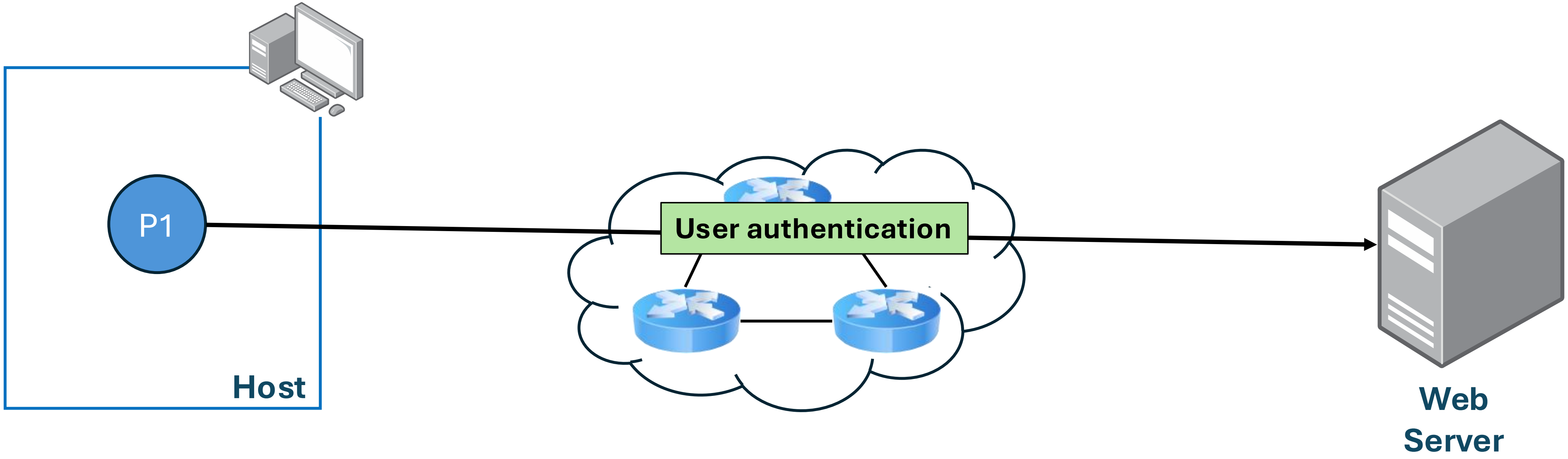
NetCap: Data-Plane Capability-Based Defense Against Token Theft in Network Access

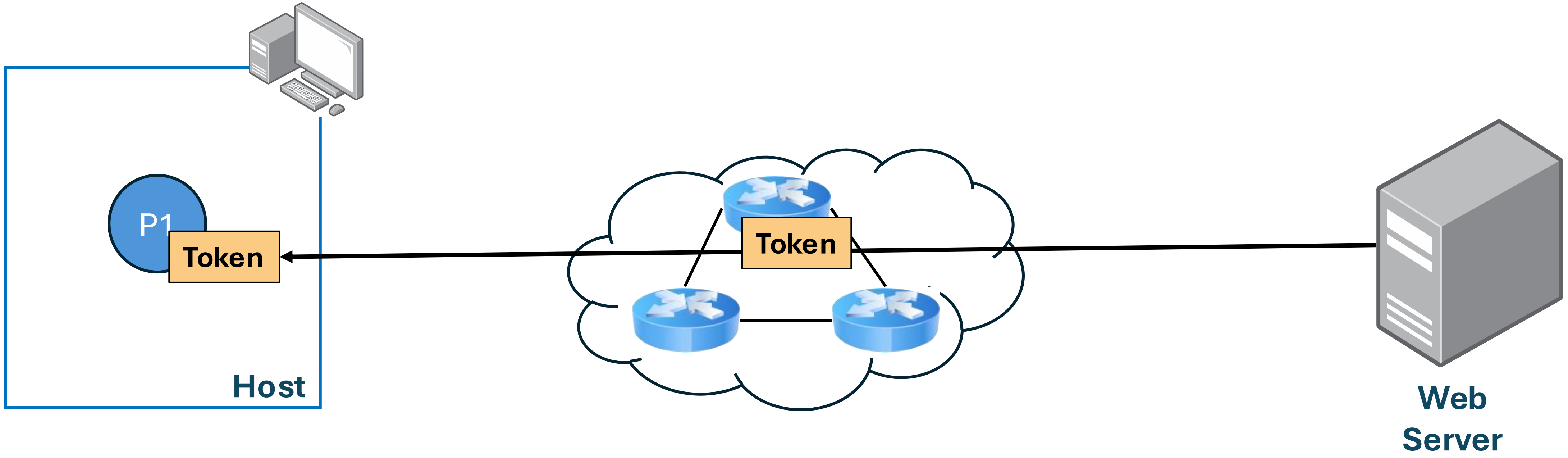
Osama Bajaber, Bo Ji, Peng Gao

VIRGINIA TECH

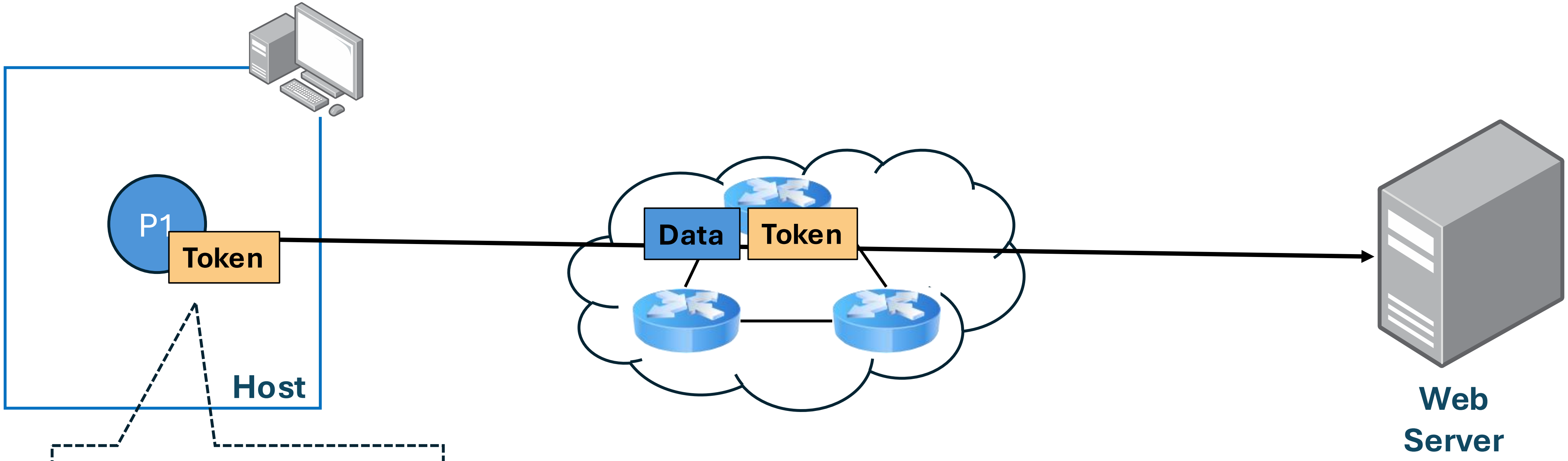
Token-Based Authentication



Token-Based Authentication



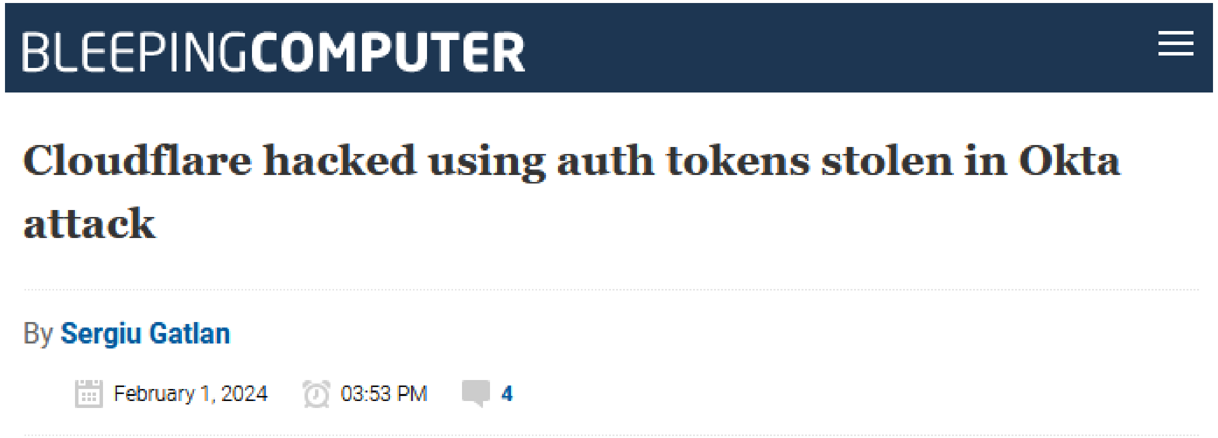
Token-Based Authentication



No need to re-enter user's credentials again

Attackers Leverage Stolen Access Tokens to Gain Unauthorized Access

Attackers Leverage Stolen Access Tokens to Gain Unauthorized Access

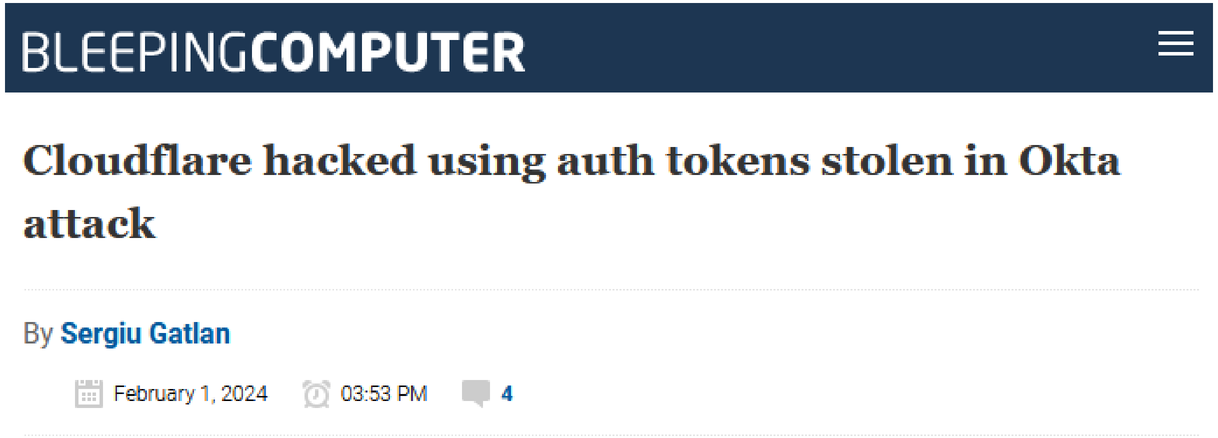


Cloudflare hacked using auth tokens stolen in Okta attack

By **Sergiu Gatlan**

February 1, 2024 03:53 PM 4

Attackers Leverage Stolen Access Tokens to Gain Unauthorized Access

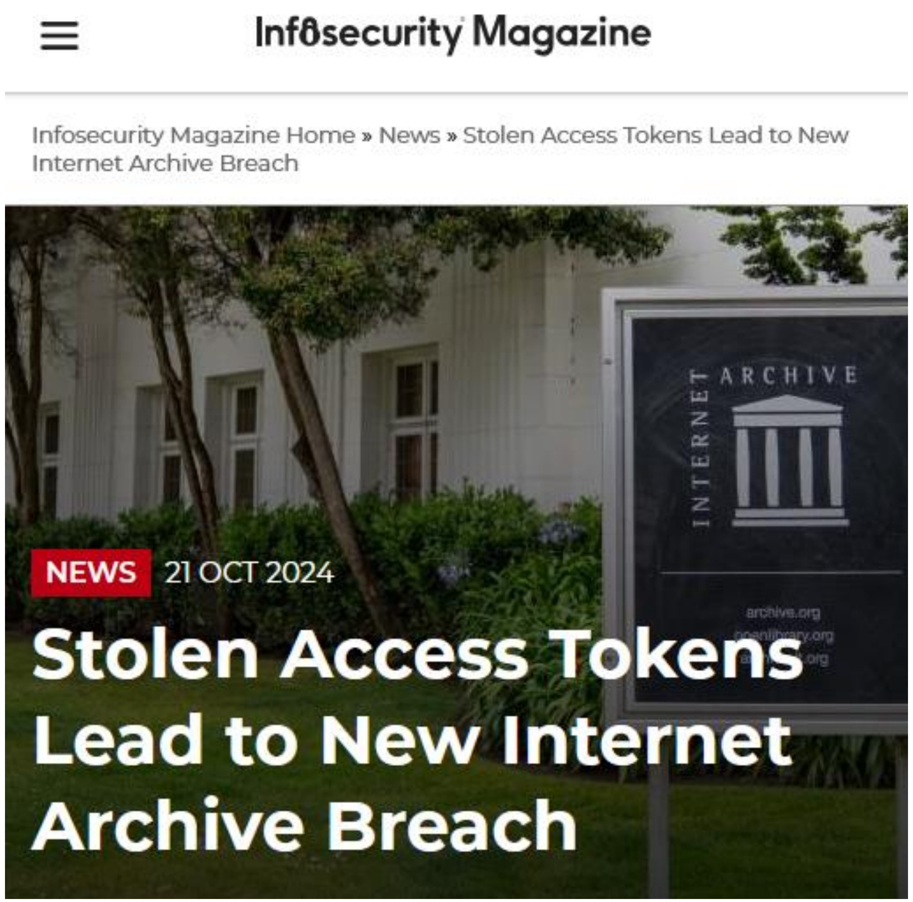


BLEEPINGCOMPUTER

Cloudflare hacked using auth tokens stolen in Okta attack

By **Sergiu Gatlan**

February 1, 2024 03:53 PM 4



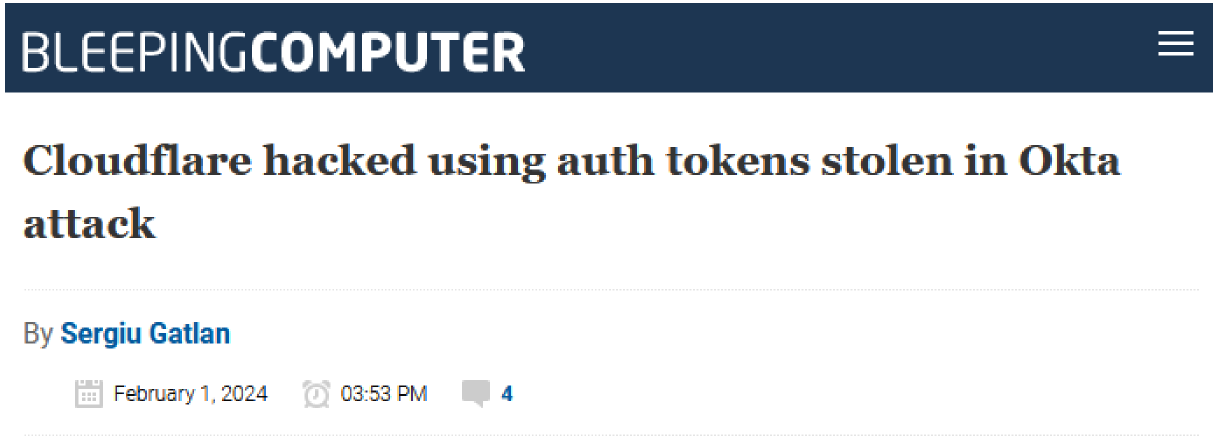
Infosecurity Magazine

Infosecurity Magazine Home » News » Stolen Access Tokens Lead to New Internet Archive Breach

NEWS 21 OCT 2024

Stolen Access Tokens Lead to New Internet Archive Breach

Attackers Leverage Stolen Access Tokens to Gain Unauthorized Access

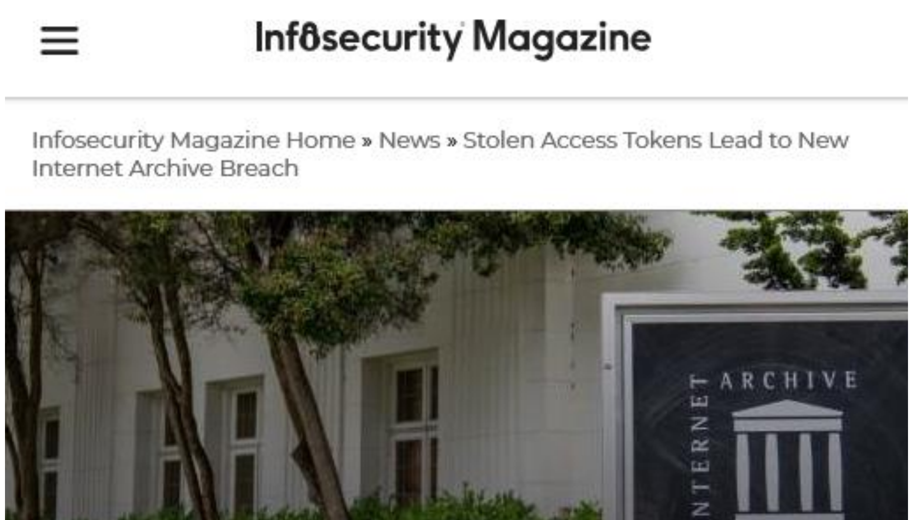


BLEEPINGCOMPUTER

Cloudflare hacked using auth tokens stolen in Okta attack

By Sergiu Gatlan

February 1, 2024 03:53 PM 4

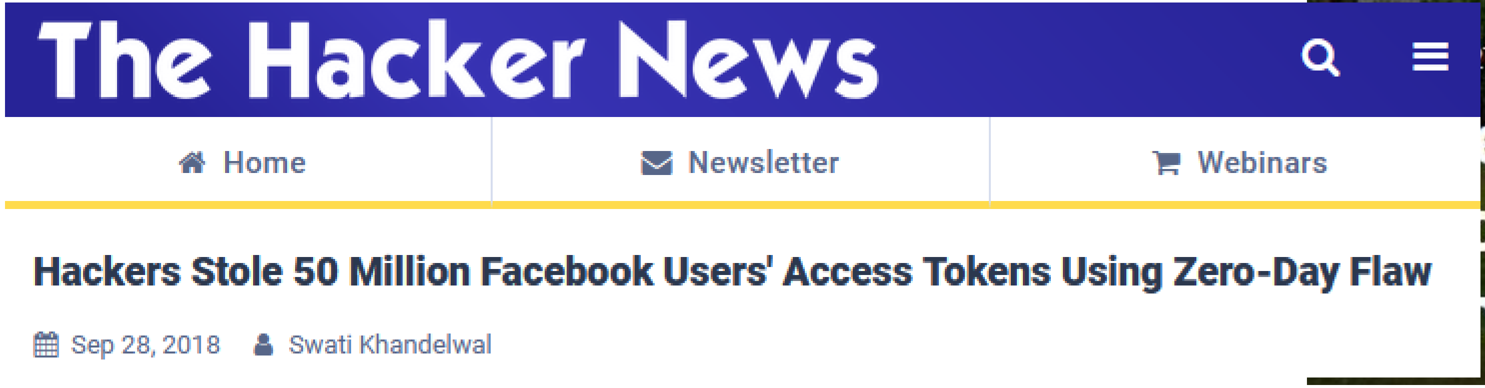


Infosecurity Magazine

Infosecurity Magazine Home » News » Stolen Access Tokens Lead to New Internet Archive Breach

1 OCT 2024

en Access Tokens
d to New Internet
hive Breach

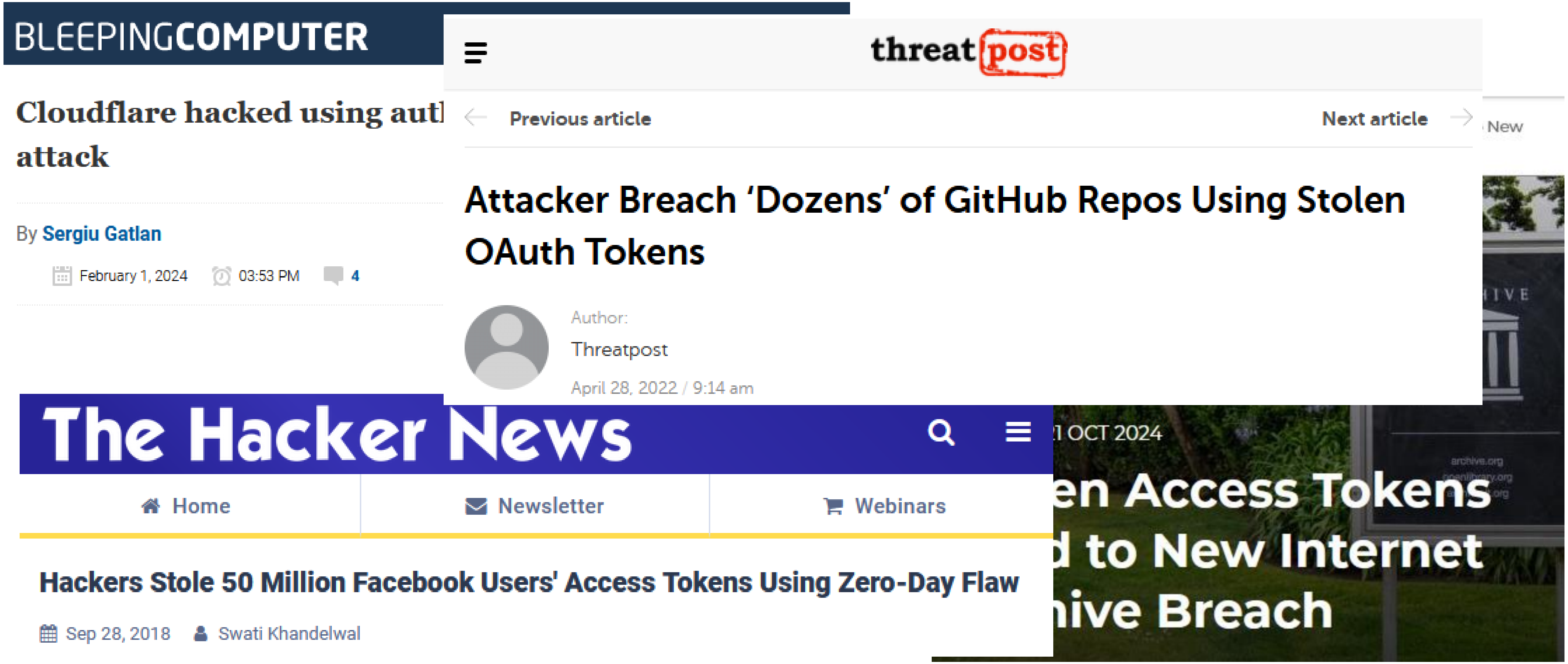


The Hacker News

Hackers Stole 50 Million Facebook Users' Access Tokens Using Zero-Day Flaw

Sep 28, 2018 Swati Khandelwal

Attackers Leverage Stolen Access Tokens to Gain Unauthorized Access



BLEEPINGCOMPUTER

Cloudflare hacked using autl attack

By Sergiu Gatlan

February 1, 2024 03:53 PM 4

threatpost

Previous article

Next article

Attacker Breach 'Dozens' of GitHub Repos Using Stolen OAuth Tokens

Author:
Threatpost
April 28, 2022 / 9:14 am

The Hacker News

Home Newsletter Webinars

Hackers Stole 50 Million Facebook Users' Access Tokens Using Zero-Day Flaw

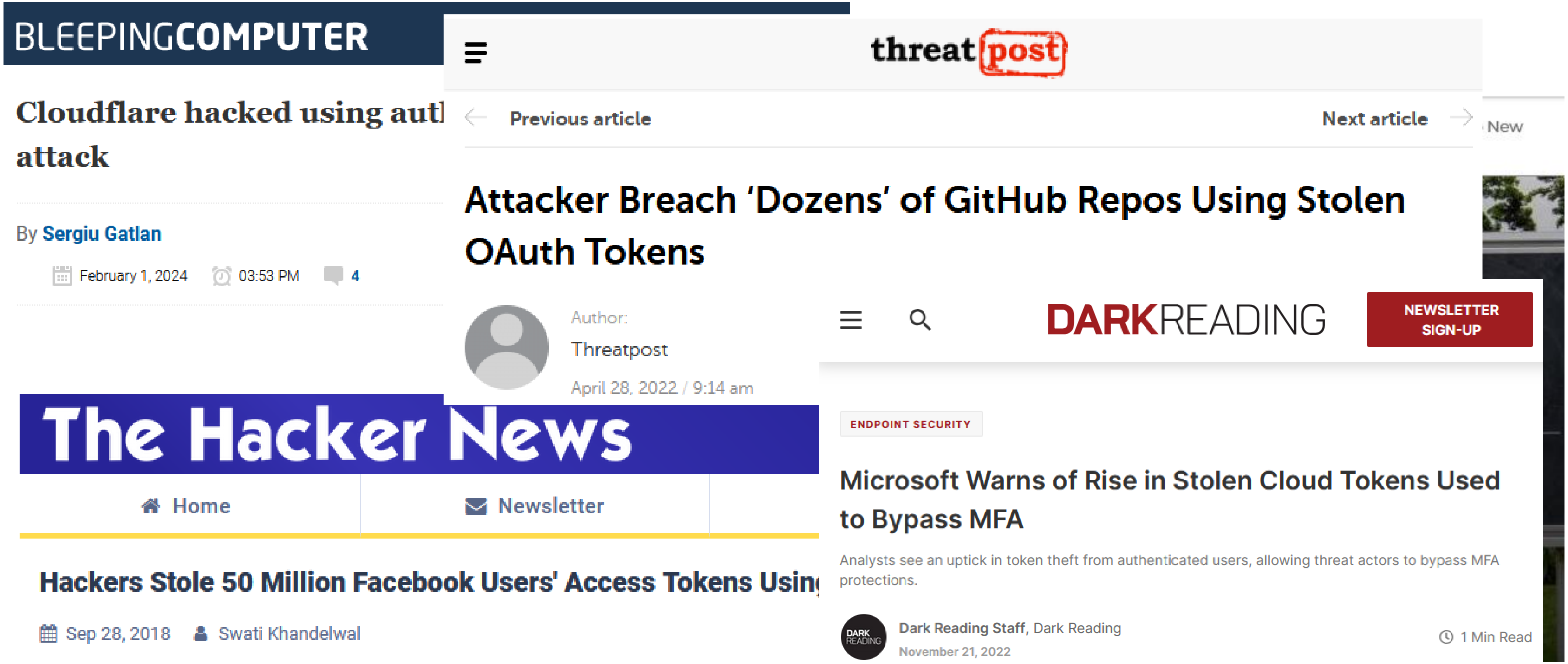
Sep 28, 2018 Swati Khandelwal

New

1 OCT 2024

en Access Tokens
d to New Internet
hive Breach

Attackers Leverage Stolen Access Tokens to Gain Unauthorized Access



Cloudflare hacked using autl attack

By Sergiu Gatlan

February 1, 2024 03:53 PM 4

Attacker Breach 'Dozens' of GitHub Repos Using Stolen OAuth Tokens

Author: Threatpost

April 28, 2022 / 9:14 am

Hackers Stole 50 Million Facebook Users' Access Tokens Usin

Sep 28, 2018 Swati Khandelwal

Microsoft Warns of Rise in Stolen Cloud Tokens Used to Bypass MFA

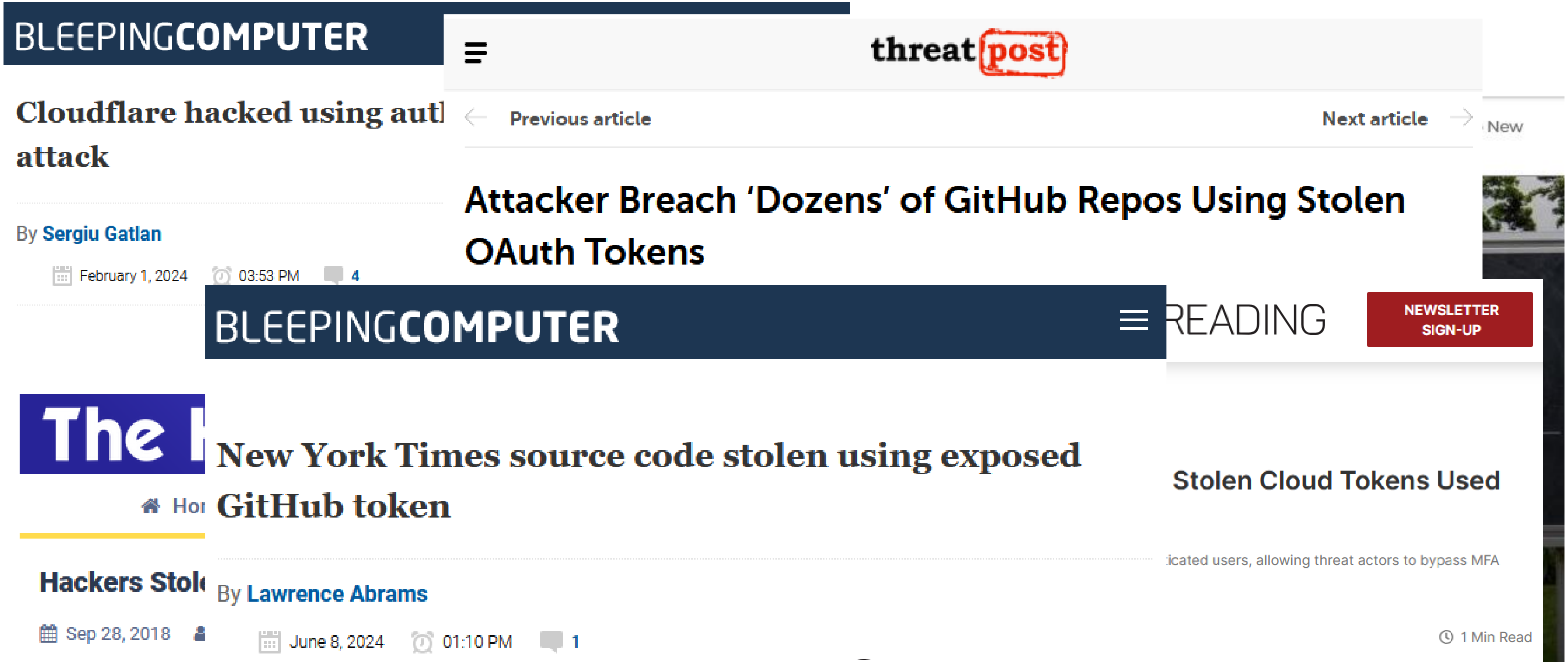
Analysts see an uptick in token theft from authenticated users, allowing threat actors to bypass MFA protections.

Dark Reading Staff, Dark Reading

November 21, 2022

1 Min Read

Attackers Leverage Stolen Access Tokens to Gain Unauthorized Access



BLEEPINGCOMPUTER

Cloudflare hacked using autl attack

By Sergiu Gatlan

February 1, 2024 03:53 PM 4

threatpost

Previous article

Next article

New

Attacker Breach 'Dozens' of GitHub Repos Using Stolen OAuth Tokens

READING

NEWSLETTER SIGN-UP

Stolen Cloud Tokens Used

icated users, allowing threat actors to bypass MFA

1 Min Read

The I

Hor

Hackers Stol

Sep 28, 2018

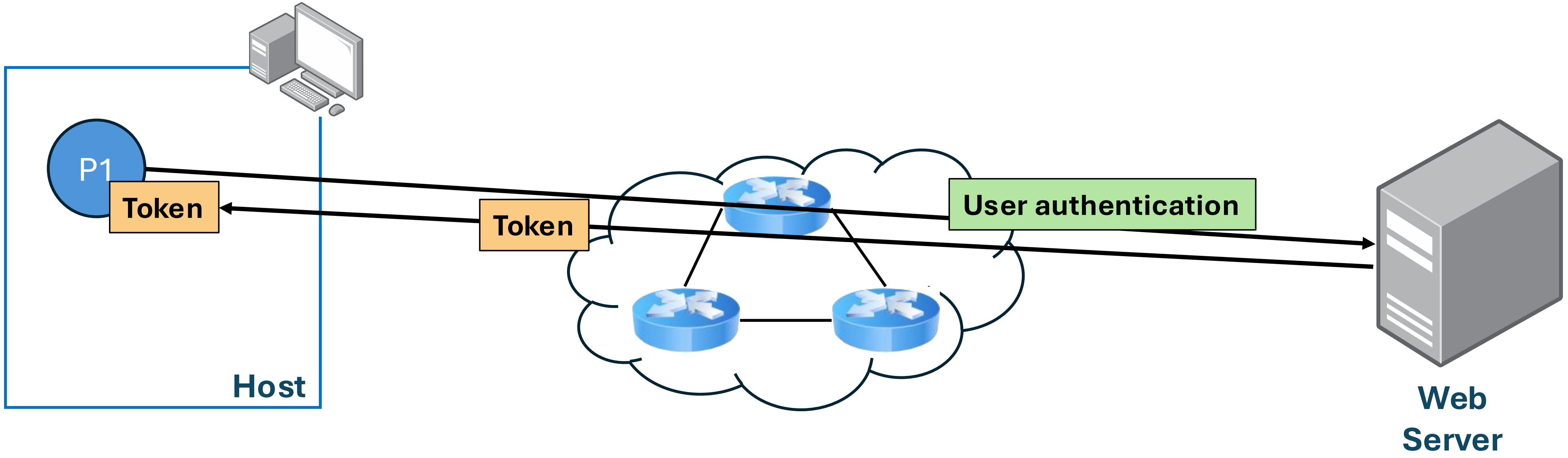
BLEEPINGCOMPUTER

New York Times source code stolen using exposed GitHub token

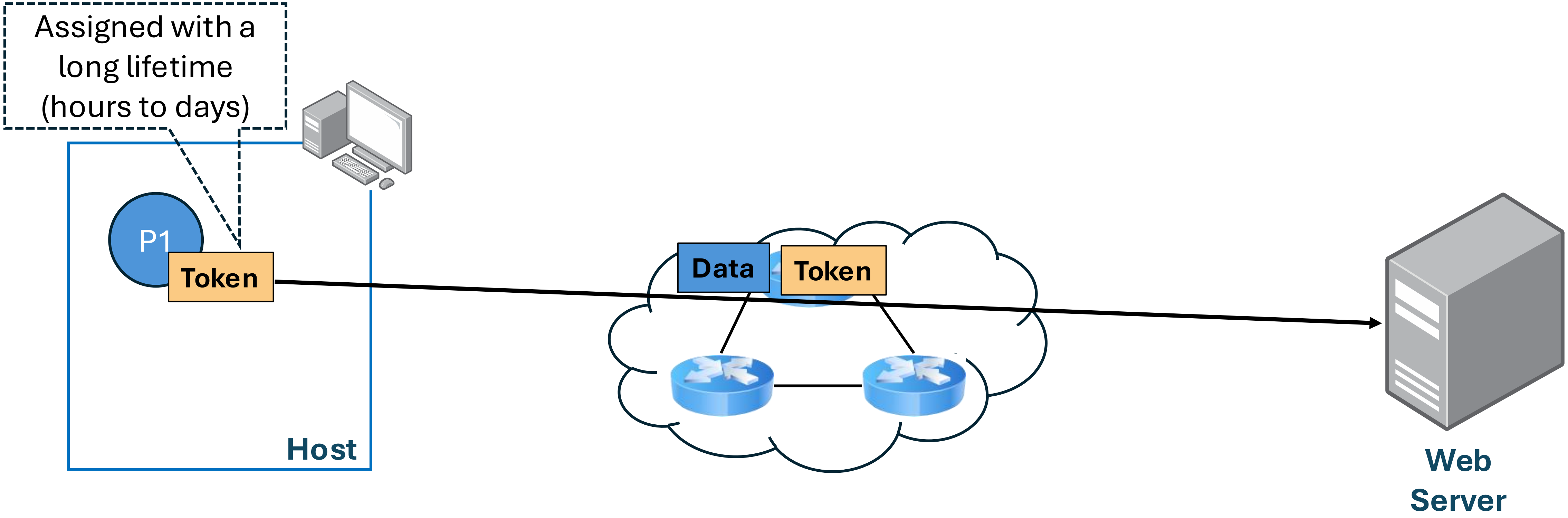
By Lawrence Abrams

June 8, 2024 01:10 PM 1

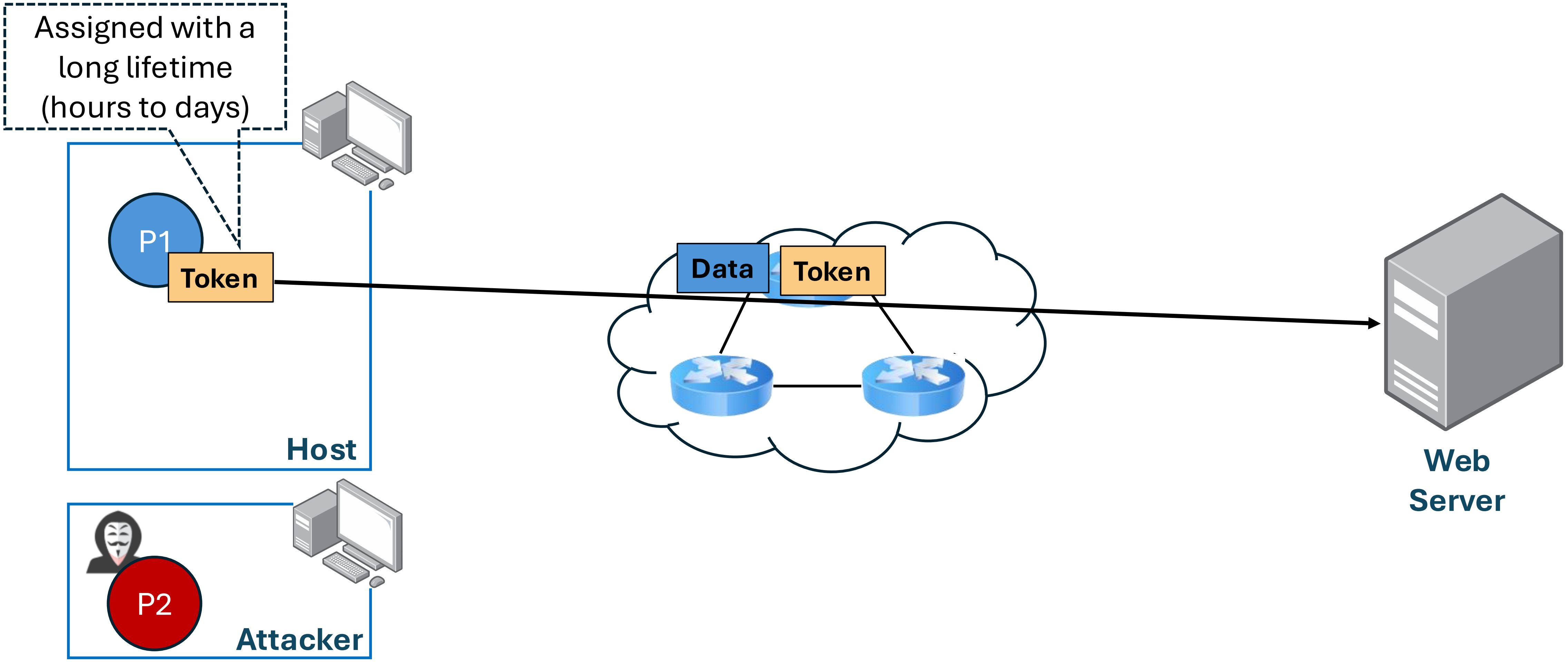
Token Theft Attack: An Illustration



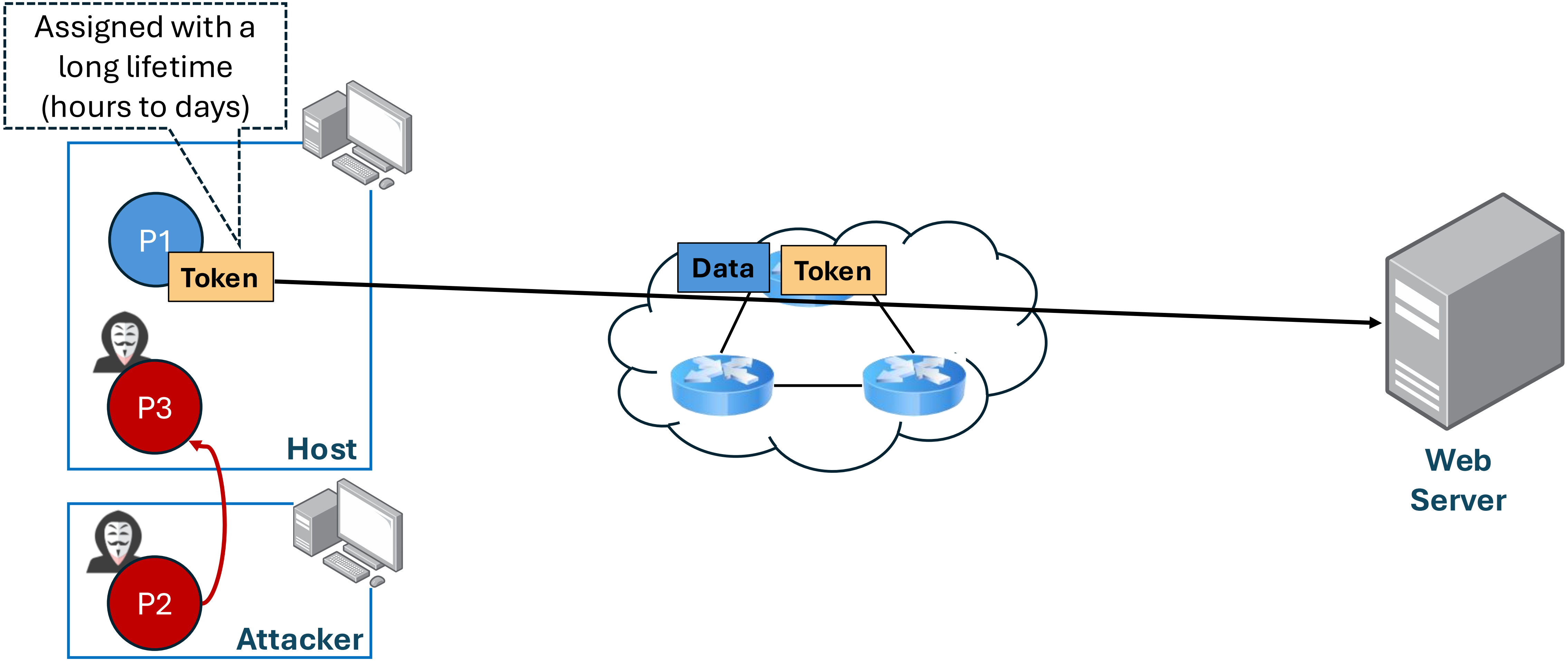
Token Theft Attack: An Illustration



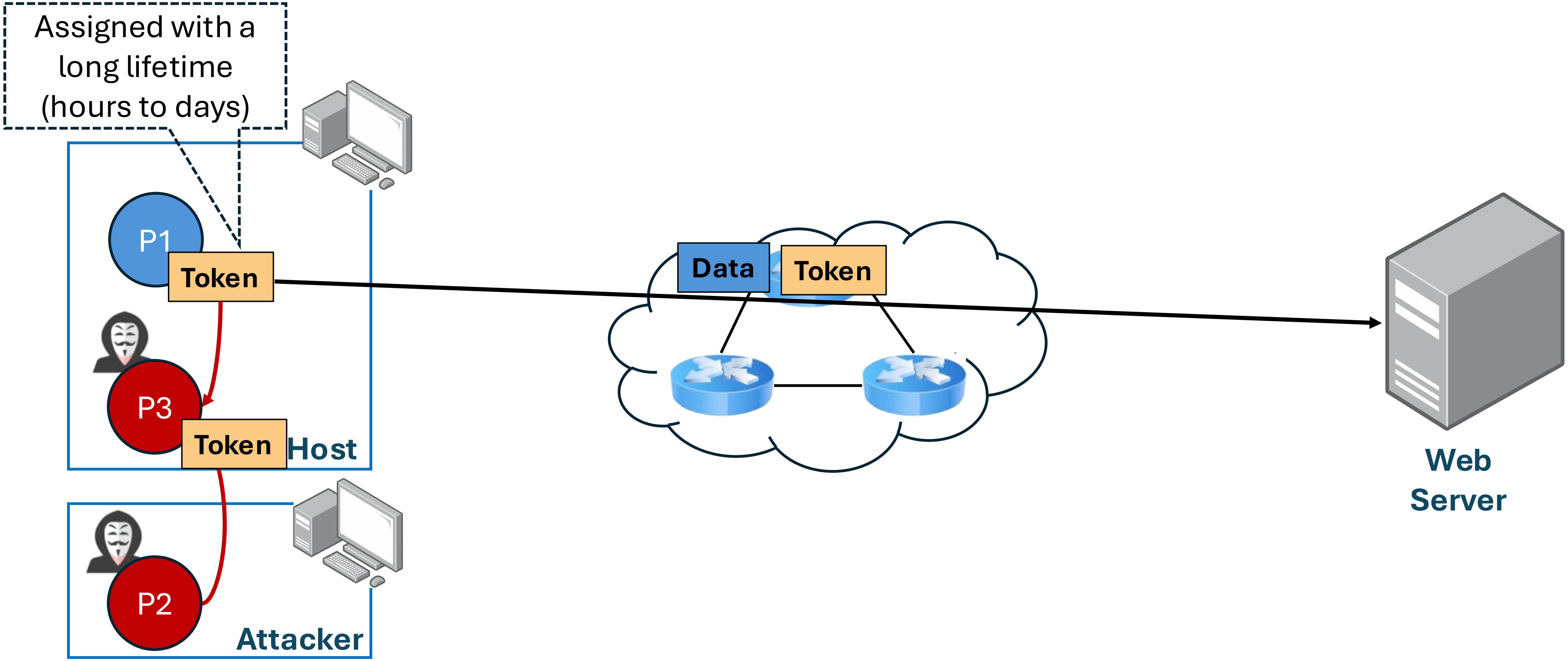
Token Theft Attack: An Illustration



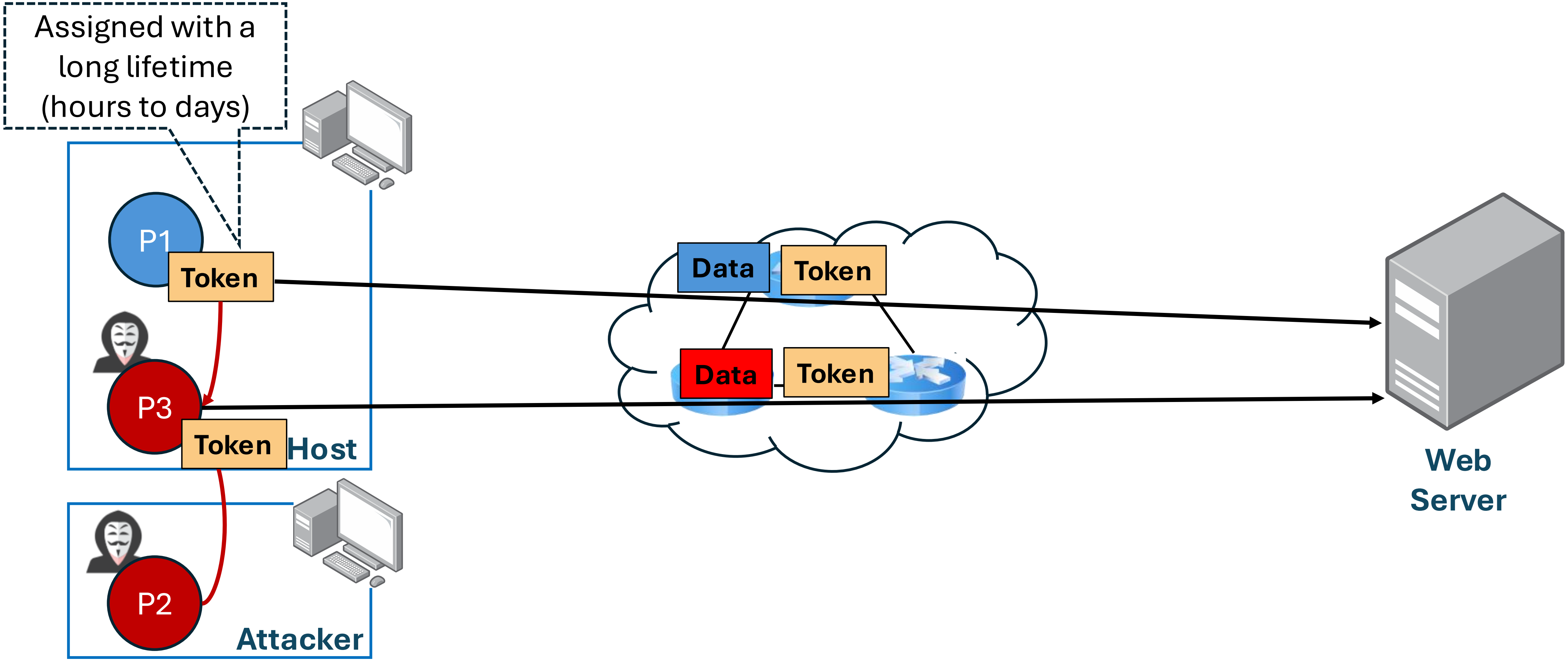
Token Theft Attack: An Illustration



Token Theft Attack: An Illustration



Token Theft Attack: An Illustration



Problem: Lack of Fine-Grained Continuous Authentication

- **Key issue 1:** Access tokens **are not bound** to authorized processes
  - Authentication servers assume **ambient trust** for all processes within hosts
    - Lack of process-level visibility

Problem: Lack of Fine-Grained Continuous Authentication

- **Key issue 1:** Access tokens **are not bound** to authorized processes
  - Authentication servers assume **ambient trust** for all processes within hosts
    - Lack of process-level visibility
- **Key issue 2:** Access tokens have **a long lifetime**
  - E.g., Hours to months for OAuth, 10 hours for Kerberos
    - **Ample time** for attackers to use the access token before it expires

Current Approaches **are not Enough!**

- Recent efforts created **protocol-specific** security enhancements (e.g., OAuth, OpenID)
  - Limited applicability across different protocols
  - Attackers continue to exploit zero-day system vulnerabilities

Current Approaches **are not Enough!**

- Recent efforts created **protocol-specific** security enhancements (e.g., OAuth, OpenID)
  - Limited applicability across different protocols
  - Attackers continue to exploit zero-day system vulnerabilities
- Current continuous authentication methods focus on **device-level** operations
  - Authenticate users through biometric data or behavioral patterns

Current Approaches **are not Enough!**

- Recent efforts created **protocol-specific** security enhancements (e.g., OAuth, OpenID)
  - Limited applicability across different protocols
  - Attackers continue to exploit zero-day system vulnerabilities

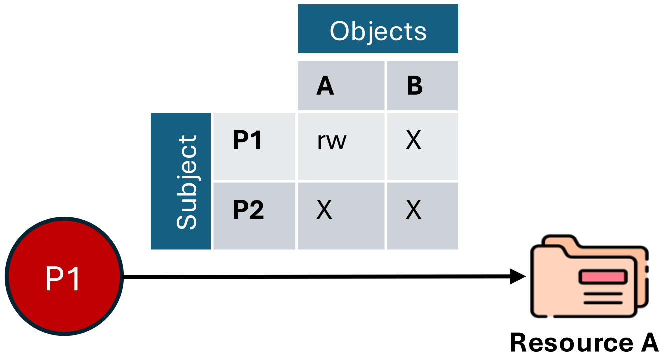
- Current continuous authentication methods focus on **device-level** operations
  - Authenticate users through biometric data or behavioral patterns

- **Goal: Enable continuous access validation at the process level for secure network access**
  - Add an extra layer of protection to a wide spectrum of protocols
  - Perform continuous authentication on every request from each process on a host

Access Control Mechanisms

Access Control Mechanisms

Access Control List (ACL)

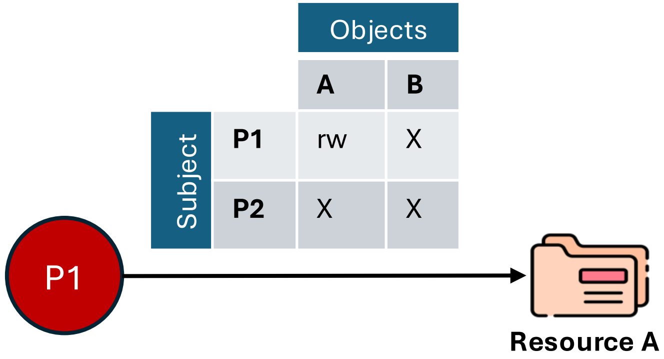


		Objects	
		A	B
Subject	P1	rw	X
	P2	X	X

- ACL defines the **list** of subjects and the operations they can perform on objects
- Inherently centralized
- **Limitation:** Not scalable with large and frequently changing volume of subjects

Access Control Mechanisms

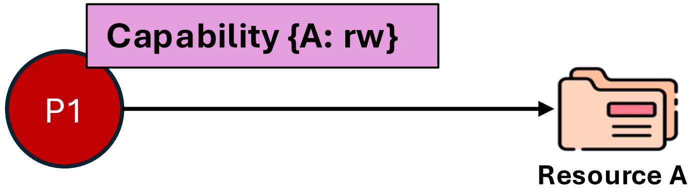
Access Control List (ACL)



		Objects	
		A	B
Subject	P1	rw	X
	P2	X	X

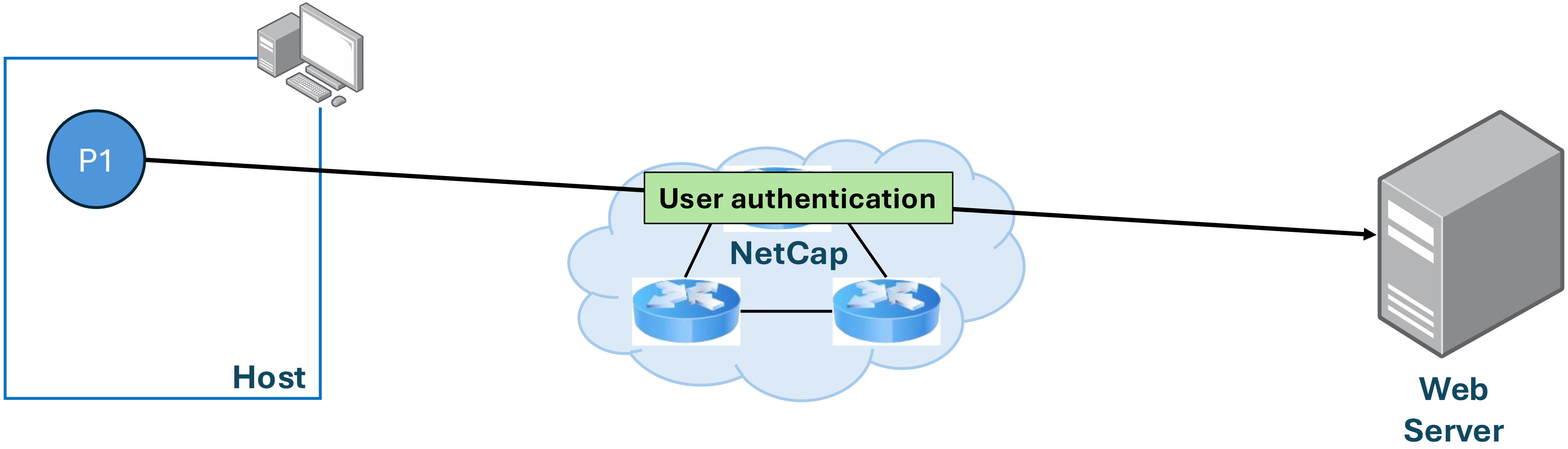
- ACL defines the **list** of subjects and the operations they can perform on objects
- Inherently centralized
- **Limitation:** Not scalable with large and frequently changing volume of subjects

Capability-Based Systems



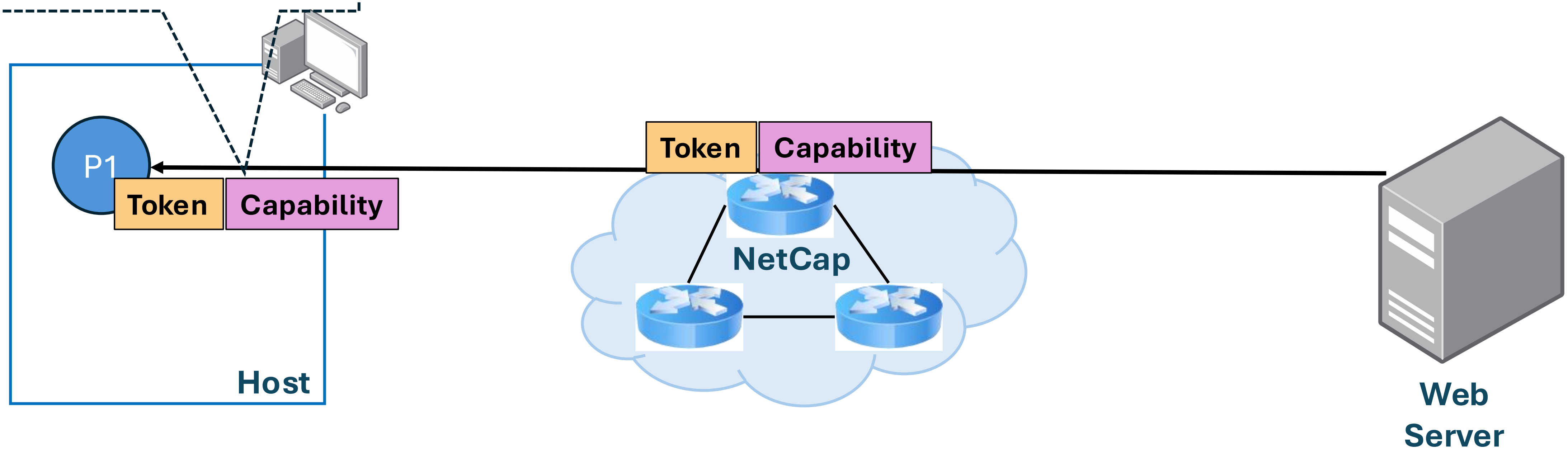
- A capability is an **unforgeable proof** that grants its holder permission to access a specific resource
- Inherently decentralized
- OSes integrated capabilities to **control process access** to memory locations
- **Limitation:** No **extended process capability control** to network accesses

Key Contribution: **In-Network**, Fine-Grained, Capability-Based Defense (NetCap)



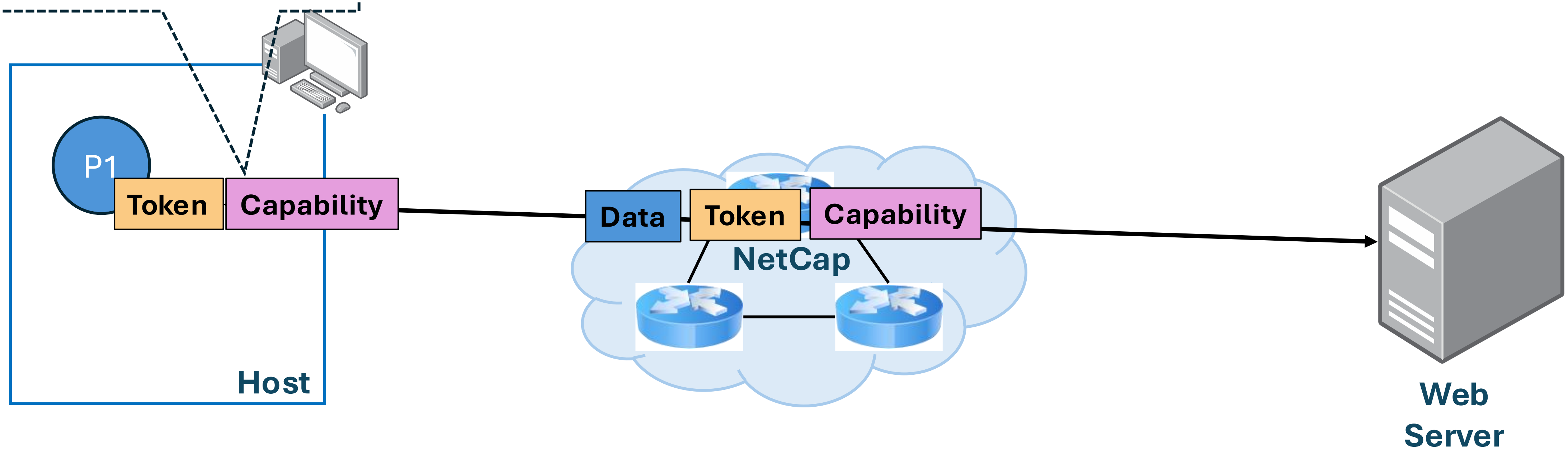
Key Contribution: **In-Network**, Fine-Grained, Capability-Based Defense (NetCap)

A capability is **cryptographically bound** to the authorized process



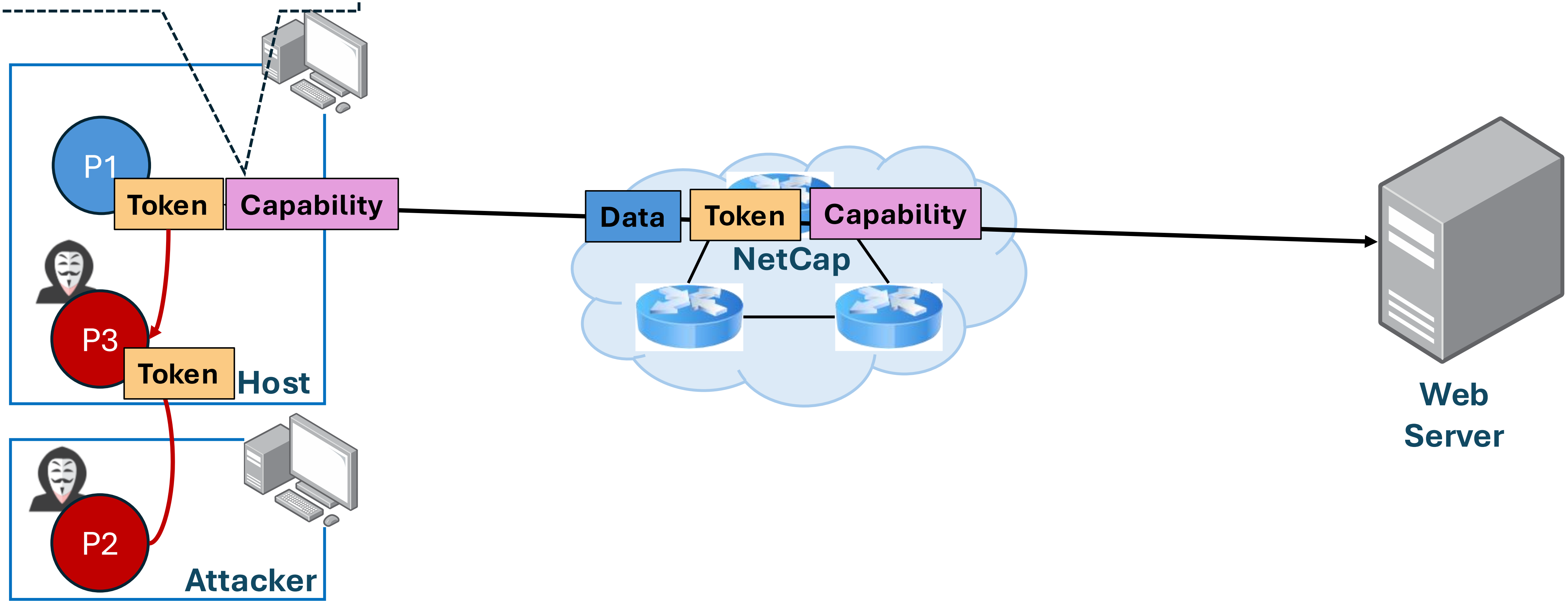
Key Contribution: **In-Network**, Fine-Grained, Capability-Based Defense (NetCap)

A capability is **cryptographically bound** to the authorized process



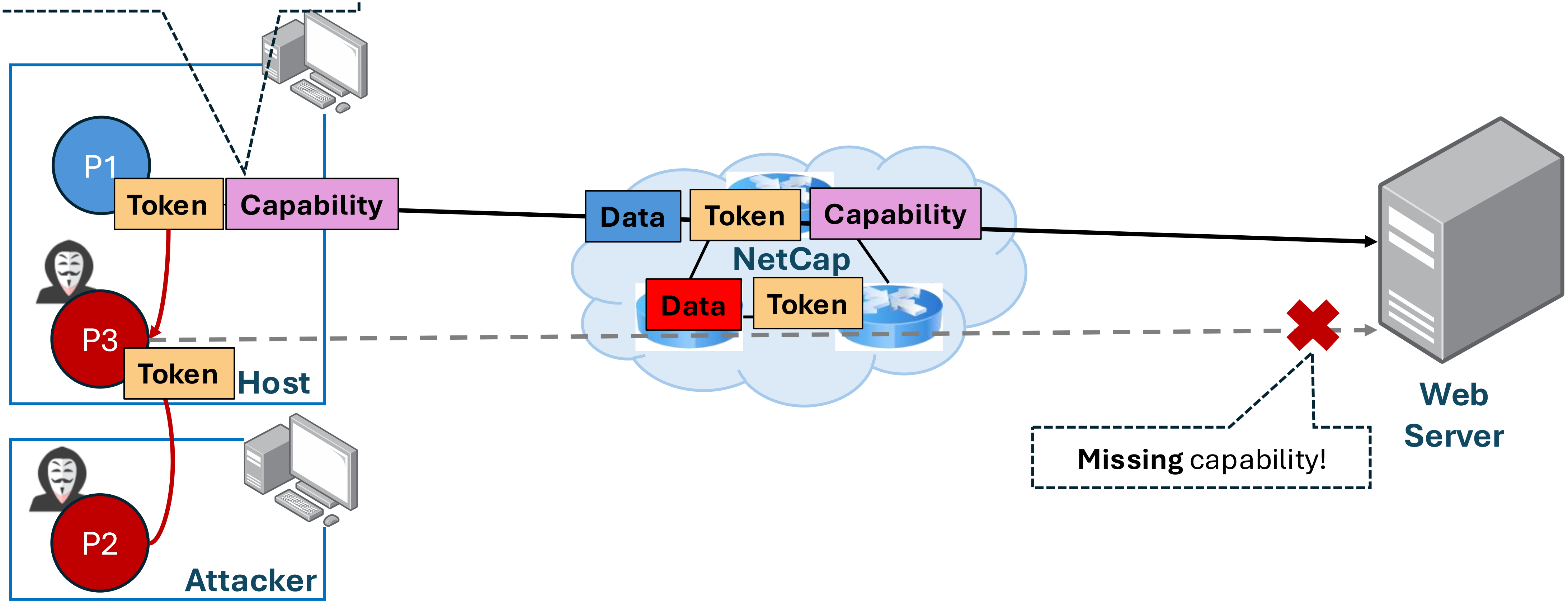
Key Contribution: **In-Network**, Fine-Grained, Capability-Based Defense (NetCap)

A capability is **cryptographically bound** to the authorized process



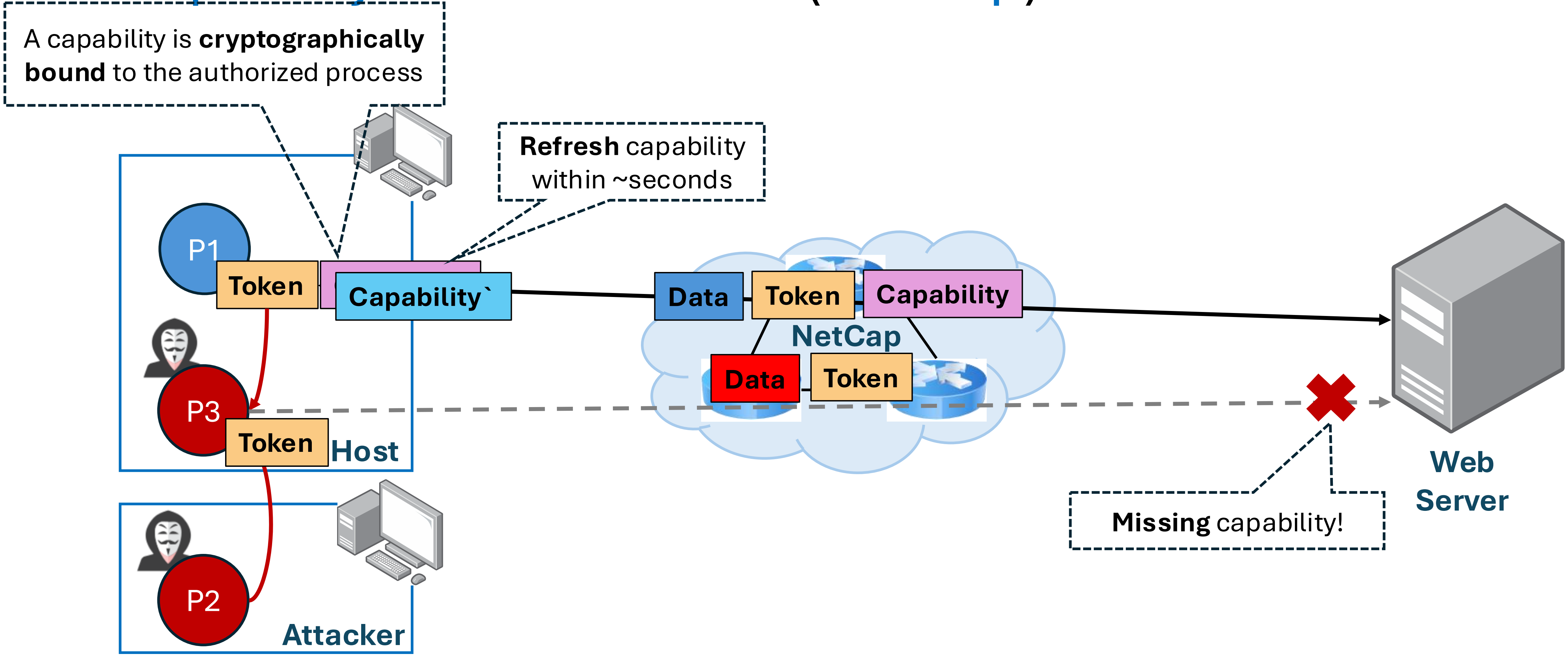
Key Contribution: **In-Network**, Fine-Grained, Capability-Based Defense (NetCap)

A capability is **cryptographically bound** to the authorized process



Missing capability!

Key Contribution: **In-Network**, Fine-Grained, Capability-Based Defense (NetCap)



A capability is **cryptographically bound** to the authorized process

Refresh capability within ~seconds

Missing capability!

Threat Model

- We aim to **prevent attackers from using stolen tokens** to access unauthorized resources in enterprise environments
- Attackers seek to steal tokens **through various techniques** like memory scraping, cross-site scripting, or network sniffing
- Attackers can be either within the same victim's host or controlling a separate host

Challenge 1: Fine-Grained Capabilities for Secure Network Access

- Capabilities need to be **unforgeable and tamper-proof**

Challenge 1: Fine-Grained Capabilities for Secure Network Access

- Capabilities need to be **unforgeable and tamper-proof**

- Need for an **extended capability mechanism** to control process access to network services
  - A capability enforces least-privilege access to a specific network service
  - A process's capabilities cannot be used by other processes

Challenge 1: Fine-Grained Capabilities for Secure Network Access

- Capabilities need to be **unforgeable and tamper-proof**
- Need for an **extended capability mechanism** to control process access to network services
  - A capability enforces least-privilege access to a specific network service
  - A process's capabilities cannot be used by other processes
- These capabilities should be valid for a **short lifetime**
  - Need a mechanism to frequently refresh capabilities

Contribution 1: **Fine-Grained, Network-Level** Capability Mechanism

$$Capability = Chaskey_{key}\ (PID||IP_{client}||IP_{service}||Port_{service}||Timestamp)$$

- **Unforgeable capabilities:** Capabilities are computed using **Chaskey**, a lightweight and secure cryptographic function

Contribution 1: **Fine-Grained, Network-Level** Capability Mechanism

$$Capability = Chaskey_{key}\ \underbrace{(PID||IP_{client}||IP_{service}||Port_{service}}||Timestamp)$$

- **Unforgeable capabilities:** Capabilities are computed using **Chaskey**, a lightweight and secure cryptographic function
- **Least-privilege:** Each capability is bound to a specific process to reach a specific network service

Contribution 1: **Fine-Grained, Network-Level** Capability Mechanism

$$Capability = Chaskey_{key}\ (\underbrace{PID||IP_{client}||IP_{service}||Port_{service}}_{}||\underbrace{Timestamp}_{})$$

- **Unforgeable capabilities:** Capabilities are computed using **Chaskey**, a lightweight and secure cryptographic function
- **Least-privilege:** Each capability is bound to a specific process to reach a specific network service
- **Short lifetime:** Capabilities expire within a short period (few seconds)

Contribution 1: **Fine-Grained, Network-Level** Capability Mechanism

$$Capability = Chaskey_{key}\ (\underbrace{PID||IP_{client}||IP_{service}||Port_{service}}_{}||\underbrace{Timestamp}_{})$$

- **Unforgeable capabilities:** Capabilities are computed using **Chaskey**, a lightweight and secure cryptographic function
- **Least-privilege:** Each capability is bound to a specific process to reach a specific network service
- **Short lifetime:** Capabilities expire within a short period (few seconds)

- **Capability-based authentication:** Conduct two critical checks:
  1. Capability is valid
  2. Capability has not expired

Challenge 2: Capability-Based Defense with Little Overhead

- Frequent capability **validation and refresh** will overwhelm commodity servers

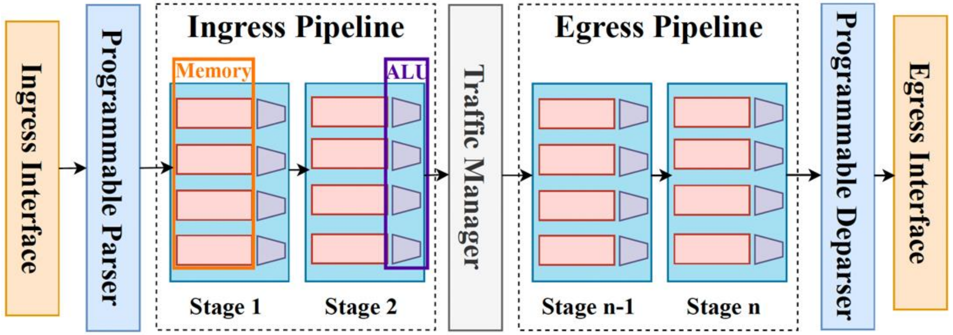
Challenge 2: Capability-Based Defense with Little Overhead

- Frequent capability **validation and refresh** will overwhelm commodity servers

- Requirements for an effective and efficient authentication scheme:
  - **Generate** capabilities to processes
  - **Validate** incoming capabilities
  - **Refresh** expired capabilities
  - **All with little overhead!**

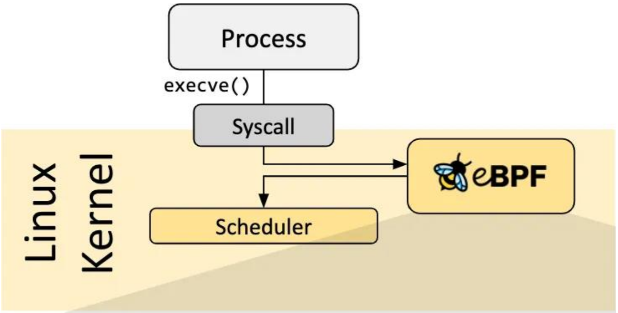
Key Enablers: Programmable Data Planes

Programmable Switches



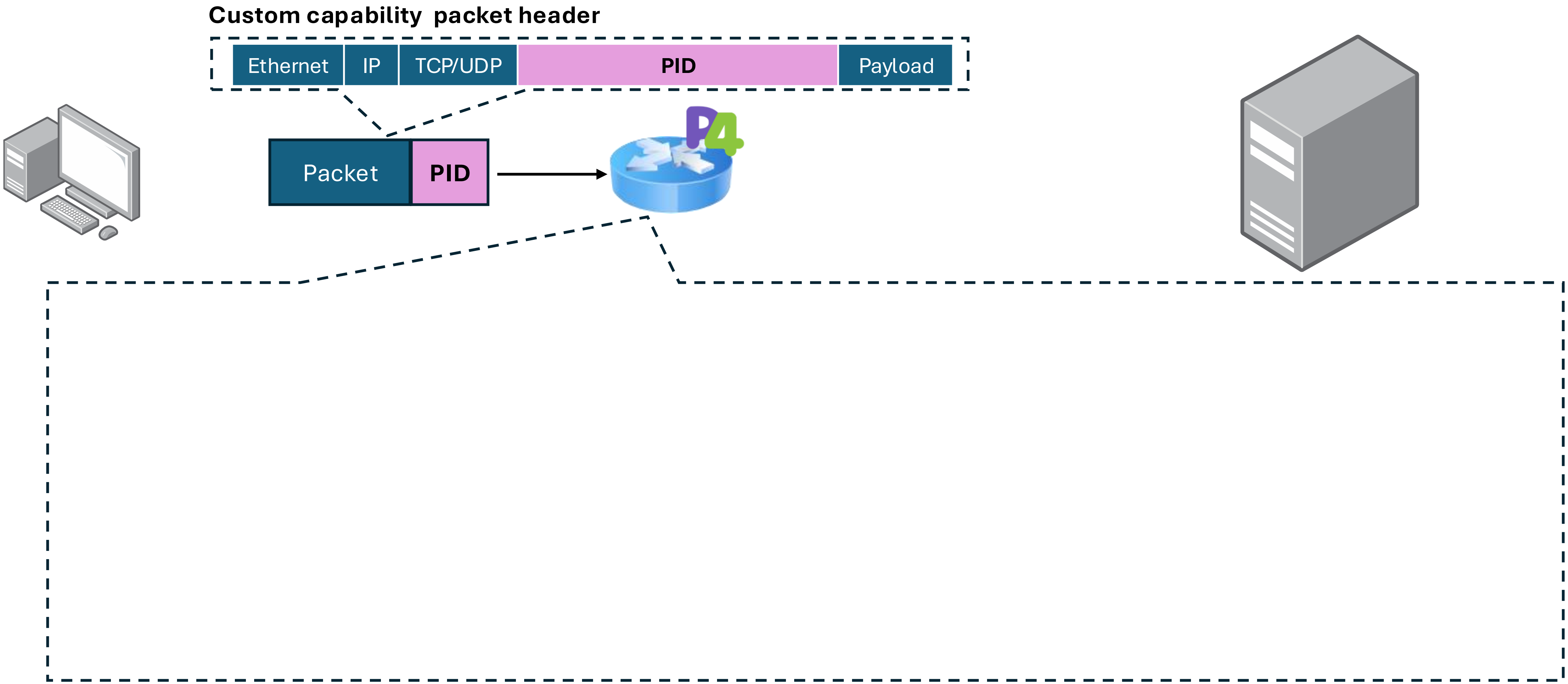
- Offer flexibility to define **custom packet processing logic**
- Support customized packet headers
  - Capability packet headers
- Run at linespeed (Tbps)
  - Enforce capability-based access controls on the fly

eBPF (Extended Berkeley Packet Filter)

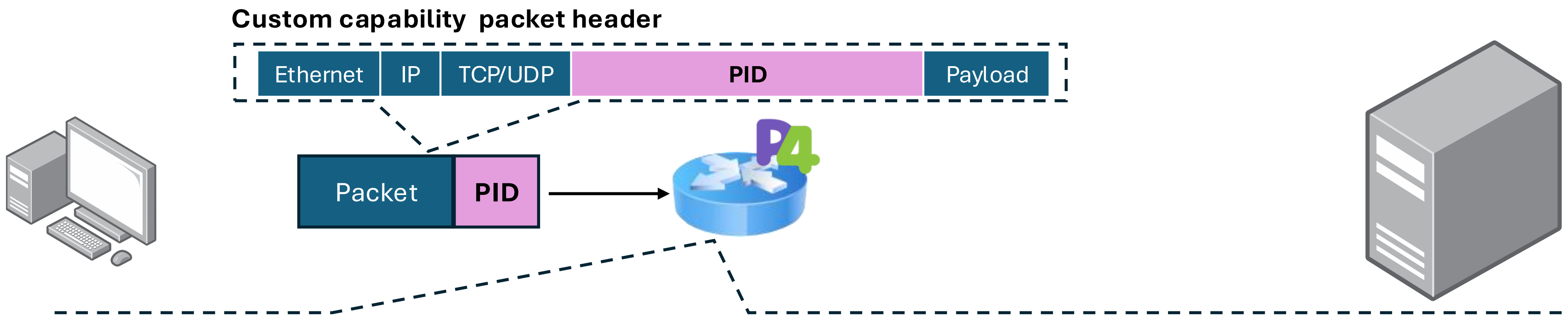


- Dynamically integrate **additional features** into the OS kernel
- Seamless integration
  - Trace process activities without any modifications to the kernel
- Modify network packets to incorporate capability packet headers
  - Maintain capability context persistence

Contribution 2: **In-Network** Capability-Based Defense



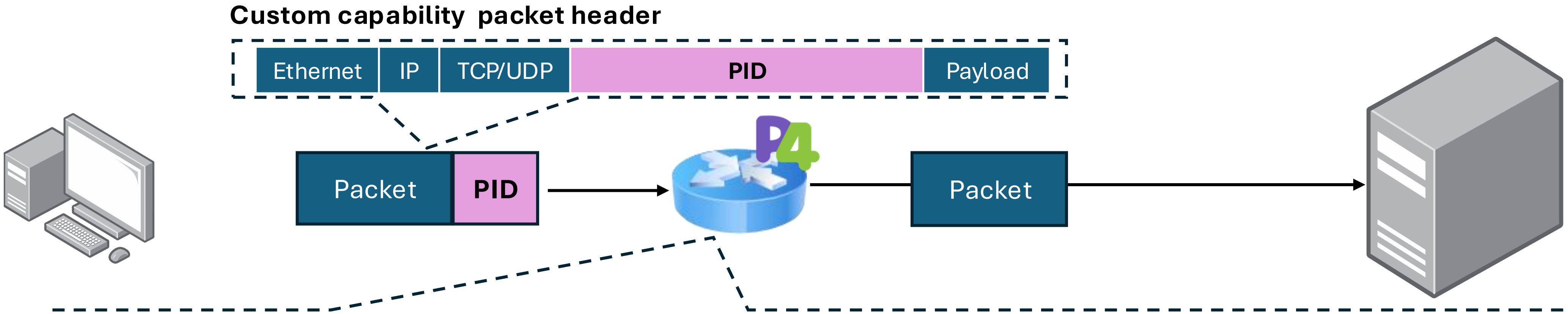
Contribution 2: **In-Network** Capability-Based Defense



Function 1:
Cache PID

Key (flow ID + PID)	Value (time)
Flow ID, PID=10	22:23
--	--

Contribution 2: **In-Network** Capability-Based Defense



Custom capability packet header

Ethernet | IP | TCP/UDP | **PID** | Payload

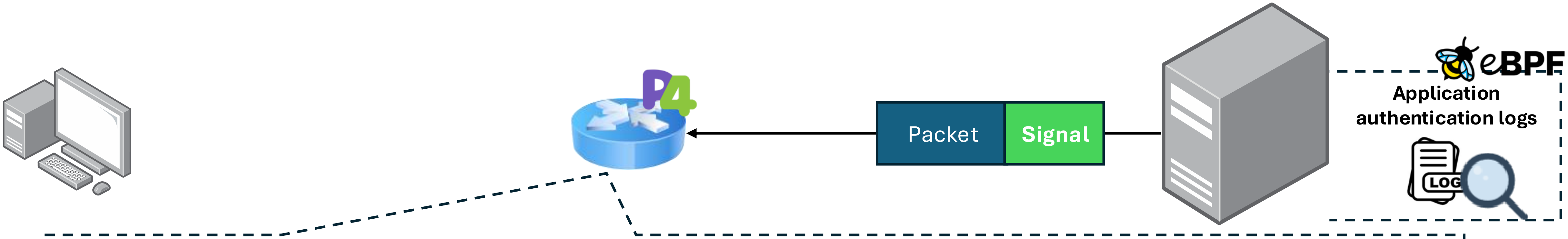
Packet | **PID**

Packet

Function 1:
Cache PID

Key (flow ID + PID)	**Value (time)**
Flow ID, PID=10	22:23
--	--

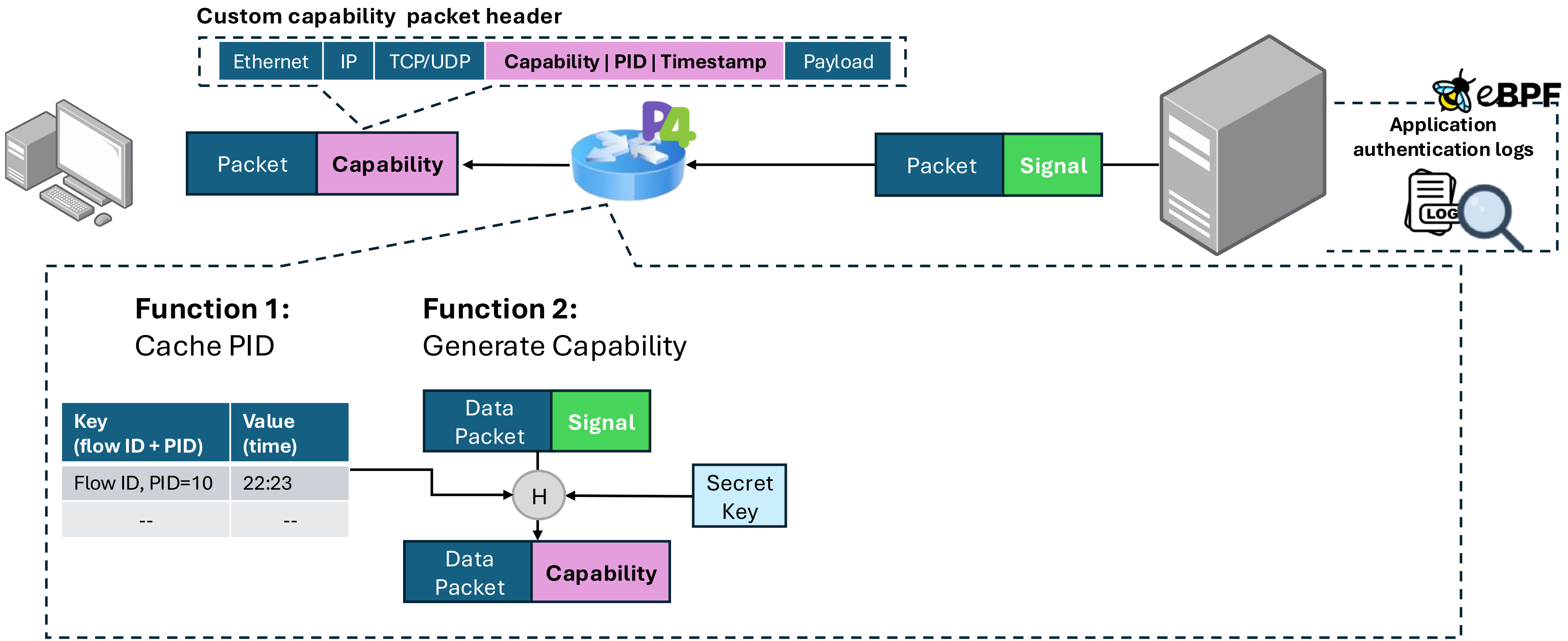
Contribution 2: **In-Network** Capability-Based Defense



Function 1:
Cache PID

Key (flow ID + PID)	Value (time)
Flow ID, PID=10	22:23
--	--

Contribution 2: **In-Network** Capability-Based Defense



Custom capability packet header

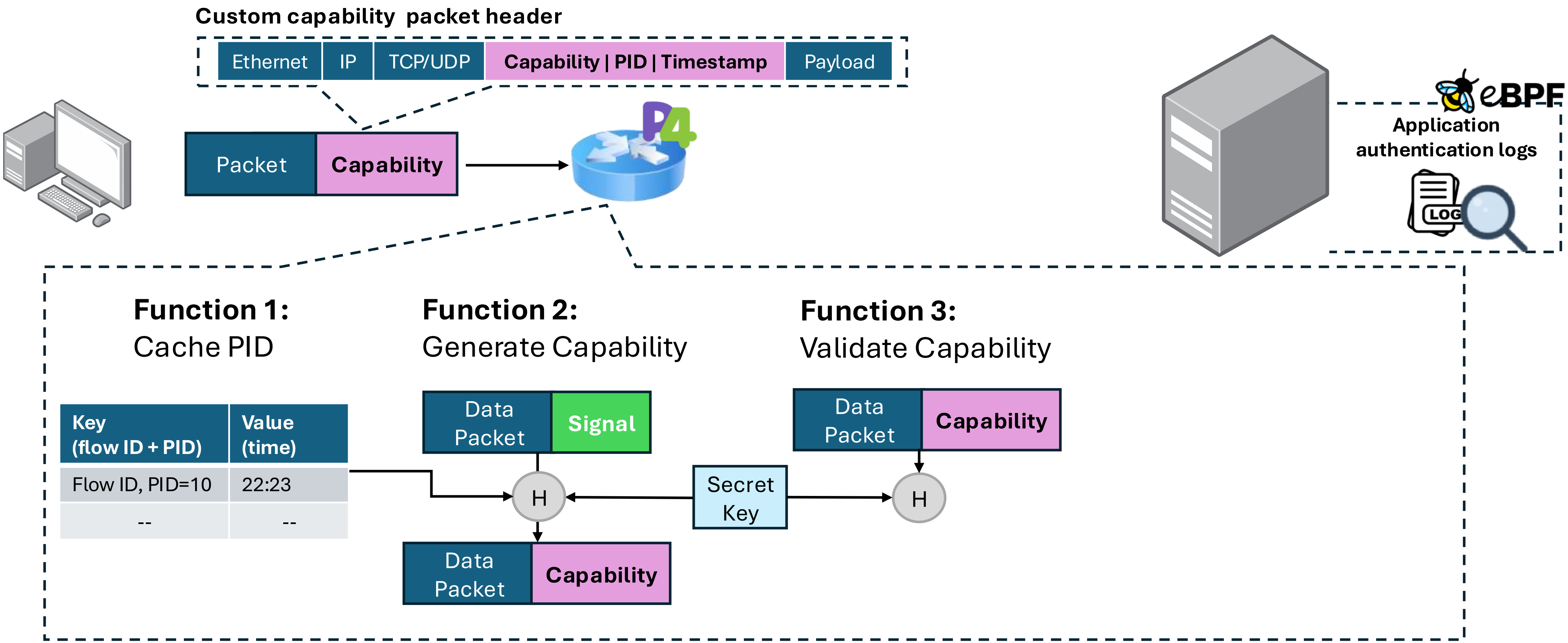
Ethernet | IP | TCP/UDP | **Capability | PID | Timestamp** | Payload

Function 1:
Cache PID

Key (flow ID + PID)	Value (time)
Flow ID, PID=10	22:23
--	--

Function 2:
Generate Capability

Contribution 2: **In-Network** Capability-Based Defense



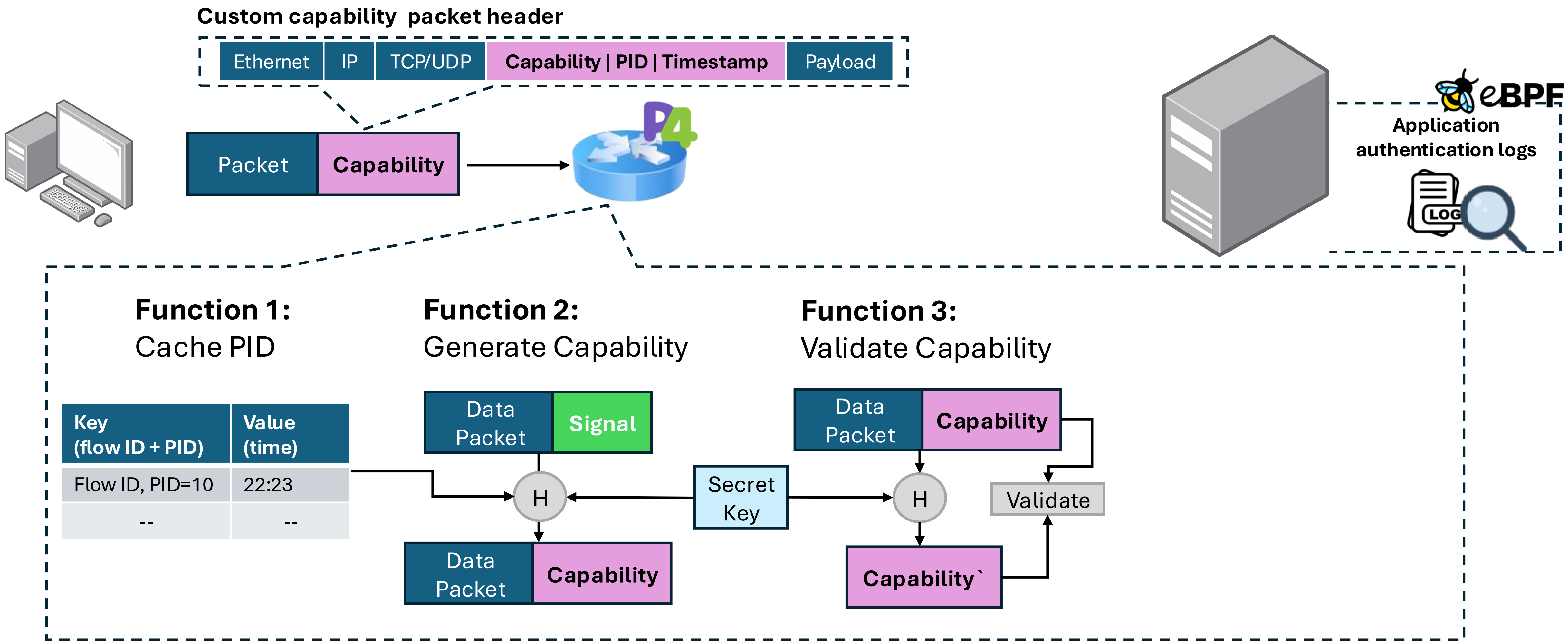
Function 1: Cache PID

Key (flow ID + PID)	Value (time)
Flow ID, PID=10	22:23
--	--

Function 2: Generate Capability

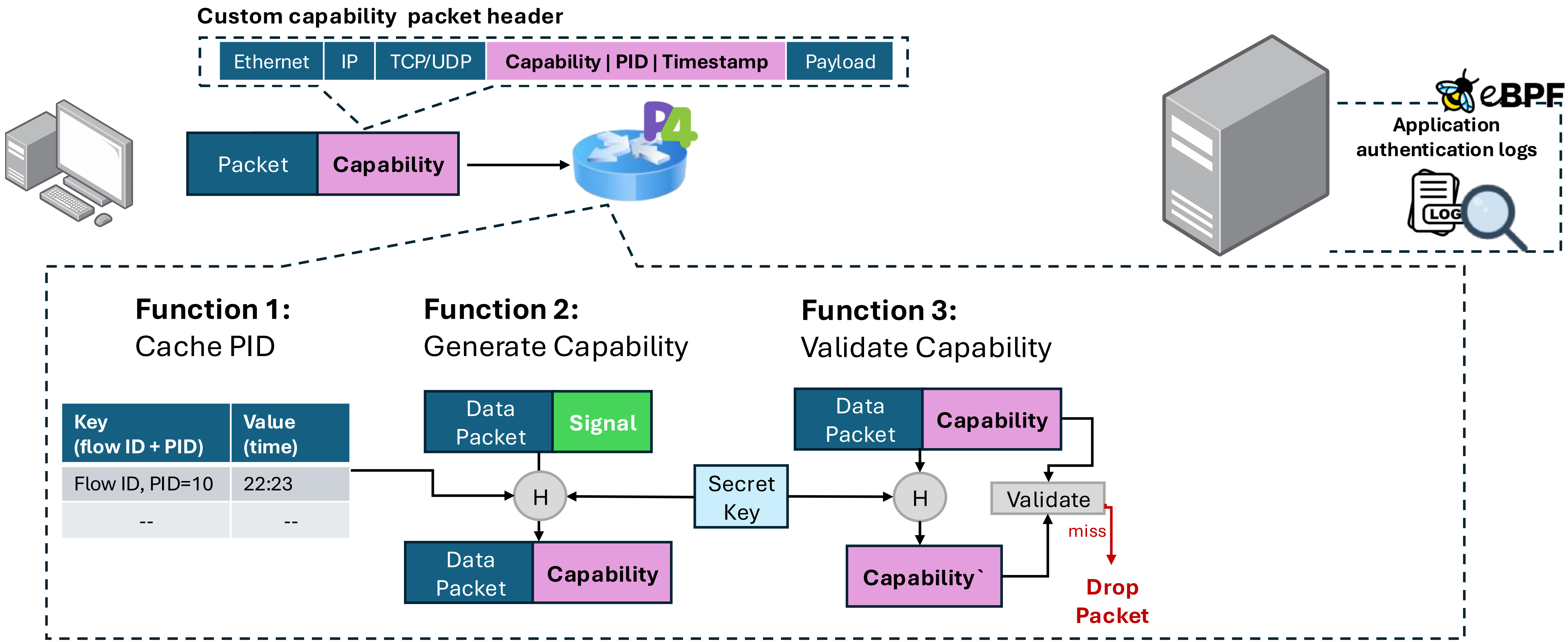
Function 3: Validate Capability

Contribution 2: **In-Network** Capability-Based Defense



Key (flow ID + PID)	Value (time)
Flow ID, PID=10	22:23
--	--

Contribution 2: **In-Network** Capability-Based Defense



Custom capability packet header

Ethernet | IP | TCP/UDP | **Capability | PID | Timestamp** | Payload

Packet **Capability**

Application authentication logs

Function 1:
Cache PID

Key (flow ID + PID)	**Value (time)**
Flow ID, PID=10	22:23
--	--

Function 2:
Generate Capability

Data Packet **Signal**

H

Data Packet **Capability**

Secret Key

Function 3:
Validate Capability

Data Packet **Capability**

H

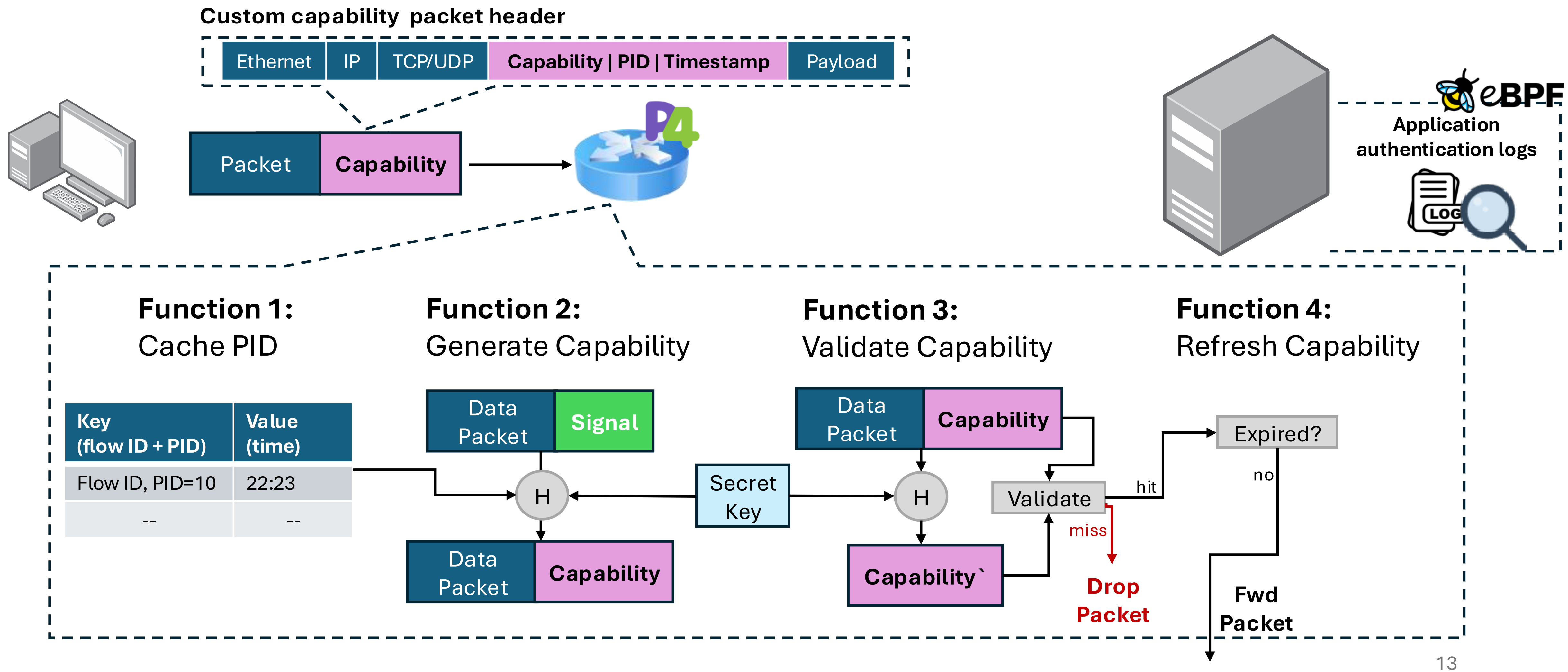
Capability`

Validate

miss

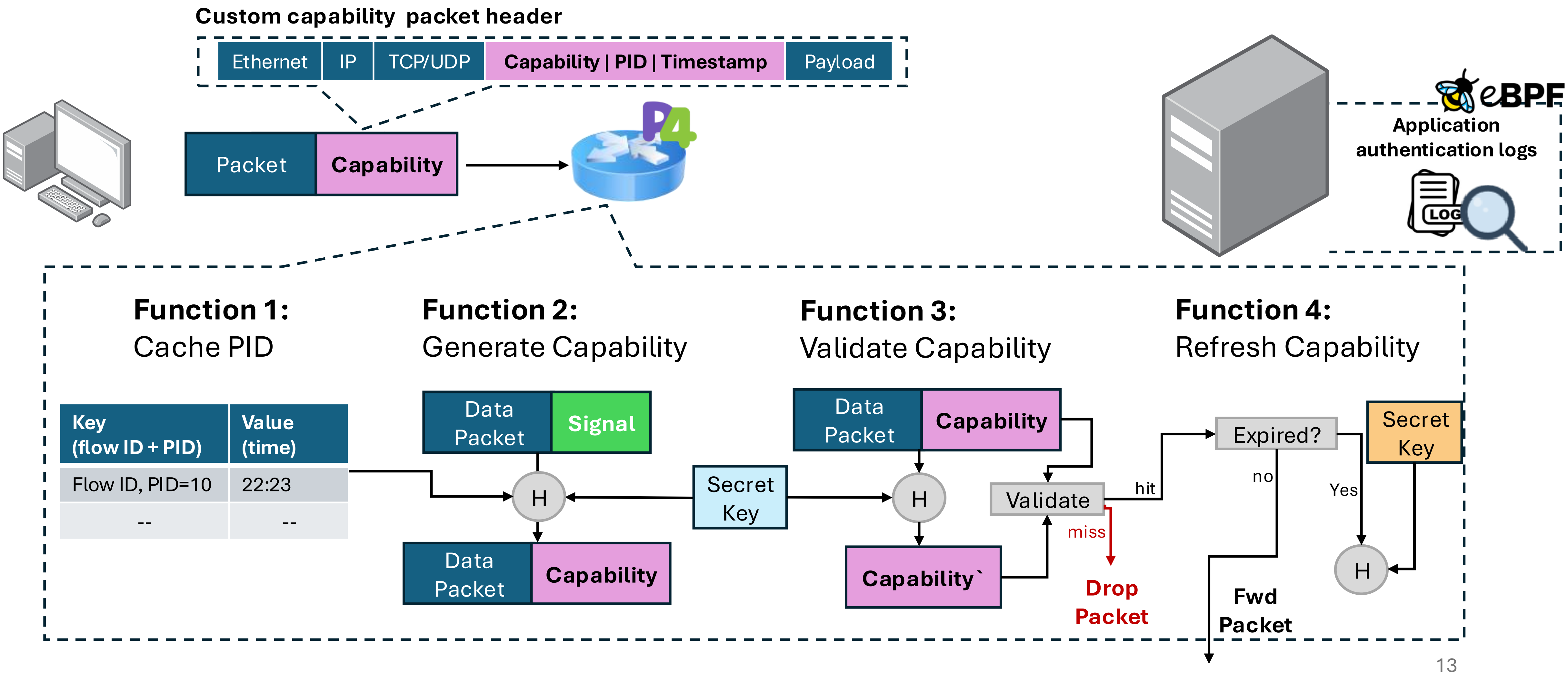
Drop Packet

Contribution 2: **In-Network** Capability-Based Defense



Key (flow ID + PID)	Value (time)
Flow ID, PID=10	22:23
--	--

Contribution 2: **In-Network** Capability-Based Defense



Custom capability packet header

Function 1: Cache PID

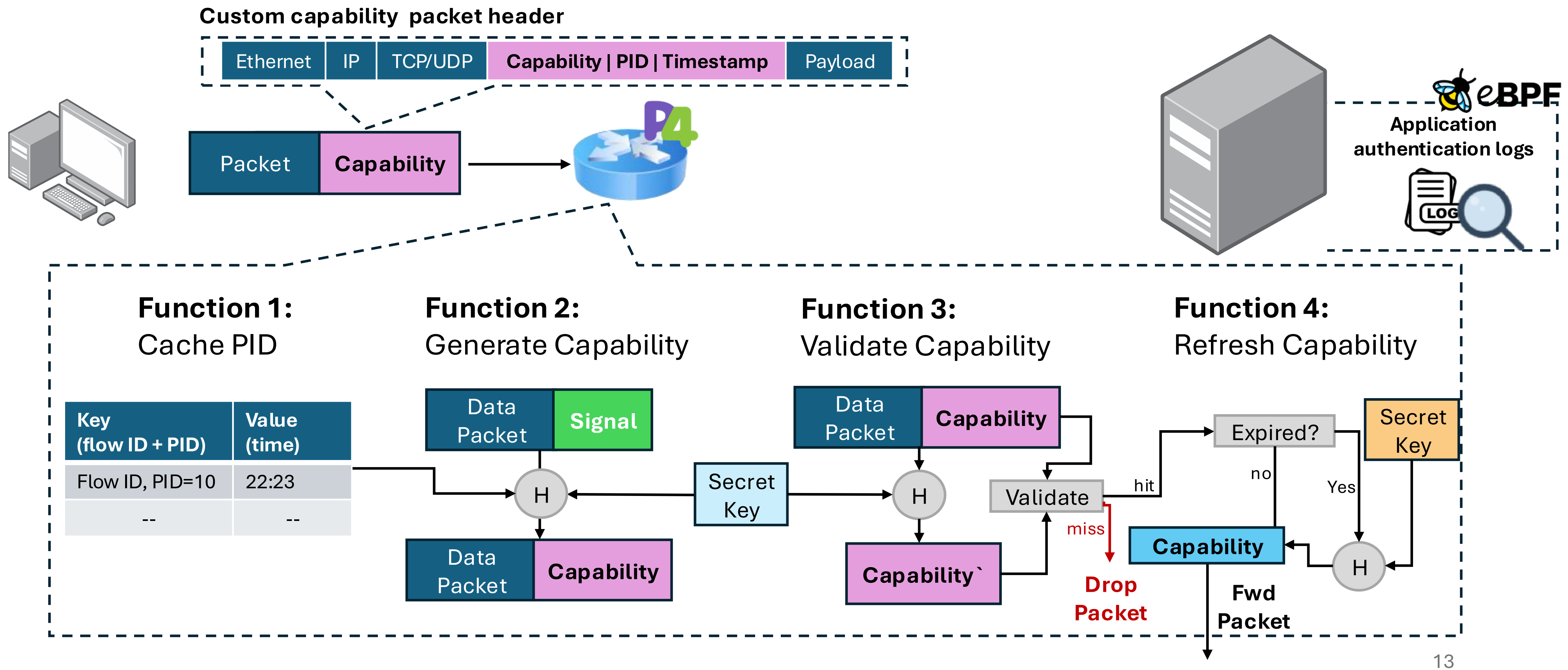
Key (flow ID + PID)	Value (time)
Flow ID, PID=10	22:23
--	--

Function 2: Generate Capability

Function 3: Validate Capability

Function 4: Refresh Capability

Contribution 2: **In-Network** Capability-Based Defense



Custom capability packet header

Ethernet | IP | TCP/UDP | **Capability | PID | Timestamp** | Payload

Packet | **Capability**

Application authentication logs

Function 1: Cache PID

Key (flow ID + PID)	**Value (time)**
Flow ID, PID=10	22:23
--	--

Function 2: Generate Capability

Function 3: Validate Capability

Function 4: Refresh Capability

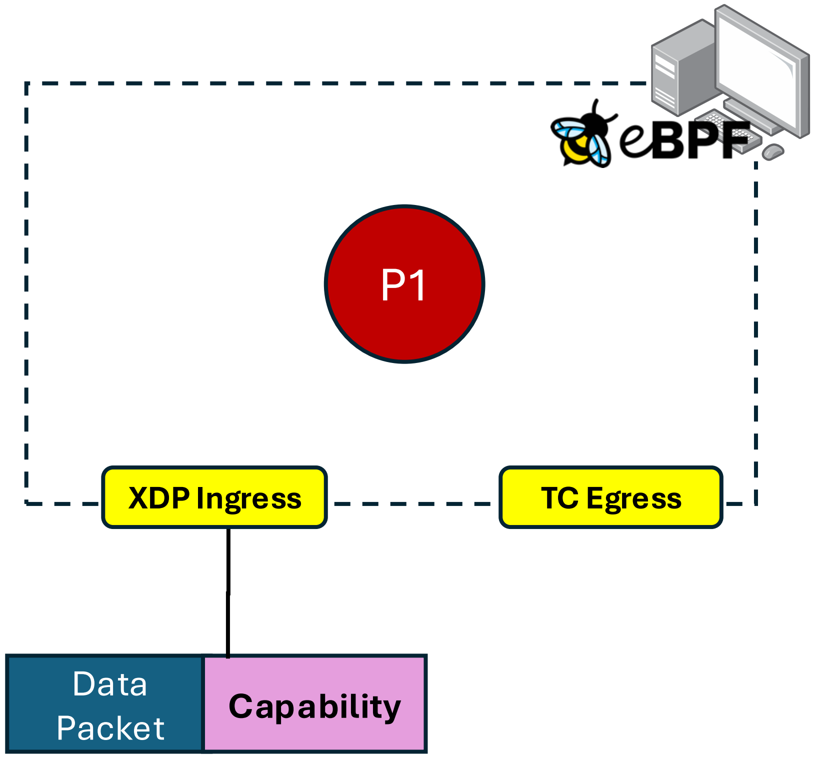
Challenge 3: Host-Level Fine-Grained Capability Management

- Need a mechanism to **manage capabilities** within hosts
  - Optimized storage, where each process can have multiple capabilities

- Need to be **transparent** to running applications and protocols
  - Supporting complex applications without modifying existing source code

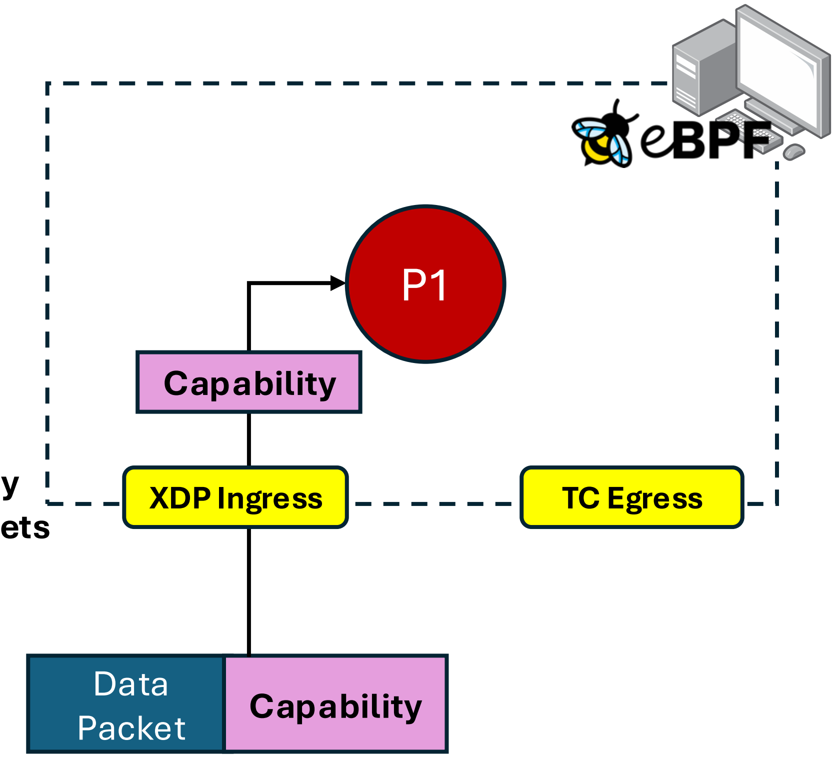
- Attach capabilities to network packets
  - Little network **performance overhead**

Contribution 3: **Lightweight** eBPF Programs



Incoming packet

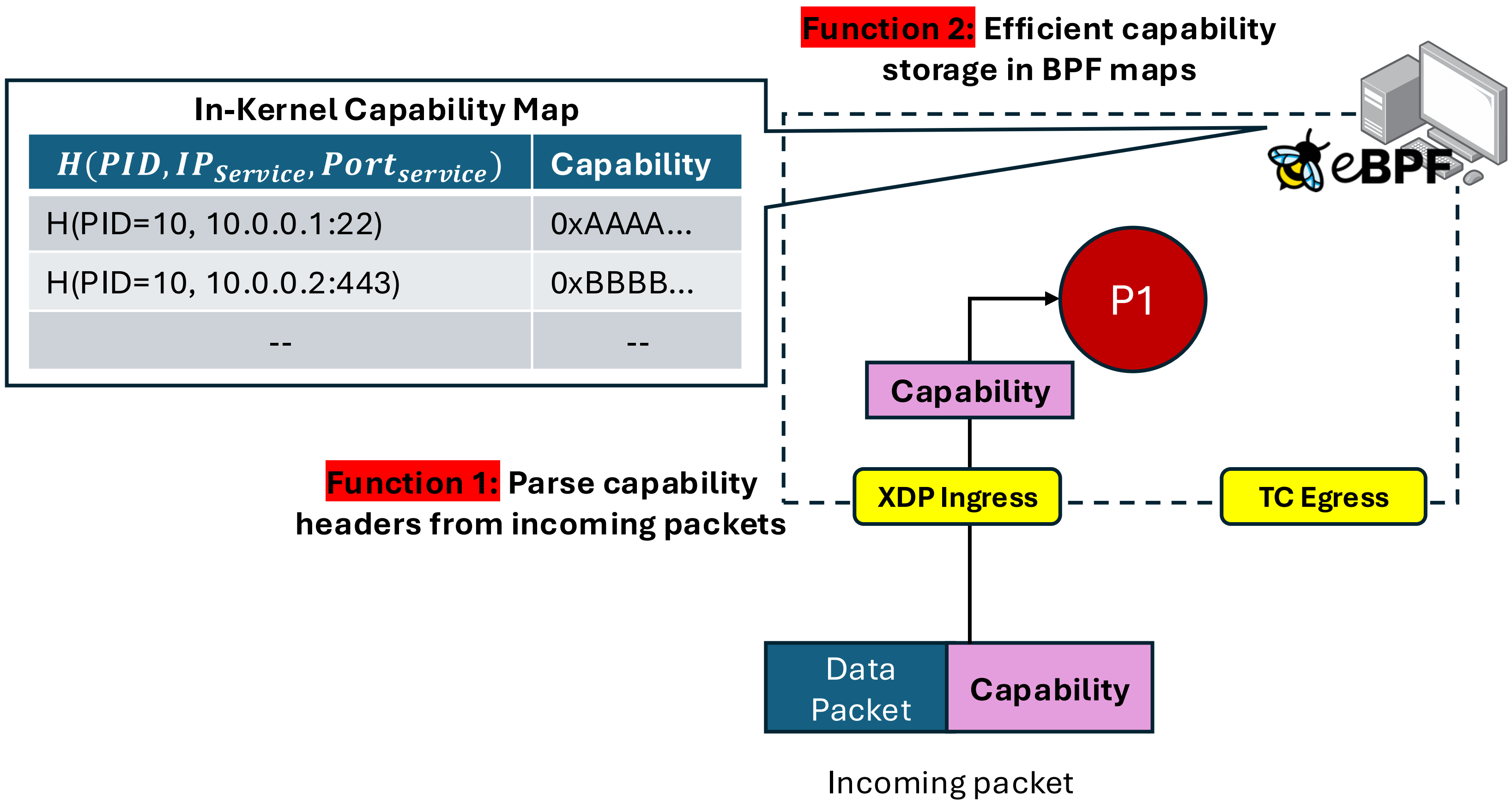
Contribution 3: **Lightweight** eBPF Programs



Function 1: **Parse capability headers from incoming packets**

Incoming packet

Contribution 3: **Lightweight** eBPF Programs



Function 2: Efficient capability storage in BPF maps

In-Kernel Capability Map

$H(PID, IP_{Service}, Port_{service})$	Capability
H(PID=10, 10.0.0.1:22)	0xAAAA...
H(PID=10, 10.0.0.2:443)	0xBBBB...
--	--

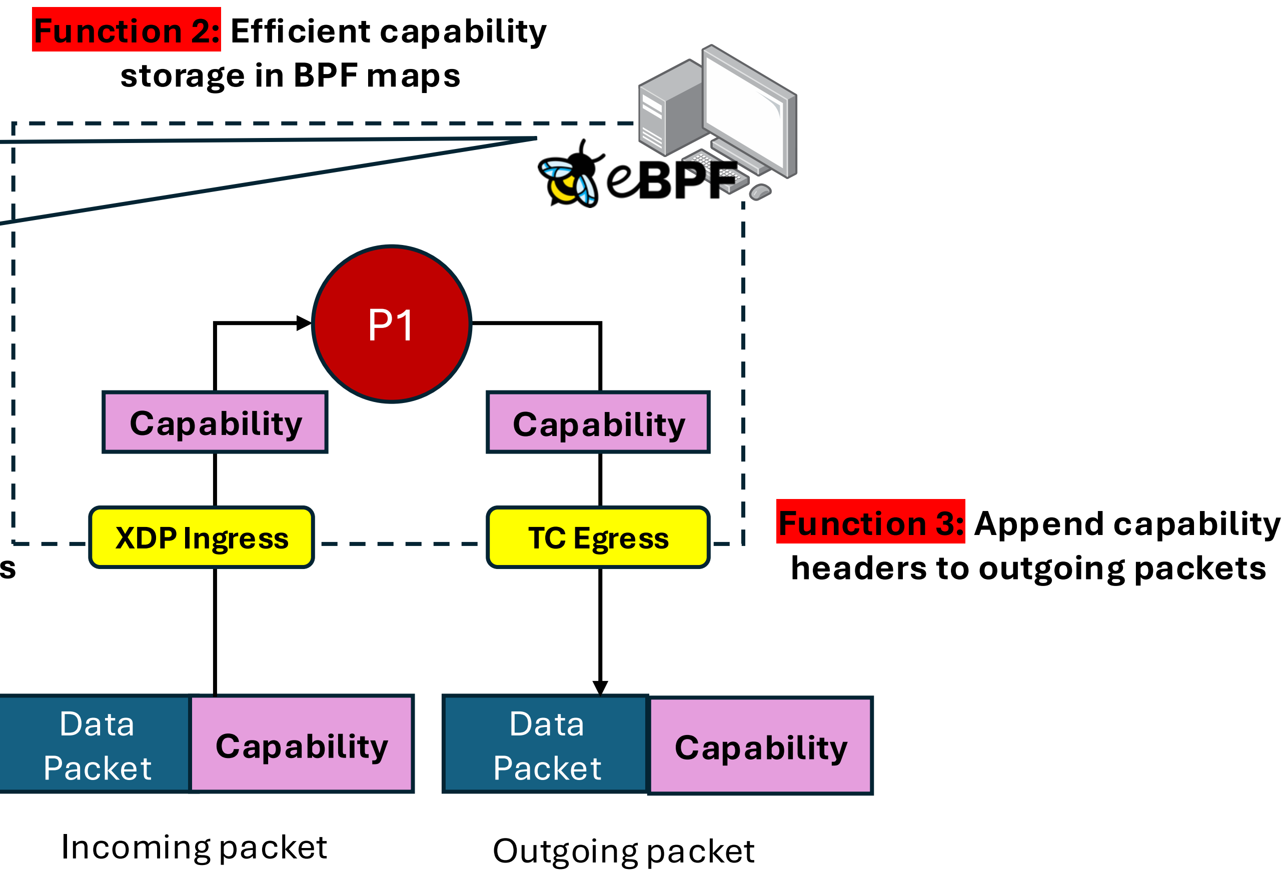
Function 1: Parse capability headers from incoming packets

Incoming packet

Contribution 3: **Lightweight** eBPF Programs

In-Kernel Capability Map

$H(PID, IP_{Service}, Port_{service})$	Capability
H(PID=10, 10.0.0.1:22)	0xAAAA…
H(PID=10, 10.0.0.2:443)	0xBBBB…
--	--

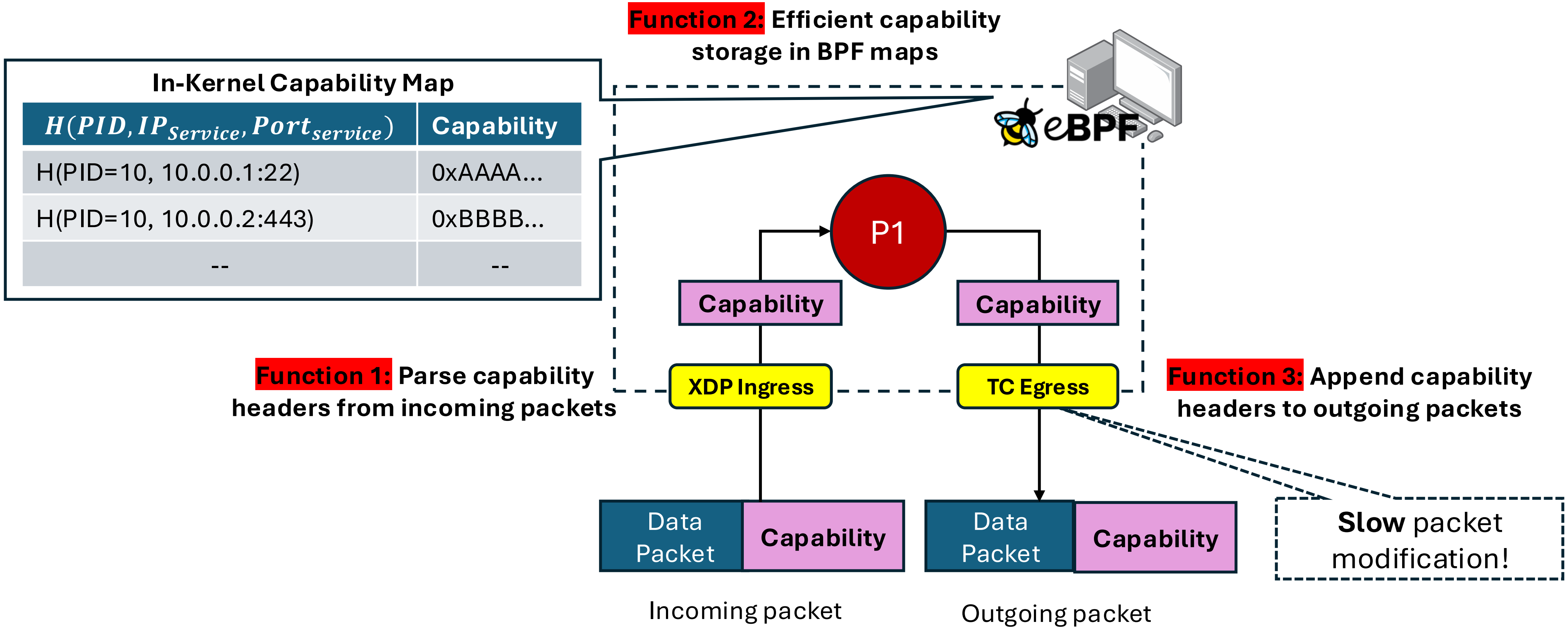


Function 2: Efficient capability storage in BPF maps

Function 1: Parse capability headers from incoming packets

Function 3: Append capability headers to outgoing packets

Contribution 3: **Lightweight** eBPF Programs



In-Kernel Capability Map

$H(PID, IP_{Service}, Port_{service})$	Capability
H(PID=10, 10.0.0.1:22)	0xAAAA…
H(PID=10, 10.0.0.2:443)	0xBBBB…
--	--

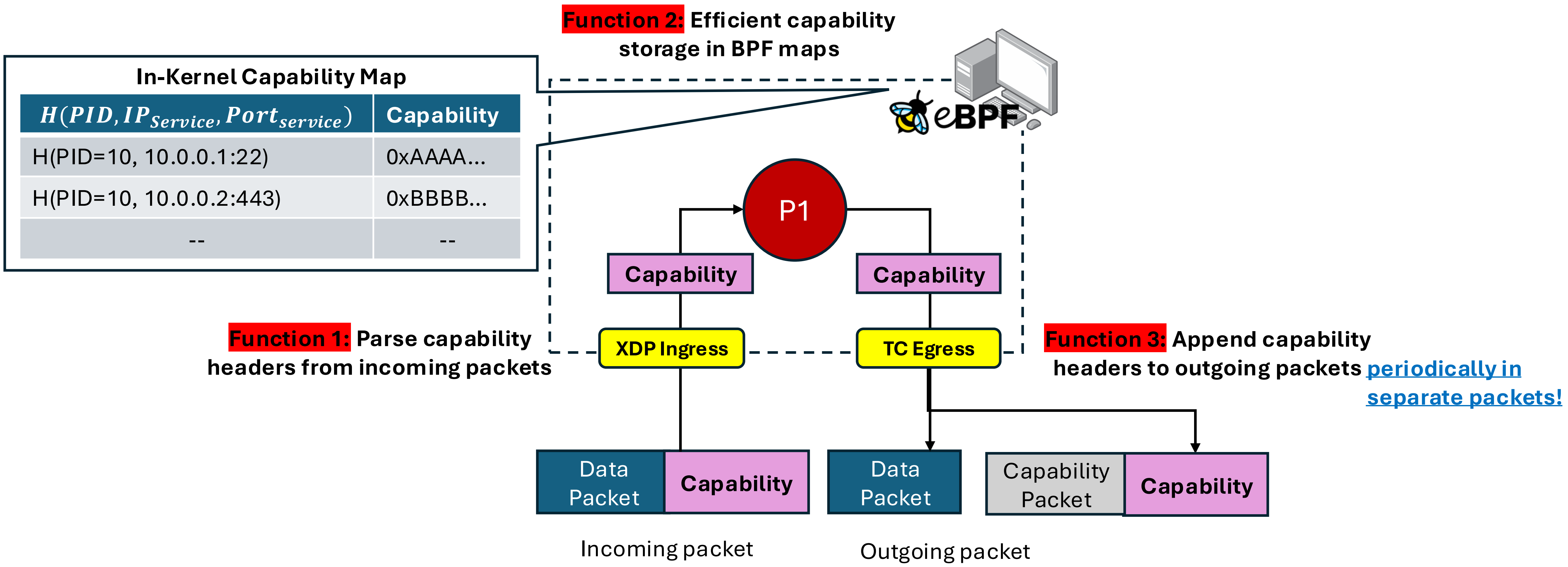
Function 2: Efficient capability storage in BPF maps

Function 1: Parse capability headers from incoming packets

Function 3: Append capability headers to outgoing packets

Slow packet modification!

Contribution 3: **Lightweight** eBPF Programs



Function 2: **Efficient capability storage in BPF maps**

In-Kernel Capability Map

$H(PID, IP_{Service}, Port_{service})$	Capability
H(PID=10, 10.0.0.1:22)	0xAAAA...
H(PID=10, 10.0.0.2:443)	0xBBBB...
--	--

Function 1: **Parse capability headers from incoming packets**

Function 3: **Append capability headers to outgoing packets periodically in separate packets!**

Evaluations

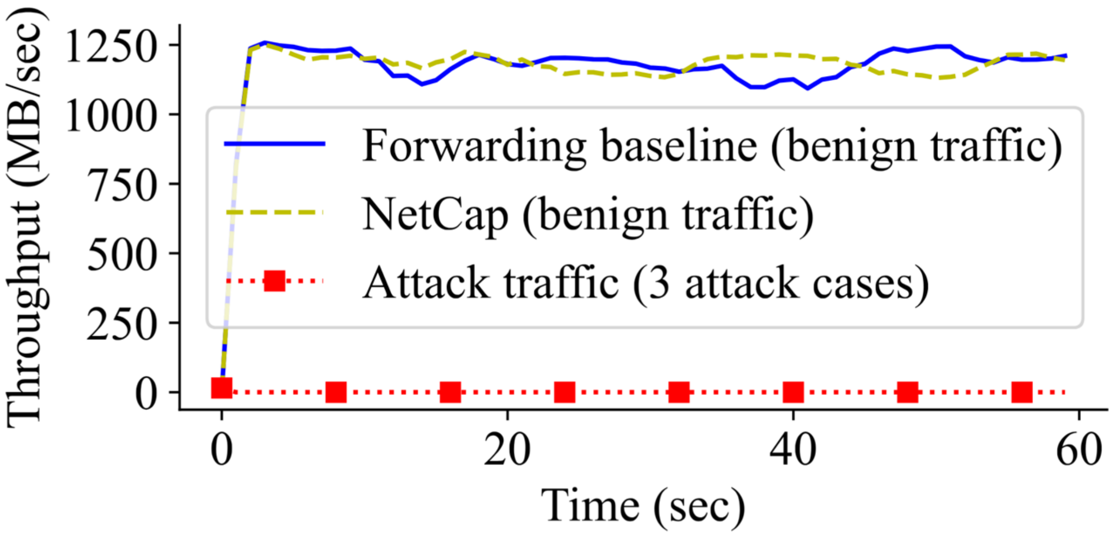
- **Prototype**
  - ~2,500 LoCs of P4, C and Python code.
  - Includes switch program, eBPF programs, and control plane functions
- **Testbed**
  - Physical Tofino P4 Switch with 32x100 Gbps ports
  - Real-world trace datasets (LANL Unified Host and Network dataset, DARPA OpTC dataset)
  - Realistic internal and external sophisticated attack implementations leveraging different protocols and real-world applications (web applications, Kerberos)
  - Real-world complex applications (Apache, FTP, Node.js)
- **Key evaluation aspects**

Authentication scheme effectiveness

Impact on network and host performance

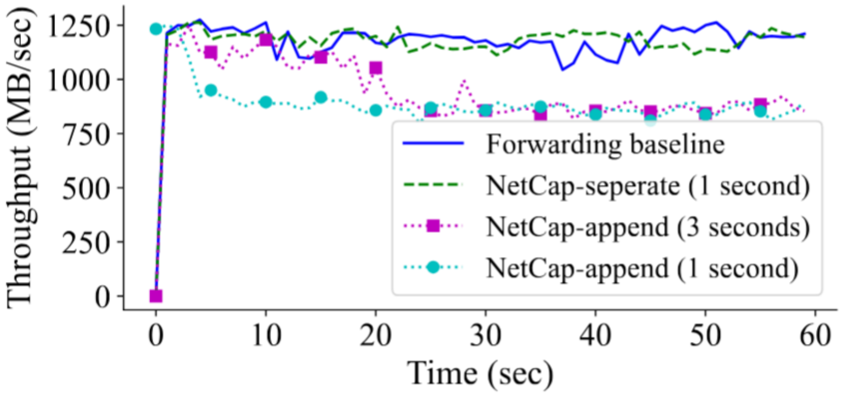
Support complex real-world applications

Authentication Scheme Effectiveness



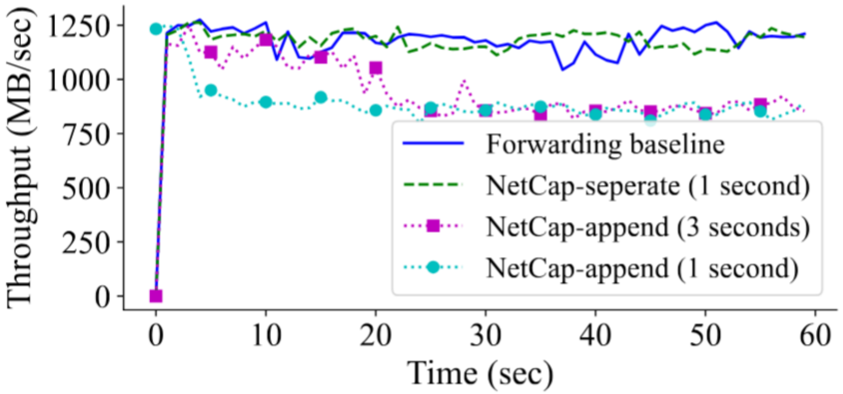
- NetCap successfully **blocks all attack traffic** across multiple attack scenarios
- NetCap preserves line-rate throughput for benign traffic, matching baseline forwarding performance

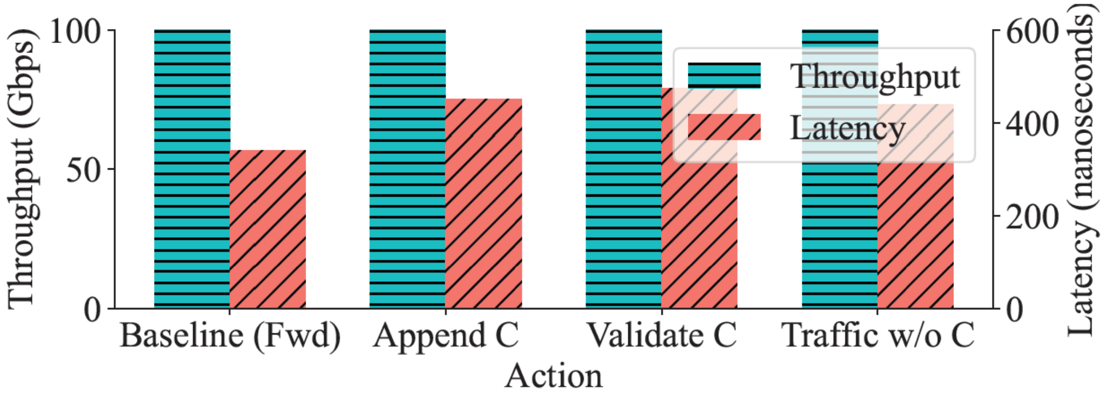
System Impact and Overhead



- NetCap's approach of **sending capabilities separately** maintains similar throughput to the forwarding baseline

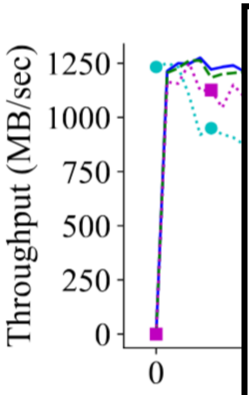
System Impact and Overhead





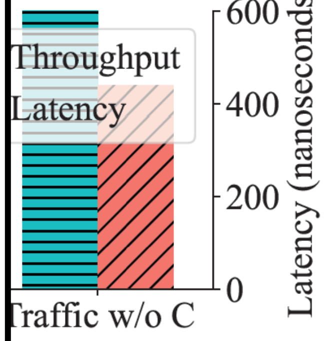
- NetCap's approach of **sending capabilities separately** maintains similar throughput to the forwarding baseline
- Across all in-network operations, NetCap achieves **99.9 Gbps** throughput, similar to the forwarding baseline with a **negligible ~130 ns** latency

System Impact and Overhead



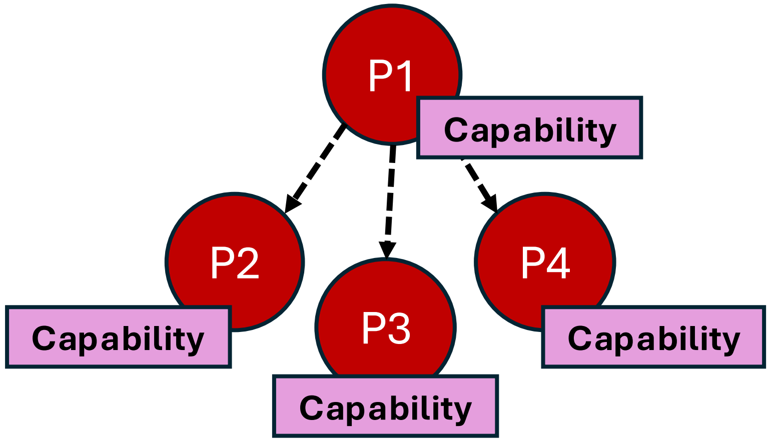
More evaluations in the paper:

RQ1: Authentication scheme effectiveness
RQ2: Scalability in real-world scenarios
RQ3: Impact on network and host performance
RQ4: Supporting complex applications
RQ5: Comparison with server-based design

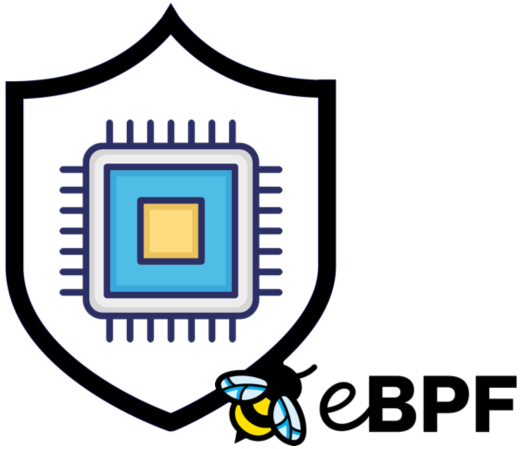


- NetCap's approach of **sending capabilities separately** maintains similar throughput to the forwarding baseline
- Across all in-network operations, NetCap achieves **99.9 Gbps** throughput, similar to the forwarding baseline with a **negligible ~130 ns** latency

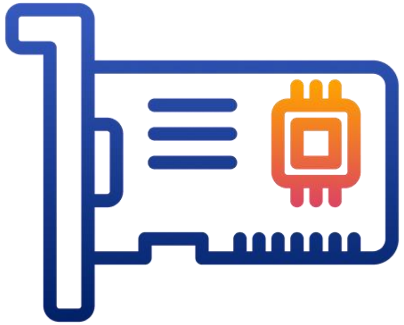
Future Research Directions



Support **dynamic capability** delegation and revocation across processes in different hosts



Extend capability enforcement to **restrict memory access** within the OS without kernel modifications using eBPF



Offload our eBPF programs to SmartNICS for faster packet modifications

Key Takeaways

- **NetCap** is an in-network, fine-grained, capability-based defense that validates network access requests at the process level

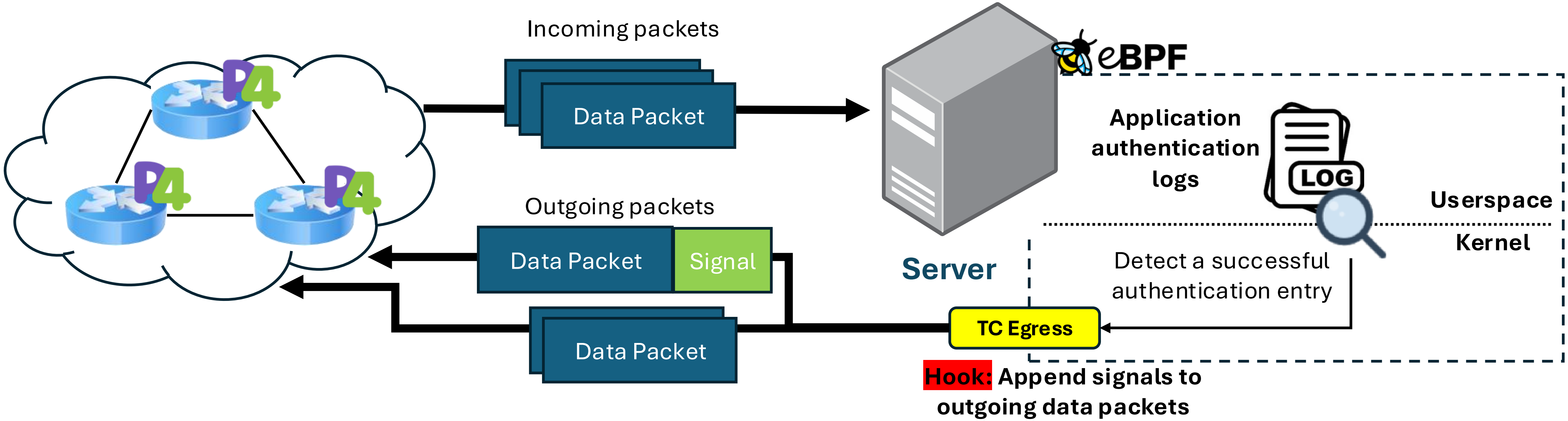
- First work to realize capabilities within the data plane
  - In-network capability-based defense to prevent exploiting stolen access tokens
  - Supports a wide range of protocols
  - Seamless deployment without modifications to the underlying protocols, applications, or kernel source code
  - Facilitates the realization of **Zero Trust** through continuous authentication

Thank You!

VIRGINIA TECH

Backup

Lightweight Server-Side Programs


Incoming packets
Data Packet
Outgoing packets
Data Packet
Signal
Data Packet
Server
eBPF
Application authentication logs
LOG
Userspace
Kernel
Detect a successful authentication entry
TC Egress
Hook: Append signals to outgoing data packets

- **Problem:** Modifying server-side applications to send signals upon successful authentication is not optimal
- **Solution: Employ a *split-design* of userspace and eBPF programs to monitor application authentication logs**
  - The **userspace program** monitors new updates to authentication logs
  - The **eBPF program** appends a successful authentication signal to a data packet from the same connection