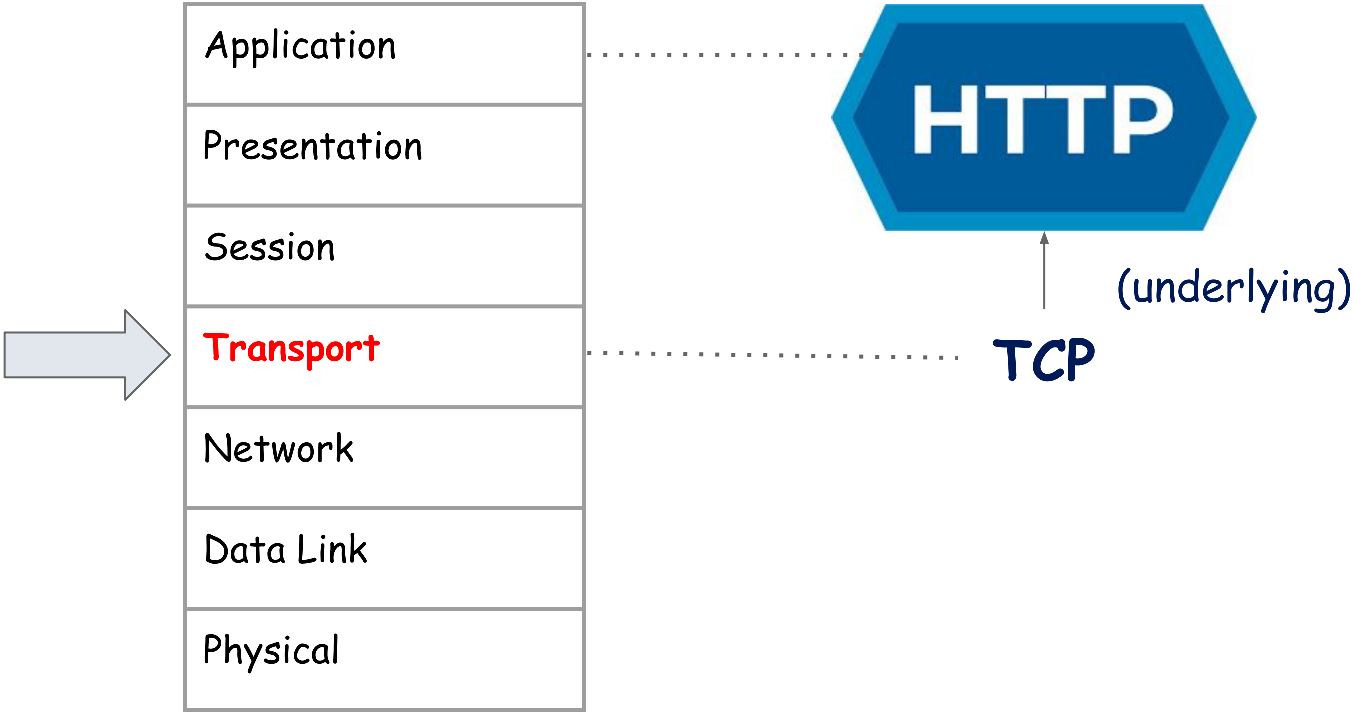
Athena: Analyzing and Quantifying Side Channels of Transport Layer Protocols

Feiyang Yu[1], Quan Zhou[2], Syed Rafiul Hussain[2], Danfeng Zhang[1]

[1]Duke University, [2]Penn State University

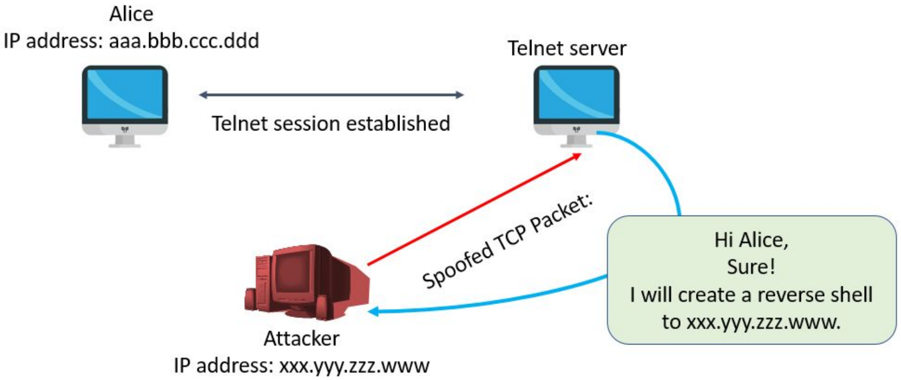
Aug 15, 2024

Transport Layers



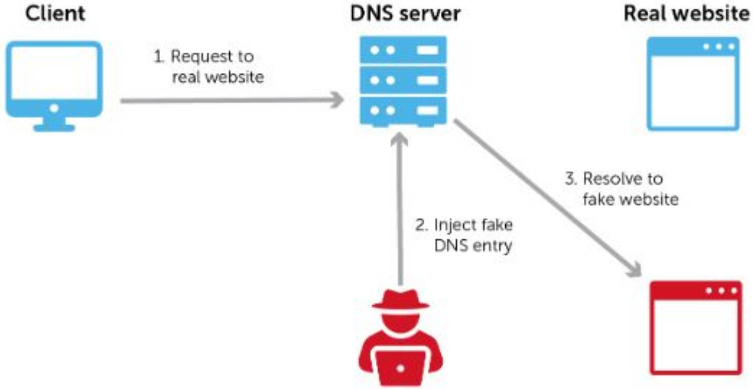
Transport Layer Attacks

TCP hijacking



Connection take-over (requires seq number)

DNS poisoning

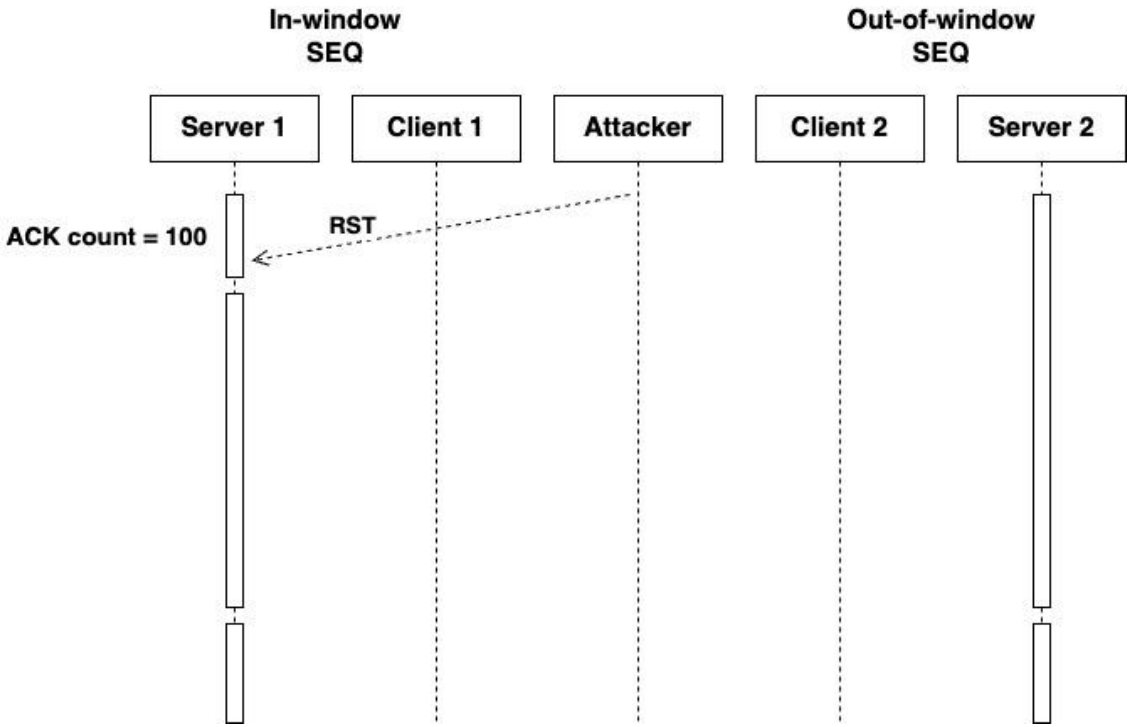


Access manipulation (requires port number)

An illustrative TCP SEQ Inference Attack

A TCP side-channel attack [Cao 2016]:

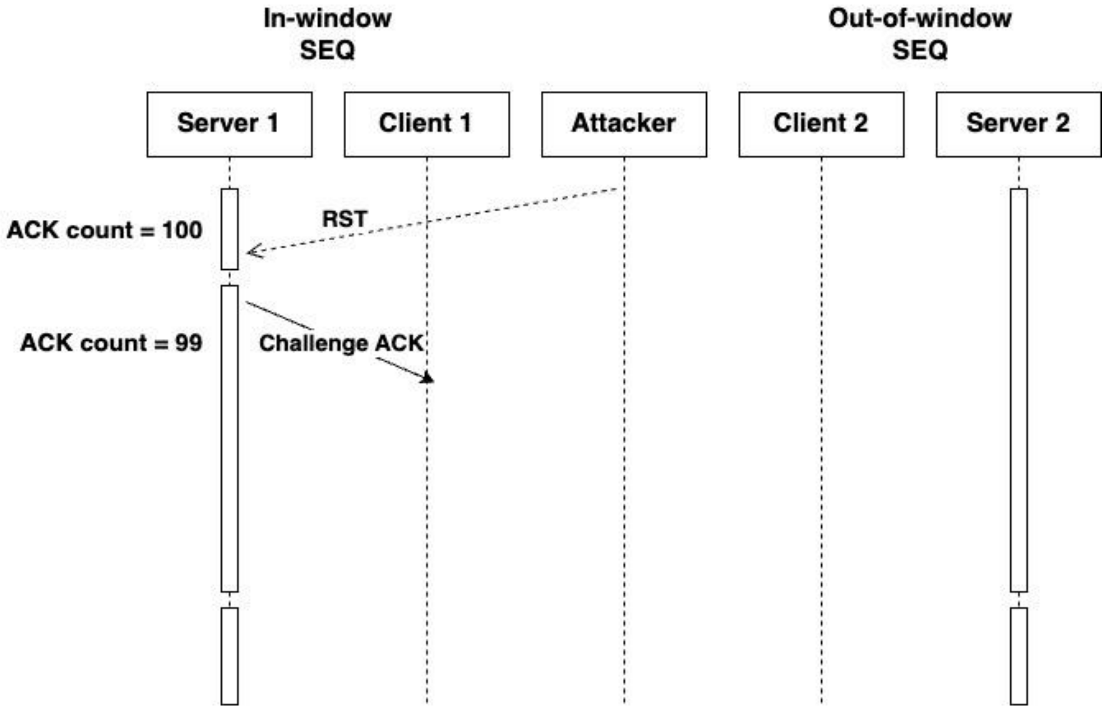
With successful inference, attackers can **hijack the session**.



An illustrative TCP SEQ Inference Attack

A TCP side-channel attack [Cao 2016]:

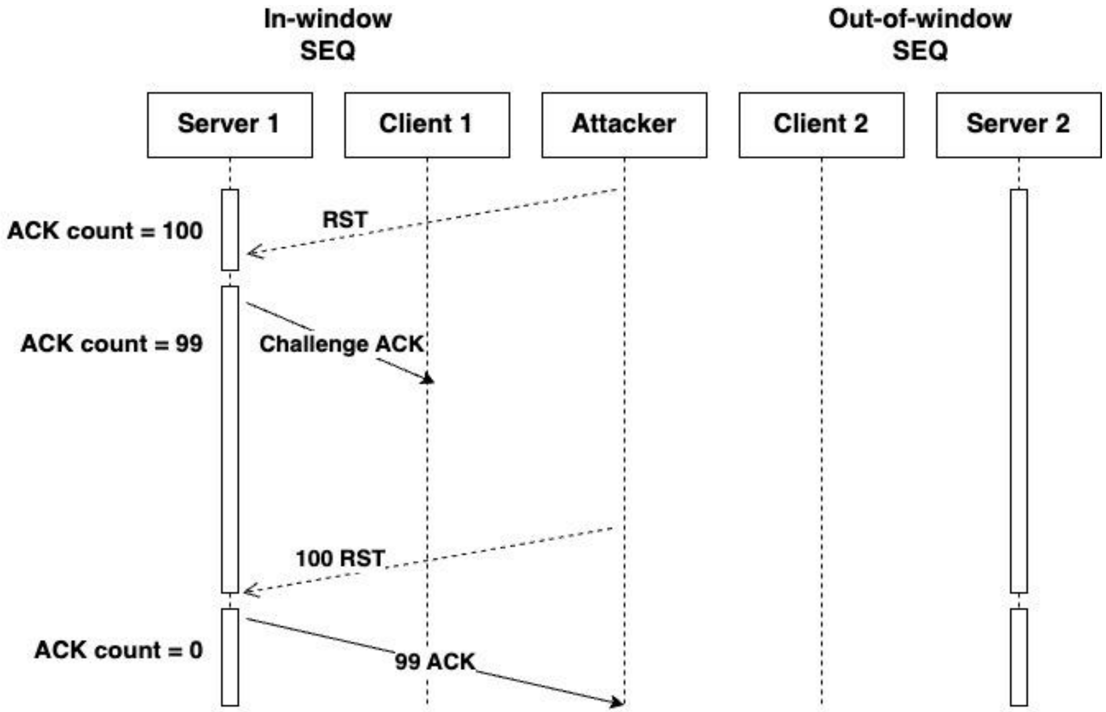
With successful inference, attackers can **hijack the session**.



An illustrative TCP SEQ Inference Attack

A TCP side-channel attack [Cao 2016]:

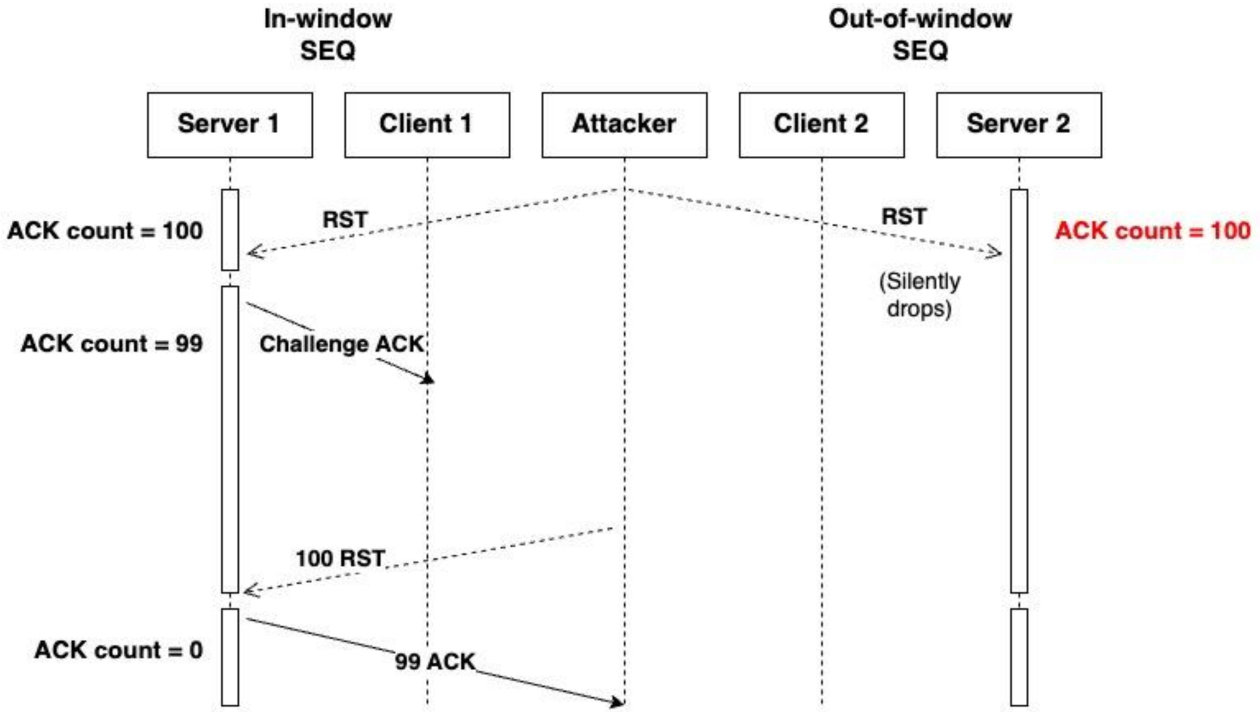
With successful inference, attackers can **hijack the session**.



An illustrative TCP SEQ Inference Attack

A TCP side-channel attack [Cao 2016]:

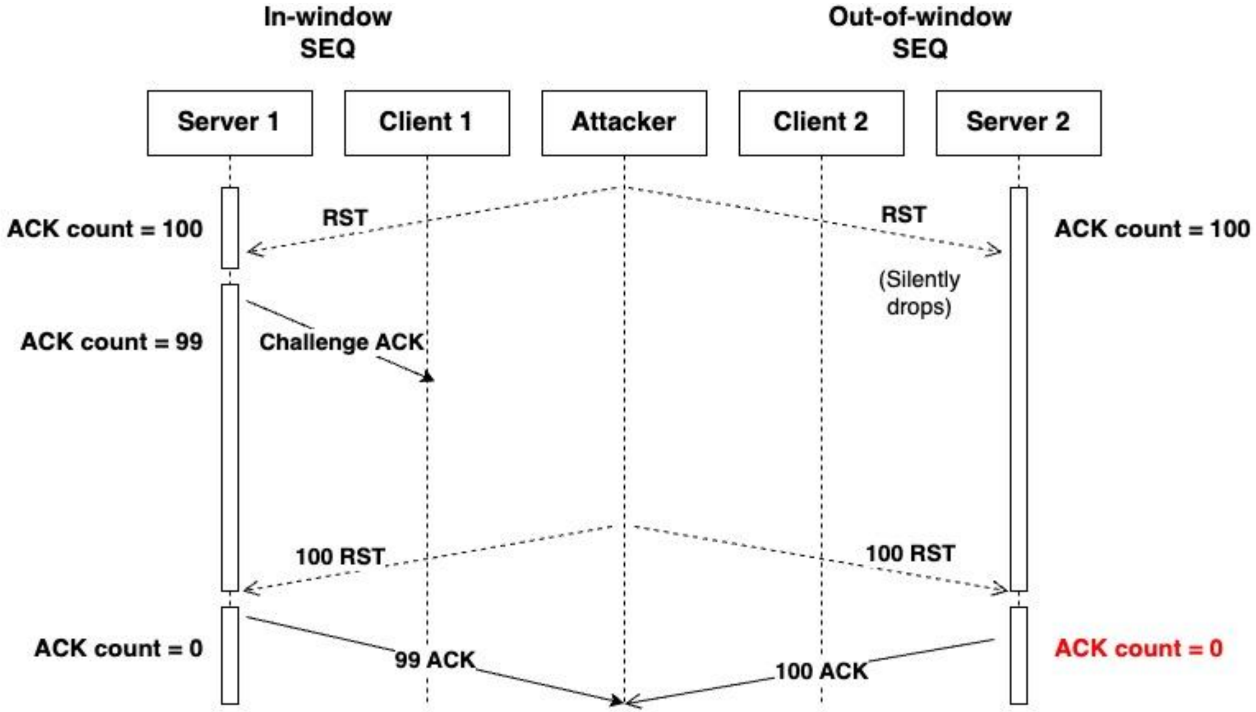
With successful inference, attackers can **hijack the session**.



An illustrative TCP SEQ Inference Attack

A TCP side-channel attack [Cao 2016]:

With successful inference, attackers can **hijack the session**.

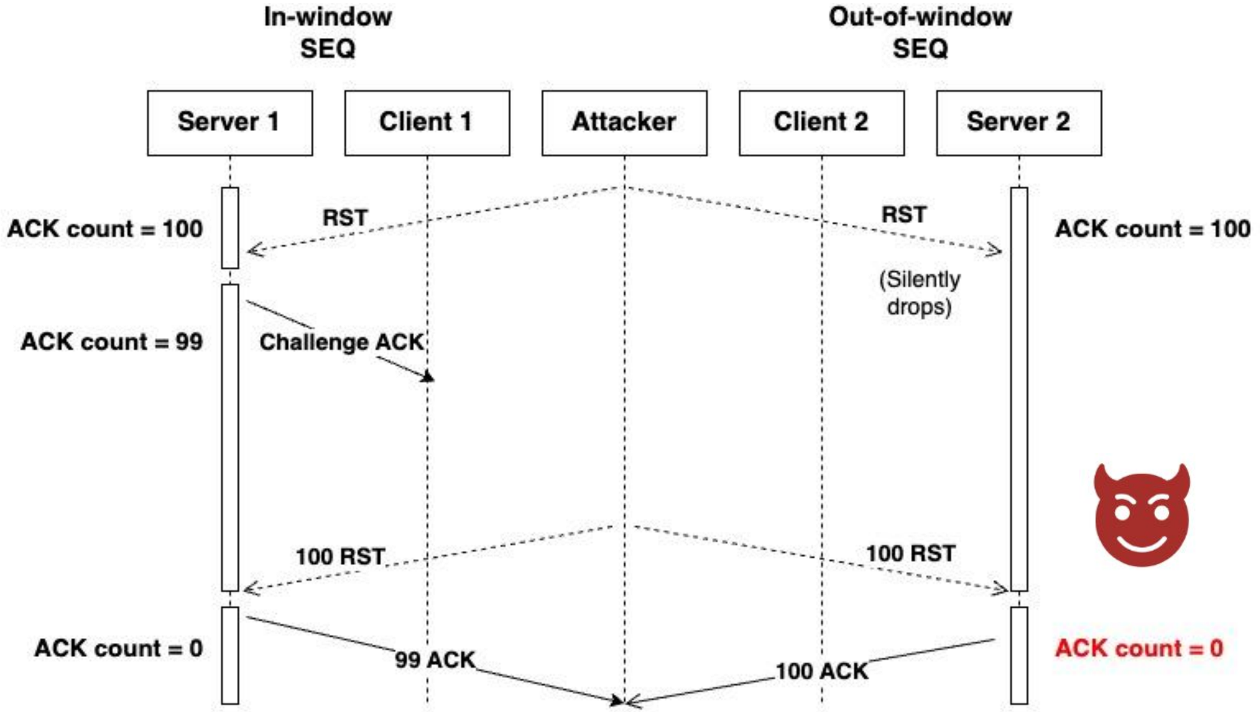


An illustrative TCP SEQ Inference Attack

The global counter is also stored as a file (procfs).

An unprivileged process can access it even more easily…

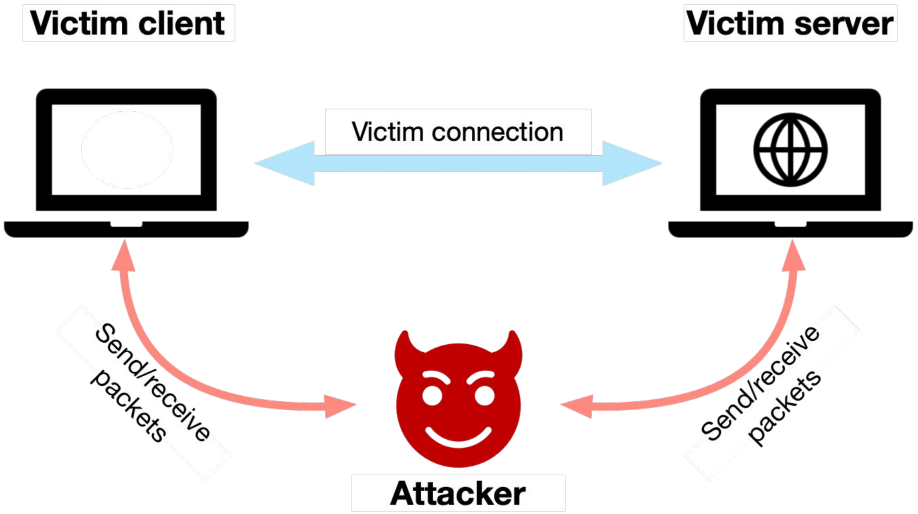
[Qian 2012]



Threat Models

Prior works consider two threat models:

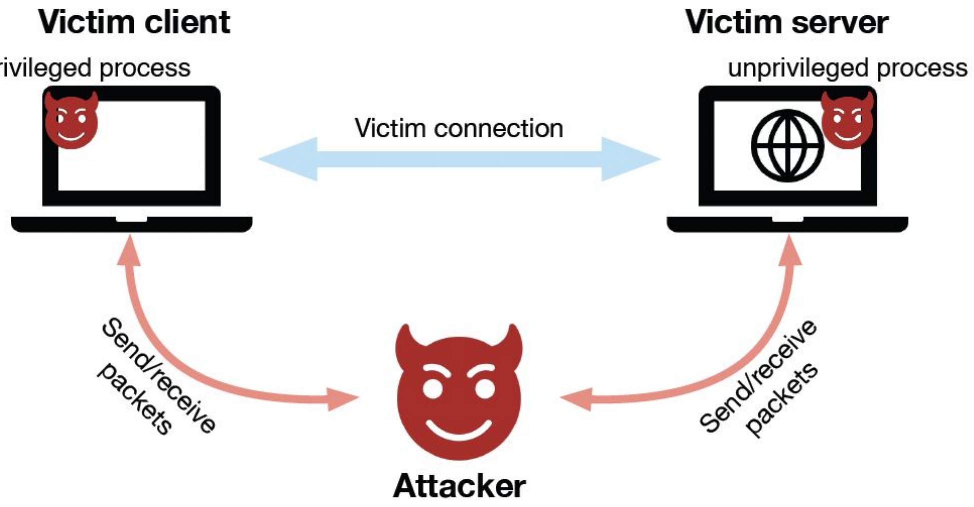
- Off-path attackers (cannot modify/eavesdrop victim connections)
- Aided off-path attackers (w/ control of an ***unprivileged*** process)



Threat Models

Prior works consider two threat models:

- Off-path attackers (cannot modify/eavesdrop victim connections)
- Aided off-path attackers (w/ control of an ***unprivileged*** process)



Root Cause

```
static void tcp_send_challenge_ack(struct sock *sk)
{
    static unsigned int ACK_COUNT;
    strict tcp_sock *tp = tcp_sk(sk);

    if (ACK_COUNT > 0) {
        NET_INC_STATS(sock_net(sk), LINUX_MIB_TCPCHALLENGEACK);
        tcp_send_ack(sk);
    }
}
```

Root cause of the side channel is the **secret-dependent branch**.

Limitation #1: Automation and Scalability

Most side channels were manually investigated:

- TCP [Qian 2012, Cao 2016, Feng 2020, Feng 2022] ...
- UDP [Alharbi 2019, Man 2020, Man 2021] …

Limitation #1: Automation and Scalability

Most side channels were manually investigated:

- TCP [Qian 2012, Cao 2016, Feng 2020, Feng 2022] ...
- UDP [Alharbi 2019, Man 2020, Man 2021] …

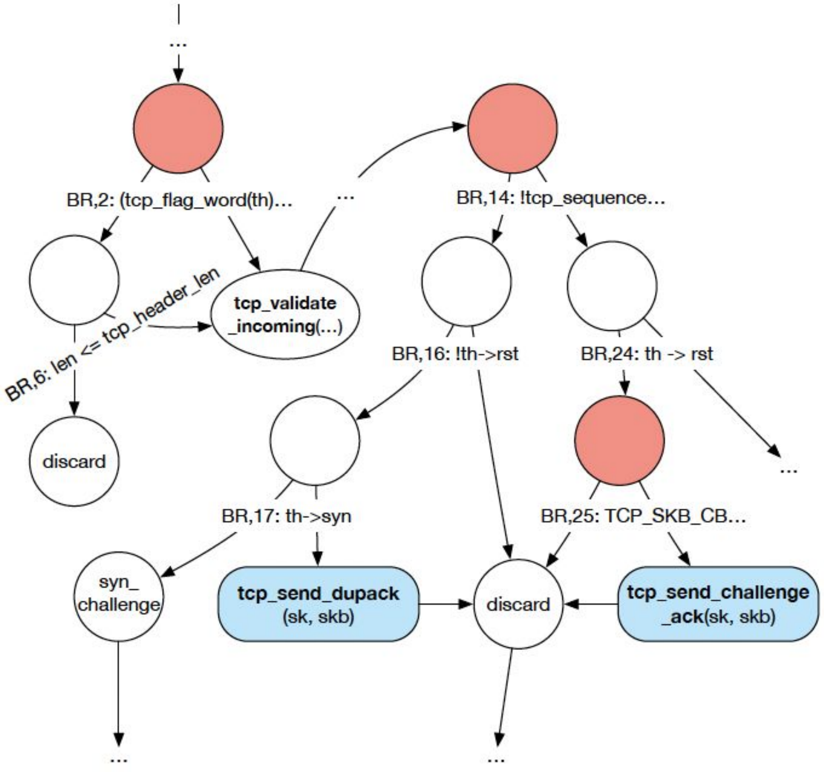
While there have been systematic work, they run into scalability issues and can only cover *a limited portion* of the code base:

- Model checking [Ensafi 2010, Cao 2019]: Very costly to build an abstract model; limited program states and interactions
- Fuzzing [Zou 2021]: Poor code coverage

Our Solution: A graph-based approach

In our work, we model detection of side-channel vulnerabilities as a graph search problem.

Time complexity: O(|V|)



Limitation #2: “Quantifying” side channels

Another limitation of prior side-channel study is lack of “quantification”: Measure of severity.

Side channel 1

- *tcp.c: L1824*

Side channel 2

- *udp.c: L505*

Side channel 3

- *icmp.c: L977*

…

Side channel 10248

- *some_random_file.c: L114514*

Our Solution: Quantifying and Ranking

Side channel 1
- *tcp.c: L1824*

Side channel 2
- *udp.c: L505*

Side channel 3
- *icmp.c: L977*

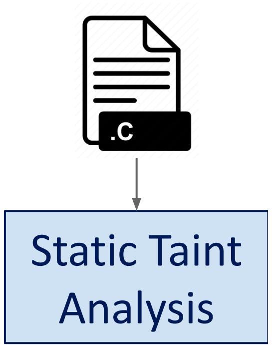
…

Branch #1
- *score: 1.00*

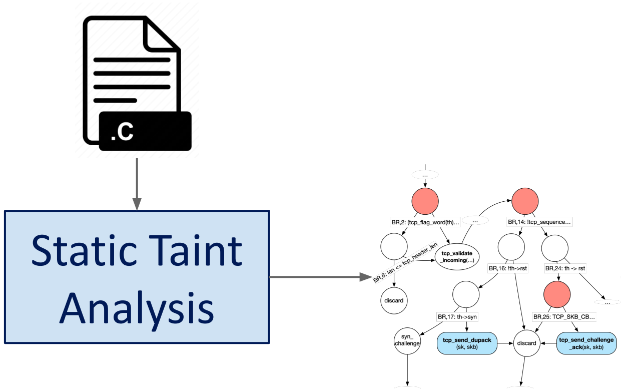
Branch #2
- *score: 0.96*

Branch #3
- *score: 0.85*

Design

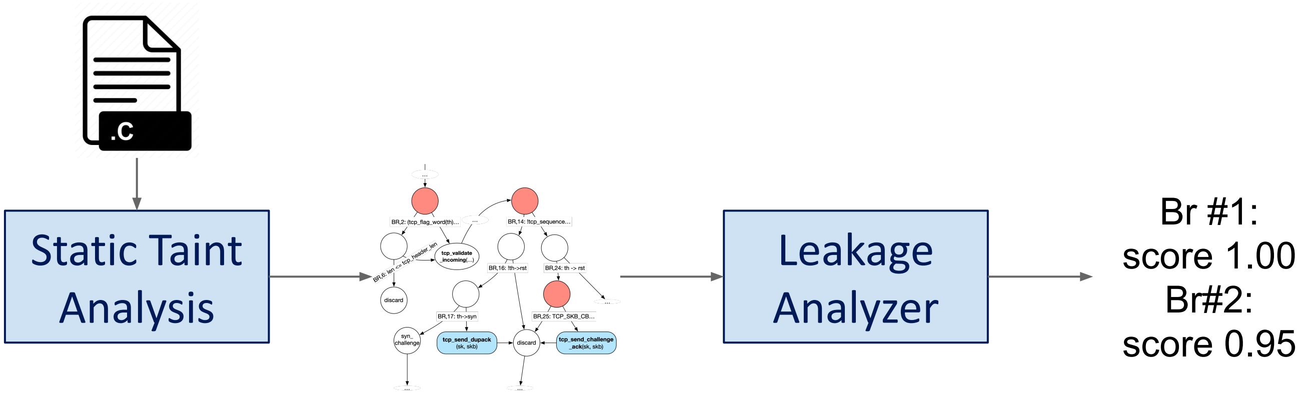


Design

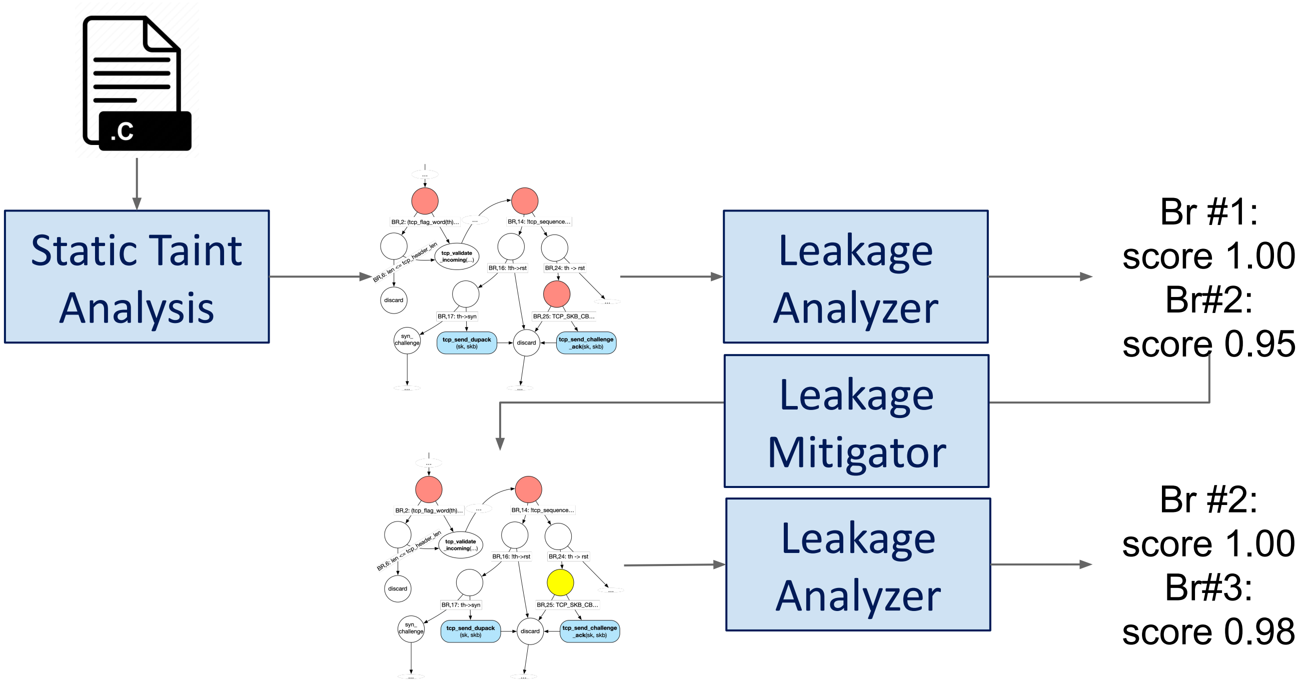


.C
Static Taint Analysis
BR,2: (tcp_flag_word(th)…
BR,14: !tcp_sequence…
tcp_validate _incoming(…)
BR,6: len <= tcp_header_len
BR,16: !th->rst
BR,24: th -> rst
discard
BR,17: th->syn
BR,25: TCP_SKB_CB…
syn_ challenge
tcp_send_dupack (sk, skb)
discard
tcp_send_challenge _ack(sk, skb)

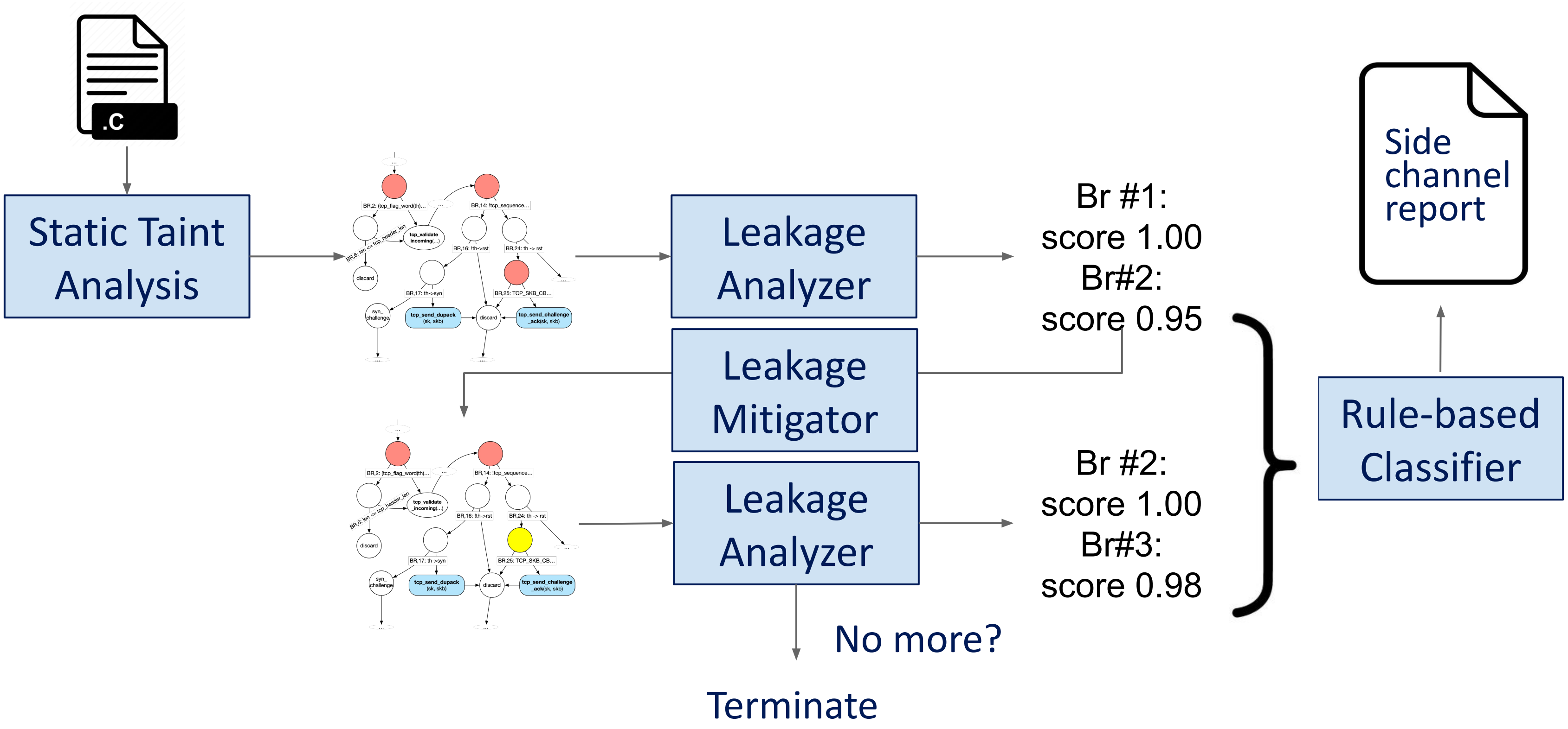
Design


.C
Static Taint Analysis
BR,2: (tcp_flag_word(th)…
BR,6: len <= tcp_header_len
discard
tcp_validate _incoming(…)
BR,14: !tcp_sequence…
BR,16: !th->rst
BR,24: th -> rst
BR,17: th->syn
BR,25: TCP_SKB_CB…
syn_ challenge
tcp_send_dupack (sk, skb)
discard
tcp_send_challenge _ack(sk, skb)
Leakage Analyzer
Br #1:
score 1.00
Br#2:
score 0.95

Design


.c
Static Taint Analysis
Leakage Analyzer
Br #1:
score 1.00
Br#2:
score 0.95
Leakage Mitigator
Leakage Analyzer
Br #2:
score 1.00
Br#3:
score 0.98

Design



.C
Static Taint Analysis
Leakage Analyzer
Br #1:
score 1.00
Br#2:
score 0.95
Leakage Mitigator
Leakage Analyzer
Br #2:
score 1.00
Br#3:
score 0.98
No more?
Terminate
Rule-based Classifier
Side channel report

Static Taint Analysis: Sensitive Branches

```
static void tcp_send_challenge_ack(struct sock *sk)
{
    static unsigned int ACK_COUNT;
    strict tcp_sock *tp = tcp_sk(sk);

    if (ACK_COUNT > 0) {
        NET_INC_STATS(sock_net(sk), LINUX_MIB_TCPCHALLENGEACK);
        tcp_send_ack(sk);
    }
}
```

Tainted by source

Sensitive Branch *(secret-dependent)*

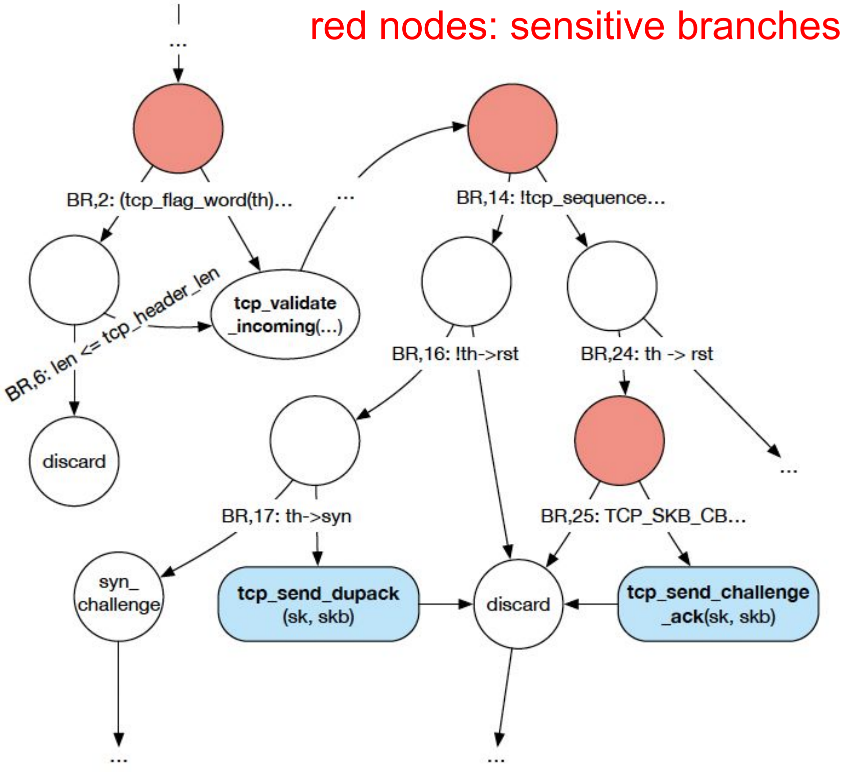
Sink / Observable outputs

Tainted Control-Flow-Graph (τCFG)

The *Tainted* CFG is a modified CFG with marked **sensitive branches**.

If a sensitive branch can reach two different observable outputs, it suggests a potential side channel (**critical branch**).

red nodes: sensitive branches

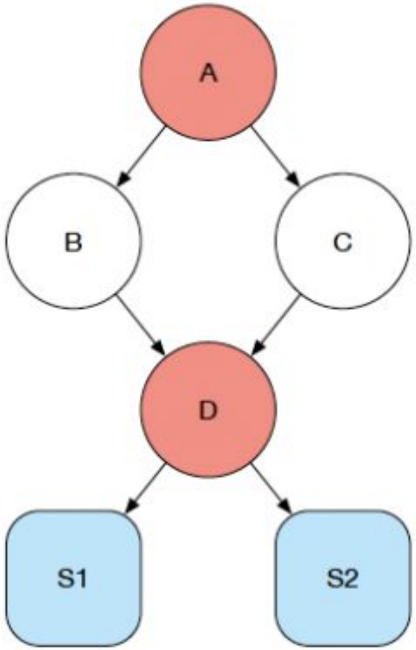


blue nodes: observable outputs

Why “Quantification”?

Q: Are both critical branches (**A** and **D**) equally severe?

- Intuitively, **A** has no control on the outcomes



Tainted CFG - Quantifying Side Channels

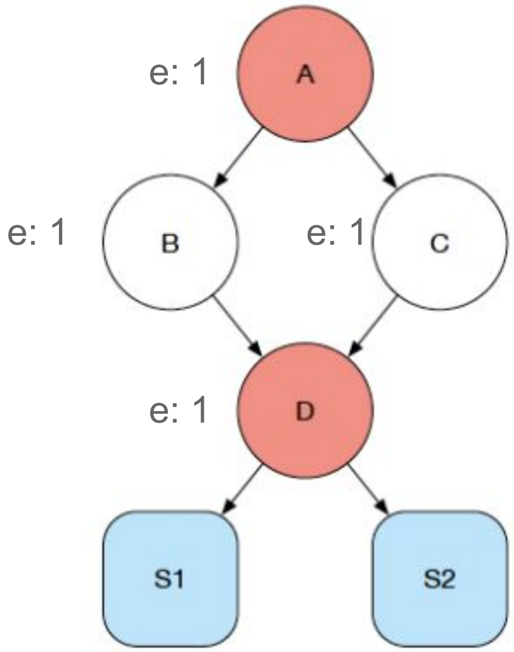
Idea of measuring leakage: *entropy difference*

Information entropy measures uncertainty, thus providing insight of how much information may be leaked at this point.

Definition 2 (Entropy of node). *Let $\tau CFG = (V, E, T, S)$ be an acyclic tainted CFG. For a node $v \in V$, let $\mathcal{H}_S(v)$ be the entropy of reaching the sink set S, defined as:*

$$\mathcal{H}_S(v) = \begin{cases} 0, & v \in S \\ -\sum_{s \in S} P(v,s) \log_2 P(v,s) & v \notin S \end{cases}$$

where $P(v,s)$ is the probability that node v reaches node s.

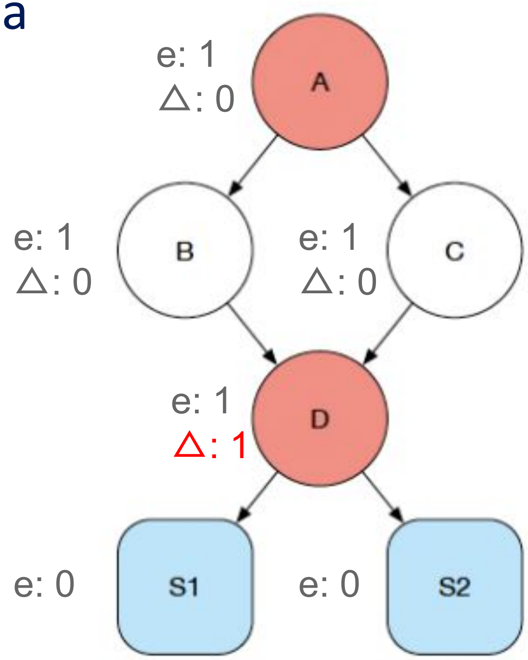


Tainted CFG - Quantifying Side Channels

Entropy difference (△) further measures how much a node ***contributes*** to the leakage.

In this example, **D** adds 1 entropy to the system, while **A** adds 0 (since either B or C already has 1 entropy), which matches the intuition that **D** is more critical.

Definition 3 (Leakage of node). *Let* $\tau CFG = \langle V, E, T, S \rangle$ *be an acyclic tainted CFG. For a node* $v \in V$*, let* $succ(v)$ *denote the set of the successors of v in* τCFG*. Let* $\mathcal{L}(v)$ *be the leakage of v defined as:* $\mathcal{L}(v) = \max_{i \in succ(v)} \mathcal{H}(v) - \mathcal{H}(i)$.



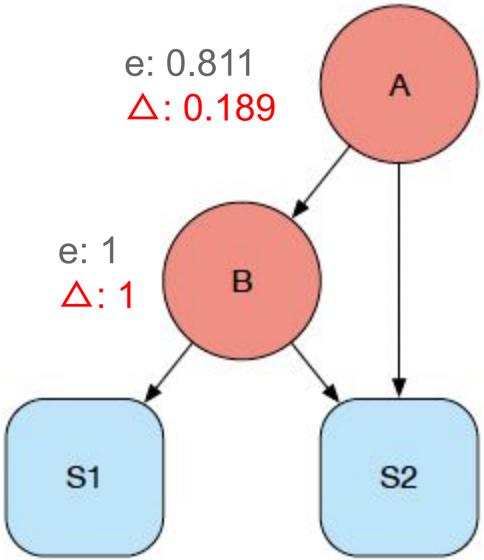
Identify All Side Channels

We have two reported branches:

- #1: B, △=1
- #2: A, △=0.189

If we fix B first, will A still remain a side channel?

But first, how would B be“fixed” in practice?



Real-world Mitigations

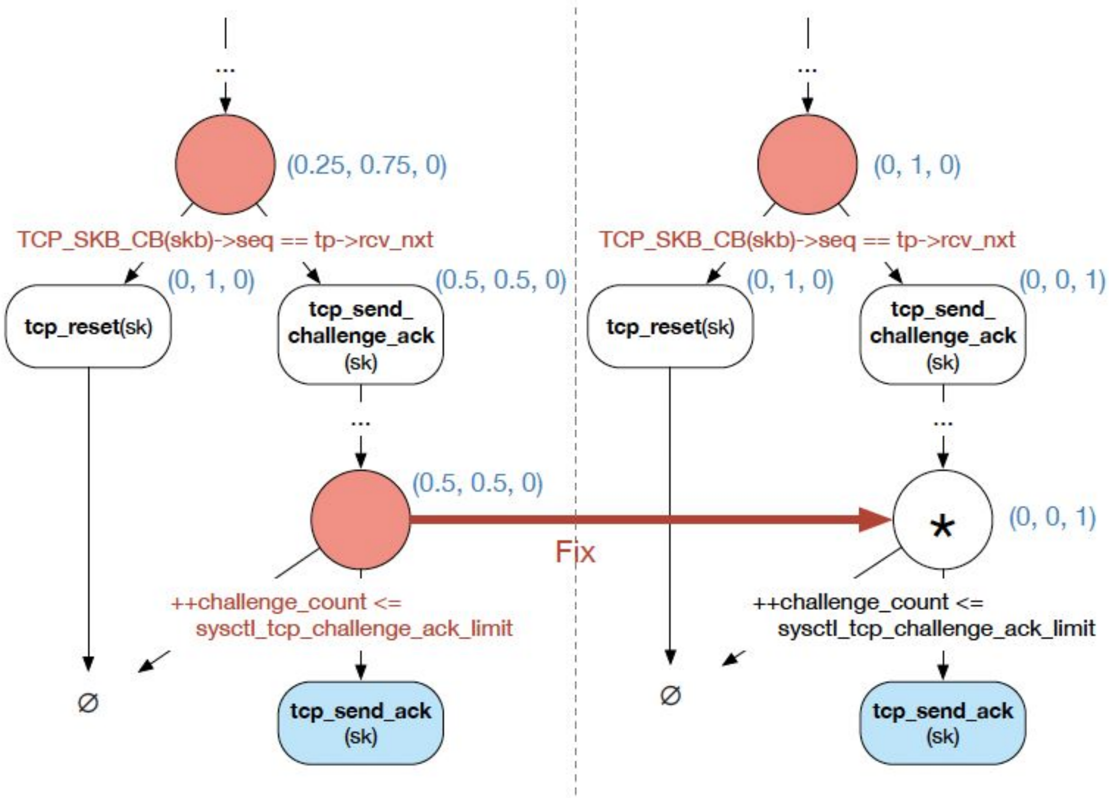
```
now = jiffies / HZ;
if (now != challenge_timestamp) {
        u32 half = (sysctl_tcp_challenge_ack_limit + 1) >> 1;

        challenge_timestamp = now;
        WRITE_ONCE(challenge_count, half +
                prandom_u32_max(sysctl_tcp_challenge_ack_limit));
}
count = READ_ONCE(challenge_count);
if (count > 0) {
        WRITE_ONCE(challenge_count, count - 1);
        NET_INC_STATS(sock_net(sk), LINUX_MIB_TCPCHALLENGEACK);
        tcp_send_ack(sk);
}
```

A mitigation in Linux v4: the ack limit is randomized.

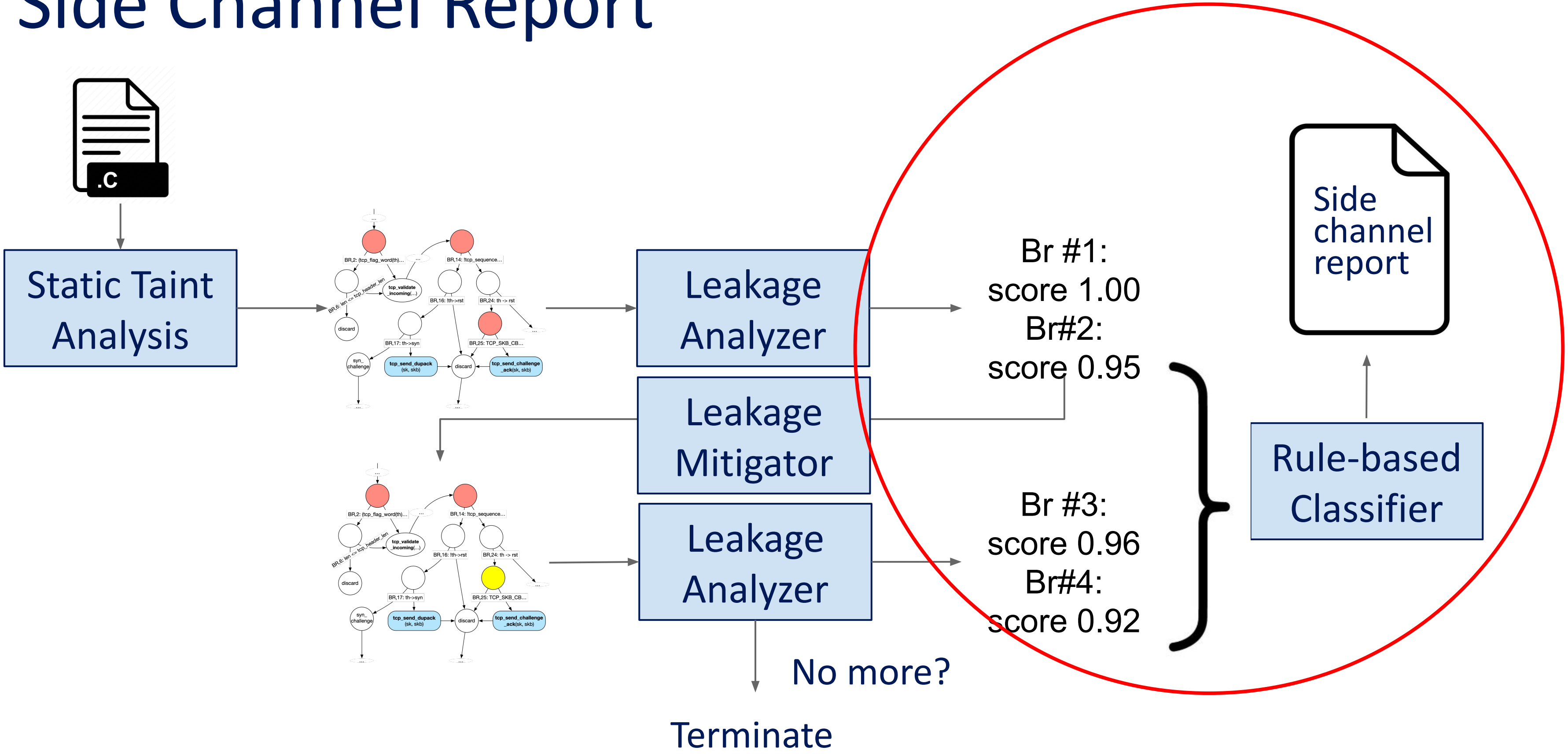
Rank-and-Replace Algorithm

We designed a replace algorithm and a special (*) node to mimic the mitigation.

Check our paper for more details.



Side Channel Report



Evaluation

Our tool is evaluted on several different TCP/UDP IPv4 implementations:

- Linux 3.12 and 4.8
- FreeBSD 13.2
- OpenBSD 7.4
- Open-source implementations:
    - Picotcp (1.1k stars)
    - Microps (1k stars)

Evaluation - Reduction

Evaluation results show that our tool ***significantly*** reduces number of candidate branches:

Tainted (sensitive): secret-dependent

Critical: non-zero entropy (reaches more than one observable)

	# tainted branches	# critical branches	# reported branches
Linux/TCP (Adv_u)	1651	185	6
Linux/TCP (Adv_a)	1651	528	5
Linux/UDP (Adv_u)	572	59	3
Linux/UDP (Adv_a)	572	354	3
FreeBSD/TCP (Adv_u)	843	199	10
FreeBSD/UDP (Adv_u)	310	28	1
OpenBSD/TCP (Adv_u)	751	173	10
OpenBSD/UDP (Adv_u)	302	27	1
microps	204	35	2
picotcp	505	75	1

Evaluation - Efficacy & Precision

- We uncovered 42 side channels, 30 of which are new.

- Compared to several prior works, our tool can detect all known side channels under the same threat model [Cao 2016, Cao 2019, Alharbi 2019, Man 2020, Man 2021, Qian 2012, Qian 2012]

- Only 5 out of 42 reported side channels are verified to be false positives.

Summary

The contributions of this work are:

- First to model the detection of TCP/UDP side-channel vulnerabilities as a graph-search problem
- Design and implement the automated tool for detecting and quantifying side channels
- Evaluated the tool on several benchmarks, uncovering 42 side channels

Our code is open-sourced at: https://github.com/athena-paper/athena

Thank you!