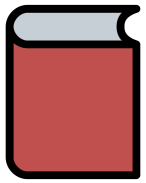
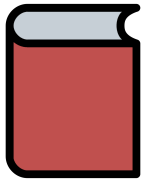


Machine Learning for Security: History

E.g., Wenke Lee's Ph.D. thesis 1994-99

A Data Mining Framework for Constructing Features and Models for Intrusion Detection Systems

(Cited by academic papers approx. 1500 times)



Machine Learning for Security: History

Getting labeled data was very hard

- Standard/reference dataset even harder

Feature construction the key to classification performance, but only semi-automated

- Encoding domain knowledge and extract features from raw data

Security community was VERY skeptical

- False positives (actionable?)
- If domain knowledge is required, why not just expert rules?

ML For Security: Recent Work

Recent Work 2005 - 2015

Data becomes more available

- Easier to get and run malware samples
  - Simply, much more are out there
- Organizations more supportive of data logging
  - Get (IRB) approval

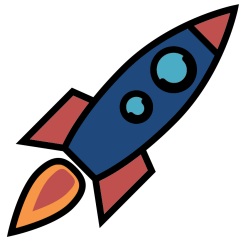
Standard/reference datasets still rare

- Privacy concerns
- Some malware samples “sensitive”

ML For Security: Recent Work

On-Going Work

- Much more acceptance in security and networking communities (e.g., many papers, and even products)
- Need to analyze huge amount of data
  - Including alerts from security sensors
- Benefits of automated data analysis
  - Correlation analysis of alerts
  - Extraction of "interesting" patterns
  - Validation of expert knowledge/intuition



ML For Security: Future

The bar is much higher than other ML applications because security implies adversarial environments

Attackers will try to defeat ML-based security mechanisms:

- pollute the training data to cause ML algorithm to produce the wrong model
- study the features and ML algorithm and adapt attacks to evade the ML-based model

Adversarial Machine Learning

Earlier research dated back more than ten years, e.g.,

- Mimicry attacks on system call sequence models (Wagner/Berkeley, 2002)
- Polymorphic blending attacks on payload entropy models (Lee/GT, 2006)
- Misleading worm signature generators (Lee/GT, 2006)

Adversarial Machine Learning

More recent research, e.g.,

- Conceptual framework (Joseph/Berkeley, 2010)
- Limitations of Deep Learning (Jha/Wisconsin, 2016)

More recent attacks/applications, e.g.,

- Black-hat SEO, e.g., with faked user actions/signals (Lee/GT, 2013)

Attacks on Machine Learning

Exploratory attack

- Attacker generates examples to probe the defender's ML system to infer its decision boundary and then craft his attack to evade the ML-based model
- Also called evasion attack

Attacks on Machine Learning

Causative attack

- Attacker injects malicious examples to affect the ML training process that as a result is not able to produce an effective ML-based model
- Also called data poisoning attack

Evasion Tactics Quiz

Match the attack to its description:

obfuscating internal data (O), confusing automated tools (C), environmental awareness(E), timing-based evasion (T)

E allows malware samples to detect the underlying runtime environment of the system it is trying to infect.

C allows malware to avoid detection by technologies such as signature based antivirus software.

T used by malware to run at certain times or following certain actions taken by the user.

O uses a number of tricks to run code that cannot be detected by the analysis system

Dyre Wolf Attack

Attack Steps

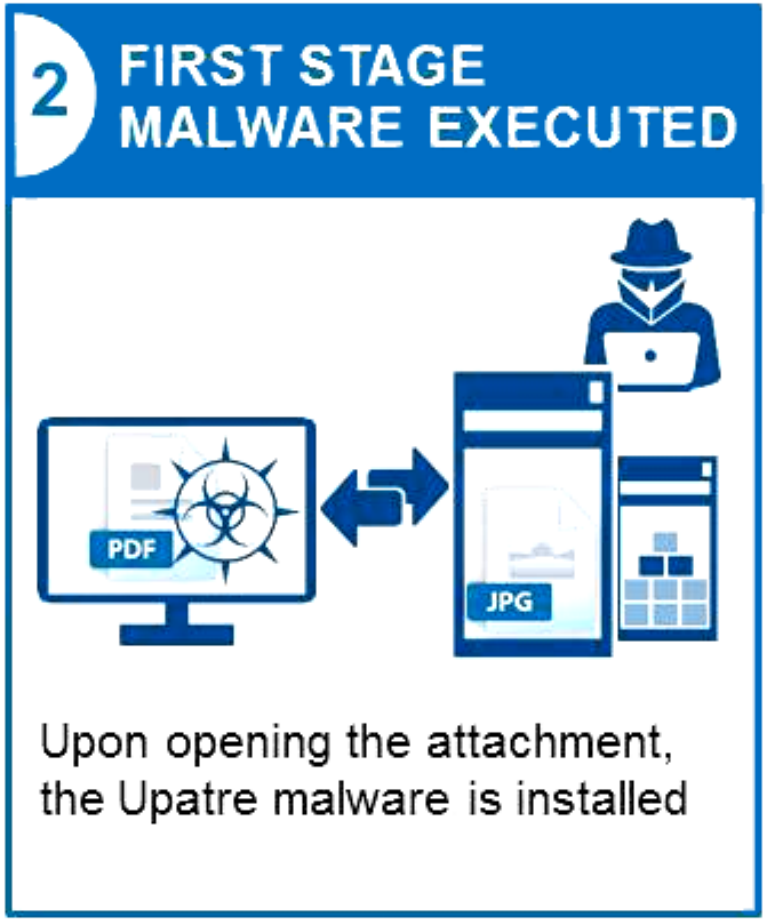
1 SPEAR PHISHING

An employee within the targeted organization receives an email with the Upatre malware

IBM

Dyre Wolf Attack

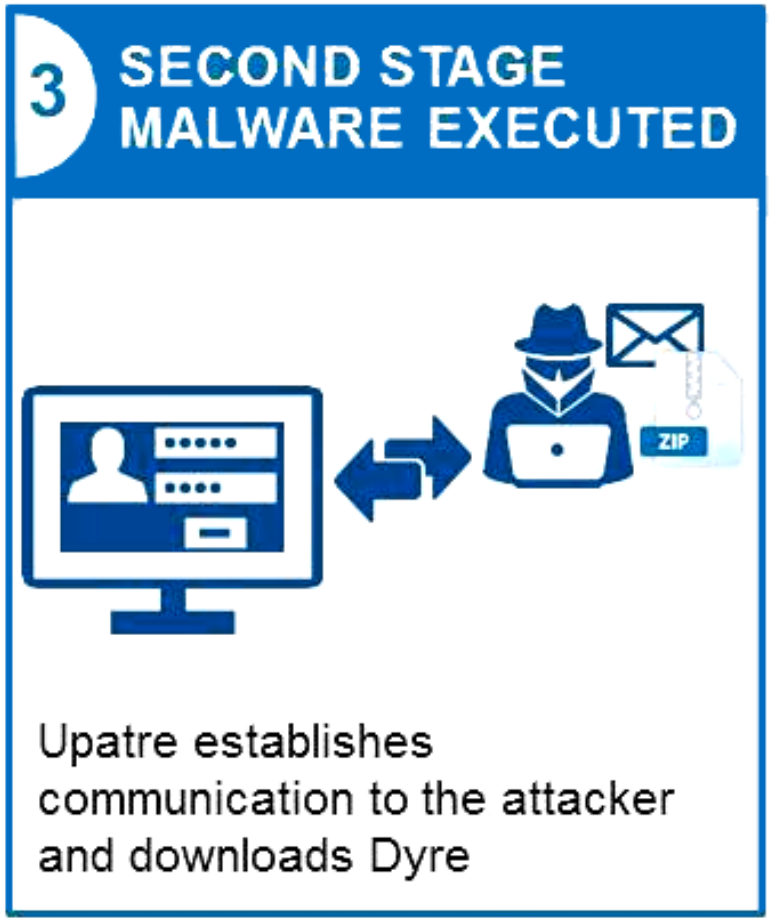
Attack Steps



IBM

Dyre Wolf Attack

Attack Steps



Dyre Wolf Attack

Attack Steps

IBM

Dyre Wolf Attack

Attack Steps

5 THE PHONE CALL
ADVANCED SOCIAL ENGINEERING

To overcome measures by the bank to protect against fraud; the Dyre Wolf social engineers critical information from the victim

IBM

Dyre Wolf Attack

Attack Steps

6 THE WIRE TRANSFER

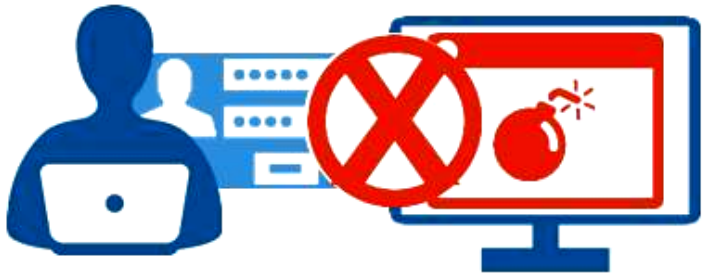
Upwards of $1.5 million USD is quickly and efficiently transferred from the victim's account to several offshore accounts

IBM

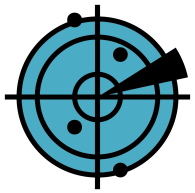
Dyre Wolf Attack

Attack Steps

7 THE DDOS



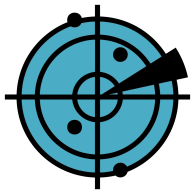
Immediately after the theft, a high volume DDoS against the victim starts; in order to distract or hinder investigation



PAYL: Payload-Based Anomaly Detection System

Measure and model the frequency distribution of *n*-grams in payloads of each network service

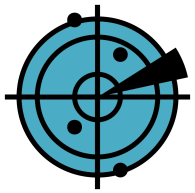
- For n=1, it models the byte, or, character, frequency distribution



PAYL: Payload-Based Anomaly Detection System

Intuition: the service (e.g., web, mail, etc.) delivered to a host or an enterprise network has unique characteristics

- E.g., the user(s) normally gets certain types of web or email contents
- Attacks embedded in network traffic, e.g., malware in email attachment, are very different kind of contents

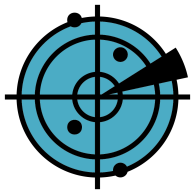


PAYL: Payload-Based Anomaly Detection System

Simple model:

For x_i, the ith gram, i.e., a character (e.g., "A") in 1-gram PAYL

- Compute its relative frequency $f(x_i)$ and the standard deviation $s(x_i)$ in normal traffic
- The set of $\{f(x_i), s(x_i))$ for all x_i's (e.g., all characters) is the model of normal traffic

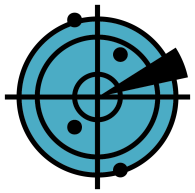


PAYL: Payload-Based Anomaly Detection System

Compare a packet to the normal profile, anomaly score is:

$$score(P) = \sum_{i}^{o} \frac{f(x_i) - f(x_i)}{\sigma(x_i) + \alpha}$$

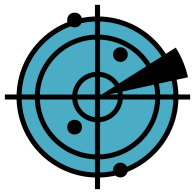
α is a smoothing factor to prevent division by 0



PAYL: Payload-Based Anomaly Detection System

Detection approach

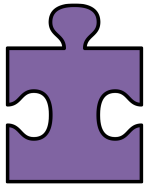
- Generate a separate payload model for each packet length
  - Packets are compared with the profiles for their lengths
- Packets with unusual length are flagged as anomalous



PAYL: Payload-Based Anomaly Detection System

Advantages:

- Very simple and efficient
- Can detect zero-day attacks, and polymorphic attacks
  - Anomaly detection, not based on signatures

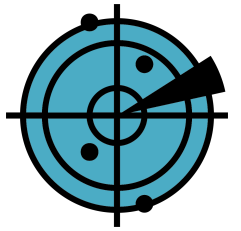


Polymorphism Quiz

Which of the following statements are true? Check all that are true:

- [x] A polymorphic attack can change its appearance with every instance.
- [x] A polymorphic attack has no predictable signature for the attack.
- [ ] Each instance of polymorphic code has different, but normal, appearance.

Polymorphic Attacks vs. Detection



A polymorphic attack has three components:

Attack vector used for exploiting vulnerability

- Some parts can be modified but there is always a set of invariant parts (the starting point of execution)
- If invariant parts are small and exist in legitimate traffic, detection can be evaded

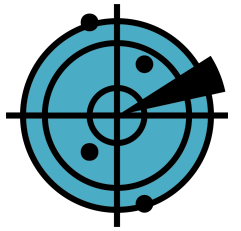
Attack body - malicious code for the attacker's purpose; shell code

- Can be transformed or encrypted

Polymorphic decryptor

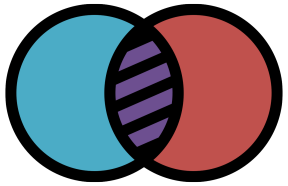
- Decrypts the shell code, can be transformed

Polymorphic Attacks vs. Detection



Byte frequency of malicious code tends to be anomalous

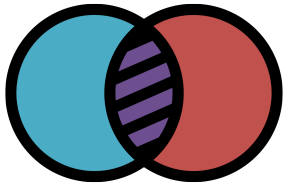
- Encrypted/transformed code different from legit contents



Evading Detection: Polymorphic Blending Attacks

Attack can blend in if it can mimic simple statistics observed in legitimate traffic

- Average size and rate of packets, byte frequency distribution, range of tokens at different offsets
- Attacker must carefully choose encryption key and pad its payload to replicate desired byte frequency



Evading Detection: Polymorphic Blending Attacks

There are more sophisticated approaches to attack detection that cannot be evaded so simply

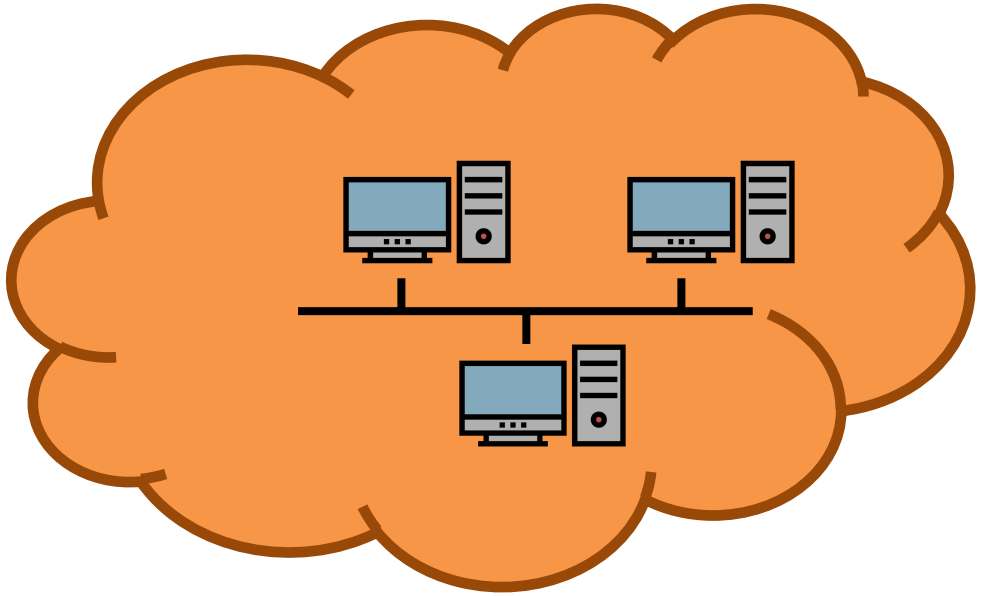
- But simpler detection approaches are more frequently used because they are efficient and scalable

Polymorphic Quiz

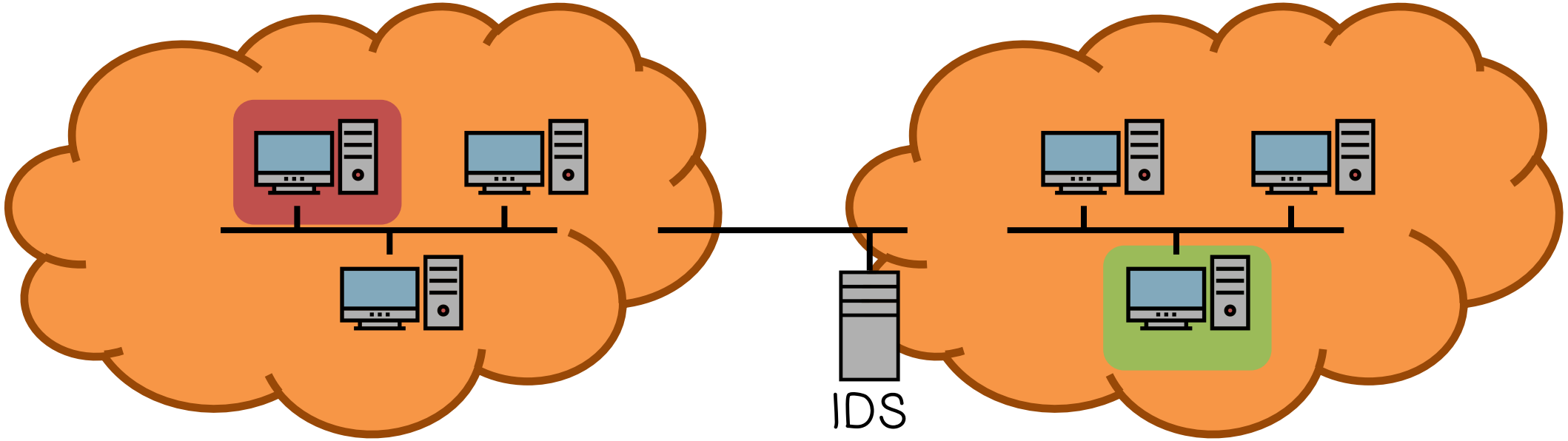
Which of the following are true statements with regards to a polymorphic blending attack?

- [x] The process should not result in an abnormally large attack size
- [x] The blending needs to be economical in time and space
- [ ] The attacker must operate under the constraint on the available resources
- [ ] Attacks of this type must collect a lot of data to learn normal statistics

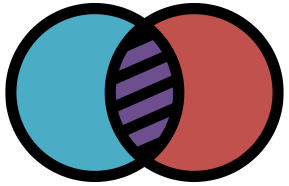
Polymorphic Attack Scenario



Polymorphic Attack Scenario



- Adversary has the knowledge of the *IDS*
- Adversary can observe some normal packets going from adversary's network to victim (or victim's network)
- Adversary has an estimation of *false positive rate* at the *IDS*



Blending Steps

Adversary compromises host X

Observe the normal traffic going from A to host Y

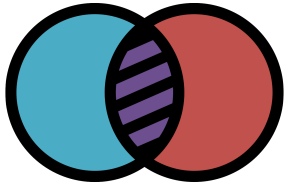
Using IDS modeling algorithm, generate an estimated (artificial) normal profile

Create an attack instance such that it matches the artificial normal profile

- Using shellcode encryption and padding

Launch the attack

- IDS should not be able to detect these attack packets as attack instance



Blending Steps

Artificial Normal Profile

Normal #1

Normal #2

Normal #3

Blending Steps

Artificial Normal Profile

Before blending

Artificial Normal Profile

After padding

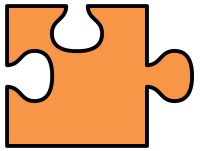
Blending Steps

Artificial Normal Profile

Before blending

Artificial Normal Profile

After padding



Polymorphic Blending Quiz

Is the following statement true or false? Check the box if it is true:

- [ ] After blending, the attack should match the normal profile perfectly

Blending Attacks Requirements

Step of Polymorphic Blending Attacks

1. Learn the IDS normal profile
2. Encrypt the attack body
3. Generate a polymorphic decryptor

Blending Attacks Requirements

Desirable properties of a polymorphic blending attack:

- Match legitimate traffic's byte frequencies
- Does not result in large attack packet size

Blending Attacks Requirements

For encryption, use a substitution cipher

Each byte is transformed into a byte from a legitimate traffic sample to match desired byte frequency

- E.g., every character in the attack body can be substituted by a character(s) observed from the normal traffic using a substitution table

Possibly some padding is added

- The encrypted attack body can be padded with more garbage normal data so that the polymorphic blended attack packet can match the normal profile even better

Encrypting Attack Contents

Greedy Method

Map the most frequent byte value in attack traffic to the most frequent byte value in normal traffic

Map the second most frequent byte in attack traffic to the second most frequent by value in normal traffic

- and so on until all byte values in attack traffic have been mapped

Decryptor

Decryption removes the padding and reverses the substitution steps

This code cannot be blended but can be transformed into equivalent instructions that contain (mostly) normal byte values

The substitution table used for decryption can be stored in an array where the i^{th} entry is the byte value in normal traffic that was used to substitute byte value i in attack traffic

- This table contains only normal byte values

Iterative Process

The attack vector, decryptor, and substitution table are not encrypted

- Including them in attack packets may cause attack traffic to not match the normal profile
  - several iterations of finding a new substitution table may be necessary to achieve the desired match

Evaluation

Create polymorphic blending attacks to evade PAYL

First create polymorphic attacks using CLET and verify that they are all detected by PAYL

Next create polymorphic *blending* attacks and demonstrate that they can evade detection

- Evaluate easiness of attack construction and cost

Evaluation Results

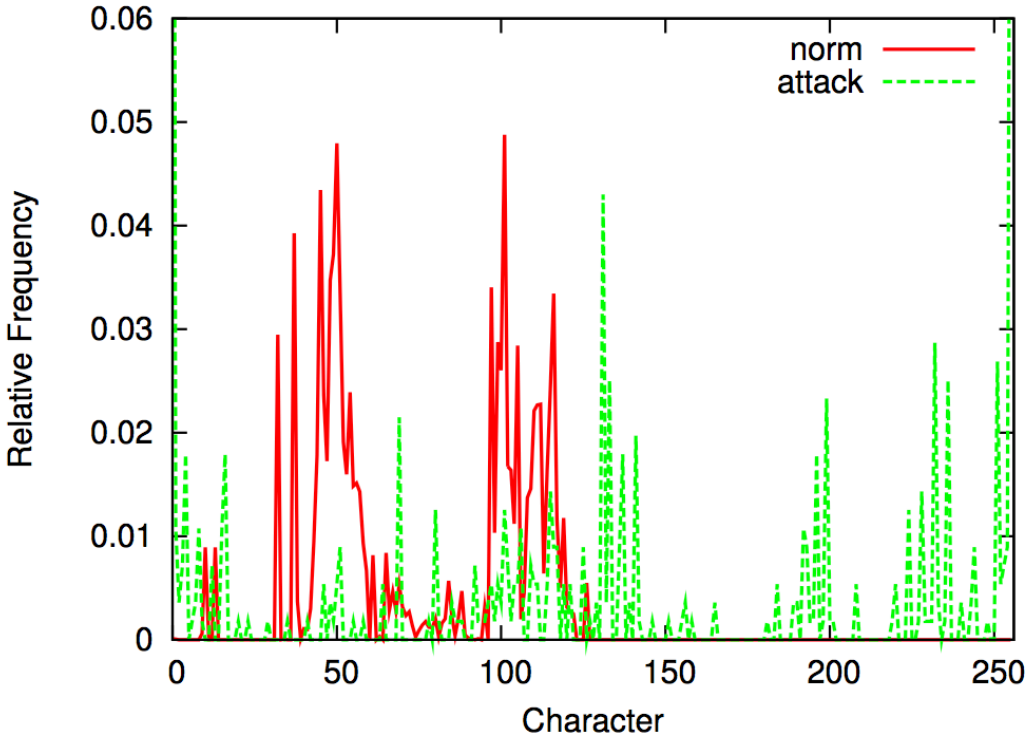
The attacker needs to learn/estimate the normal profile used by PAYL at the target network

- Training of the artificial profile is stopped when there is no significant improvement over existing profile (measured using Manhattan distance) within two packets
- Number of packets required for convergence

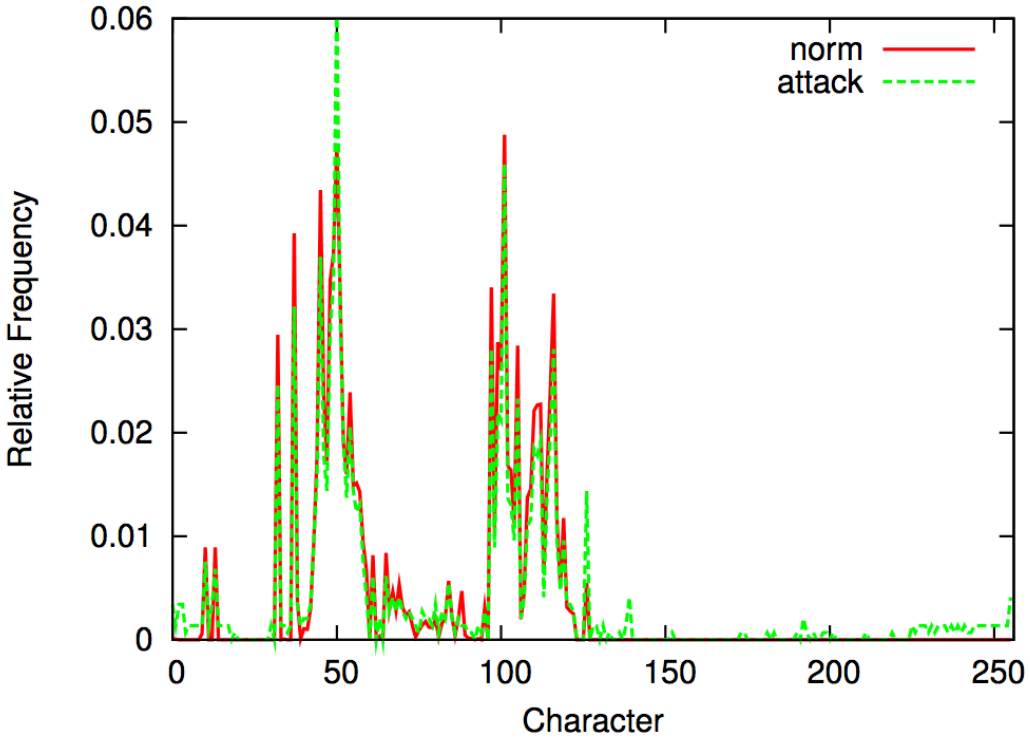
Packet Length	1-gram	2-gram
418	8	20
730	8	18
1460	14	40

Evaluation Results

Frequency Distribution



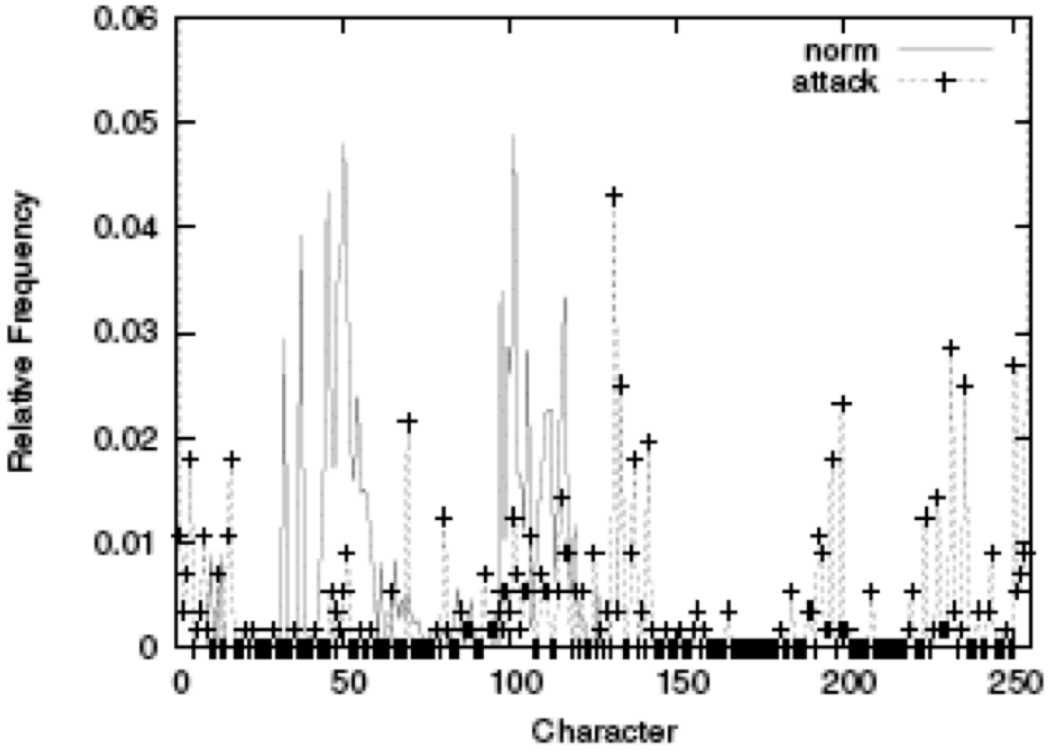
Before blending



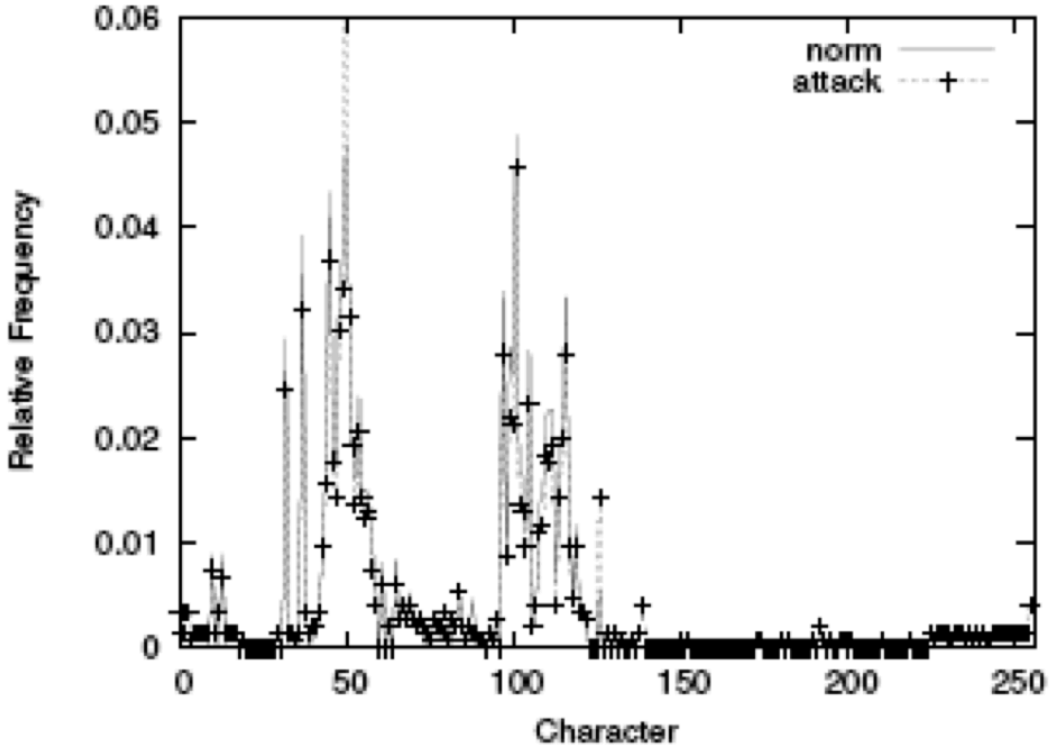
After blending

Evaluation Results

Byte Frequencies



Original attack packet



1-gram Blending packet for packet length 1460

Countermeasures

More complex models are needed

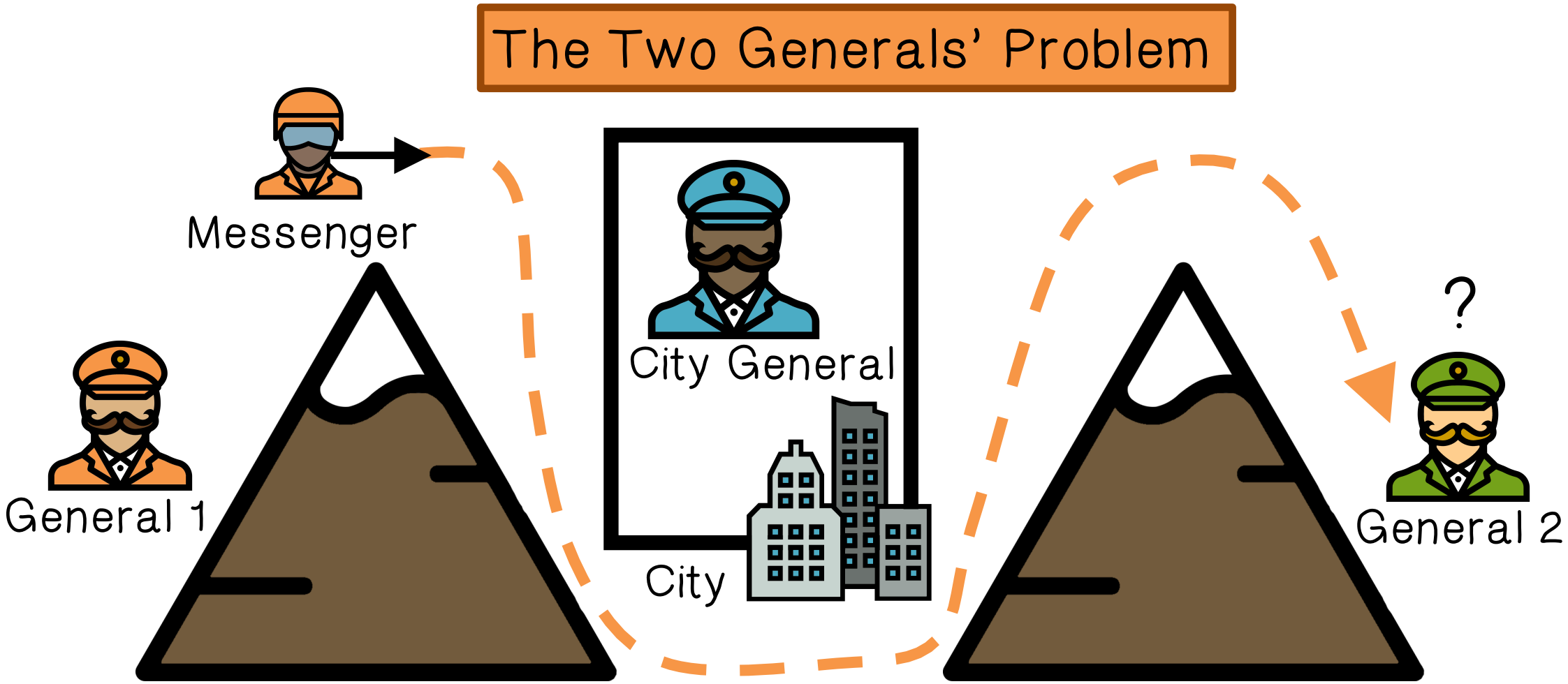
- Observe additional traffic features in addition to statistical ones, e.g., syntactic and semantic information

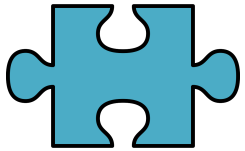
Use multiple simple IDSs that model different features

Introduce randomness into the IDS model, e.g.,

- Model byte pairs that are *v* characters apart
- Choose *v* at random and fix it for a given IDS
- Combine several such systems

Poisoning Attack Quiz

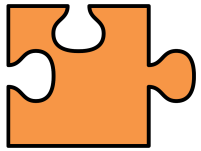




Poisoning Attack Quiz

Keeping in mind the “Two Generals’ Problem”, list some possible motivations for a poisoning attack:

- Modify lists of trusted sites
- Change training data of IDS
- Plant false information
- Propagate false information about routing
- etc.



Poisoning Attack Goals Quiz

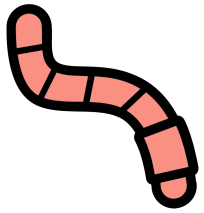
List the goals of a successful poison attack:

- Is undetected
- Continues for a period of time
- Cause damage to data

Poisoning Attack Goals Quiz

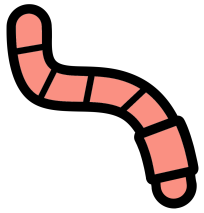
LA residents poison data of navigation app Waze

- Neighborhoods inundated with traffic diverted from congested freeways
- Neighbors used the app to falsely report their streets as congested
- The app would learn on false data and not direct traffic through neighborhood streets
- There were enough Waze users driving through the local streets to offset the poisoned data.



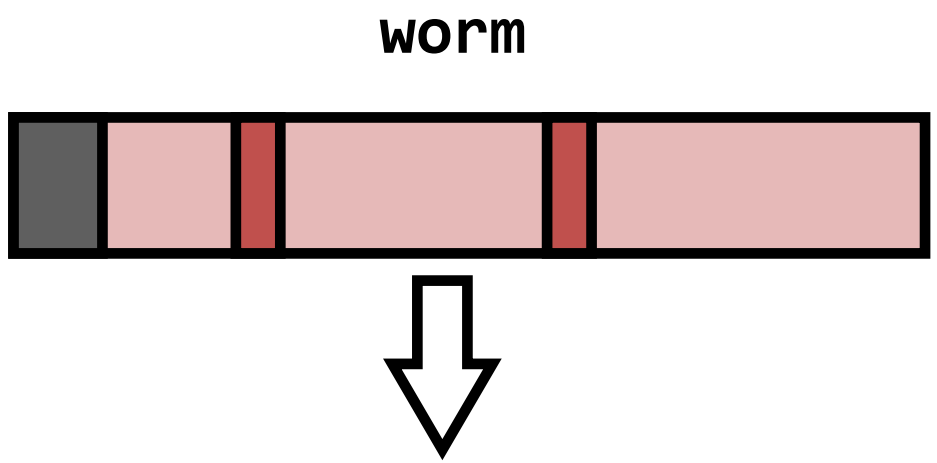
Syntactic Worms

Automatic signature generators look for invariant parts of polymorphic worms

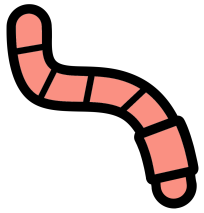


Syntactic Worms

Syntax-based:
Network packets/flows content inspection



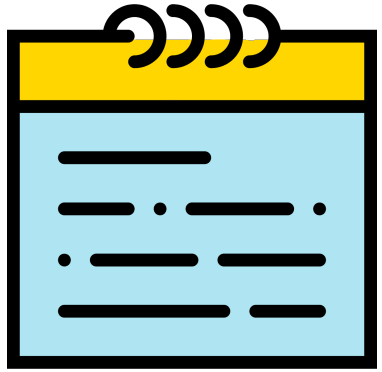

```
GET .* HTTP/1.1 .* Host: .* Host: .* \xFF\xBF .*
```

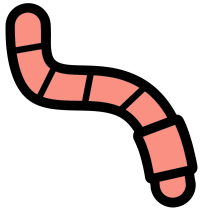


Syntactic Worms

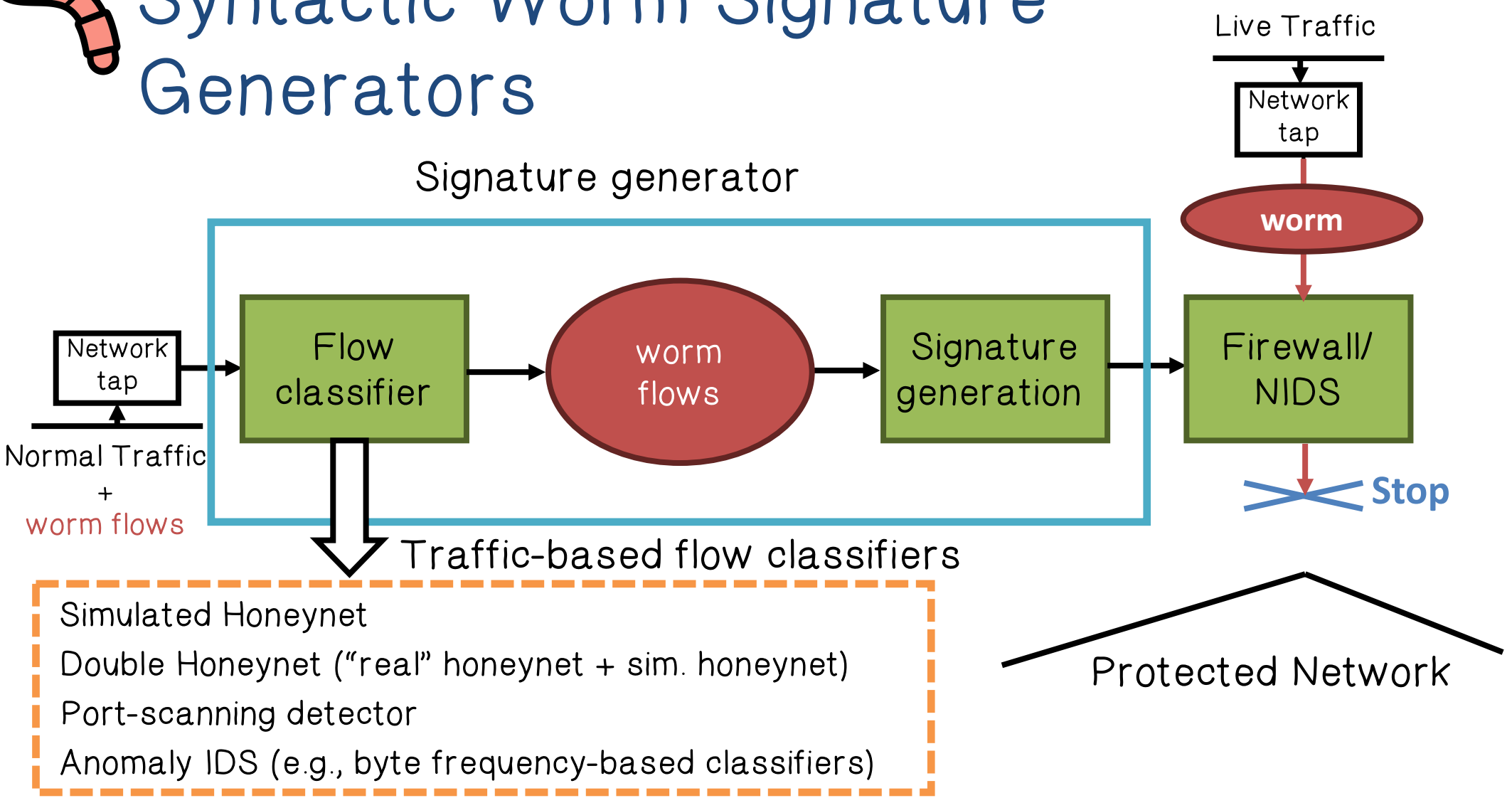
Our study showed:

Syntactic signature generators are vulnerable to Noise Injection Attack and generate a useless signatures

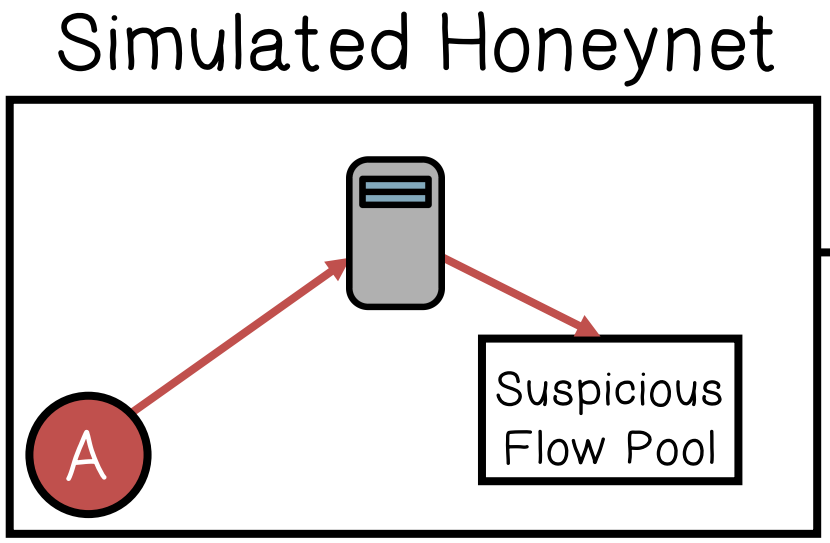




Syntactic Worm Signature Generators



Traffic Based Flow Classifiers

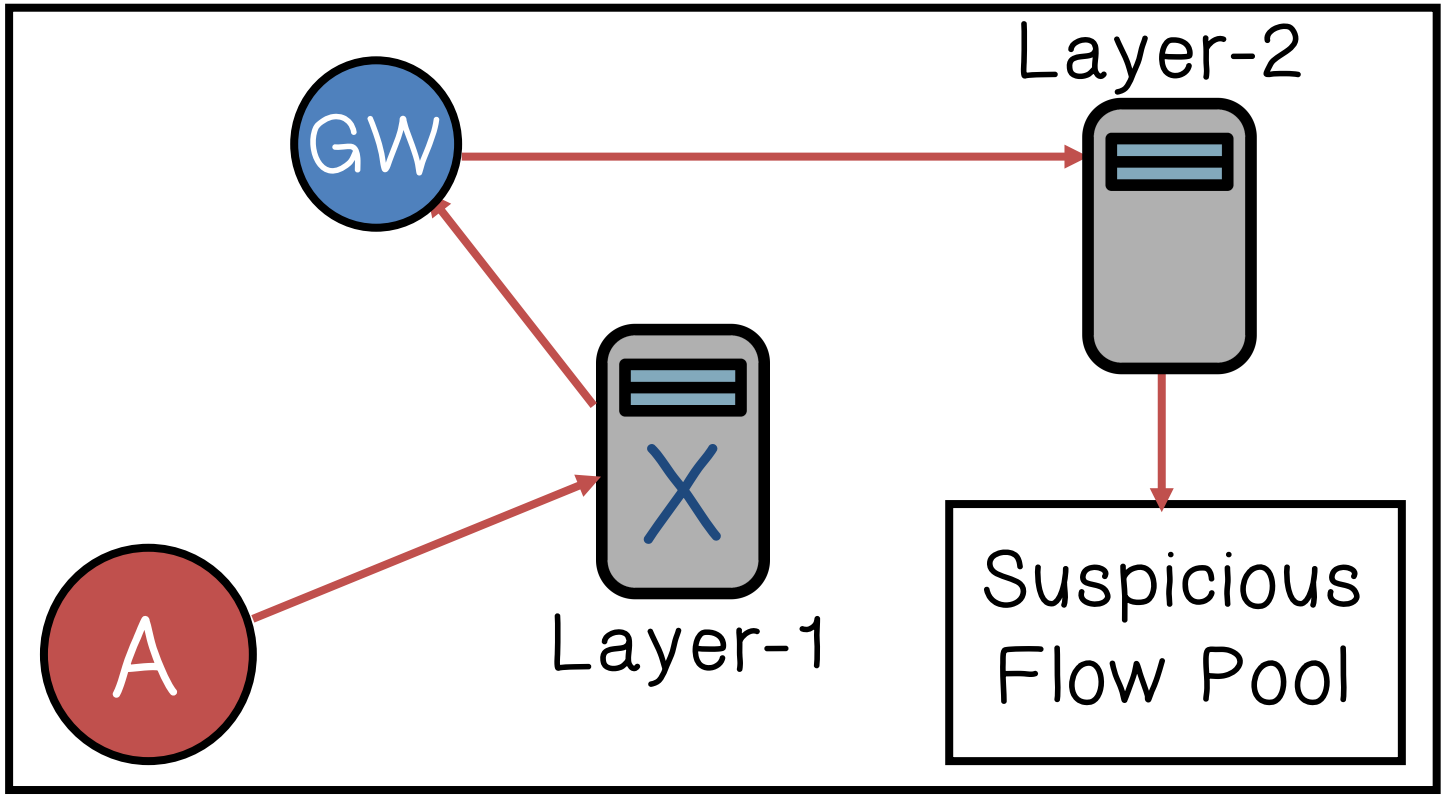


C. Kreibich et al., Honeycomb: creating intrusion detection signatures using honeypots. *Hot Topics in Networks*, 2003.

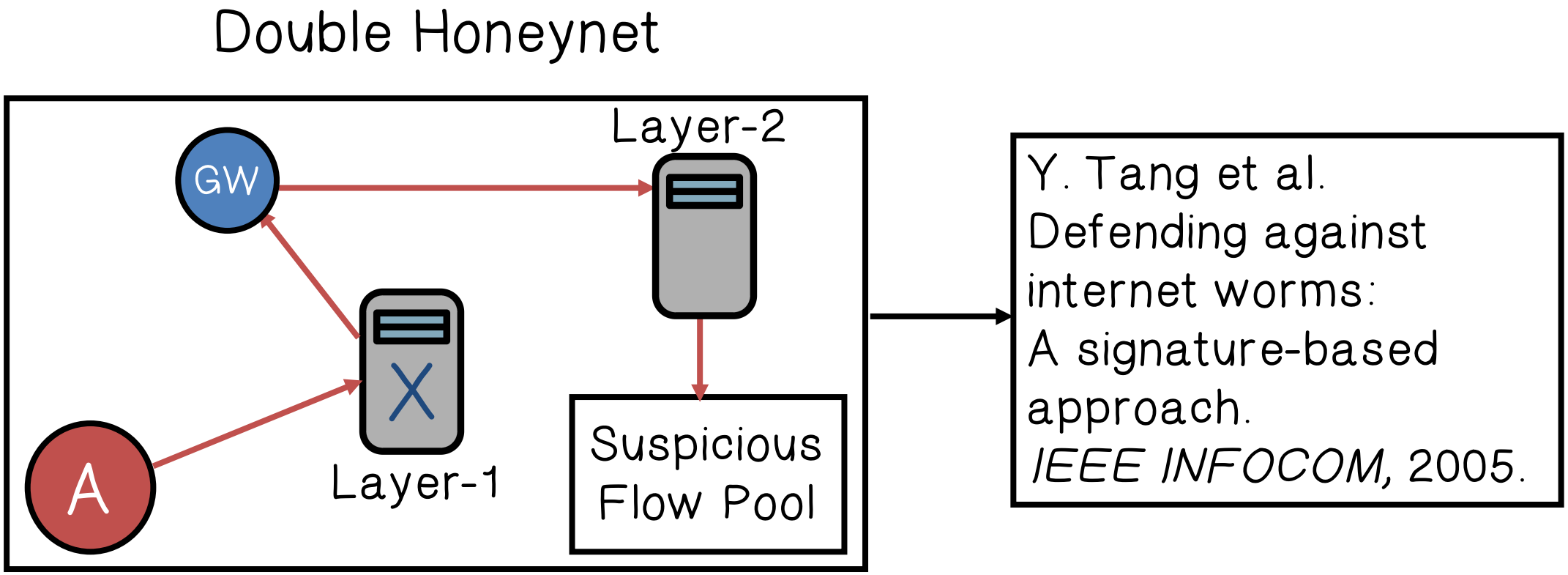
V. Yegneswaran et al., An architecture for generating semantics-aware signatures. *USENIX Security*, 2005.

Traffic Based Flow Classifiers

Double Honeynet

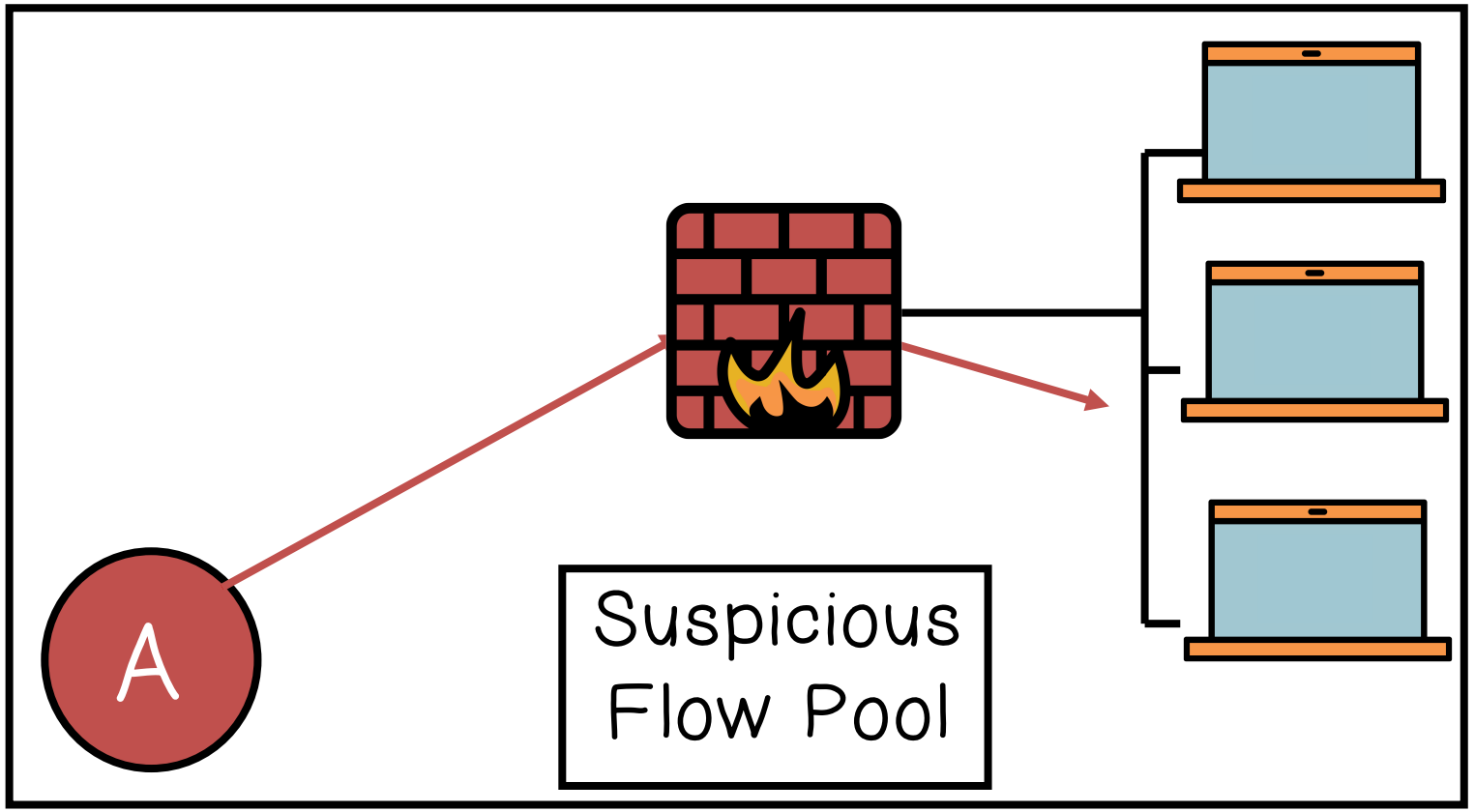


Traffic Based Flow Classifiers



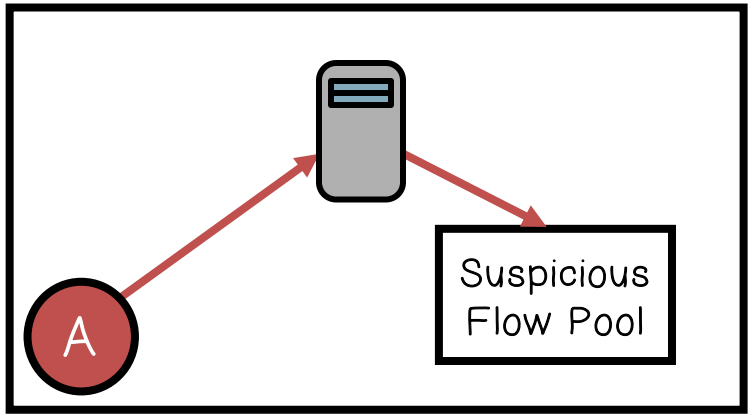
Traffic Based Flow Classifiers

Port-scanning detection

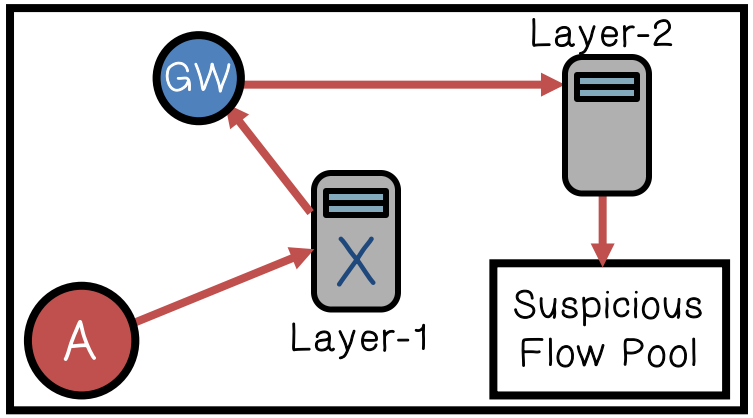


Traffic Based Flow Classifiers

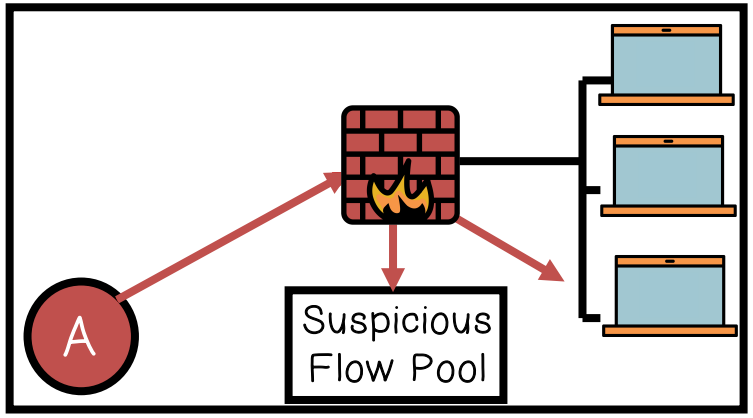
Simulated Honeynet



Double Honeynet



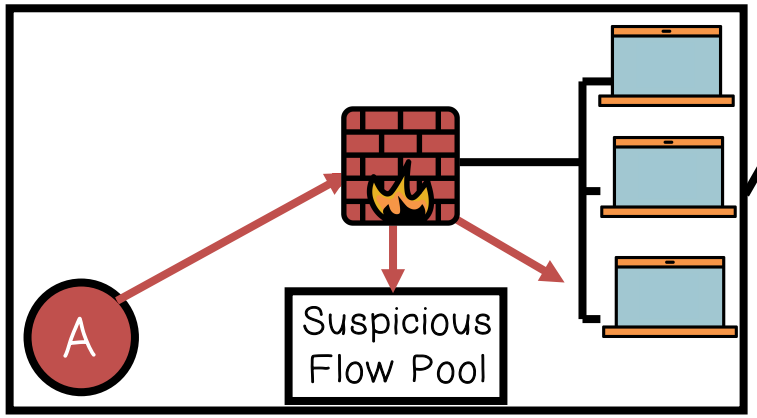
Port-scanning detection



Traffic Based Flow Classifiers

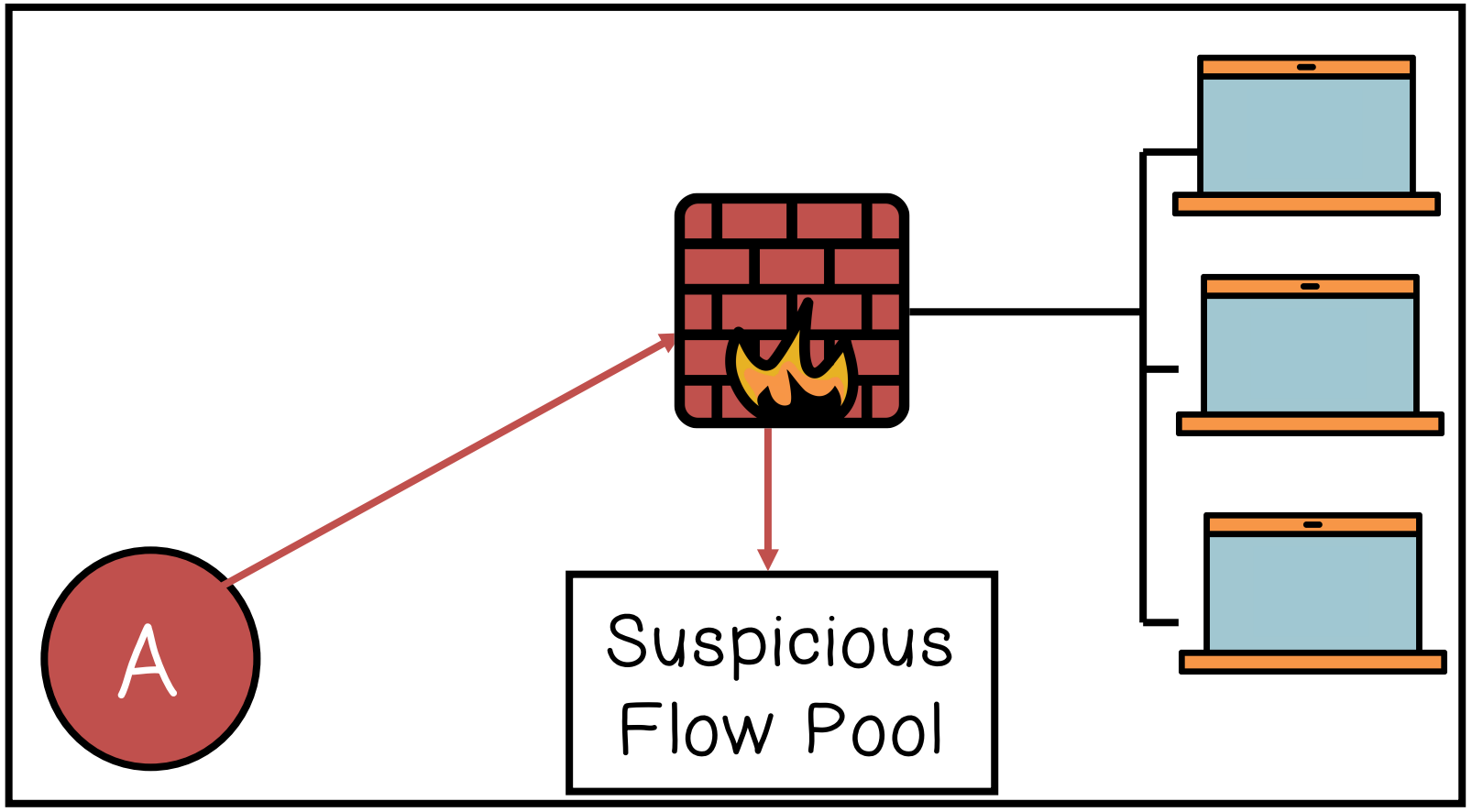
H.A. Kim et al., **Autograph: Toward automated, distributed worm signature detection.** *USENIX Security*, 2004.

Port-scanning detection



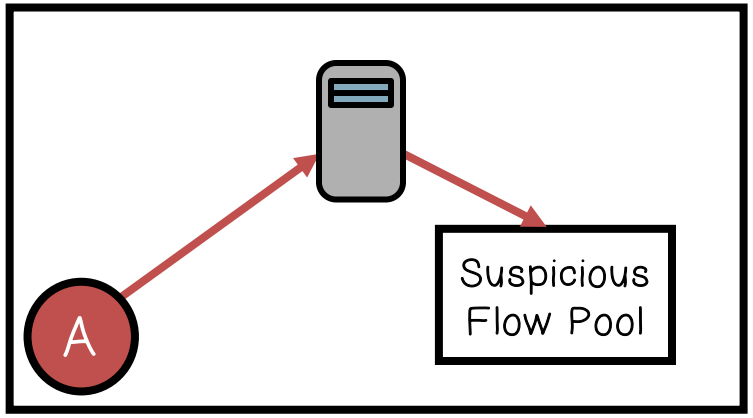
Traffic Based Flow Classifiers

Anomaly IDS

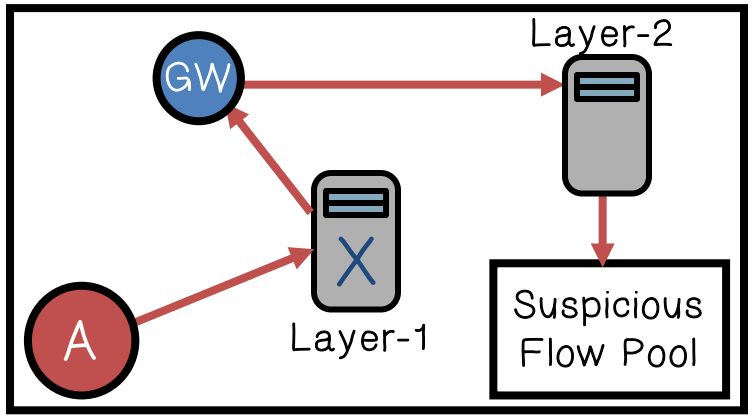


Traffic Based Flow Classifiers

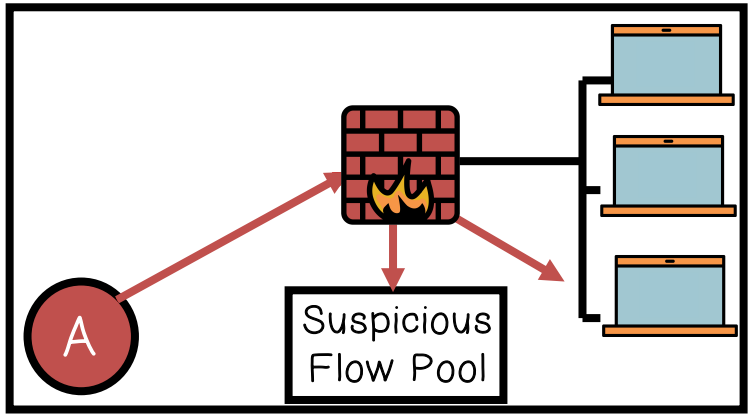
Simulated Honeynet



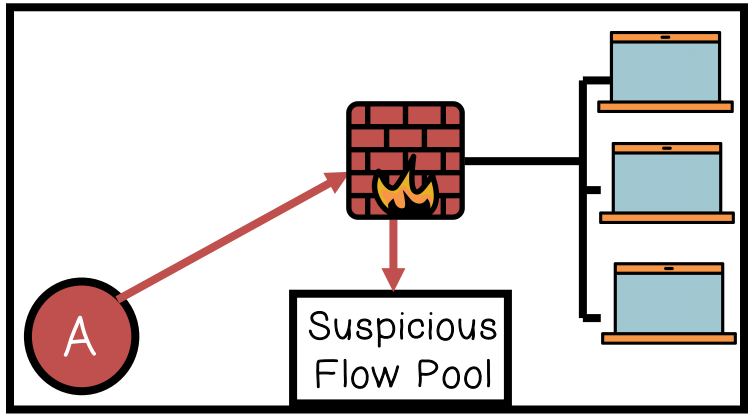
Double Honeynet

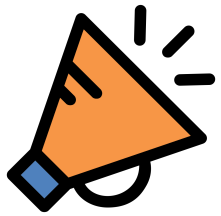


Port-scanning detection

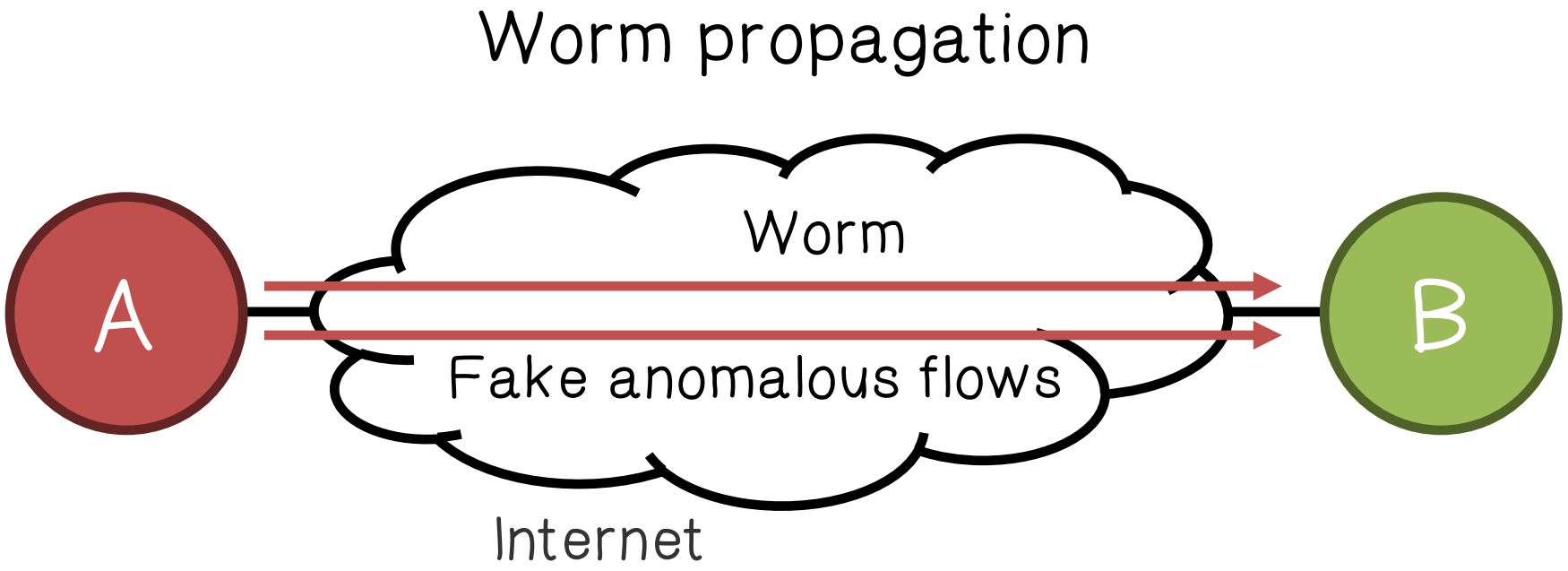


Anomaly IDS

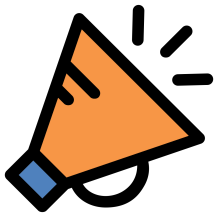




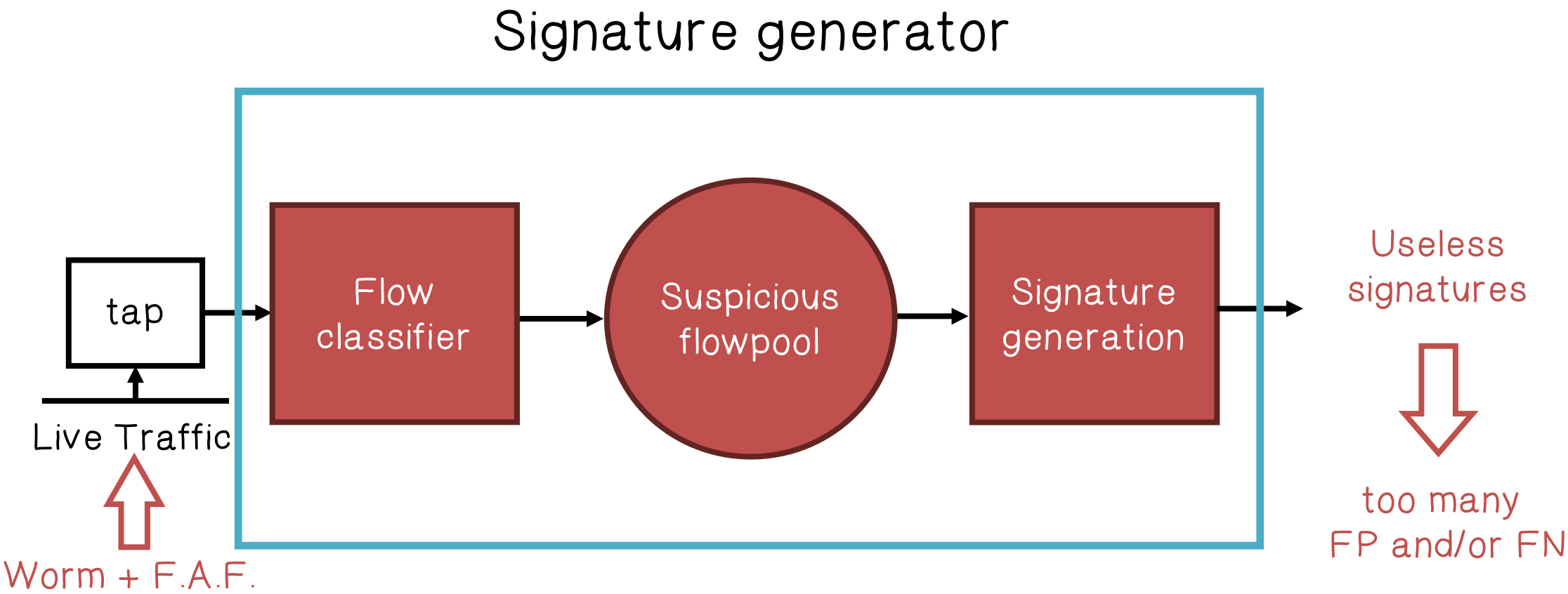
Noise Injection Attack



Fake anomalous flows do not need to exploit the vulnerability

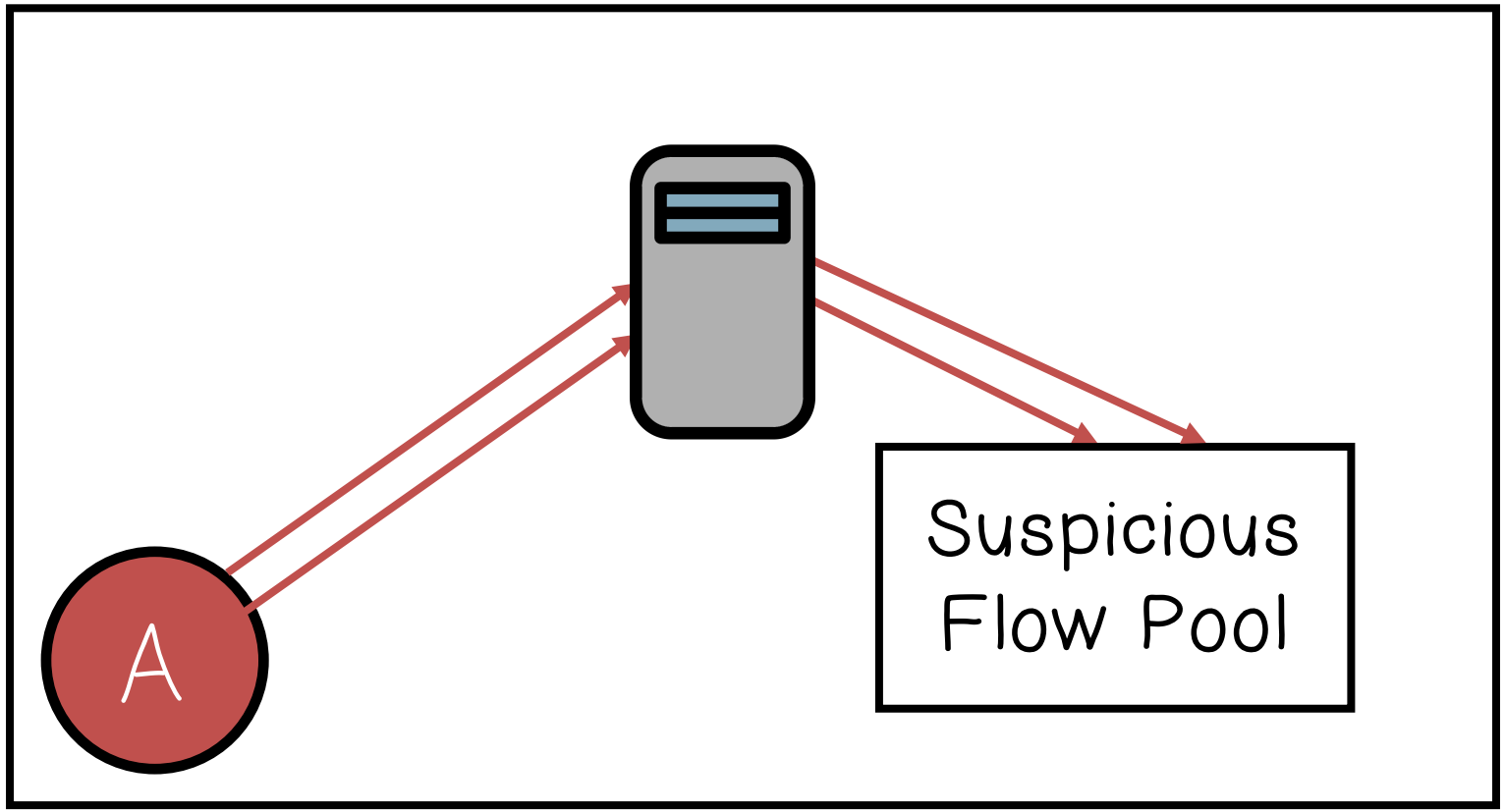


Noise Injection Attack



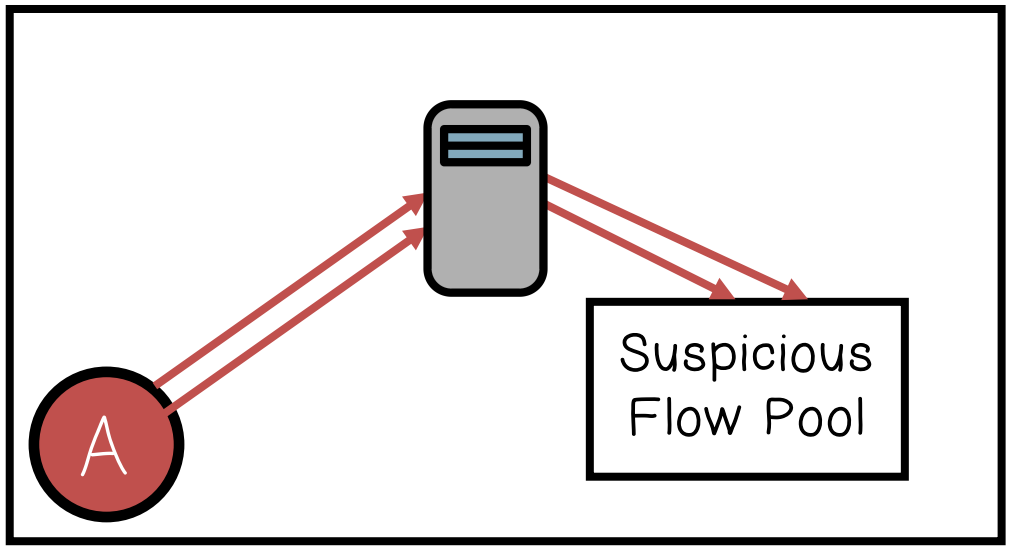
Injecting Fake Anomalous Flows

Simulated Honeynet

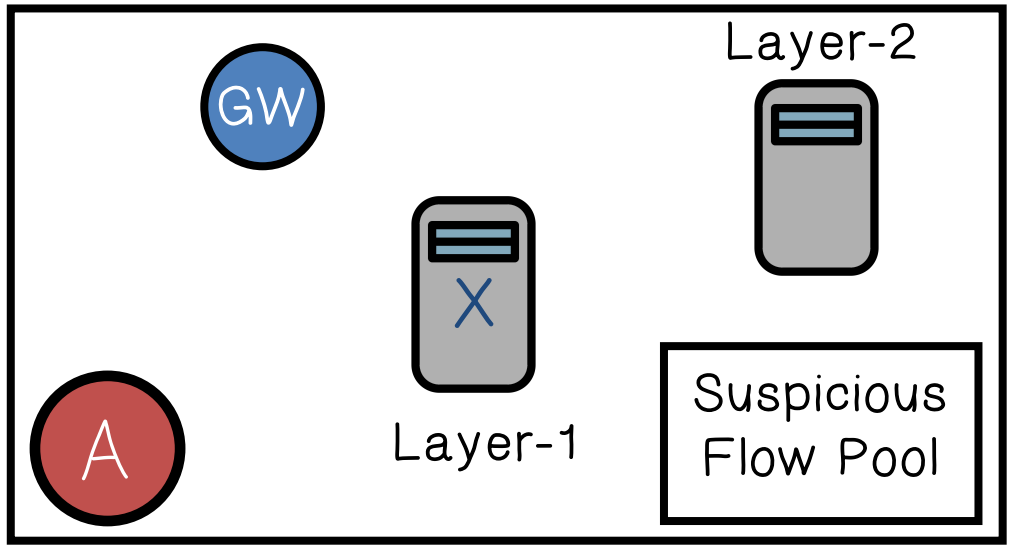


Injecting Fake Anomalous Flows

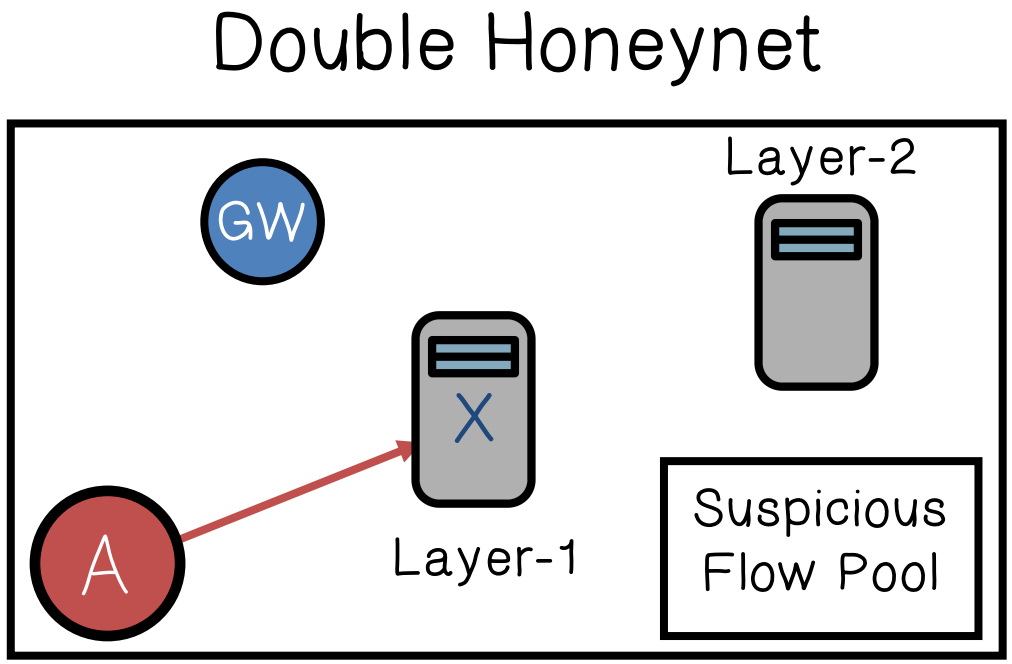
Simulated Honeynet



Double Honeynet

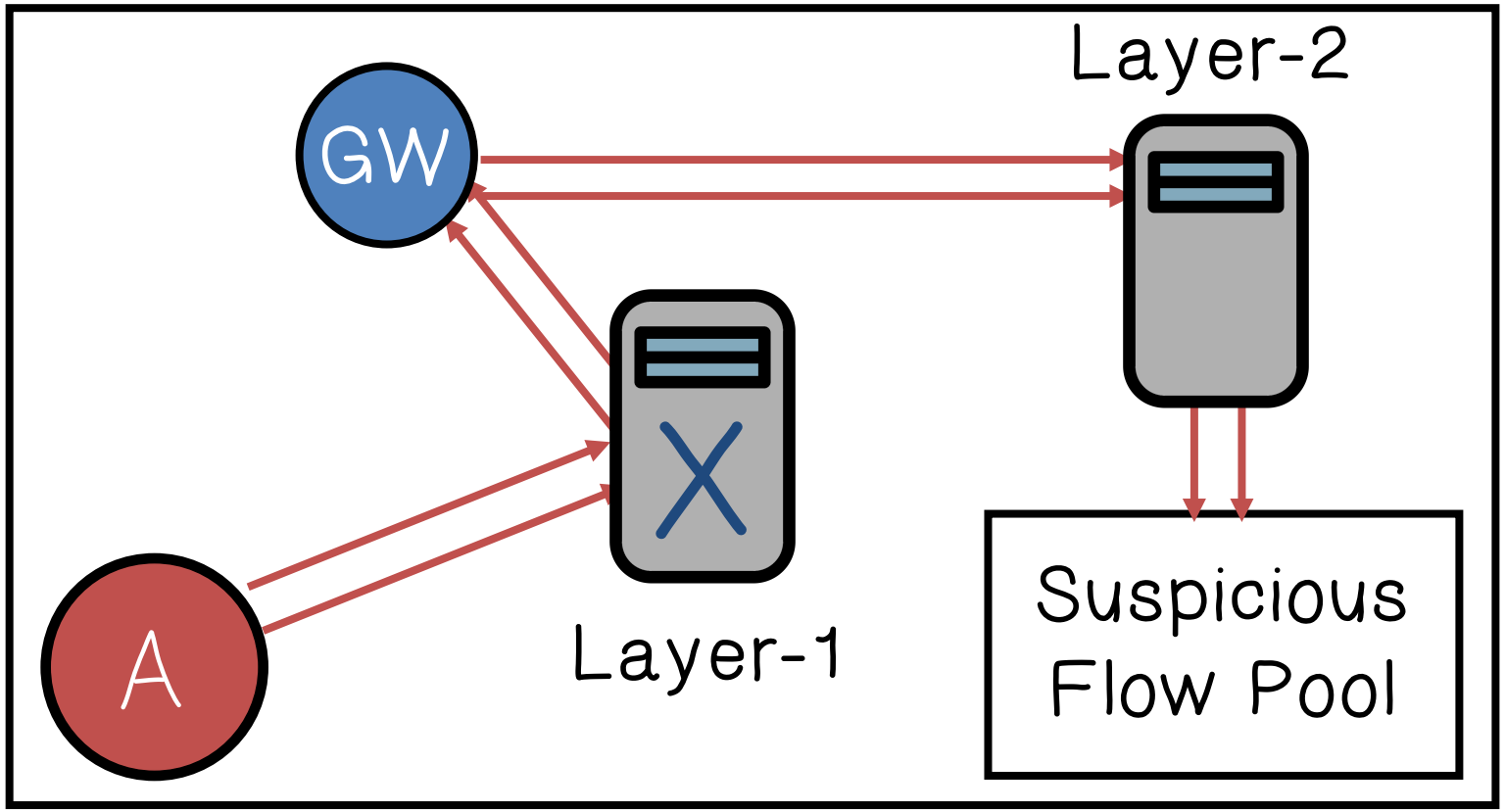


Injecting Fake Anomalous Flows



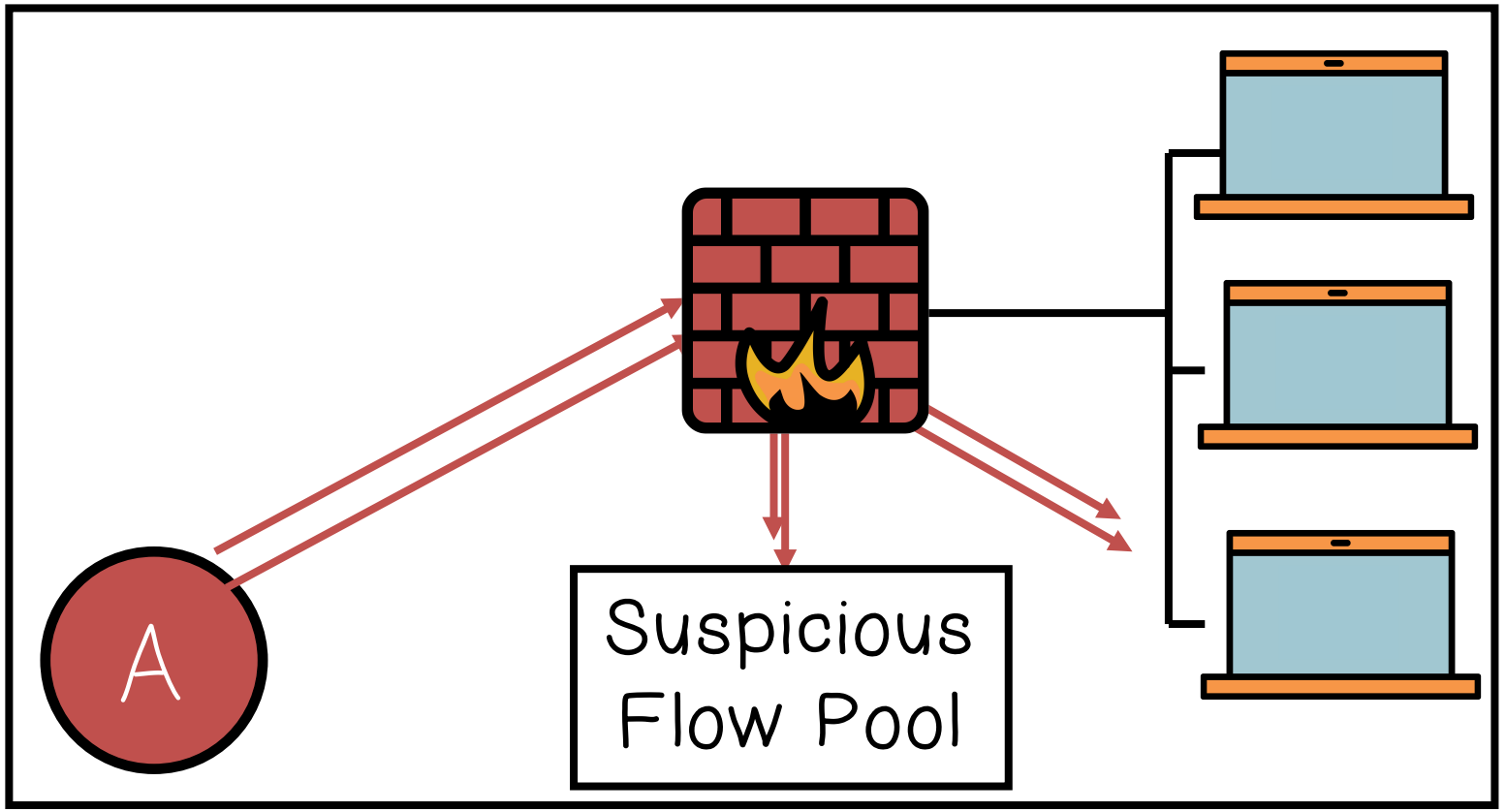
Injecting Fake Anomalous Flows

Double Honeynet



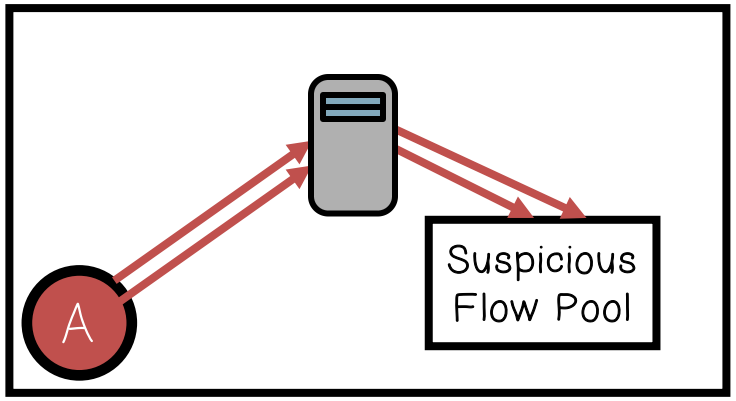
Injecting Fake Anomalous Flows

Port-scanning detection

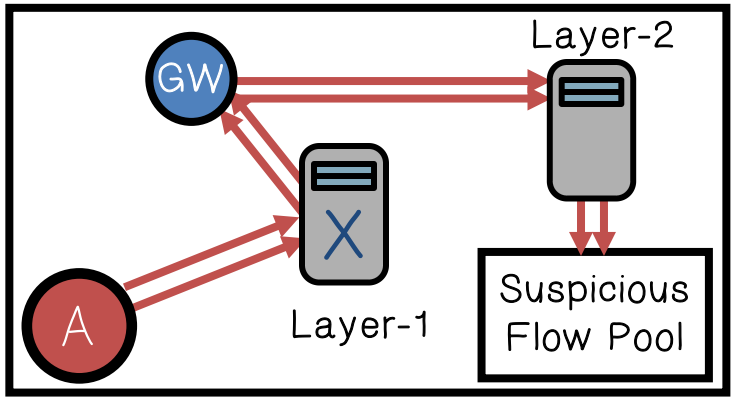


Injecting Fake Anomalous Flows

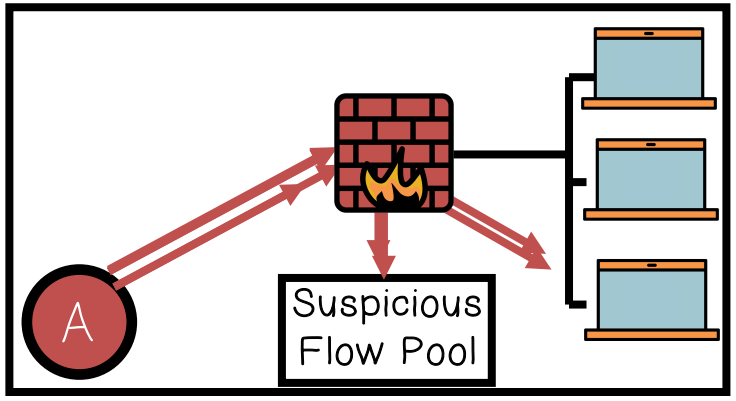
Simulated Honeynet



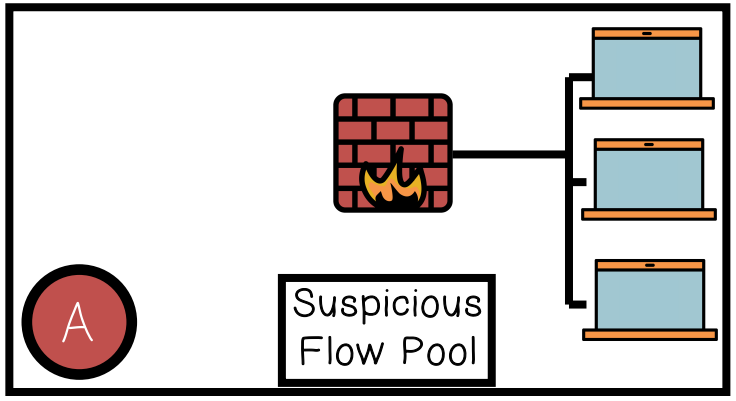
Double Honeynet



Port-scanning detection

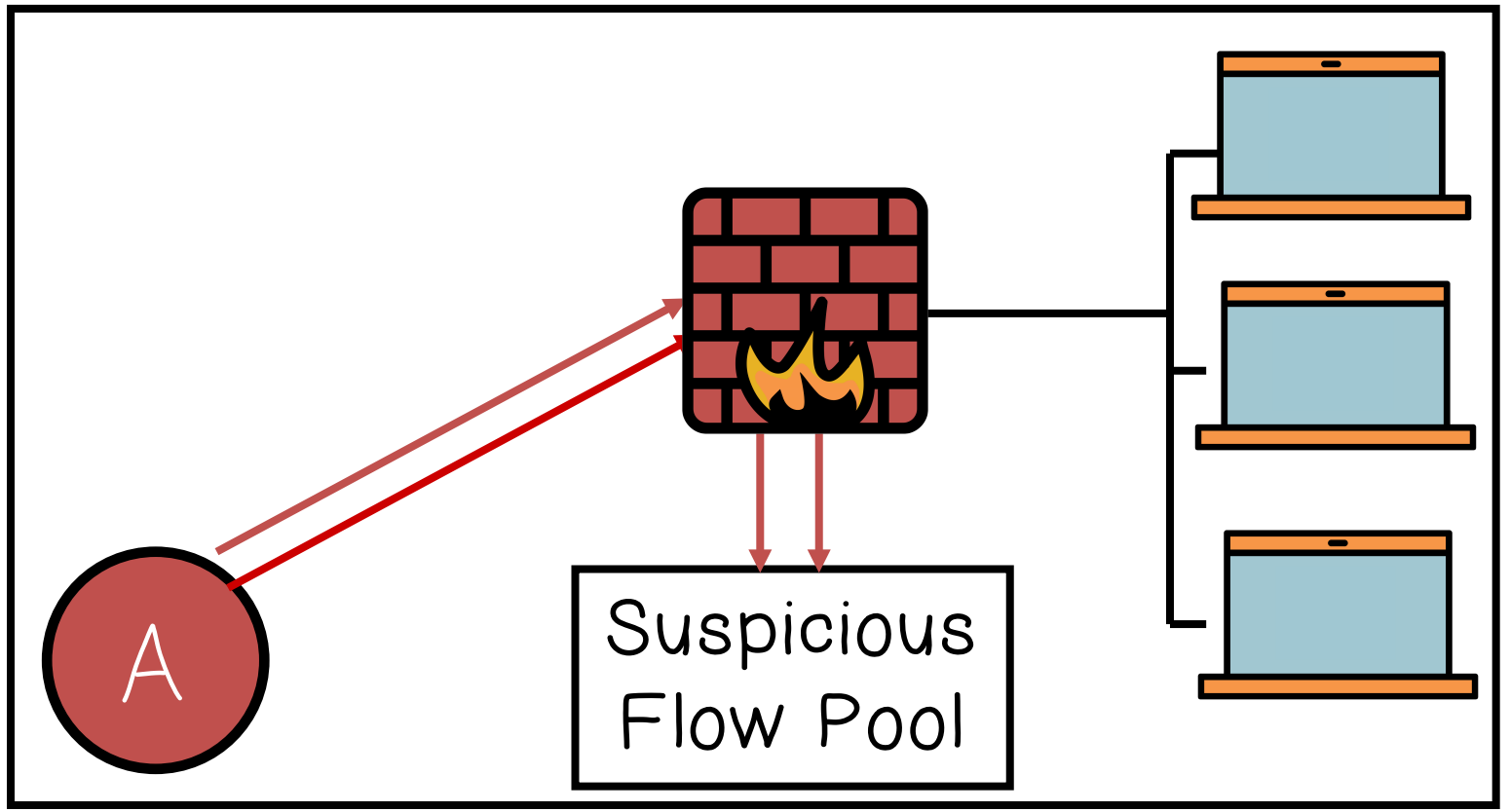


Anomaly IDS



Injecting Fake Anomalous Flows

Anomaly IDS



Case Study: Polygraph

The flow classification technique was not specified

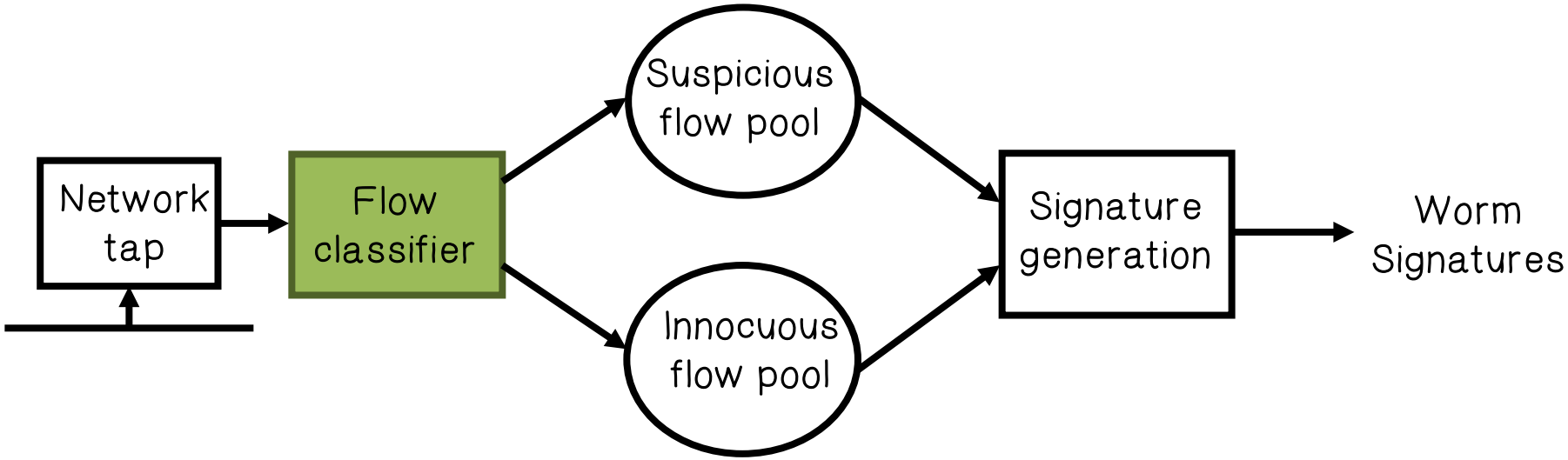
Signature generation for polymorphic worms

The authors assumed that the flow classifier is not perfect

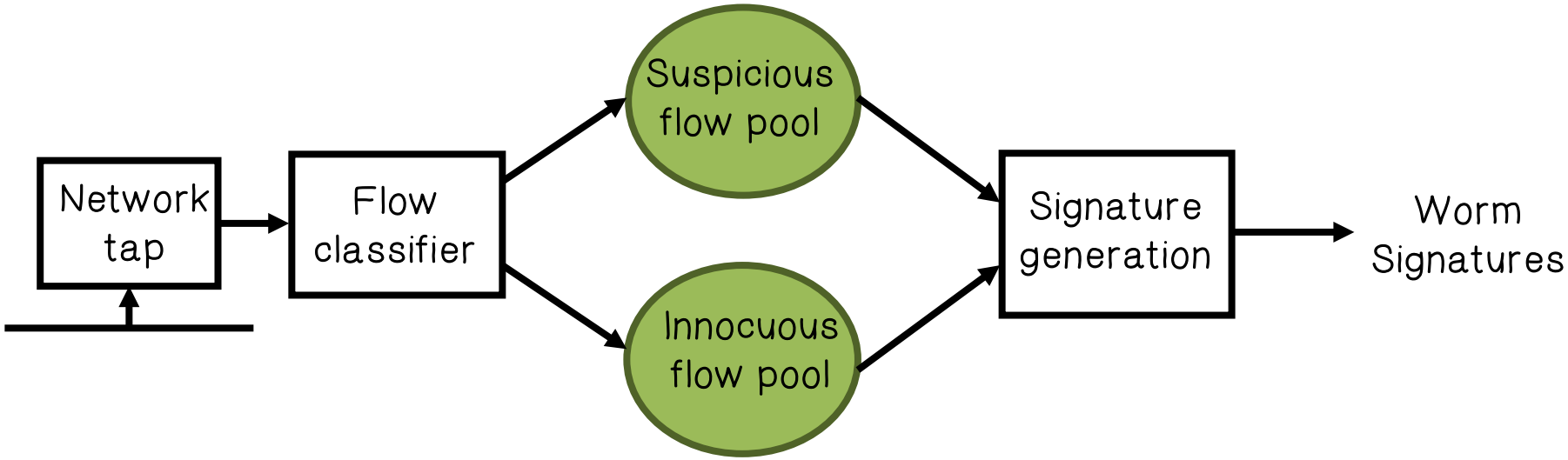
- Noise could be stored into the suspicious pool

J. Newsom et al.: “even in presence of noise Polygraph generates high-quality signatures”

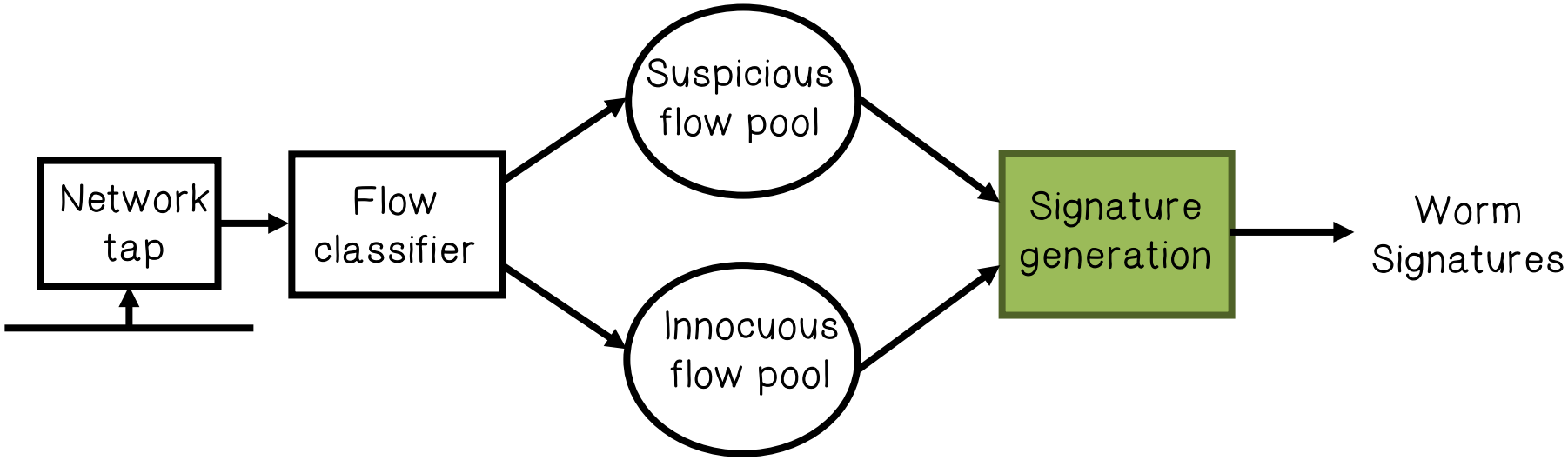
Case Study: Polygraph



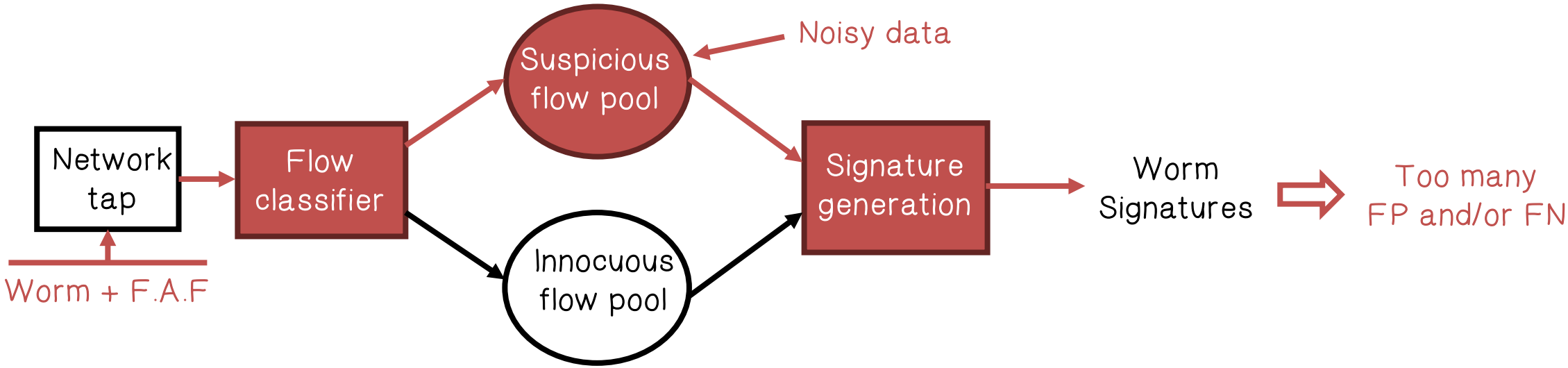
Case Study: Polygraph



Case Study: Polygraph



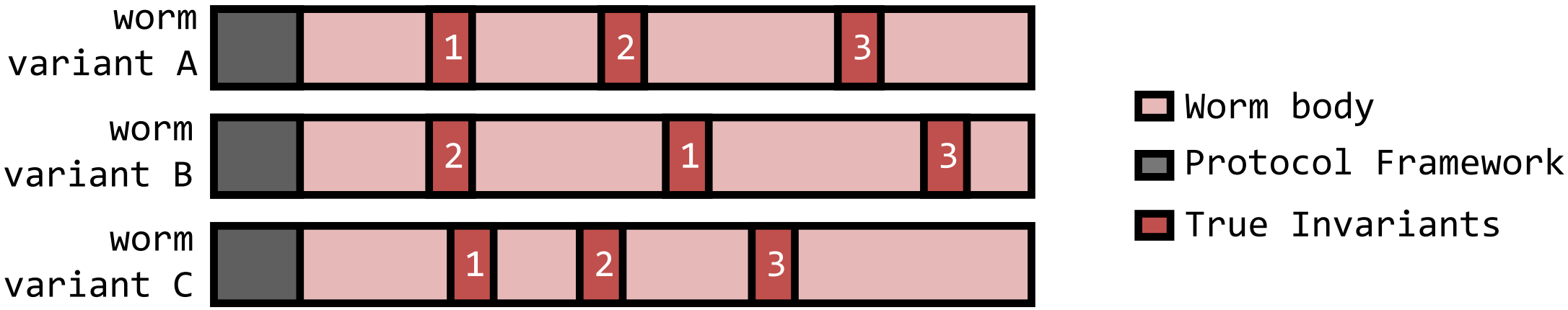
Case Study: Polygraph



This is not true if the noise is deliberately well-crafted and injected by the attacker

Case Study: Polygraph

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

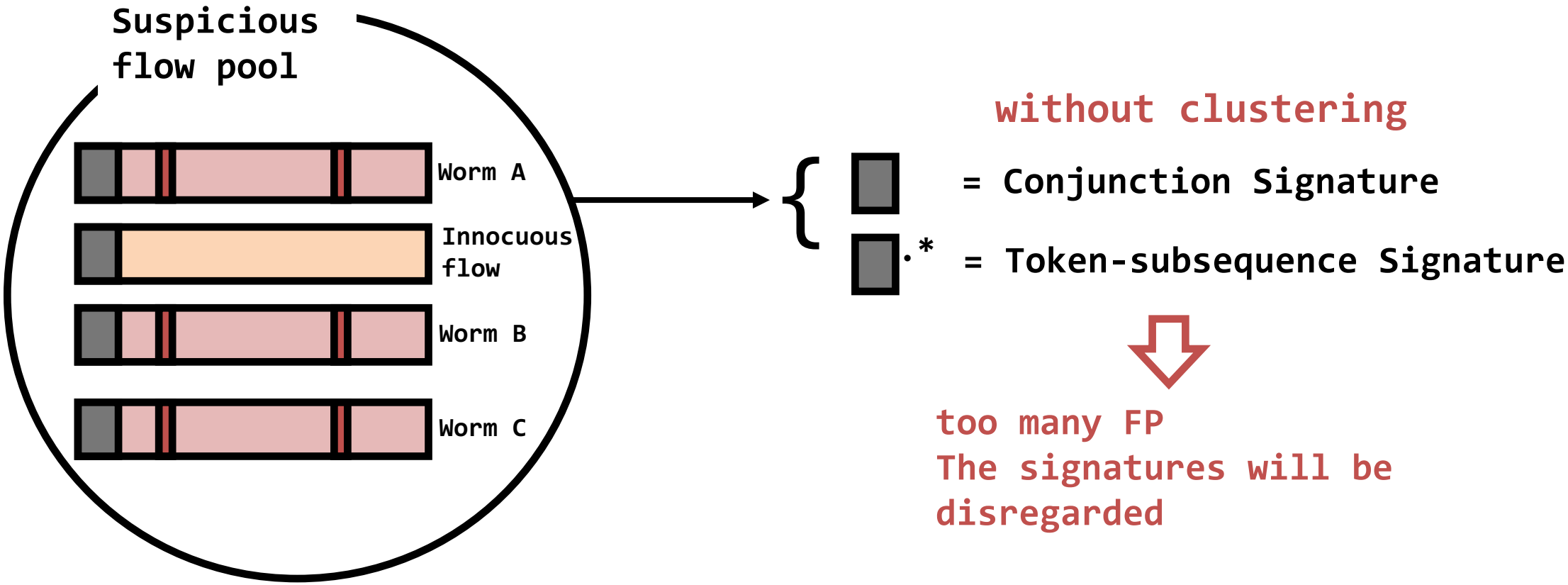


Conjunction Signature
{PF, TI-1, TI-2, TI-3}

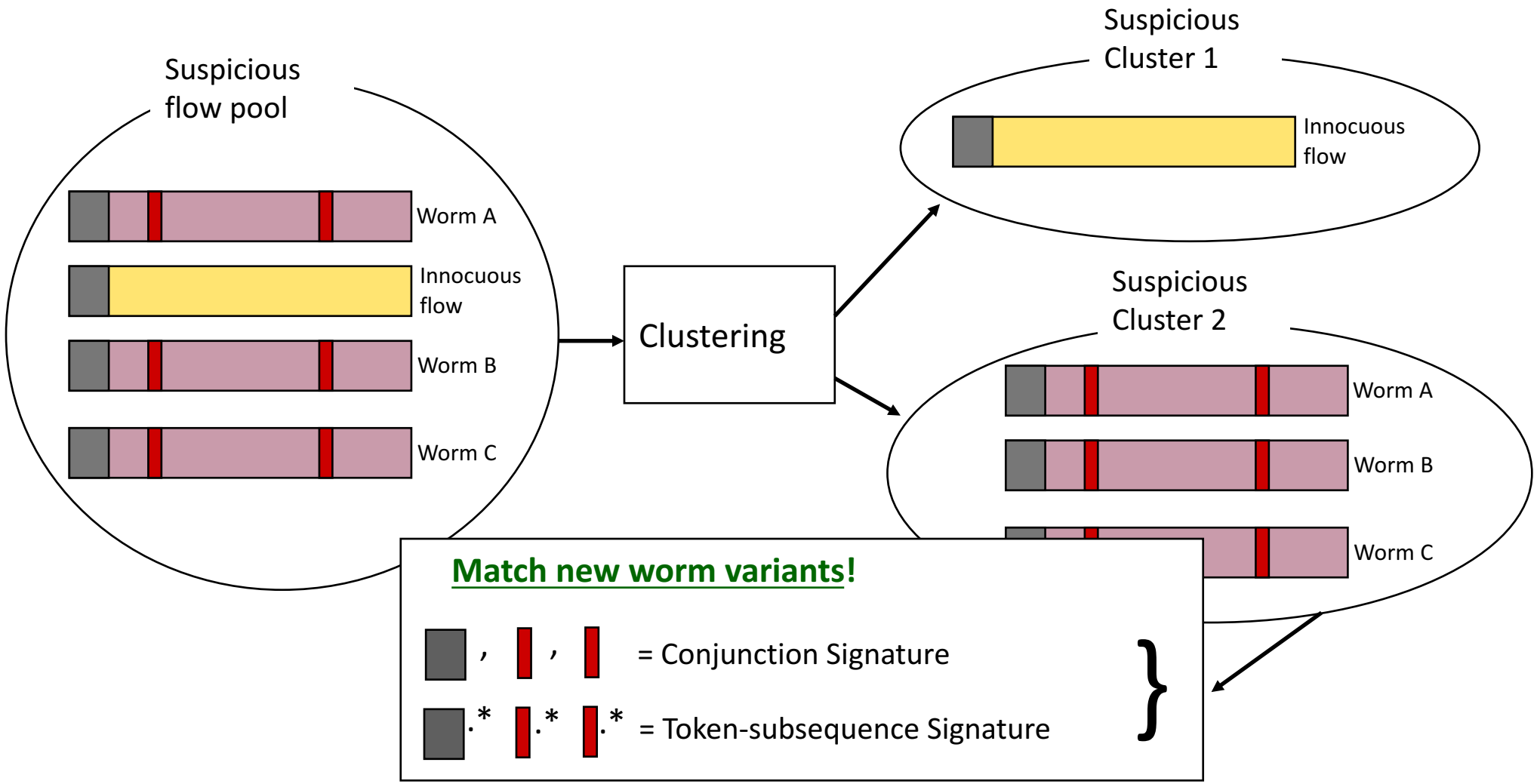
Token-subsequence Signature
PF .* TI-1 .* TI-3 .*

Case Study: Polygraph

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



Hierarchical Clustering

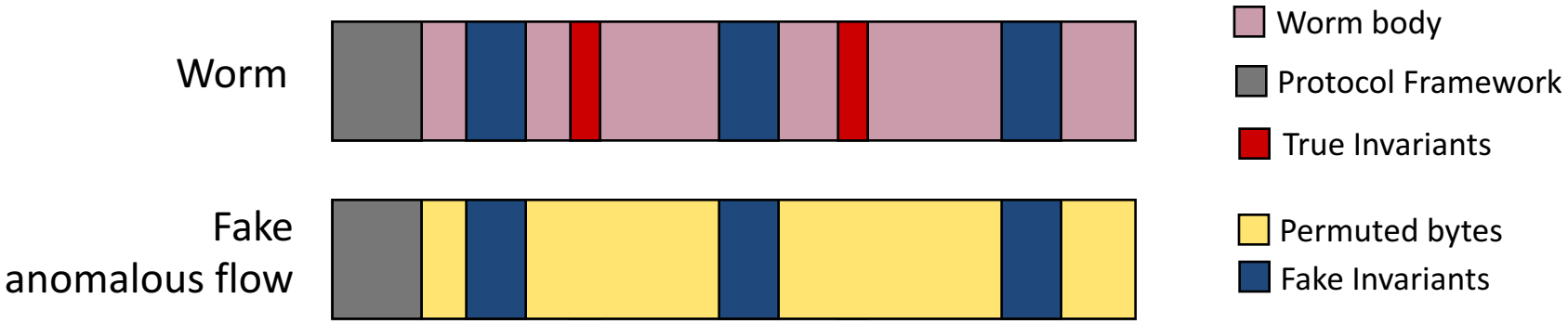


Misleading Conjunction and Token-Subsequence Signatures

Objective: Caft the fake anomalous flows so that the Hierarchical Clustering cannot filter the noise

- the extracted signatures will produce False Negatives

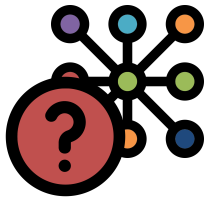
Misleading Conjunction and Token-Subsequence Signatures



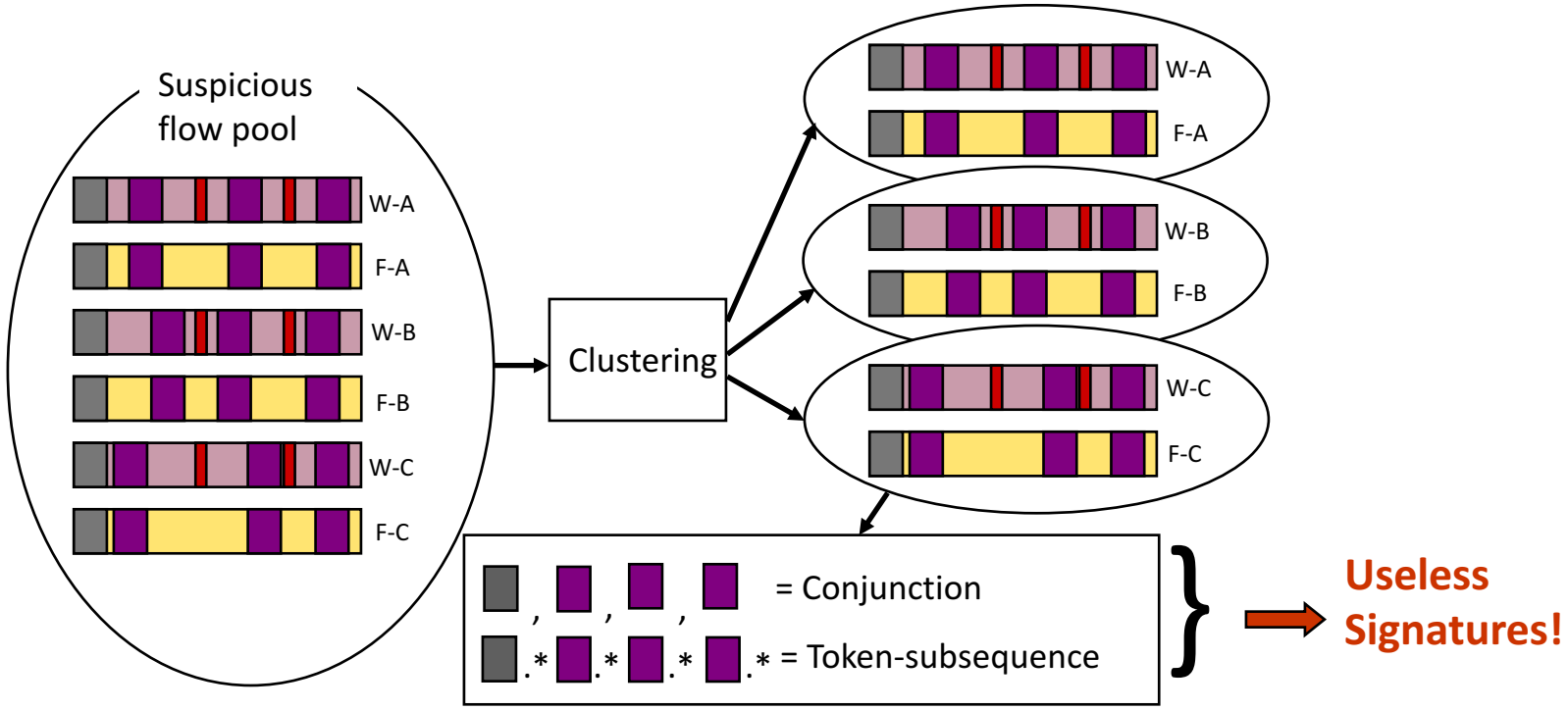
P(FI | innocuous flow) < P(TI | innocuous flow)

=

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



Misleading Hierarchical Clustering

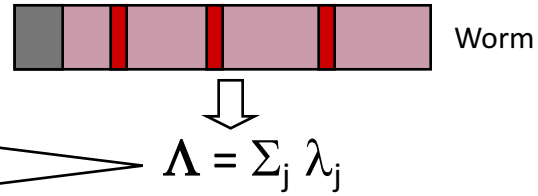


The signatures **do not contain True Invariants**

too many False Negatives!

Polygraph

- 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, λ_{PF}>, <TI-1, λ_{TI-1}>, <TI-2, λ_{TI-2}>< TI-3, λ_{TI-3}>}

If $\Lambda = \Sigma_j \lambda_j < \theta$

The flow is Innocuous!

- θ is computed during training to obtain high DR and low FP

Misleading Bayes Signatures

- Consider a "normal" HTTP string ν = "Pragma: no-cache"
  - Suppose P(ν | Innocuous Flow) = 0.10
- Suppose the worm injects substrings of ν into all the fake anomalous flows

Bayes signature

<GET,λ_1>; <HTTP/1.1,λ_2>; … ; <\xFF\xBF, λ_{TI}>;
<Pragma: no, $\lambda_{\nu 1}$>; <ragma: no-,$\lambda_{\nu 2}$>; … <: no-cache,$\lambda_{\nu 7}$>;

  - $\lambda_{\nu i} \cong \lambda_\nu = \log(0.5/0.1)$
- **Score multiplier effect** for innocuous flows which contain ν

Innocuous flow

- GET .* HTTP/1.1 .* Pragma: no-cache .*
- $\Lambda = \Sigma_k \lambda_k + \Sigma_i \lambda_{\nu i}$ → $\Lambda >> \Sigma_k \lambda_k$

Misleading Bayes Signatures

- Normal HTTP string ν = "Pragma: no-cache"
- The worm inject substrings of ν into all the F.A.F

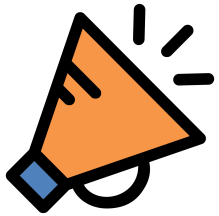
Bayes signature

<GET,λ_1>; <HTTP/1.1,λ_2>; … ; <\xFF\xBF, λ_{TI}>;
<Pragma: no, λ_{v1}>; <ragma: no-,λ_{v2}>; … <: no-cache,λ_{v7}>;

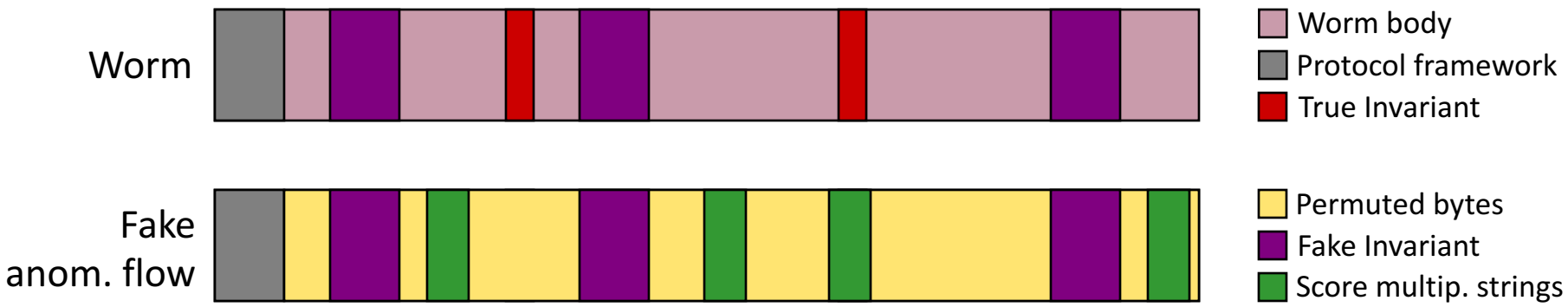
- **Score multiplier effect**
  - GET .* HTTP/1.1 .* Pragma: no-cache .* — Innocuous flow
  - $\Lambda = \Sigma_k \lambda_k + \Sigma_i \lambda_{vi}$ ➡ $\Lambda >> \Sigma_k \lambda_k$

Objective of the attack:
Inject "normal" substrings into the fake anomalous flows
so that POLYGRAPH cannot find a "good" threshold θ

⇨ **too many FALSE POSITIVES or FALSE NEGATIVES!**



Crafting the Noise



- 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" HTTP 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 HTTP requests (0.007% FP)

] J. Newsome, B. Karp, and D. Song.
Polygraph: Automatically generating signatures for polymorphic worms.
In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2005.

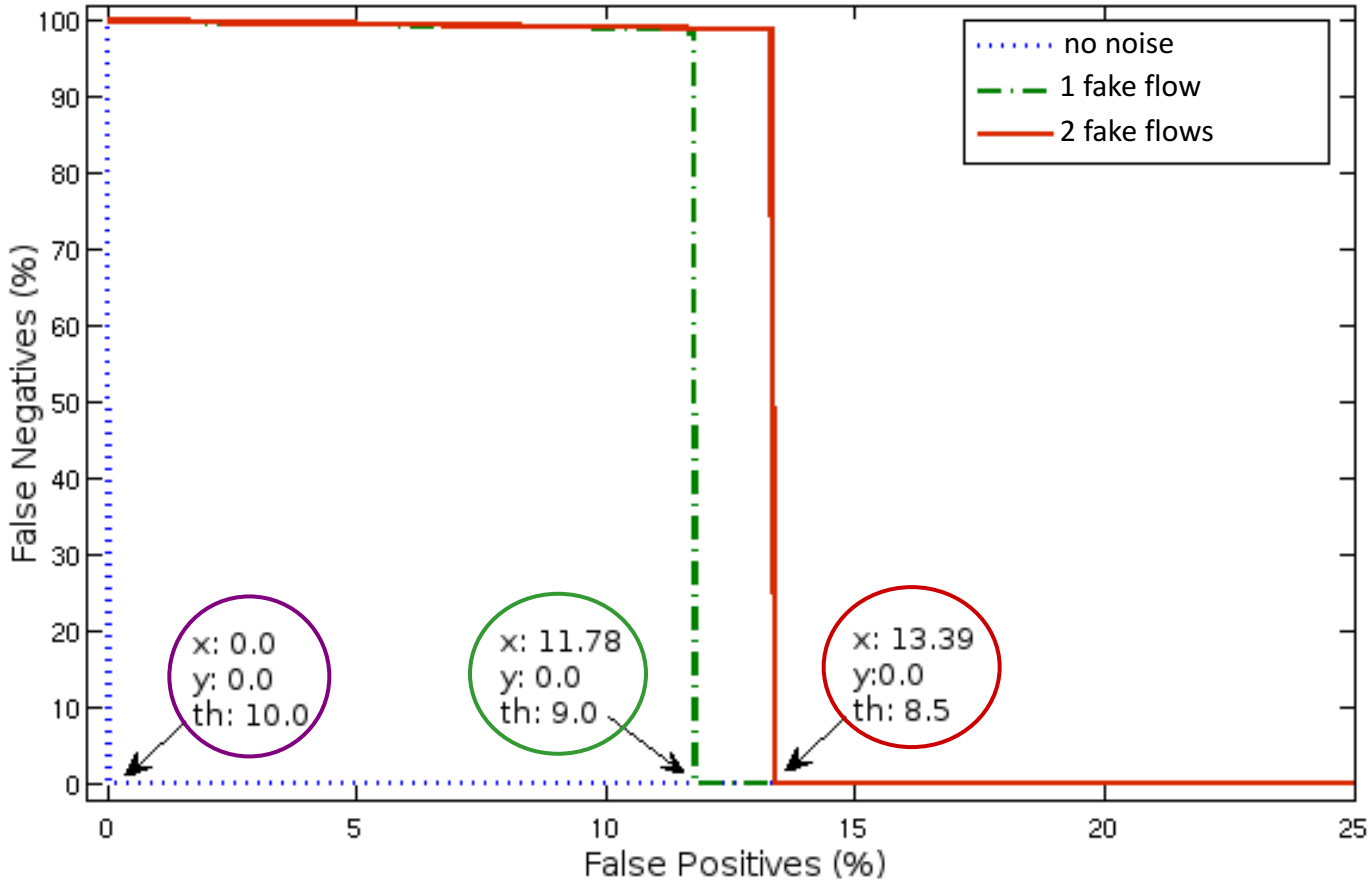
Experimental Results

- Test dataset
  - "Suspicious" Test flow pool = 100 worm variants
  - "Normal" Test flow pool = 217,164 HTTP requests (0.0% FP)
- Attacker's dataset
  - 300 Candidate *Score Multiplier Strings* extracted from 5,000 flows

] J. Newsome, B. Karp, and D. Song.
Polygraph: Automatically generating signatures for polymorphic worms.
In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2005.

Results with Bayes Signatures

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



Under attack it is **impossible to find a threshold**

Results with all 3 Signature Types

- 20 rounds - The attack is considered successful if
  - Conjunction and Token-subsequence produce 100% False Negatives
  - Bayes produces more than 1% of False Positives

	1 F.A.F./worm	2 F.A.F./worm
Conjunction	65%	95%
Token-subsequence	40%	90%
Bayes	90%	100%
All the 3 signatures	20%	85%

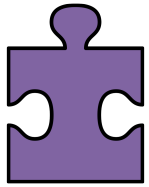
Conclusion

Noise Injection Attack has a high chance to mislead syntactic worm signature generators

- Forces extraction of useless signatures!

Need a precise flow classifier that can effectively filter the noise

Open problem... (to be solved!)



Misleading Worm Signature Quiz

Which of the following statements are true?

- [x] If we can completely control the process of generating or collecting the training data and ascertain the authenticity and integrity of the dataset, we don't have to worry about data poisoning attacks

- [x] If the training data is obtained in an open environment, e.g., the Web, there is always the potential of poisoning attacks (i.e., such attacks can't be eliminated)