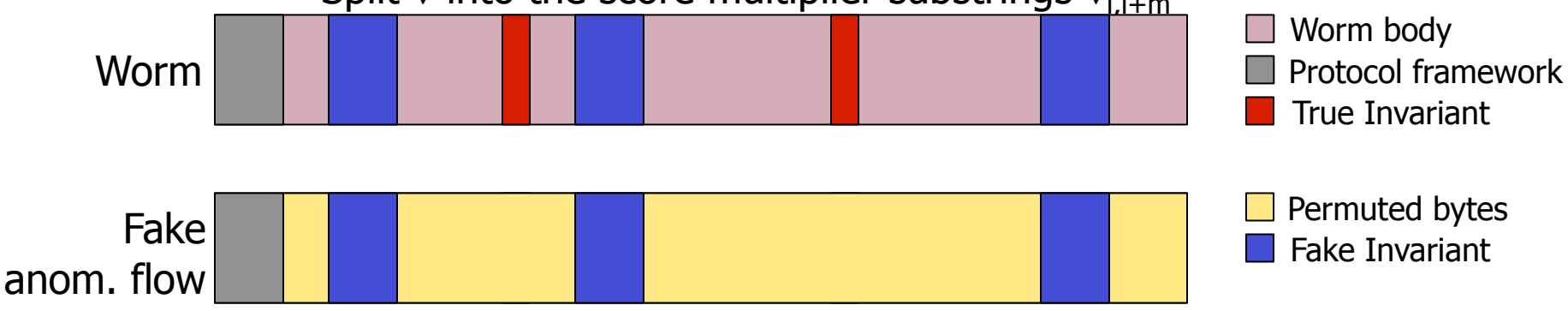
    $0.05 < p(\nu \mid \text{“normal” flow}) < 0.20$
  - Split ν into the score multiplier substrings $\nu_{j,i+m}$



Crafting the Noise

- Find candidate strings ν to produce score multiplier effect
  - Analyze normal traffic samples
  - Look for all the strings ν of length between l_1 and l_2 whereby

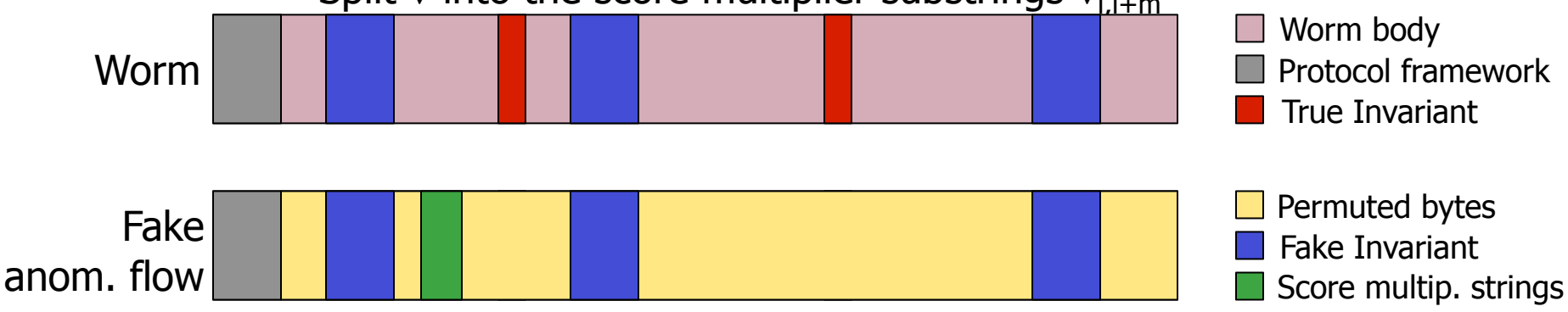
    $0.05 < p(\nu \mid \text{“normal” flow}) < 0.20$
  - Split ν into the score multiplier substrings $\nu_{j,i+m}$



Crafting the Noise

- Find candidate strings ν to produce score multiplier effect
  - Analyze normal traffic samples
  - Look for all the strings ν of length between l_1 and l_2 whereby

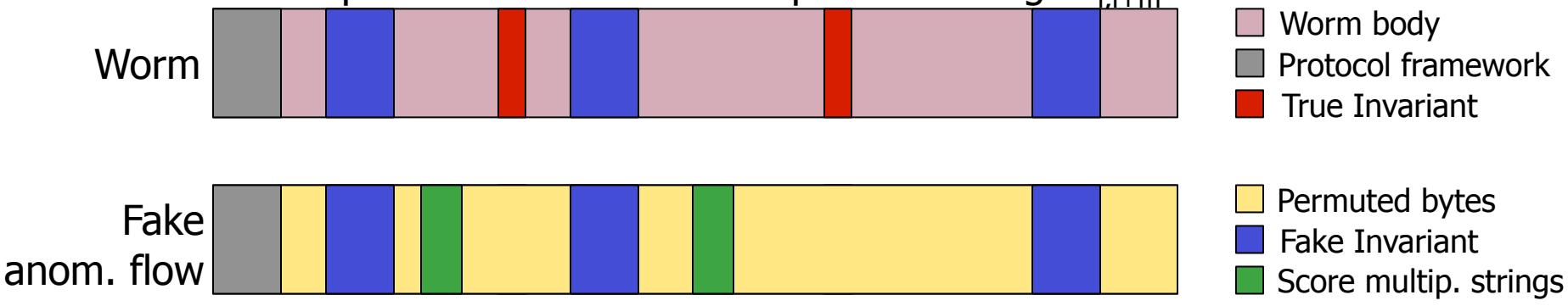
    $0.05 < p(\nu \mid \text{“normal” flow}) < 0.20$
  - Split ν into the score multiplier substrings $\nu_{j,i+m}$



Crafting the Noise

- Find candidate strings ν to produce score multiplier effect
  - Analyze normal traffic samples
  - Look for all the strings ν of length between l_1 and l_2 whereby

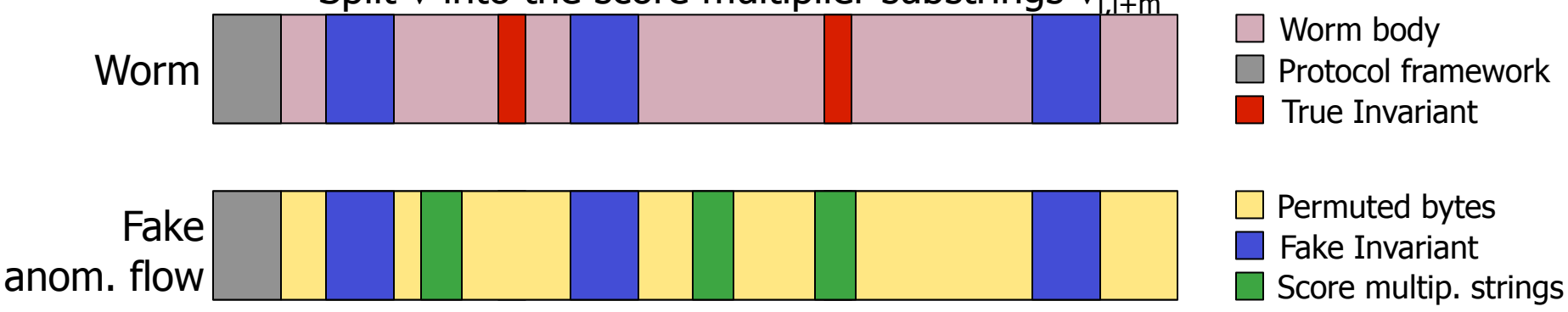
    $0.05 < p(\nu \mid \text{“normal” flow}) < 0.20$
  - Split ν into the score multiplier substrings $\nu_{j,i+m}$



Crafting the Noise

- Find candidate strings ν to produce score multiplier effect
  - Analyze normal traffic samples
  - Look for all the strings ν of length between l_1 and l_2 whereby

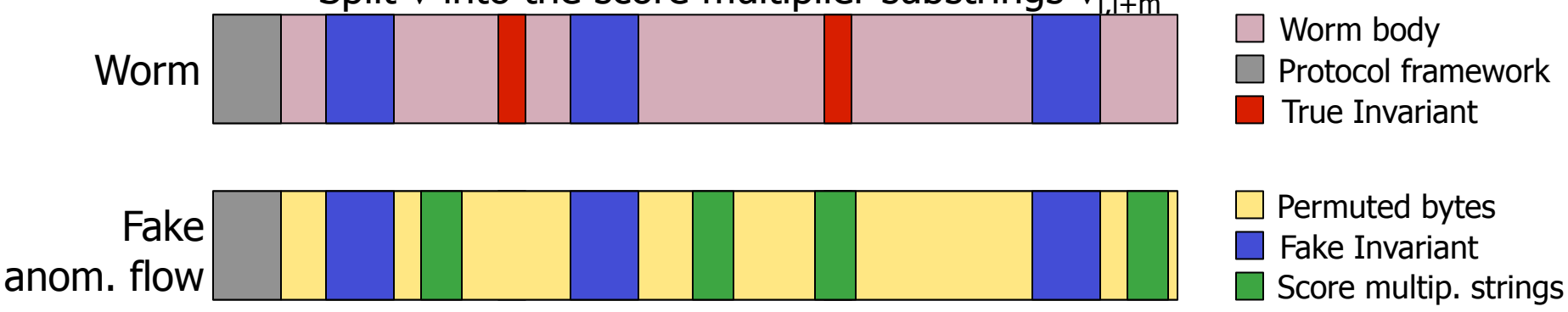
    $0.05 < p(\nu \mid \text{“normal” flow}) < 0.20$
  - Split ν into the score multiplier substrings $\nu_{j,i+m}$



Crafting the Noise

- Find candidate strings ν to produce score multiplier effect
  - Analyze normal traffic samples
  - Look for all the strings ν of length between l_1 and l_2 whereby

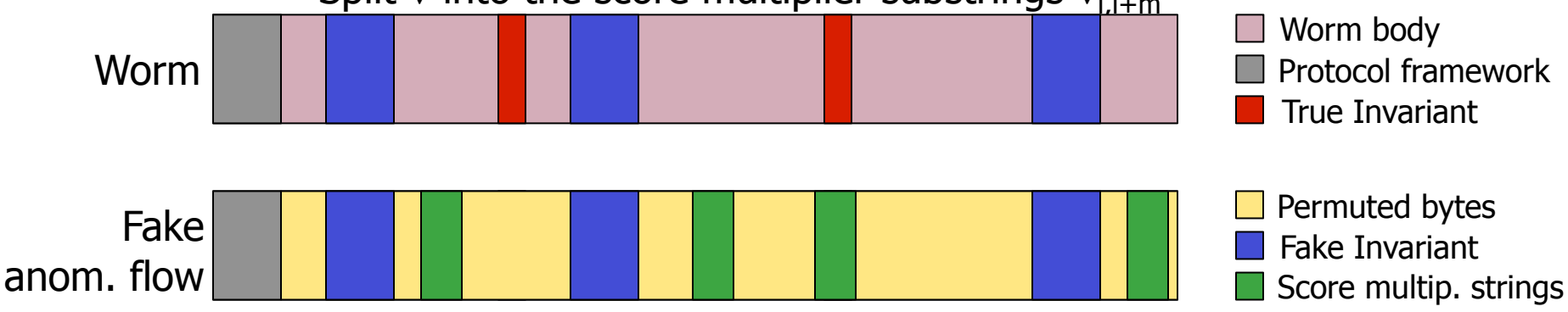
    $0.05 < p(\nu \mid \text{“normal” flow}) < 0.20$
  - Split ν into the score multiplier substrings $\nu_{j,i+m}$



Crafting the Noise

- Find candidate strings ν to produce score multiplier effect
  - Analyze normal traffic samples
  - Look for all the strings ν of length between l_1 and l_2 whereby

    $0.05 < p(\nu \mid \text{“normal” flow}) < 0.20$
  - Split ν into the score multiplier substrings $\nu_{j,i+m}$



- The Fake Invariants are specific for each worm and its fake anomalous flows
- The score multiplier strings have to be common to all the fake anomalous flows

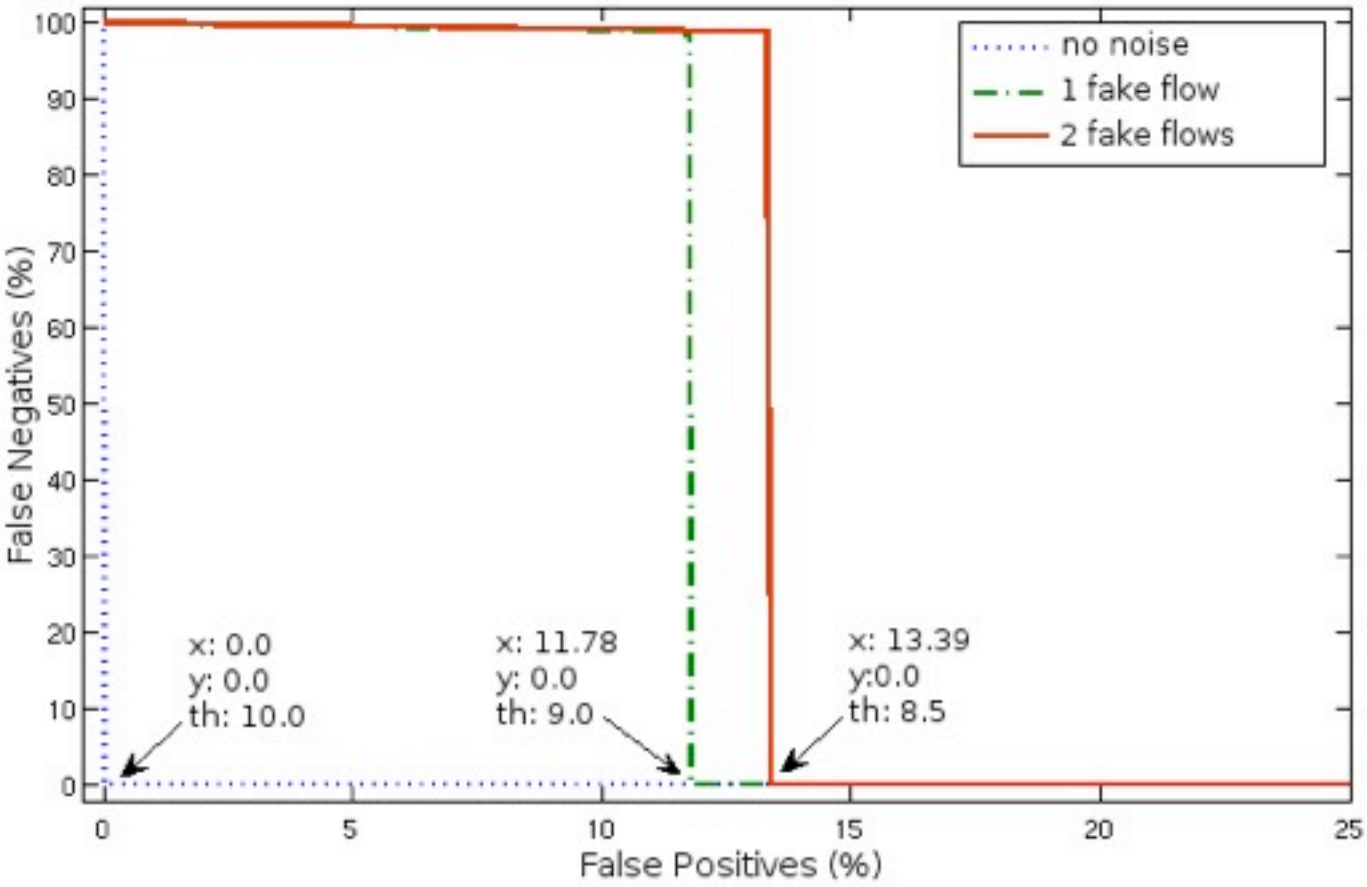
Experimental results

- **Experimental Setup**
  - We implemented POLYGRAPH according to the description in [1]
  - "Apache-Knacker" Worm:

    `GET .* HTTP/1.1\r\n.*\r\nHost: .*\r\n .*\r\nHost: .*\xFF\xBF.*\r\n`
- **Training dataset**
  - Suspicious flow pool = 10 worm variants
  - Innocuous flow pool = 100,459 flows (0.007% FP)
- **Test dataset**
  - "Normal" Test flow pool = 217,164 (0.0% FP)
  - "Suspicious" Test flow pool = 100 worm variants
- **Attacker's dataset**
  - Candidate Score Multiplier Strings extracted from 5,000 flows

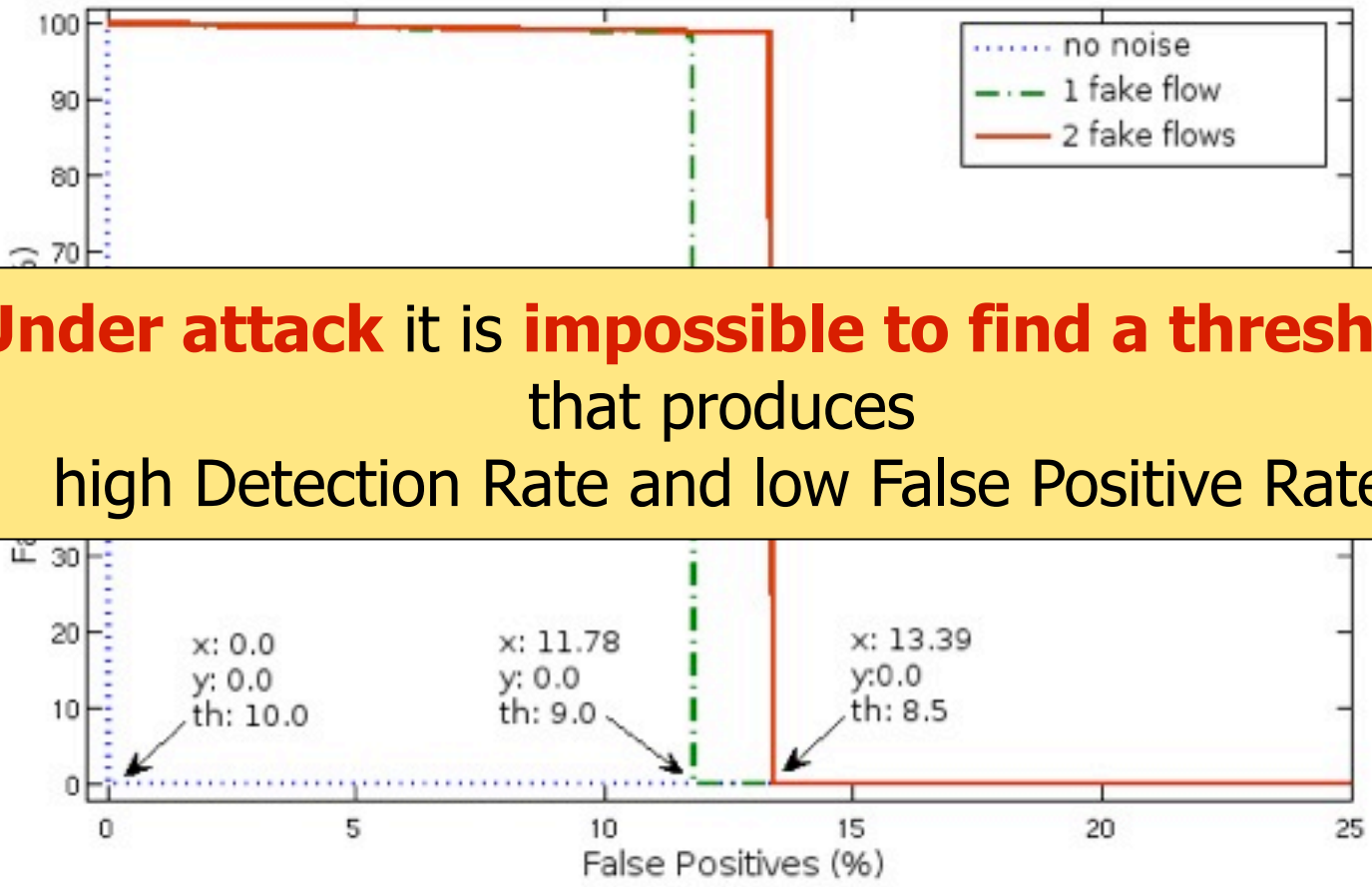
Experimental results with Bayes signatures

Score Multip. Srings (m=4): “Pragma: no-cache”, “-powerpoint”



Experimental results with Bayes signatures

Score Multip. Srings (m=4): "Pragma: no-cache", "-powerpoint"



Under attack it is **impossible to find a threshold** that produces high Detection Rate and low False Positive Rate

Experimental results with all the 3 types of signatures

- The results are not deterministically predictable (due to tokens that are common just by chance)
  - Simulations: 2 groups of tests
    - For each group of tests we simulated 2 scenarios:
      - 1 faf/worm and 2 faf/worm
    - 1° group of tests: 45 rounds
    - 2° group of tests: 20 rounds

	1 fake anomalous flow	2 fake anomalous flows
Conjunction	73.3%	88.9%
Token-subsequences	60.0%	73.3%
Bayes	100%	100%
All three signatures	44.4%	62.2%

Table 1: Percentage of successful attacks (using "Forwarded-For" and "Modified-Since")

	1 fake anomalous flow	2 fake anomalous flows
Conjunction	65%	95%
Token-subsequences	40%	90%
Bayes	90%	100%
All three signatures	20%	85%

Table 2: Percentage of successful attacks (using "Cache-Control" and "Range: bytes")

Experimental results with all the 3 types of signatures

- The results are not deterministically predictable (due to tokens that are common just by chance)
  - Simulations: 2 groups of tests
    - For each group of tests we simulated 2 scenarios:
      - 1 faf/worm and 2 faf/worm
    - 1° group of tests: 45 rounds
    - 2° group of tests: 20 rounds

	1 fake anomalous flow	2 fake anomalous flows
Conjunction	73.3%	88.9%
Token-subsequences	60.0%	73.3%
Bayes	100%	100%
All three signatures	44.4%	62.2%

Table 1: Percentage of successful attacks (using "Forwarded-For" and "Modified-Since")

	1 fake anomalous flow	2 fake anomalous flows
Conjunction	65%	95%
Token-subsequences	40%	90%
Bayes	90%	100%
All three signatures	20%	85%

Table 2: Percentage of successful attacks (using "Cache-Control" and "Range: bytes")

Conclusion

- Can Machine Learning be secure? [2]
- The Noise Injection Attack has a good chance to mislead the signature generation process
- Unless a precise semantic-based (thus expensive) flow classifier + signature generation scheme is used
  - Syntactic-based Worm Signature Generators are vulnerable to the Noise Injection Attack

[2] M. Barreno, B. Nelson, R. Sears, A.D. Joseph, J.D. Tygar
Can Machine Learning be Secure? (*ASIACCS* 2006)

Thank you!

Combining our attack with the Red Herring

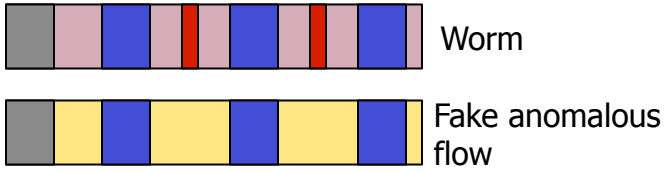
- <u>Increases the probability of success</u>
  - The worm includes some **temporay invariants**
  - This invariants **expire over time**
  - This means that even if POLYGRAPH generates useful Conjunction and Token-subsequence signatures, after a while they will become useless
  - The second time POLYGRAPH generates the signature, it could be not as "fortunate" as the first time
  - Further, if the temporary invariants are chosen among "high frequency normal tokens", the combination of the attacks **will not interphere with the attack to Bayes Signatures**

Misleading Conjunction and Token-Subsequence Signatures

- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant



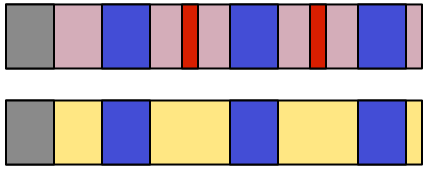
Misleading Conjunction and Token-Subsequence Signatures

W1 F1 W2 F2 W3 F3

- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant



Misleading Conjunction and Token-Subsequence Signatures

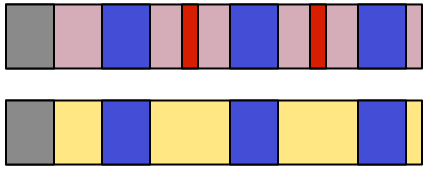
W1 F1 W2 F2 W3 F3

PF + FI-1

- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant



Worm

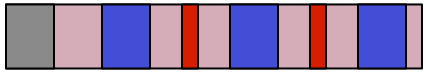
Fake anomalous flow

Misleading Conjunction and Token-Subsequence Signatures

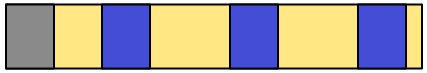
- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant



Worm



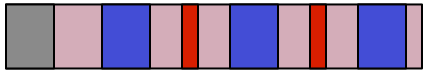
Fake anomalous flow

Misleading Conjunction and Token-Subsequence Signatures

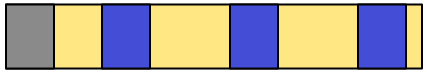
- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant



Worm



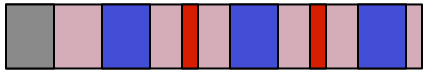
Fake anomalous flow

Misleading Conjunction and Token-Subsequence Signatures

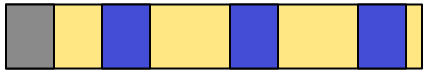
- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

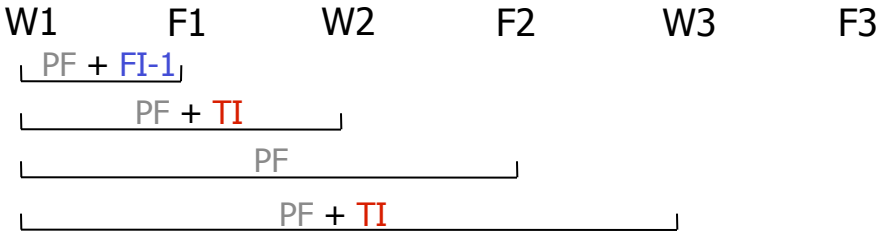
- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant



Worm



Fake anomalous flow

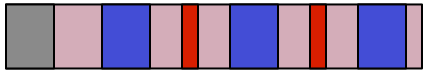


Misleading Conjunction and Token-Subsequence Signatures

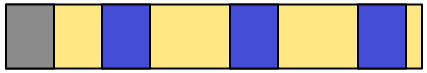
- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

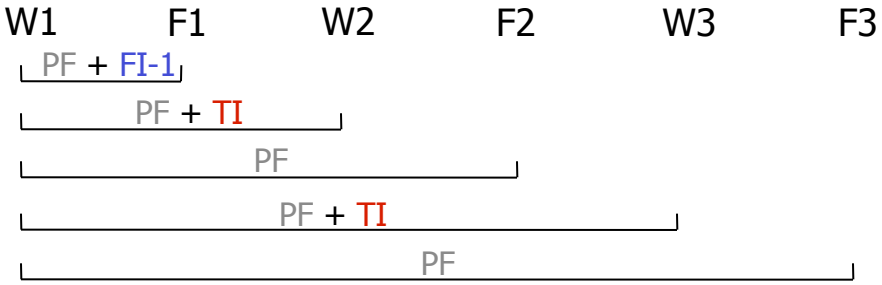
- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant



Worm



Fake anomalous flow

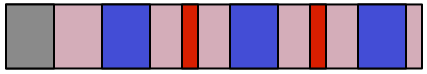


Misleading Conjunction and Token-Subsequence Signatures

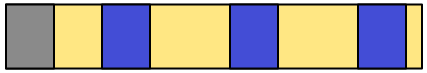
- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

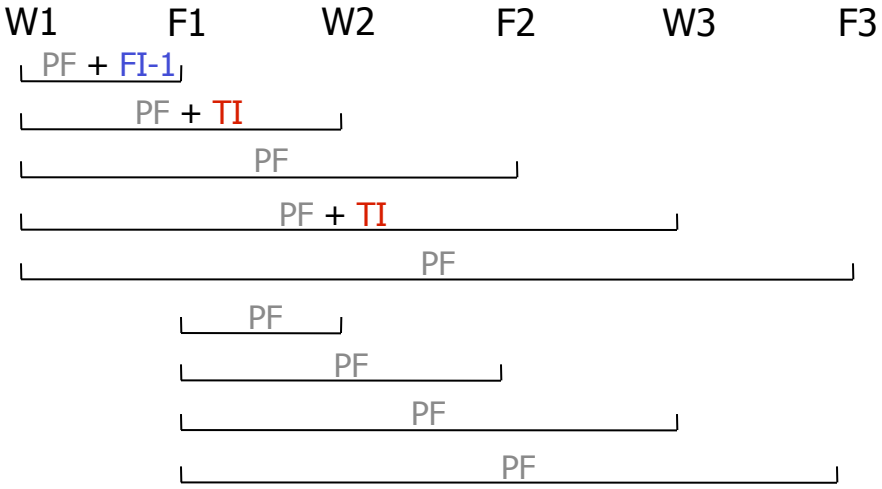
- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant



Worm



Fake anomalous flow



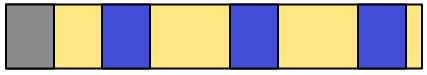
Misleading Conjunction and Token-Subsequence Signatures

- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

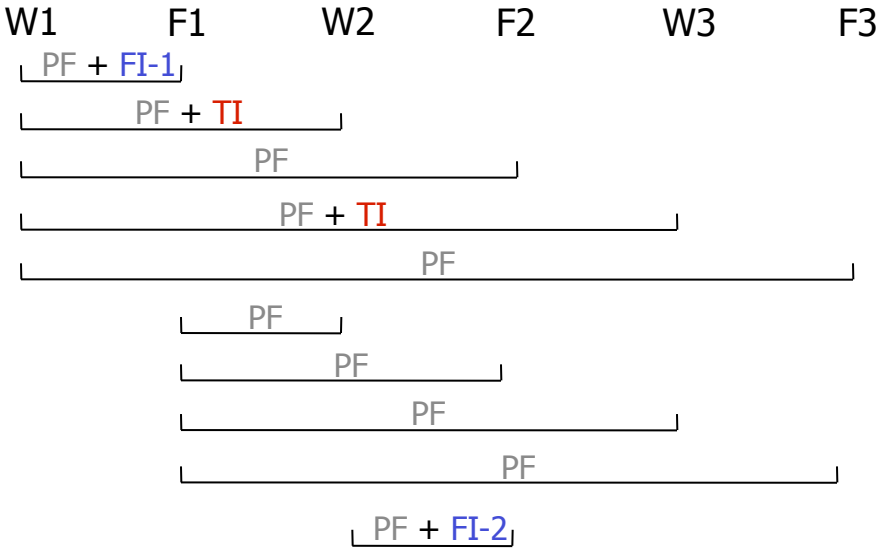
- P(FP | FI) < P(FP | TI)

- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant

Worm



Fake anomalous flow

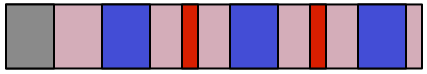


Misleading Conjunction and Token-Subsequence Signatures

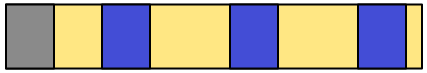
- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

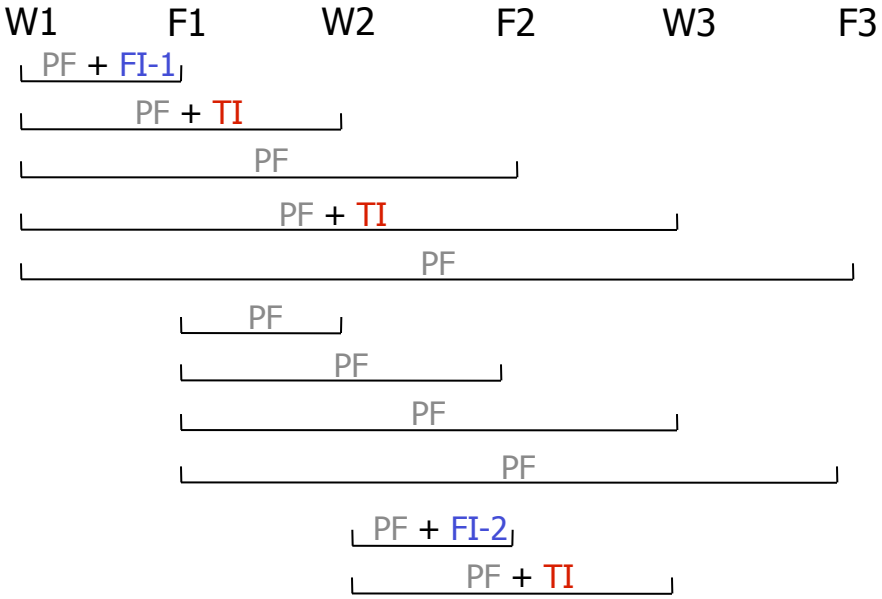
- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant



Worm



Fake anomalous flow

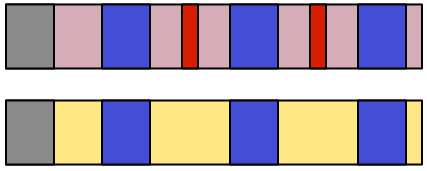


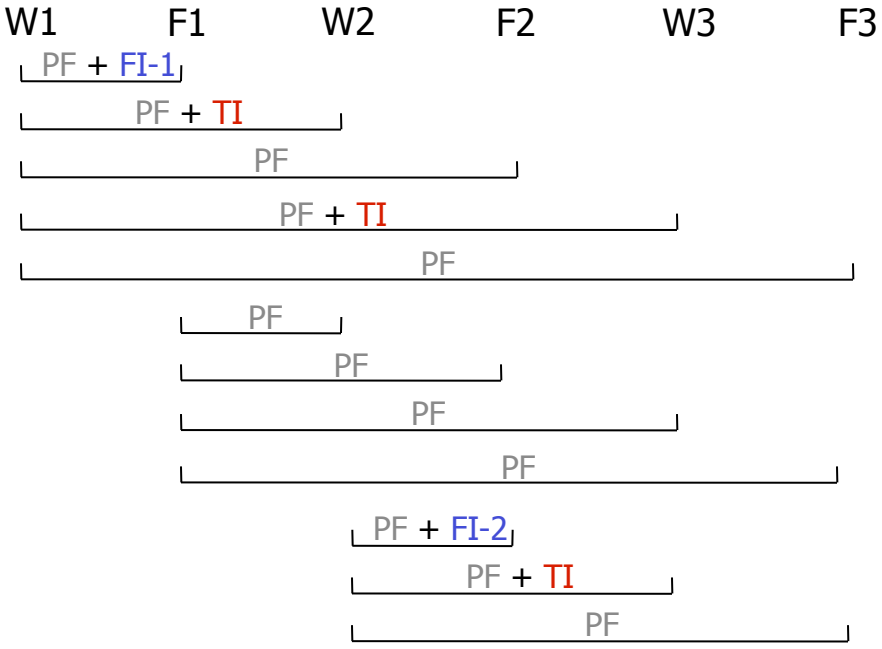
Misleading Conjunction and Token-Subsequence Signatures

- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant



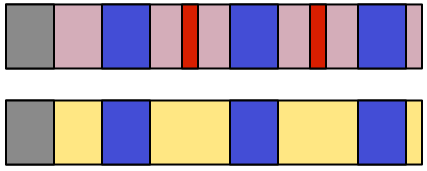


Misleading Conjunction and Token-Subsequence Signatures

- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

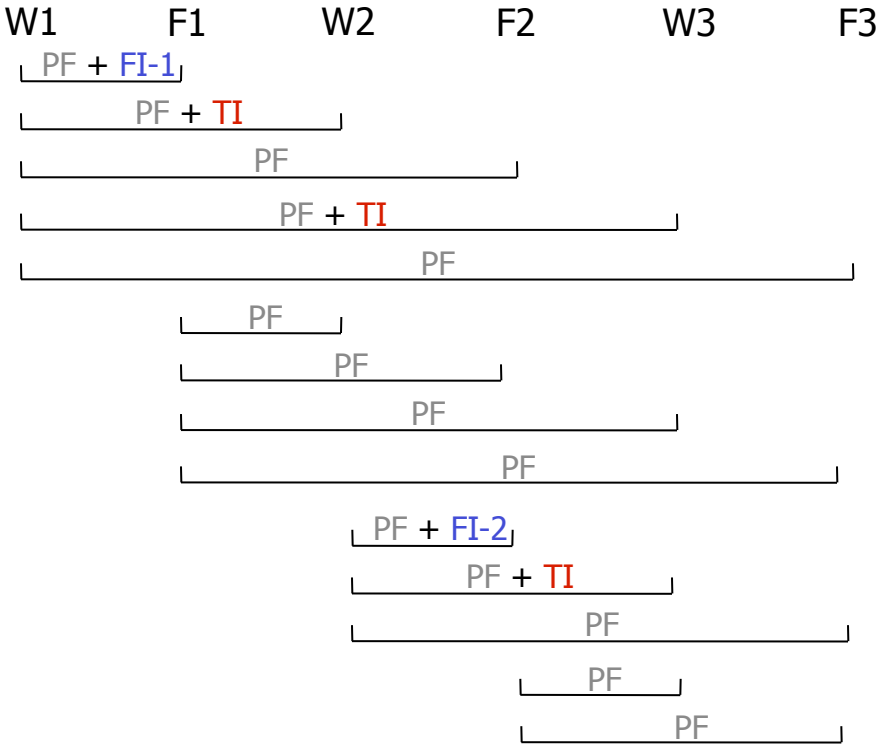
- P(FP | FI) < P(FP | TI)

- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant



Worm

Fake anomalous flow

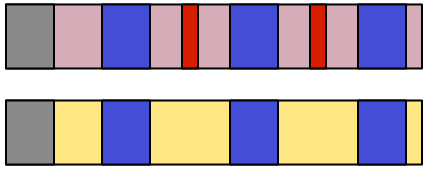


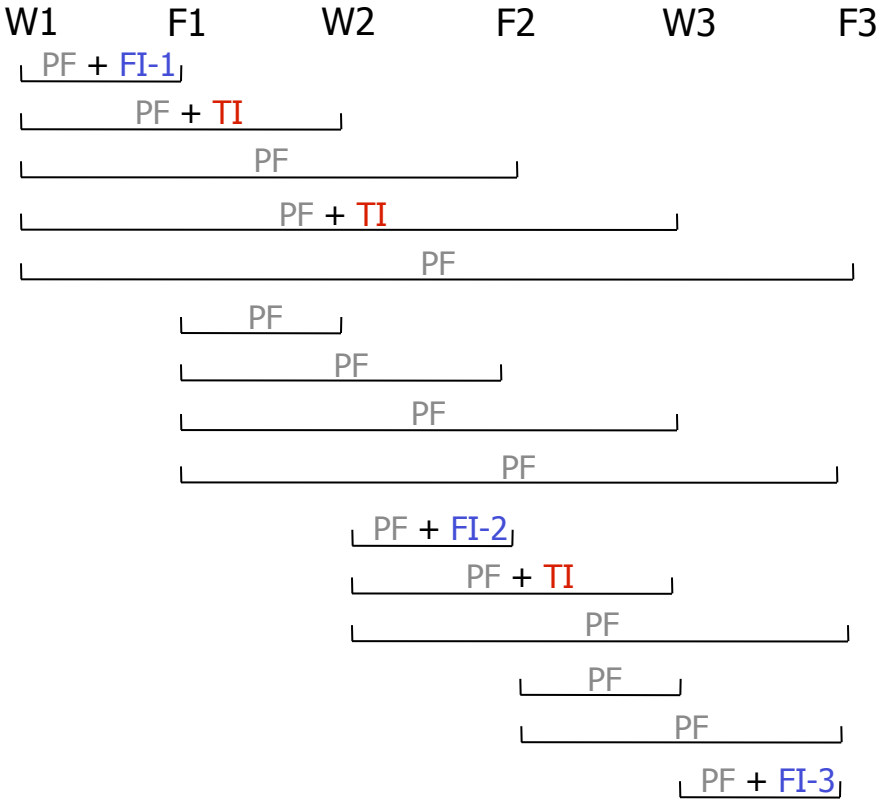
Misleading Conjunction and Token-Subsequence Signatures

- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

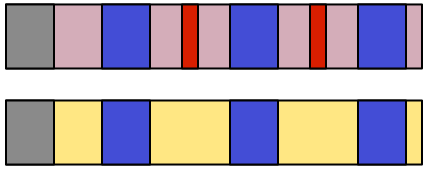
- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant

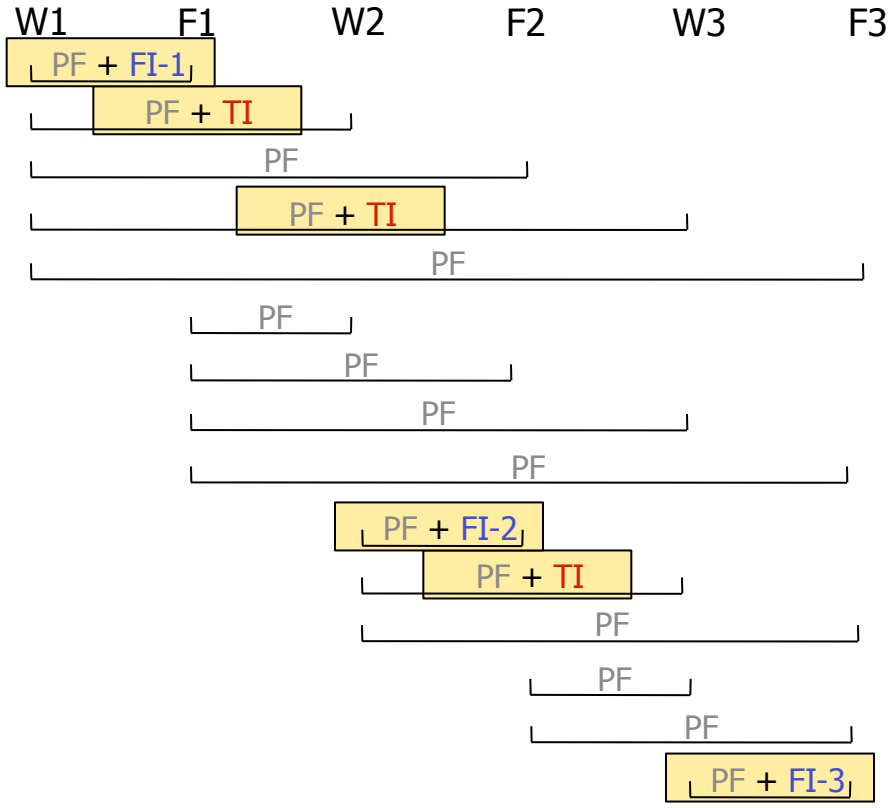




Misleading Conjunction and Token-Subsequence Signatures

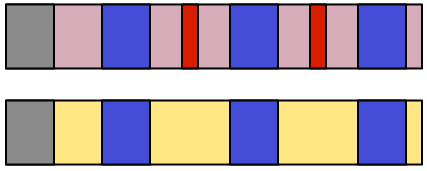
- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow
- P(FP | FI) < P(FP | TI)
- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant

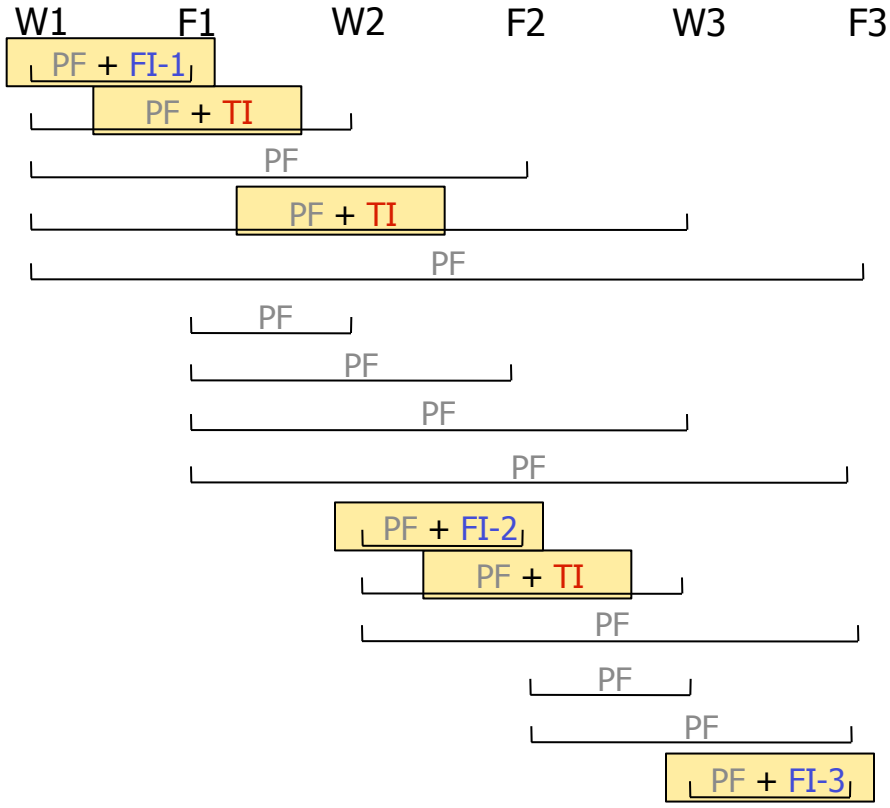




Misleading Conjunction and Token-Subsequence Signatures

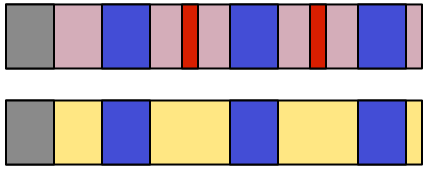
- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow
- P(FP | FI) < P(FP | TI)
- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant

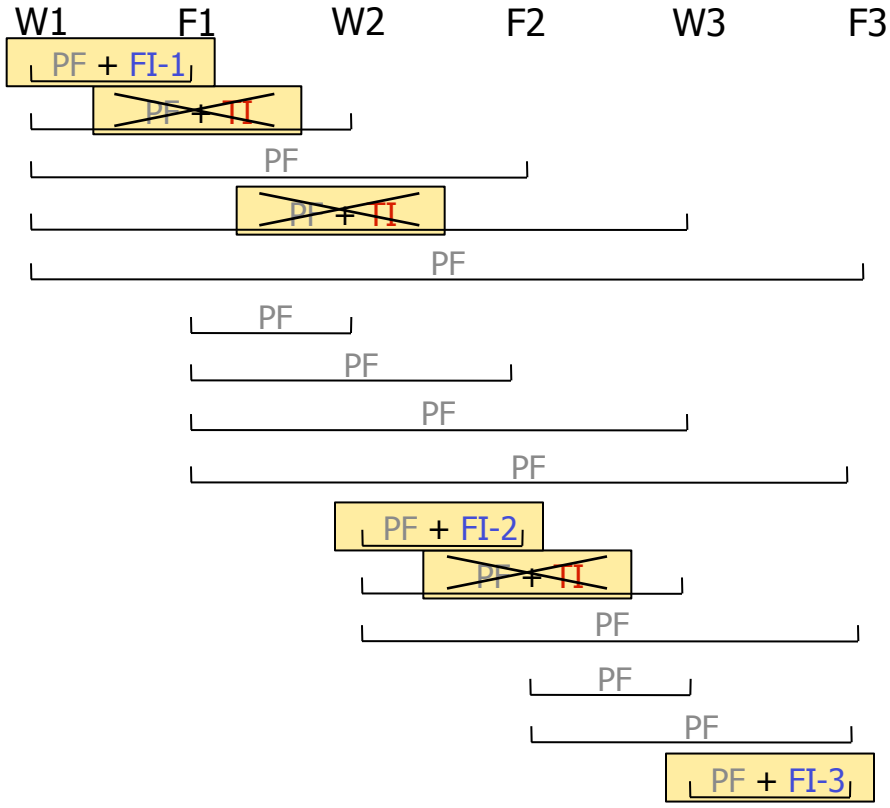




Misleading Conjunction and Token-Subsequence Signatures

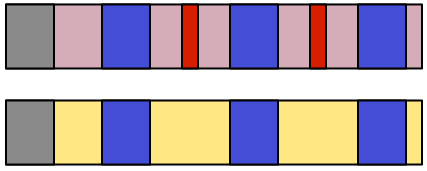
- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow
- P(FP | FI) < P(FP | TI)
- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant

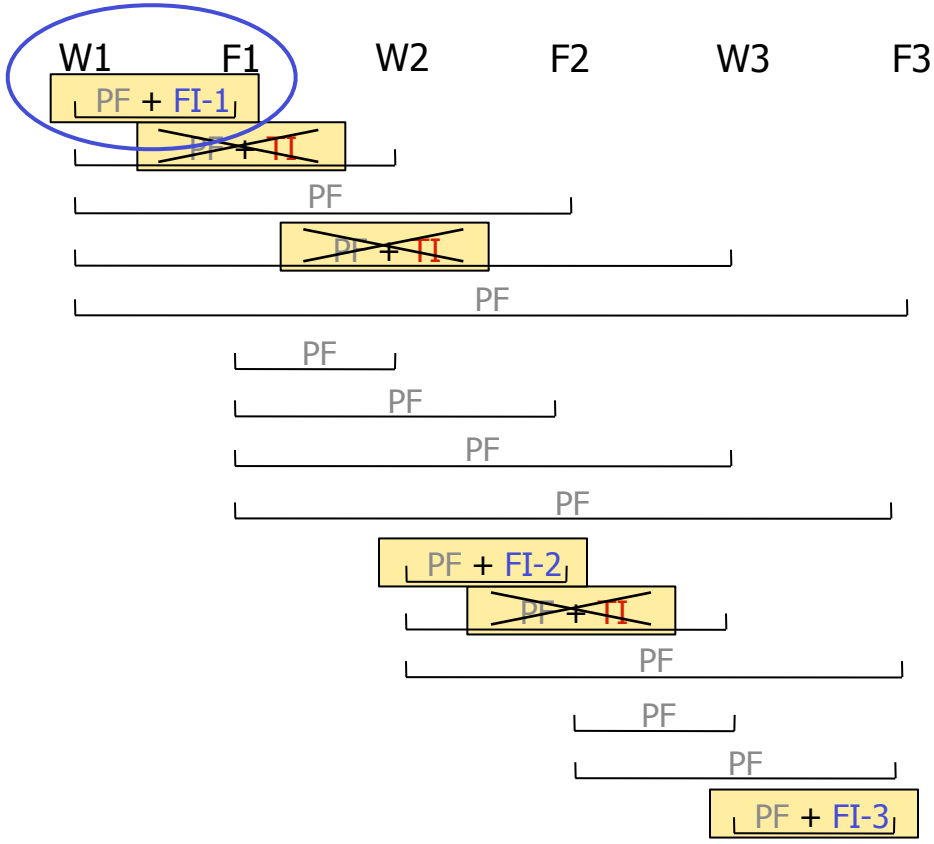




Misleading Conjunction and Token-Subsequence Signatures

- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow
- P(FP | FI) < P(FP | TI)
- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant



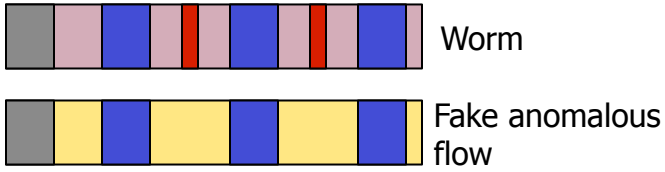


Misleading Conjunction and Token-Subsequence Signatures

- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant



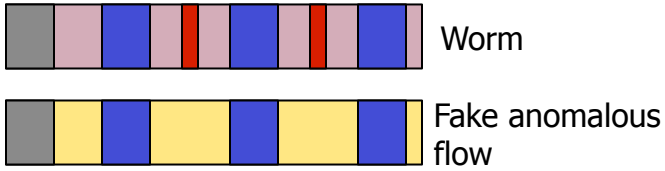
Misleading Conjunction and Token-Subsequence Signatures

[W1 , F1] W2 F2 W3 F3

- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant

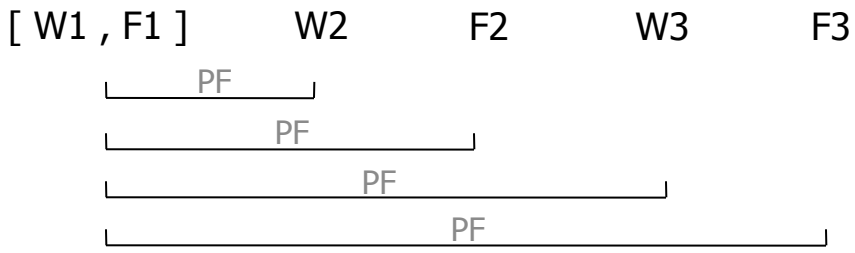


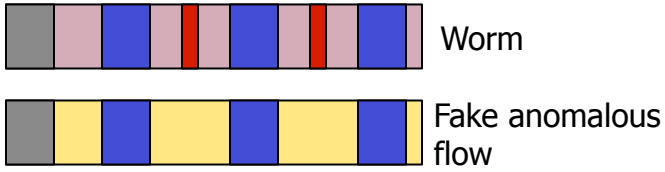
Misleading Conjunction and Token-Subsequence Signatures

- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant



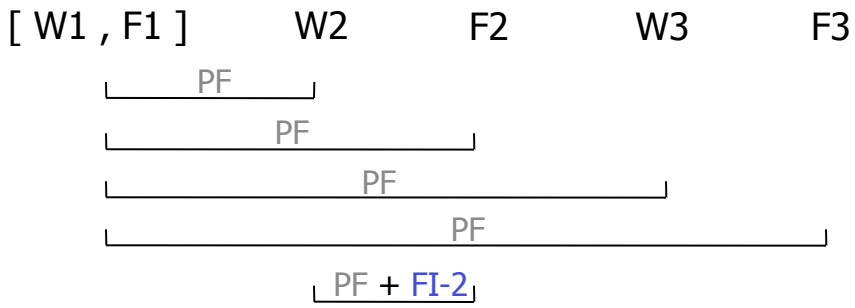


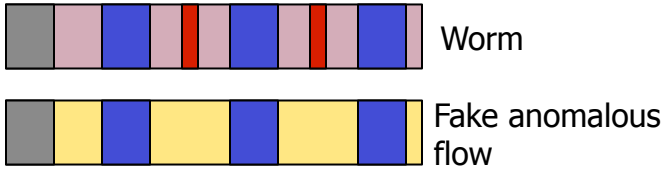
Misleading Conjunction and Token-Subsequence Signatures

- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

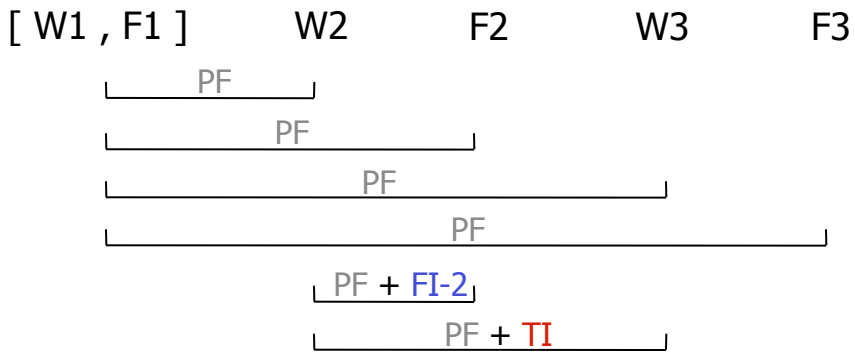
- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant

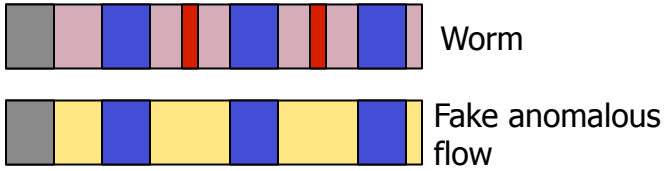




Misleading Conjunction and Token-Subsequence Signatures

- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow
- P(FP | FI) < P(FP | TI)
- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant



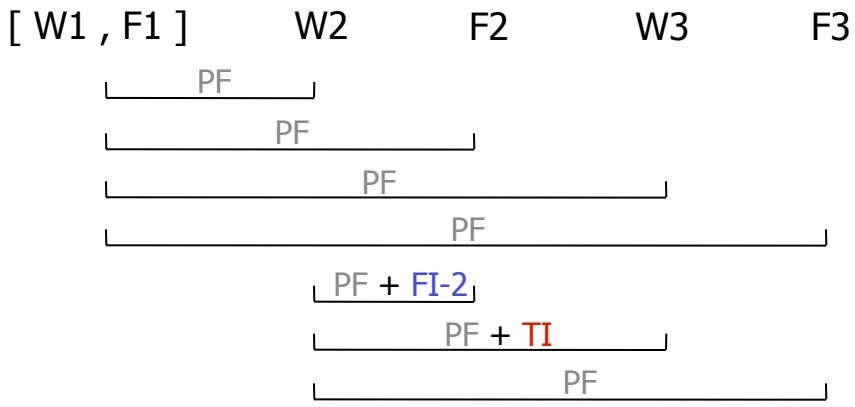


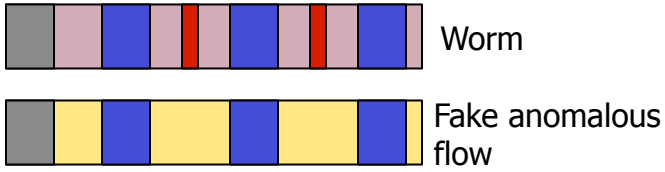
Misleading Conjunction and Token-Subsequence Signatures

- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant



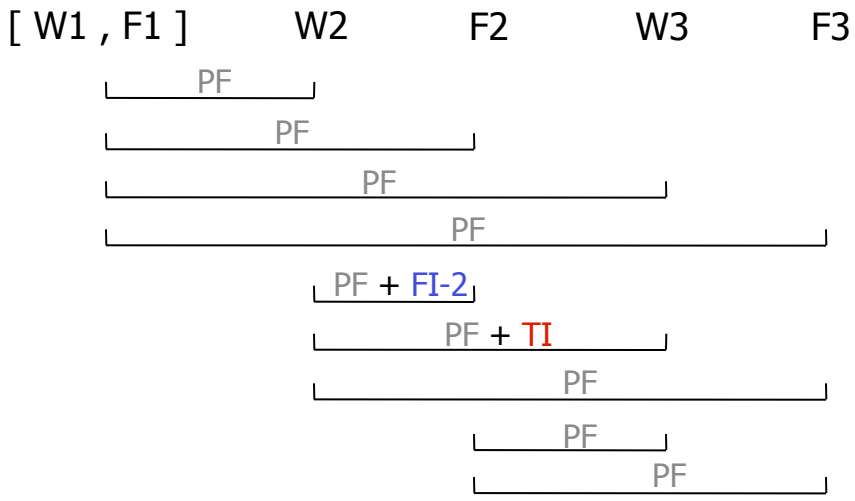


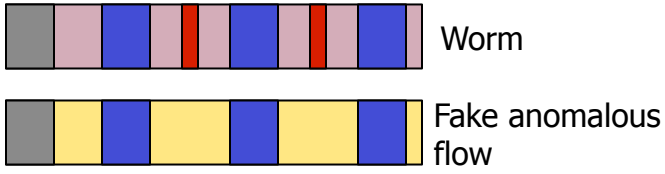
Misleading Conjunction and Token-Subsequence Signatures

- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant



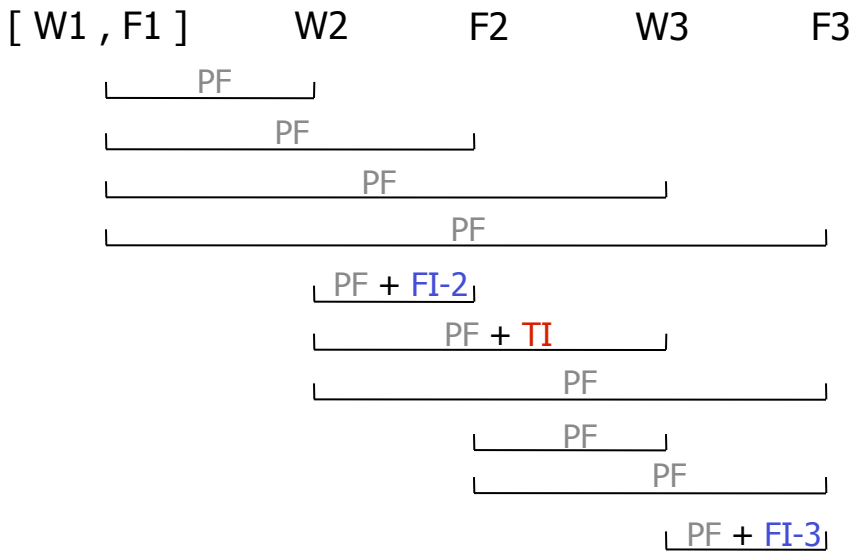


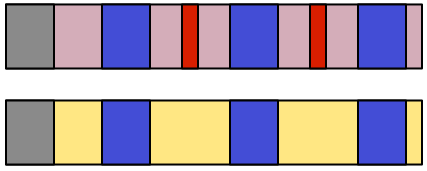
Misleading Conjunction and Token-Subsequence Signatures

- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant



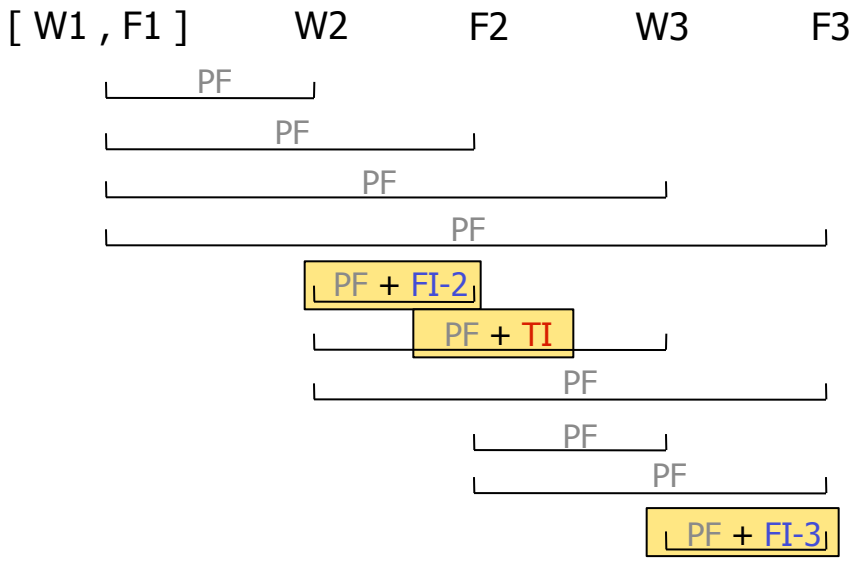


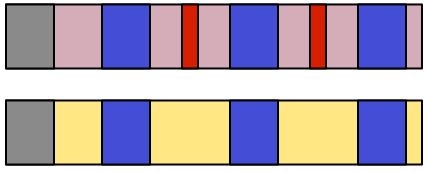
Misleading Conjunction and Token-Subsequence Signatures

- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

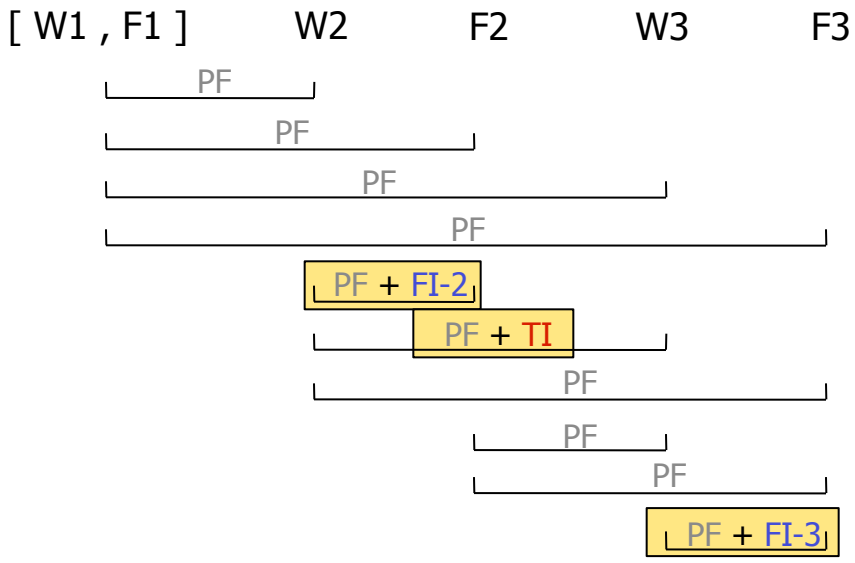
- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant

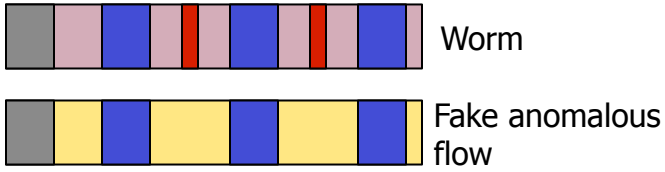




Misleading Conjunction and Token-Subsequence Signatures

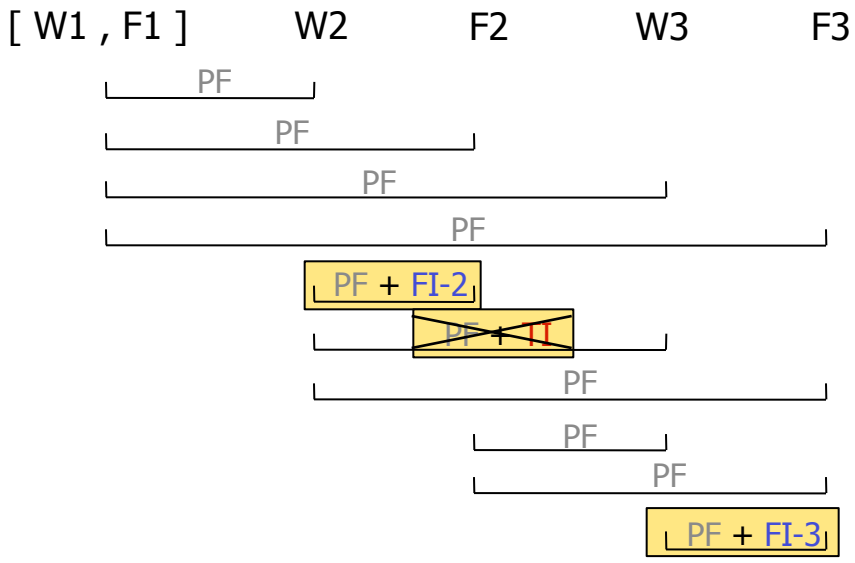
- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow
- P(FP | FI) < P(FP | TI)
- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant

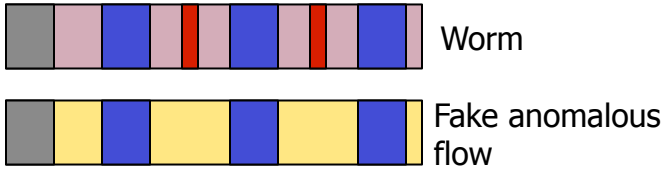




Misleading Conjunction and Token-Subsequence Signatures

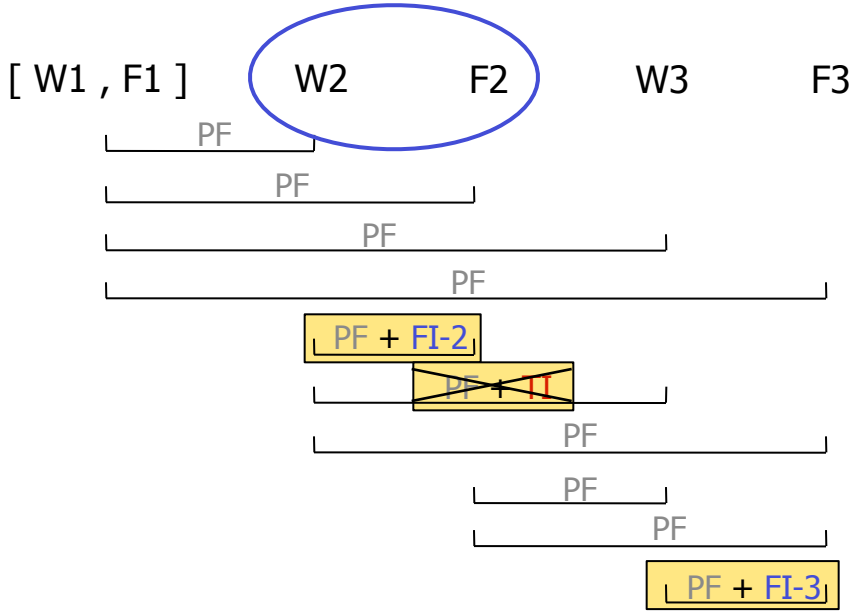
- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow
- P(FP | FI) < P(FP | TI)
- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant

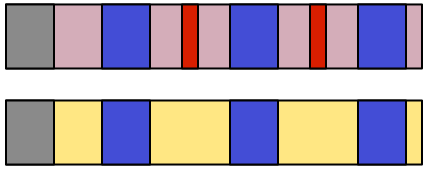




Misleading Conjunction and Token-Subsequence Signatures

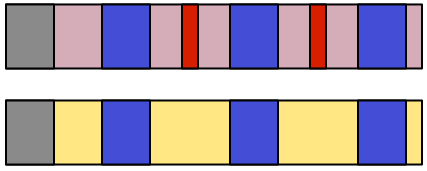
- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow
- P(FP | FI) < P(FP | TI)
- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant

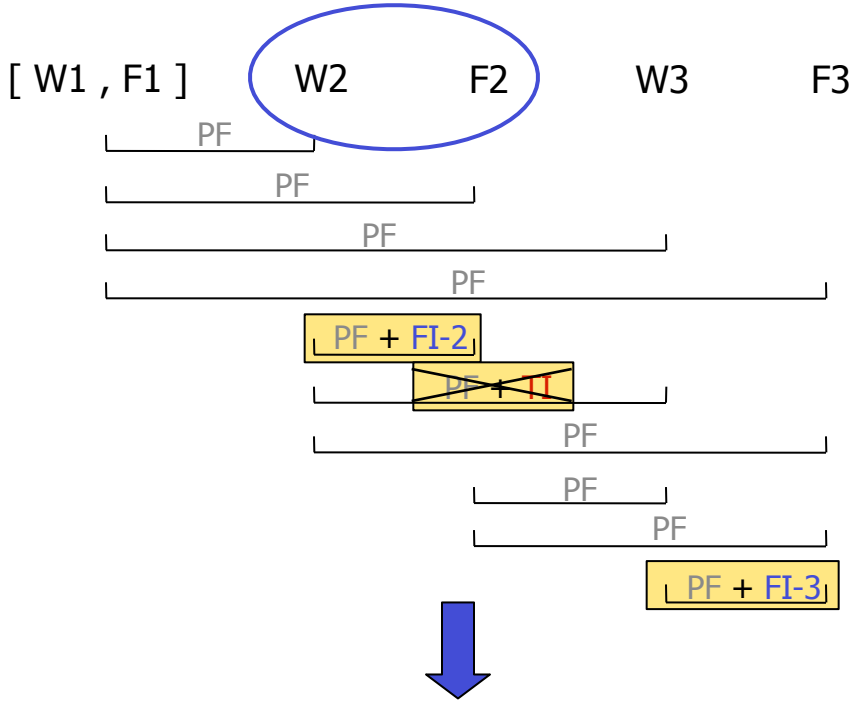




Misleading Conjunction and Token-Subsequence Signatures

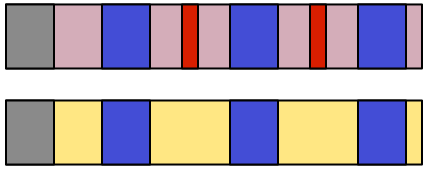
- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow
- P(FP | FI) < P(FP | TI)
- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant

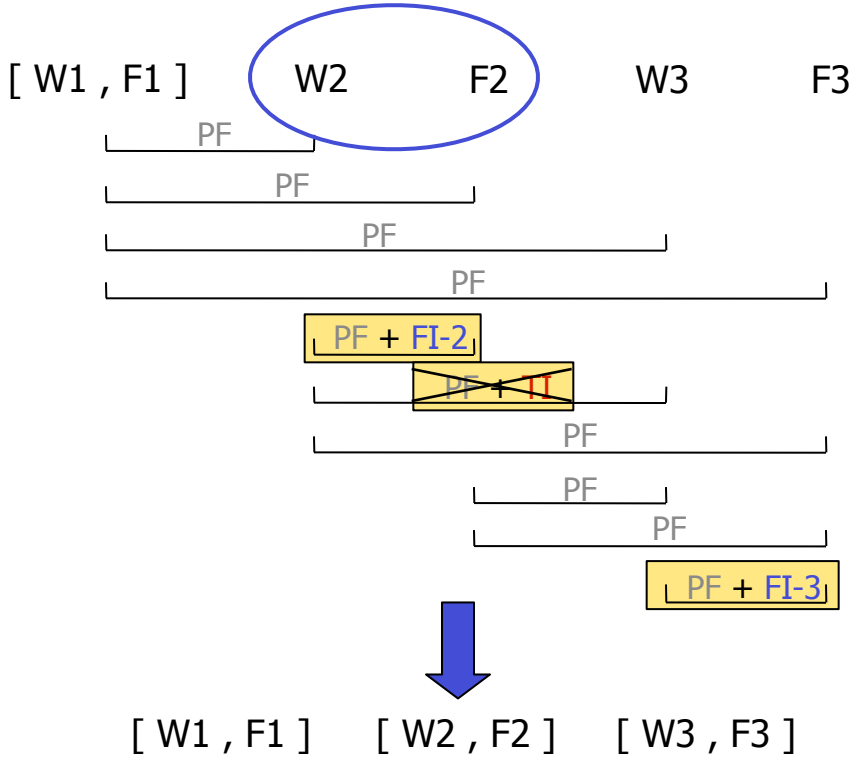




Misleading Conjunction and Token-Subsequence Signatures

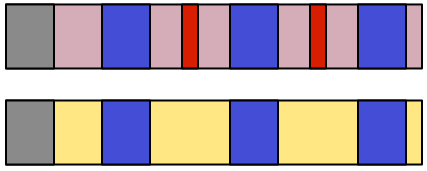
- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow
- P(FP | FI) < P(FP | TI)
- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant

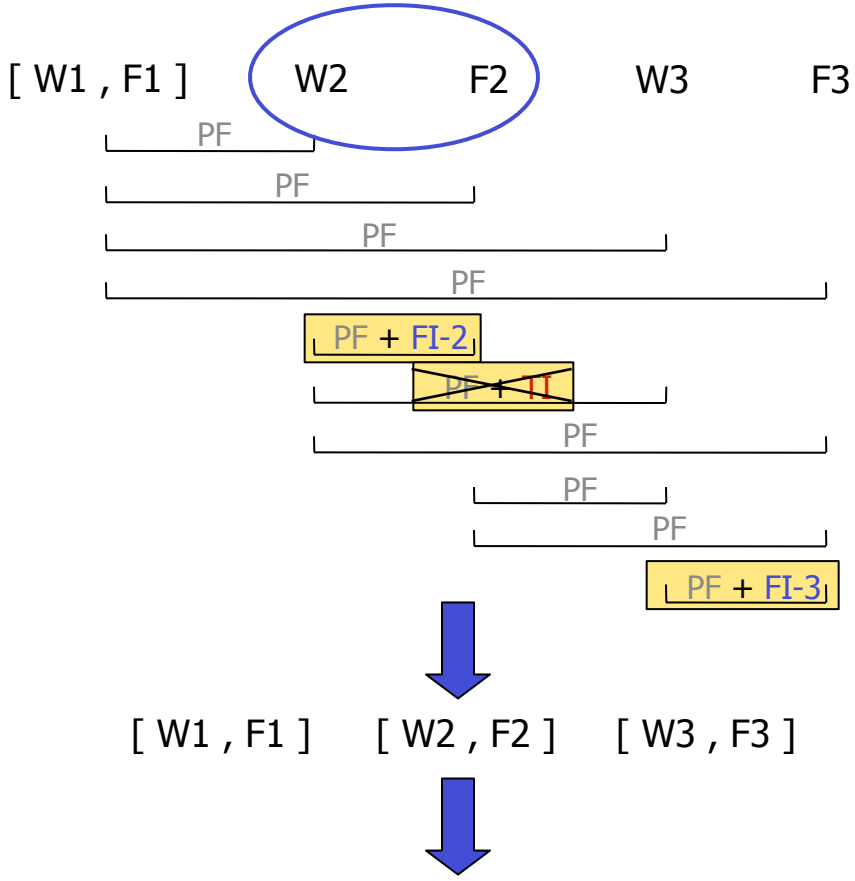




Misleading Conjunction and Token-Subsequence Signatures

- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow
- P(FP | FI) < P(FP | TI)
- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant





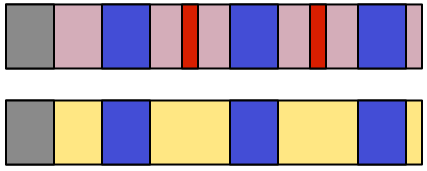
Min num of flows = 3 ➡ NO SIGNATURE!

The results are not Deterministically Predictable

- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant
- **C** = Common Token **by chance!**



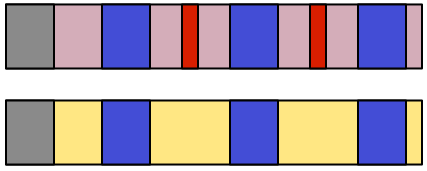
The results are not Deterministically Predictable

W1 F1 W2 F2 W3 F3

- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant
- **C** = Common Token **by chance!**



The results are not Deterministically Predictable

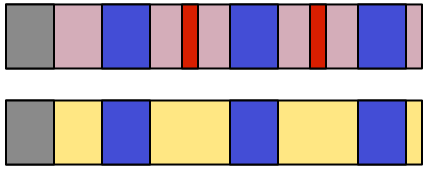
W1 F1 W2 F2 W3 F3

PF + FI-1

- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant
- **C** = Common Token **by chance!**



Worm

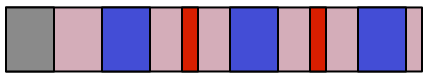
Fake anomalous flow

The results are not Deterministically Predictable

- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

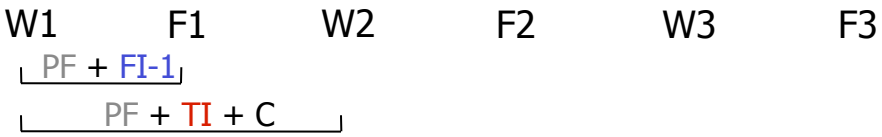
- P(FP | FI) < P(FP | TI)

- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant
- **C** = Common Token **by chance!**



Worm

Fake anomalous flow

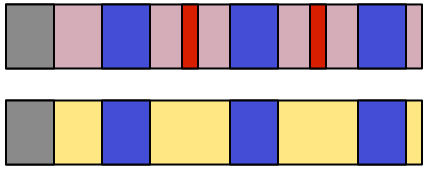


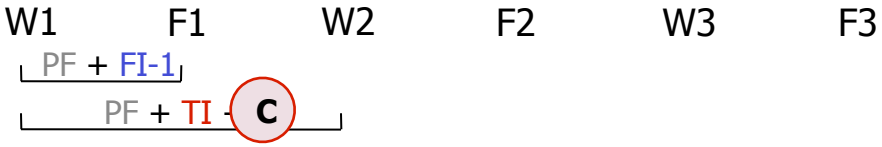
The results are not Deterministically Predictable

- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant
- **C** = Common Token **by chance!**



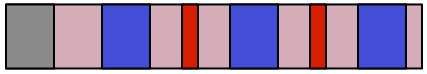


The results are not Deterministically Predictable

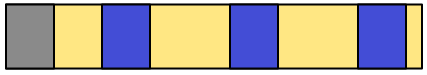
- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant
- **C** = Common Token **by chance!**



Worm



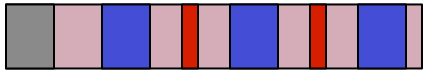
Fake anomalous flow

The results are not Deterministically Predictable

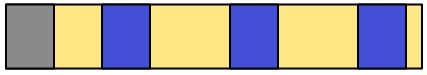
- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

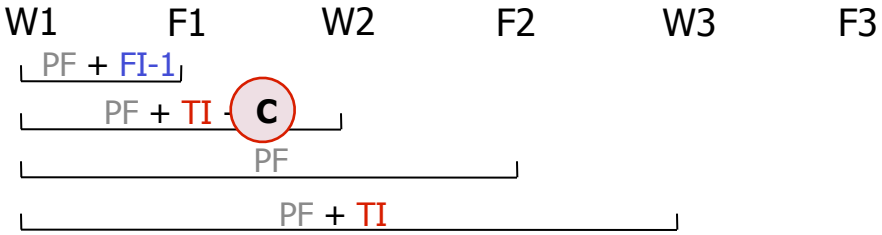
- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant
- **C** = Common Token **by chance!**



Worm



Fake anomalous flow

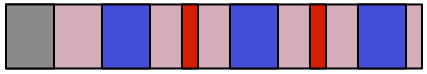


The results are not Deterministically Predictable

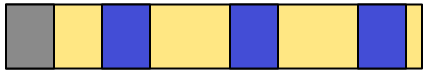
- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

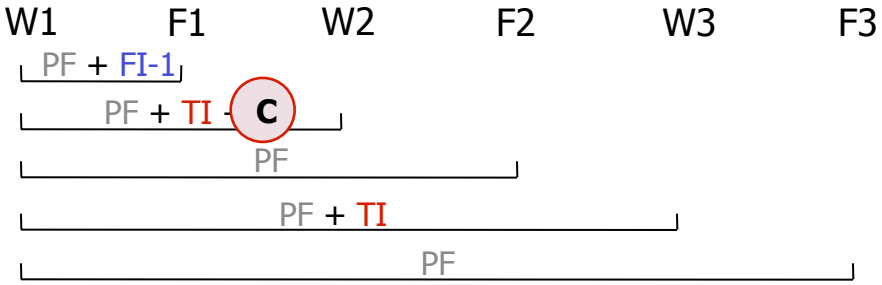
- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant
- **C** = Common Token **by chance!**



Worm



Fake anomalous flow

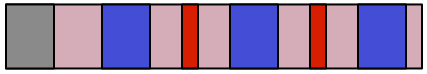


The results are not Deterministically Predictable

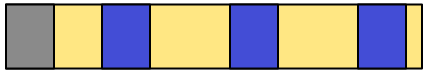
- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

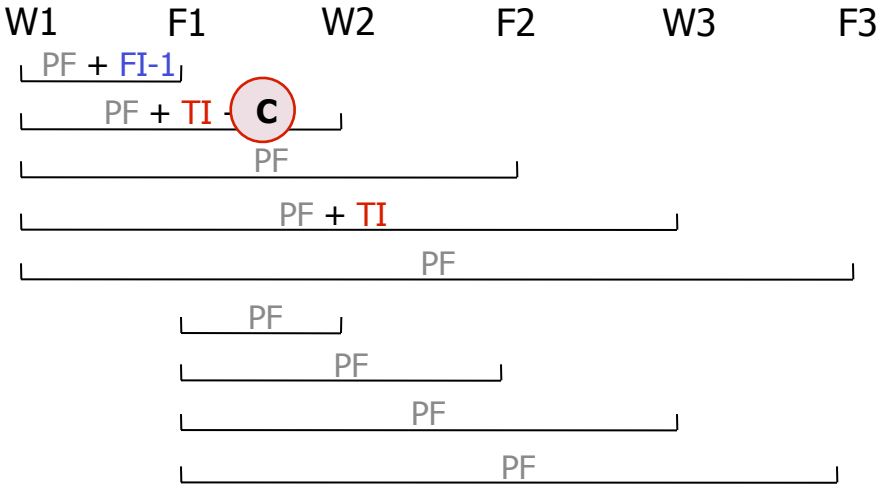
- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant
- **C** = Common Token **by chance!**



Worm



Fake anomalous flow

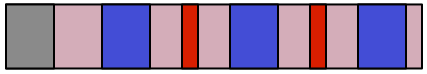


The results are not Deterministically Predictable

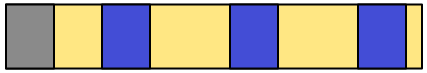
- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

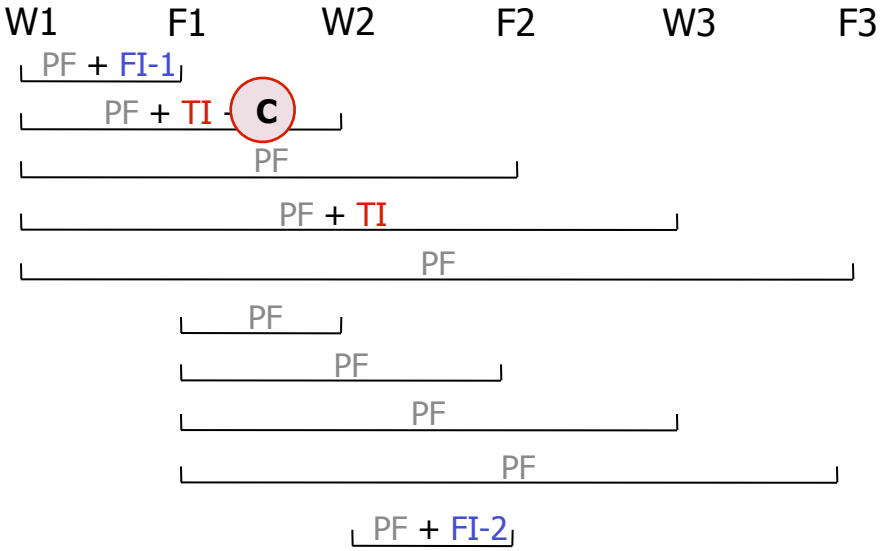
- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant
- **C** = Common Token **by chance!**



Worm



Fake anomalous flow

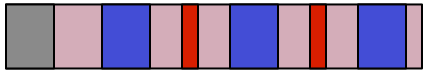


The results are not Deterministically Predictable

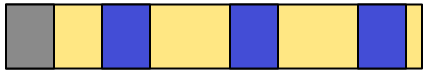
- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

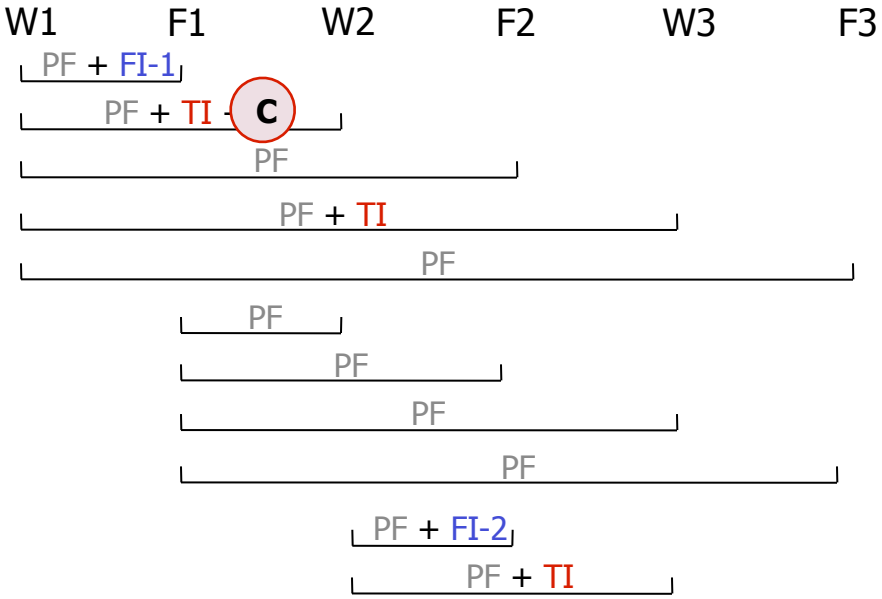
- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant
- **C** = Common Token **by chance!**



Worm



Fake anomalous flow

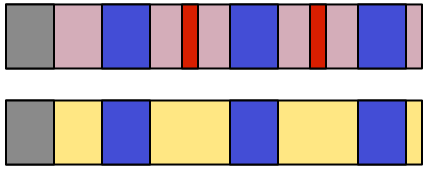


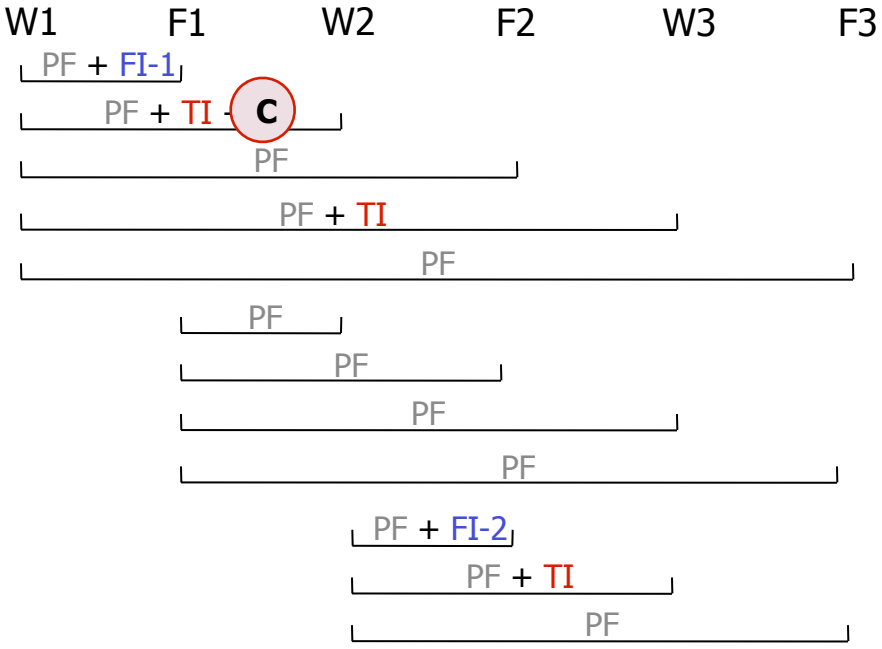
The results are not Deterministically Predictable

- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant
- **C** = Common Token **by chance!**



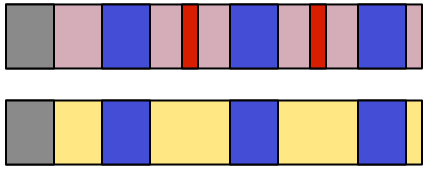


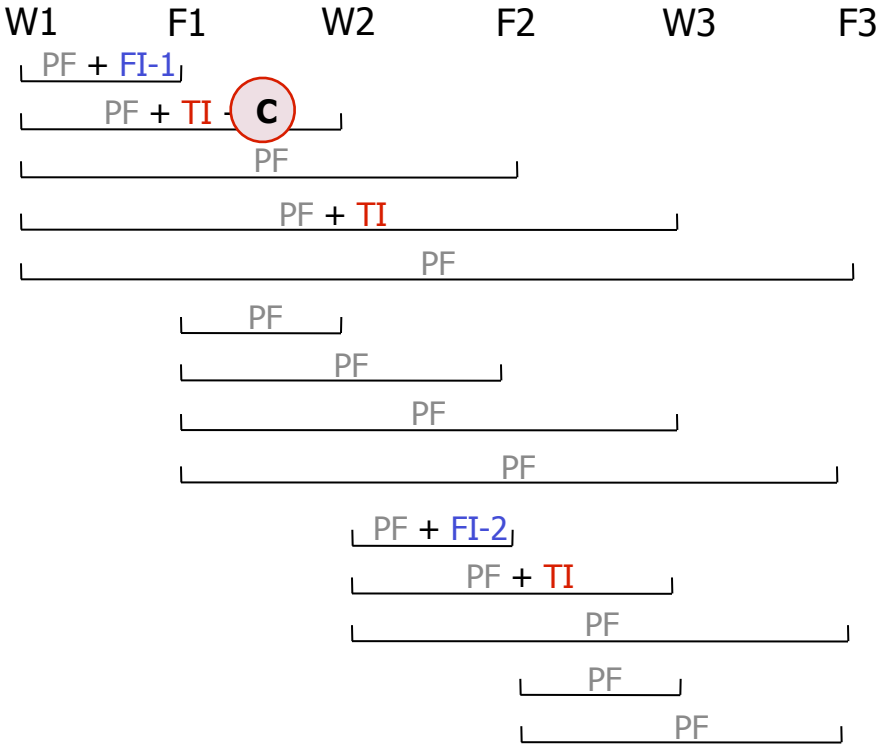
The results are not Deterministically Predictable

- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant
- **C** = Common Token **by chance!**



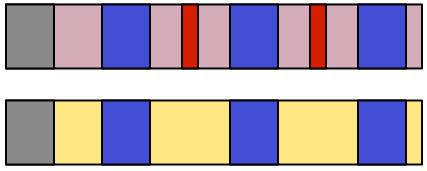


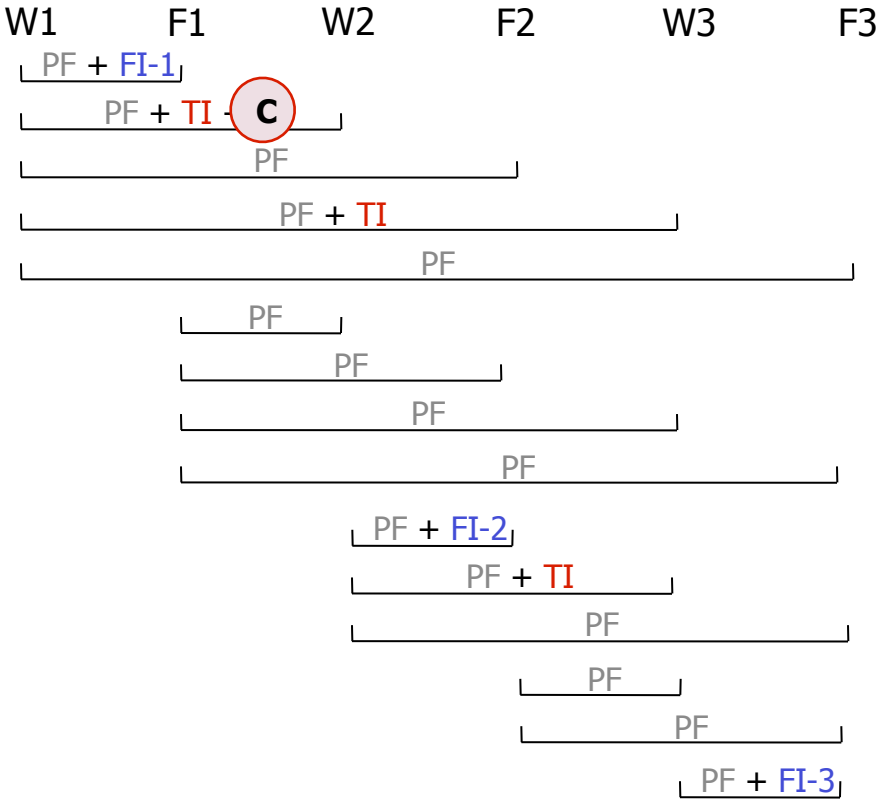
The results are not Deterministically Predictable

- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant
- **C** = Common Token **by chance!**



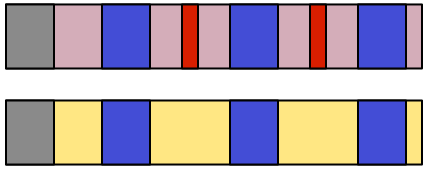


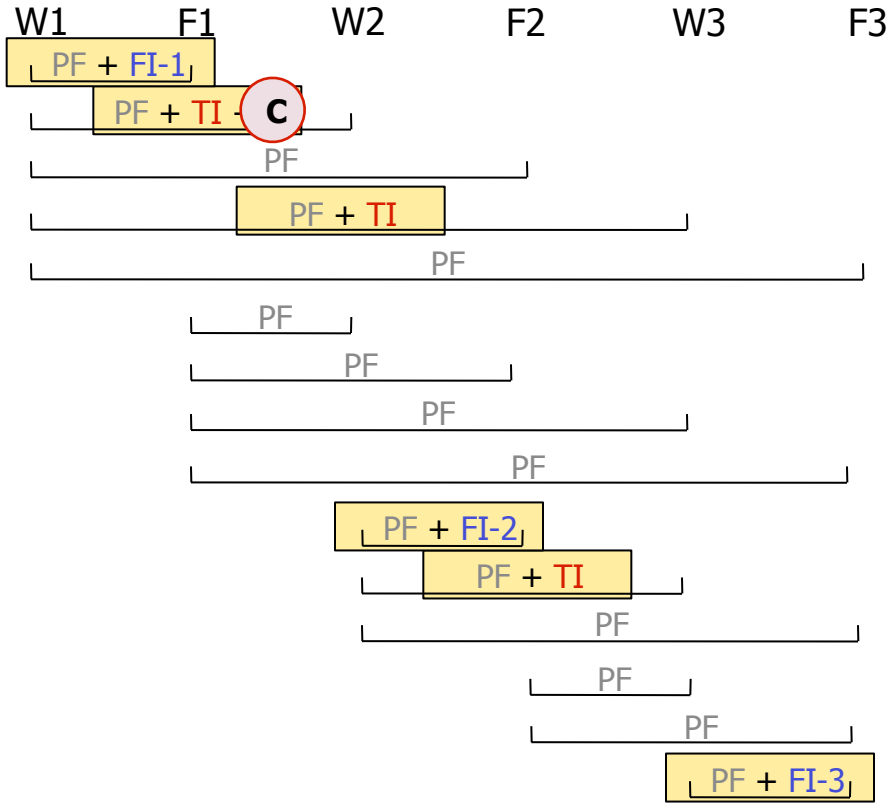
The results are not Deterministically Predictable

- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant
- **C** = Common Token **by chance!**



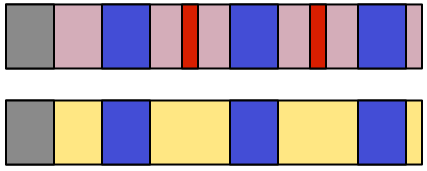


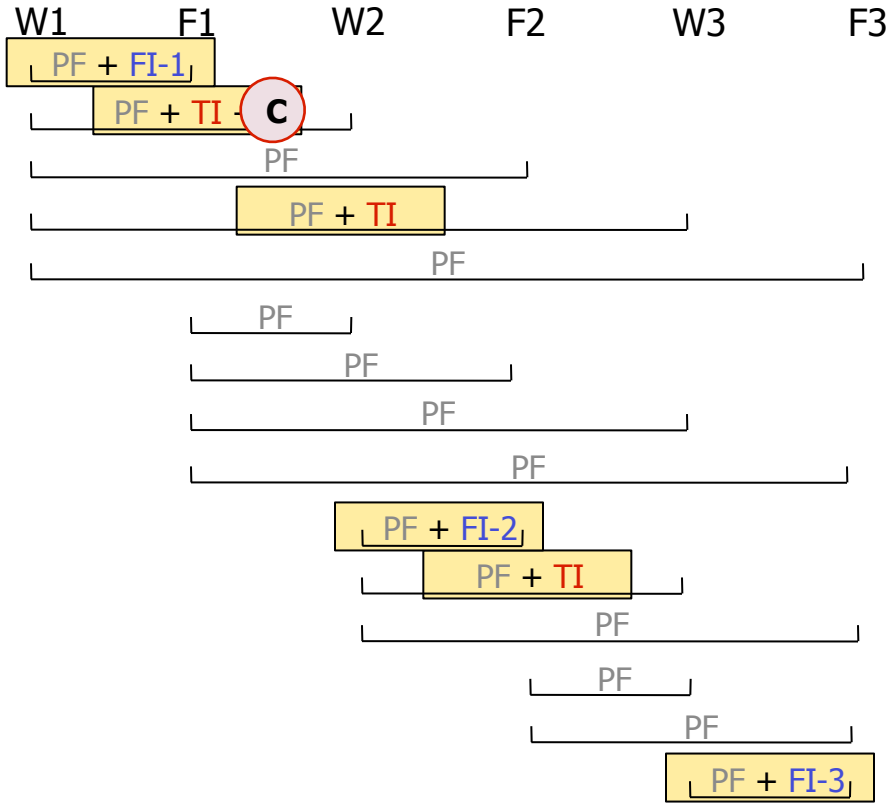
The results are not Deterministically Predictable

- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow

- P(FP | FI) < P(FP | TI)

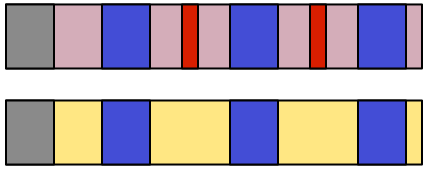
- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant
- **C** = Common Token **by chance!**

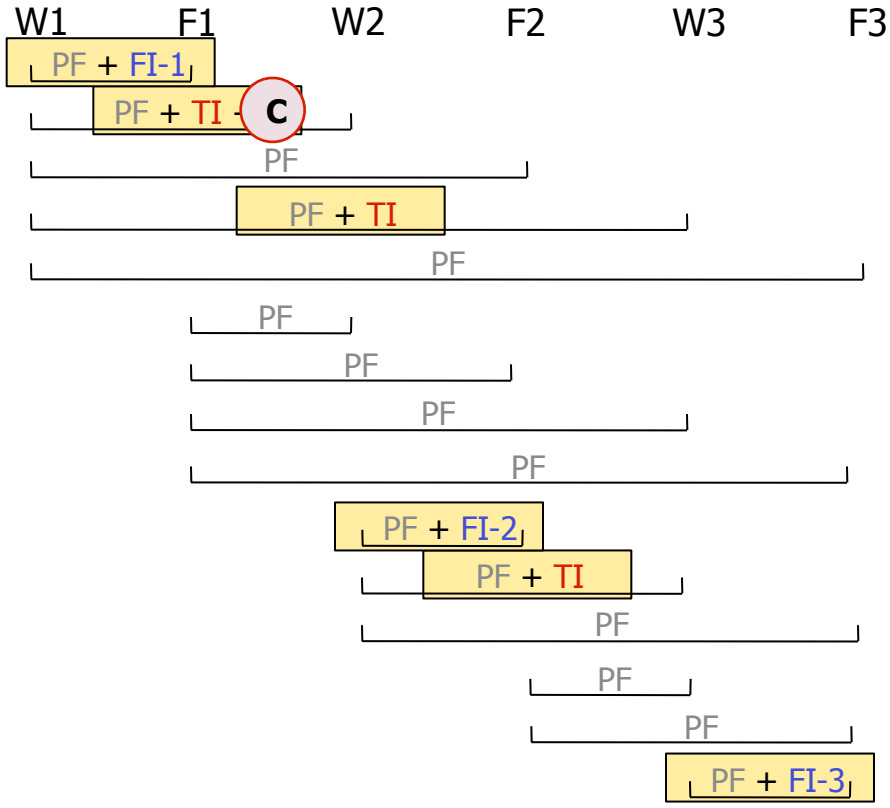




The results are not Deterministically Predictable

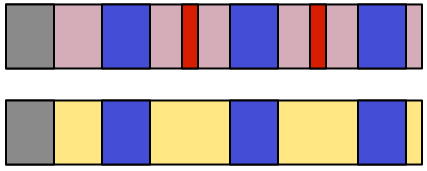
- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow
- P(FP | FI) >= P(FP | TI,C)
- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant
- **C** = Common Token **by chance!**

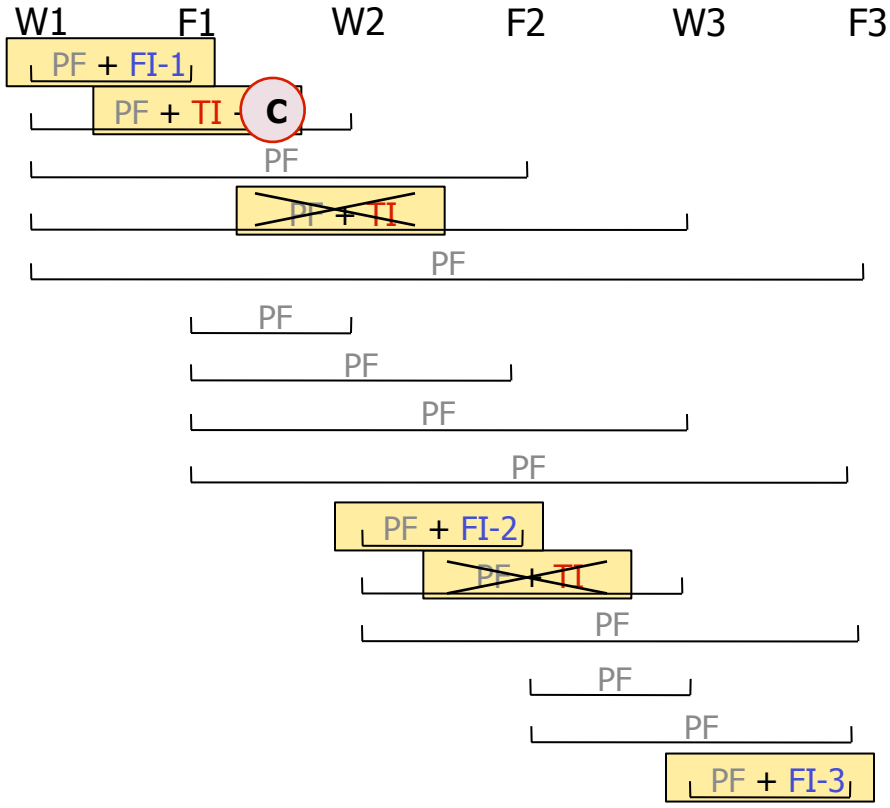




The results are not Deterministically Predictable

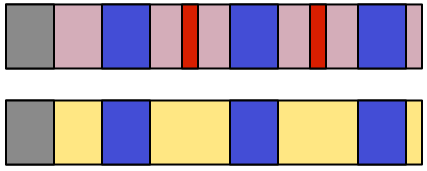
- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow
- P(FP | FI) >= P(FP | TI,C)
- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant
- **C** = Common Token **by chance!**





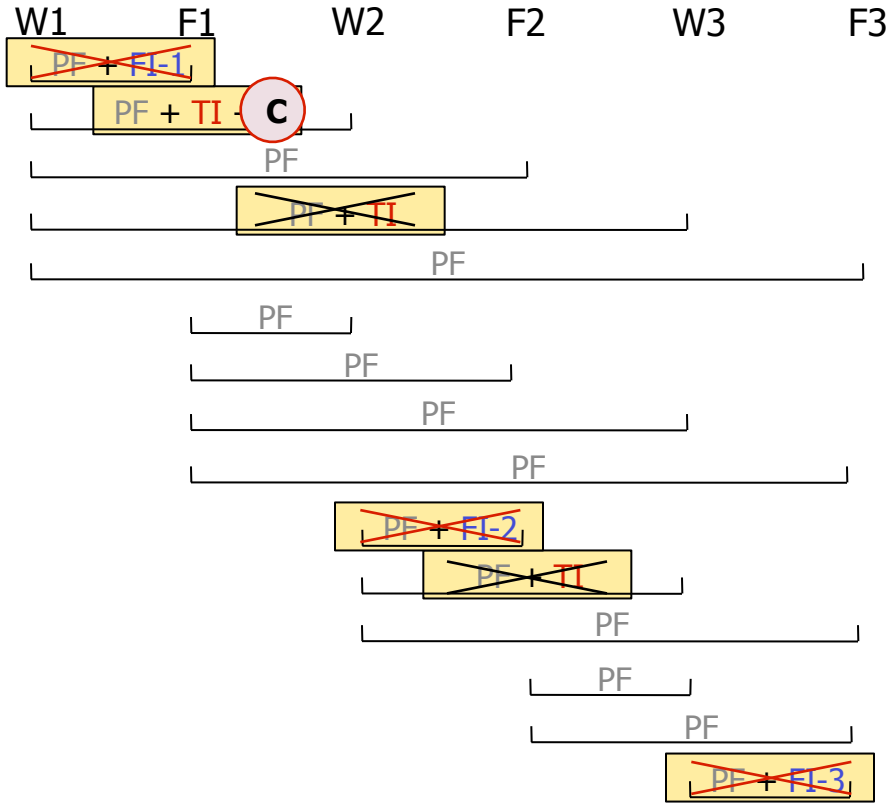
The results are not Deterministically Predictable

- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow
- P(FP | FI) >= P(FP | TI,C)
- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant
- **C** = Common Token **by chance!**



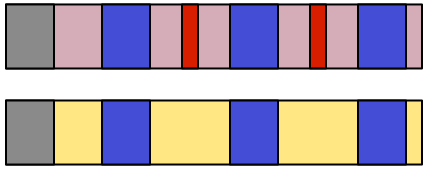
Worm

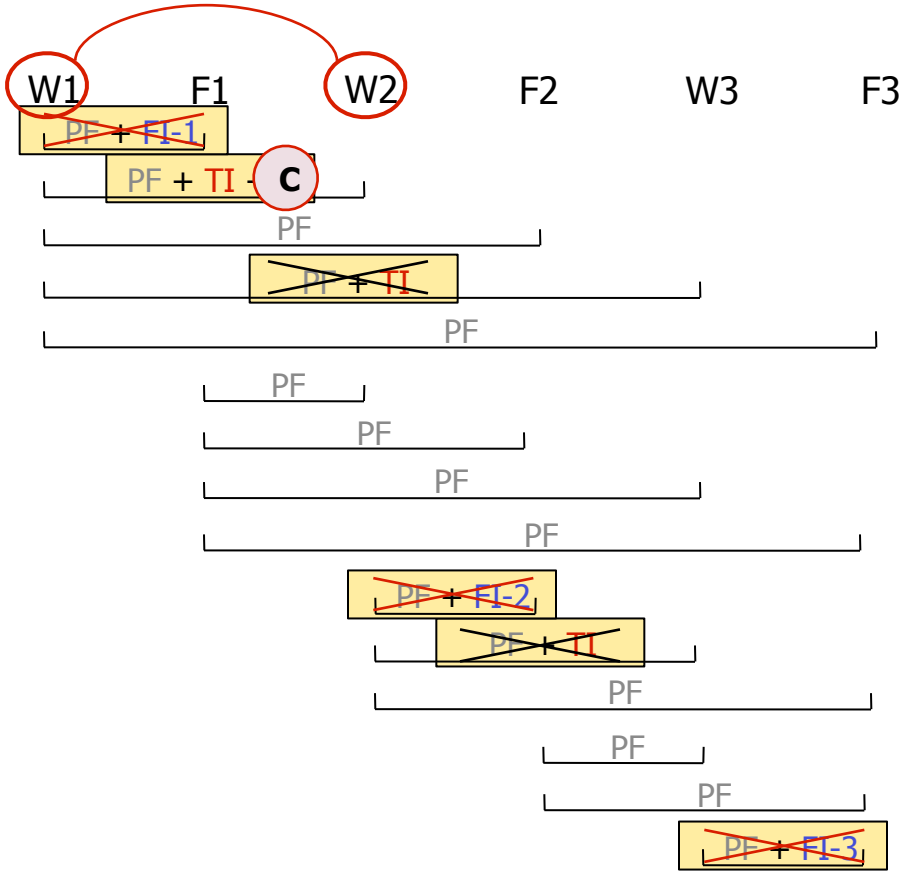
Fake anomalous flow



The results are not Deterministically Predictable

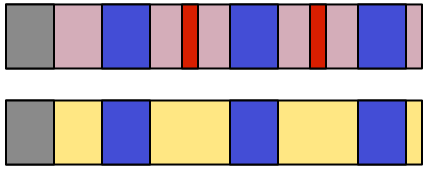
- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow
- P(FP | FI) >= P(FP | TI,C)
- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant
- **C** = Common Token **by chance!**

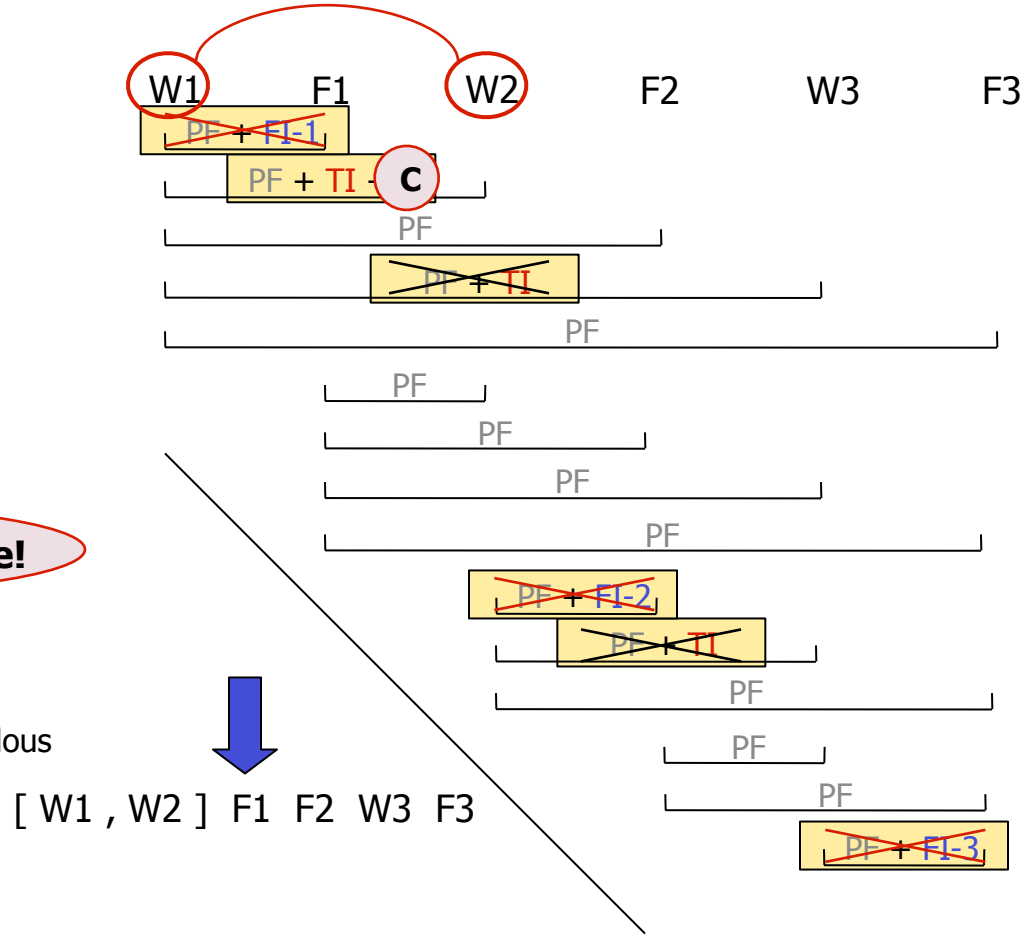




The results are not Deterministically Predictable

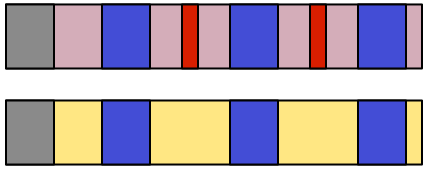
- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow
- P(FP | FI) >= P(FP | TI,C)
- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant
- **C** = Common Token **by chance!**

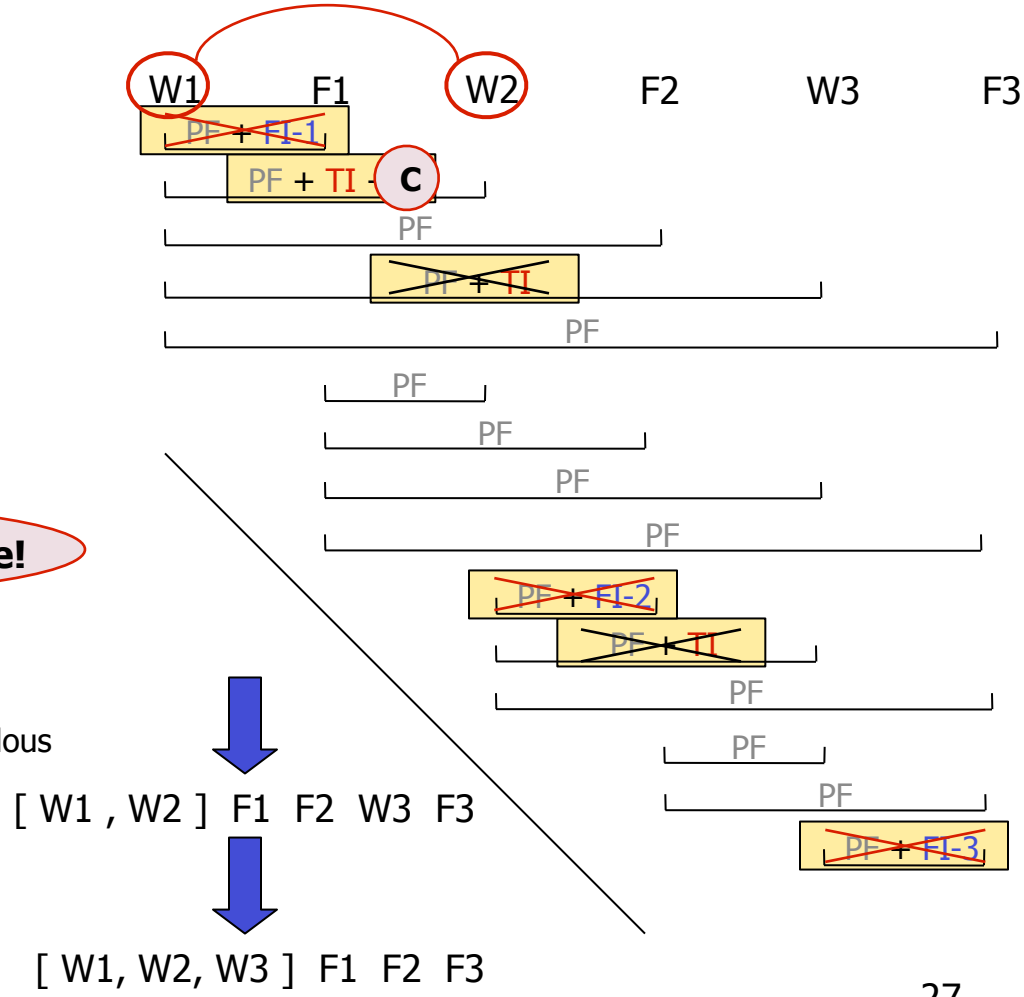




The results are not Deterministically Predictable

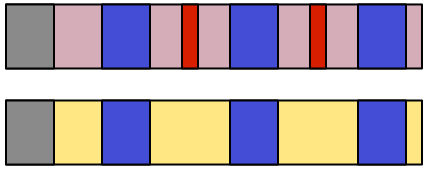
- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow
- P(FP | FI) >= P(FP | TI,C)
- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant
- **C** = Common Token **by chance!**

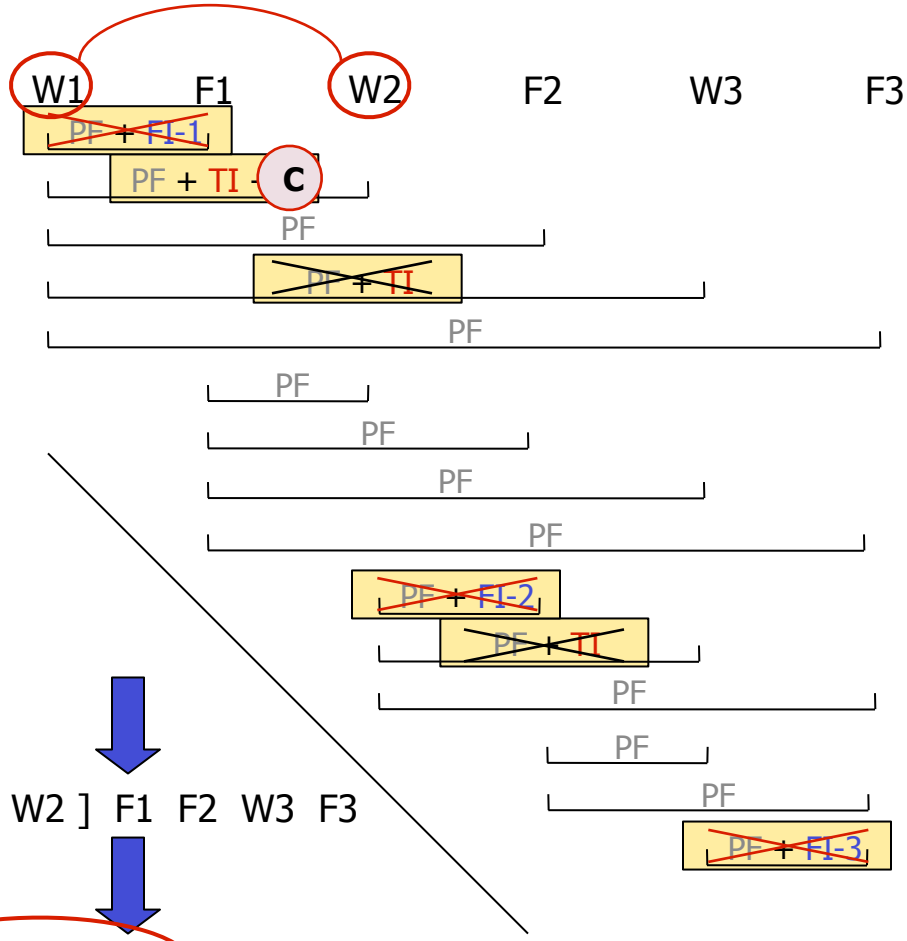




The results are not Deterministically Predictable

- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow
- P(FP | FI) >= P(FP | TI,C)
- PF = Protocol Framework
- TI = True Invariant
- FI = Fake Invariant
- **C** = Common Token **by chance!**





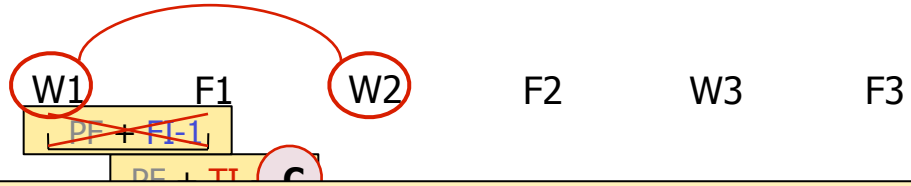
[W1 , W2] F1 F2 W3 F3

[W1, W2, W3] F1 F2 F3

Good signature!

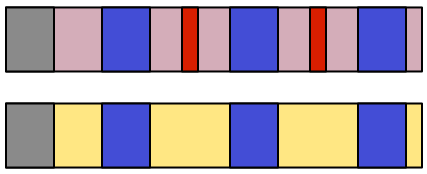
The results are not Deterministically Predictable

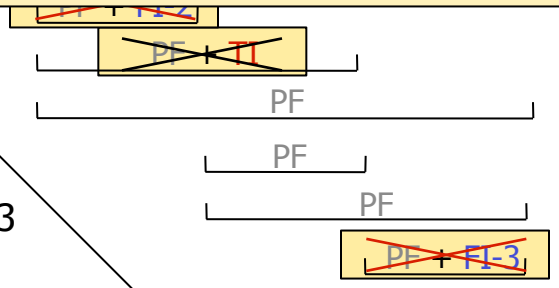
- Wi = i-th Worm variant
- Fi = i-th Fake anomalous flow



A useful signature could be produced, by chance

However, our experiments showed that the
Noise Injection Attach has a high probability of success





[W1 , W2] F1 F2 W3 F3

[W1, W2, W3] F1 F2 F3

Good signature!

Possible countermeasures

- White list
- “Coloring” technique