

DNS Data Exfiltration Prevention: Kernel-Enforced Endpoint Security

*Scalable Framework to Disrupt
DNS C2 and Tunneling*

Vedang Parasnis

University of Washington Bothell








What is Data Exfiltration

[[Singamaneni et al.](#)]

Definition: Unauthorized extraction or transmission of sensitive data from a system

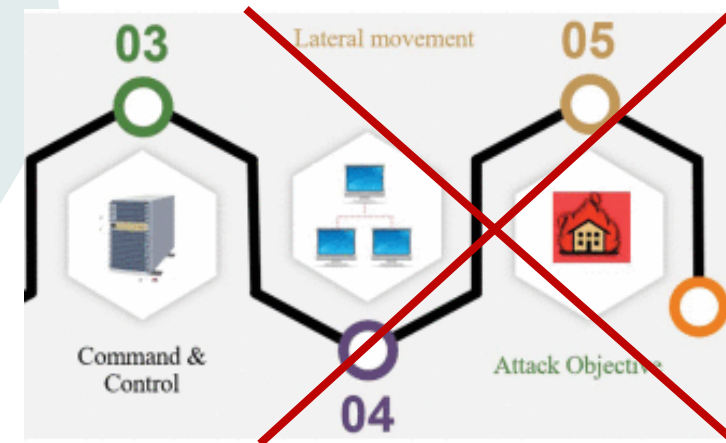
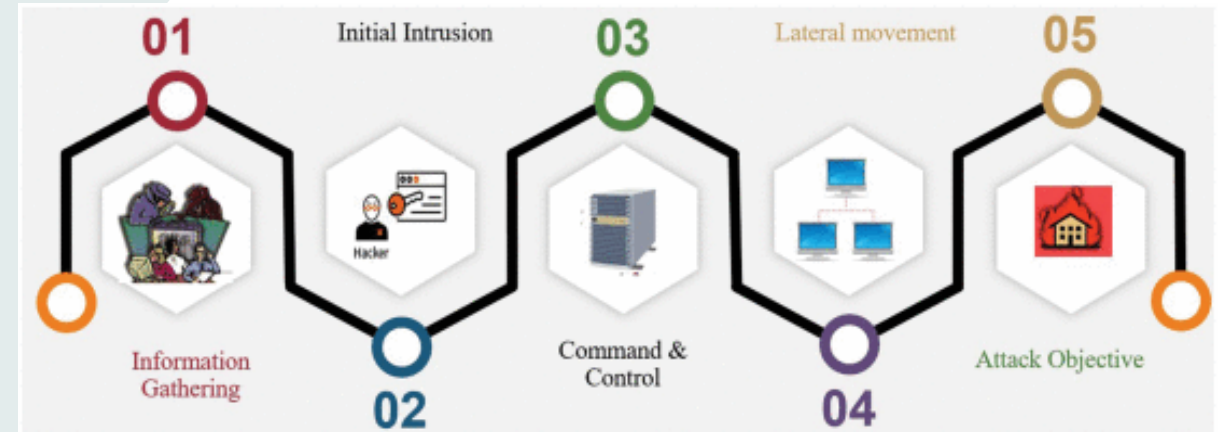
Impact: Reputation, Financial Losses to Enterprises

Attack Lifecycle

-  Information Reconnaissance
-  Initial Intrusion / Infiltration
-  **Command and Control**
-  Lateral Movement
-  Command Execution and Data Breaches

Core Defense Strategy

-  Endpoint Security (EDR / XDR)

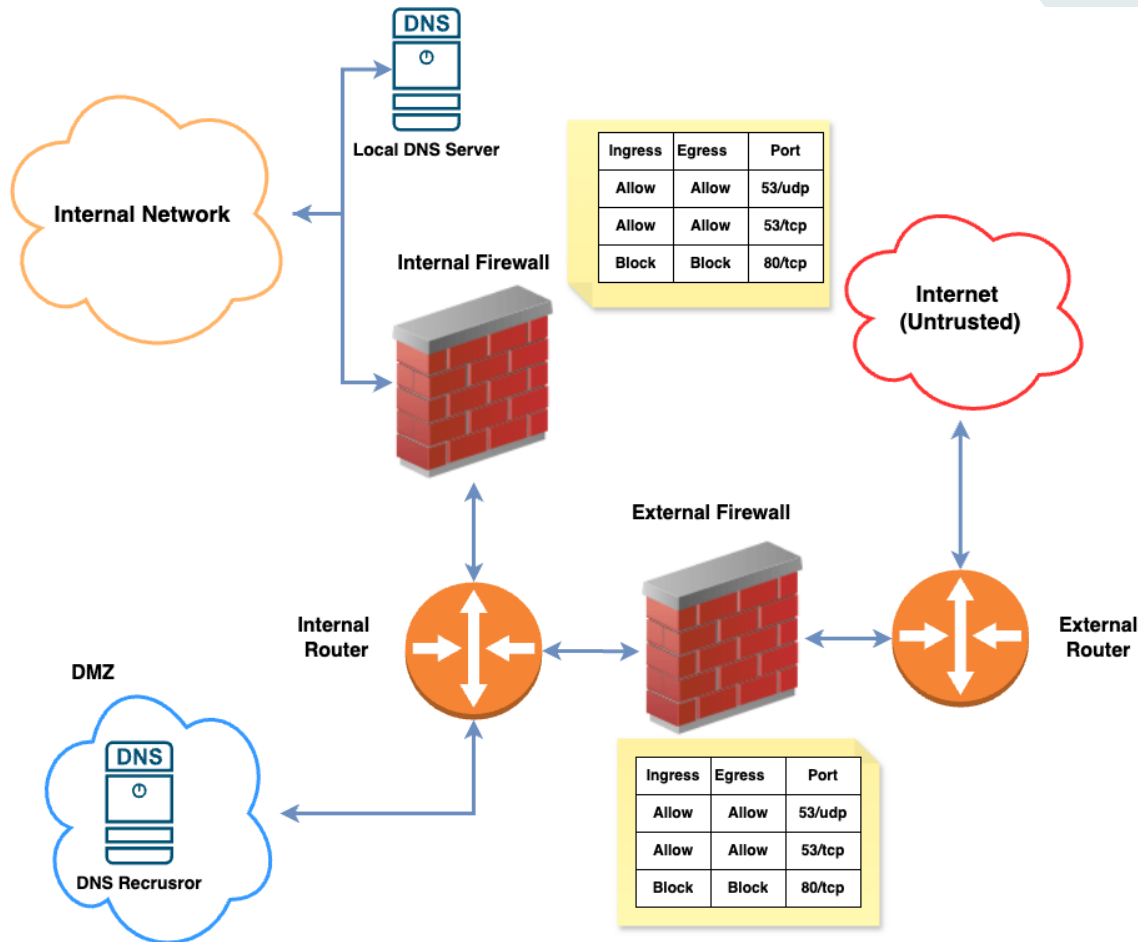


Why DNS is a Blind Spot

Unencrypted by default – Allows attackers to hide malicious payloads in plain sight.

Rarely monitored deeply – DNS logs are often ignored, giving adversaries a free channel.

Firewall blindspot – DNS ports (53 UDP/TCP) stay open, bypassing most traditional defenses.

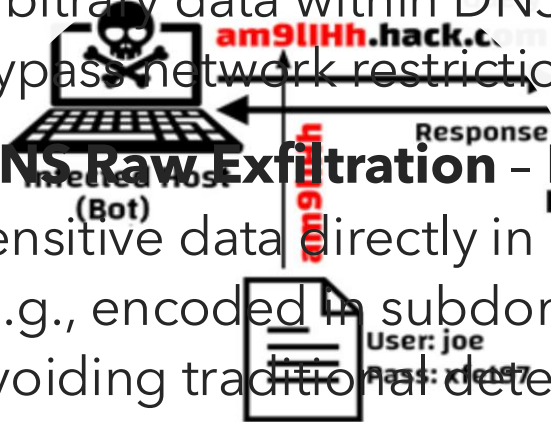


DNS Data Exfiltration

DNS C2 - Uses DNS queries and responses to maintain covert communication with attacker infrastructure.

DNS Tunneling - Encapsulates arbitrary data within DNS packets to bypass network restrictions.

DNS Raw Exfiltration - leaks sensitive data directly in DNS queries (e.g., encoded in subdomains), avoiding traditional detection.



Malware sends username and password data encoded in base64 as hostname label

Remote Code Execution (RCE)

- Shell code exploits
- Script executions, File corruptions
- Process Side channeling exploits
- Example: **Sliver C2, Hexane, APT29 (Cozy Bear)**.

Persistent Backdoors

- Deployment rootkits, ransomwares
- Example: **Turla group**

Network Pivoting (Port Forwarding)

- Compromised machines act as proxies to reach deeper into private infrastructure
- Example: **Cobalt Strike, Hexane, DNSSystem**

Existing Approaches

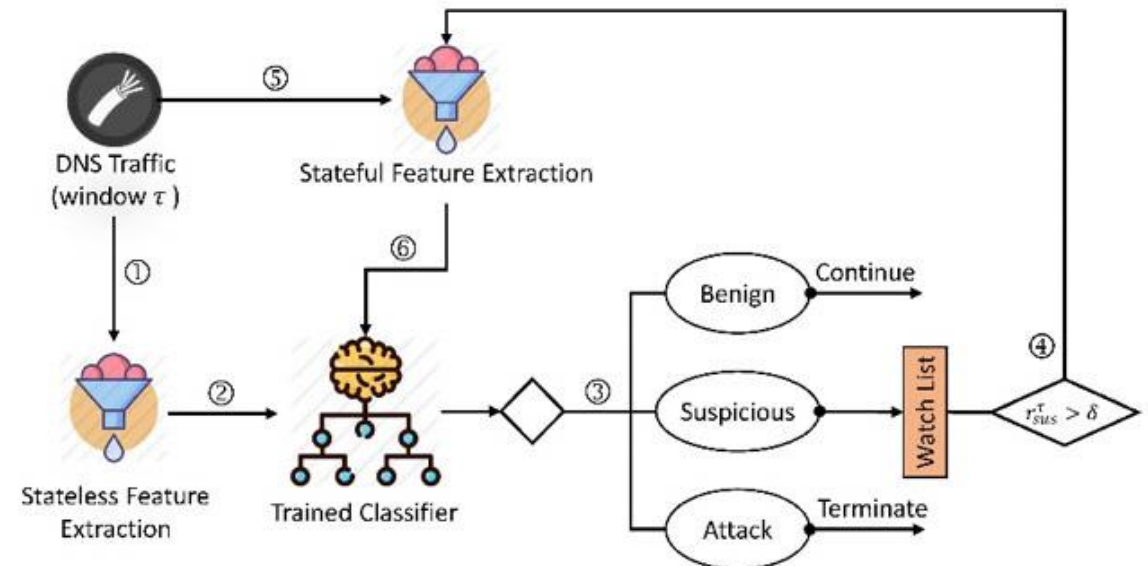
- **Active Analysis**
 - **DNS Exfiltration Security as Middleware**
 - Palo Alto Precision Guard AI Security
 - Infoblox DNS exfiltration security
- **Passive Analysis**
 - Anomaly Detection [[Bilge et al.](#)]
 - Threat Signatures, Domain Reputation scoring [[Antonakakis et al.](#)]

Existing Approaches – Passive Analysis

- **Anomaly Detection:**
 - **Traffic Behavior Analysis**
 - DNS Passive Traffic Volume Analysis
 - DNS Passive Traffic timing Statistical Analysis
 - **Machine Learning-based Threat Intelligence**
 - Uses machine learning models to identify traffic anomalies.
- **Threat Signatures:**
 - DNS Domain Scoring
 - Malicious domain signature

Stateless Features – Lexical Analysis

Stateful Features – Statistical Analysis



[[Samaneh et al.](#), [Jawad et al.](#)]

Issues with current approaches

- **Slow Detection → High Dwell Time → More Damage**
- **Extremely slow to Advanced C2 Attacks**
- **Dynamic Threat Patterns:**
 - Varying Throughput
 - Slow and Stealthy Rate
 - Kernel Encapsulated Traffic
 - Port Obfuscation
- **Centralized monitoring and analysis systems don't scale**
- **Ineffective over IP Masquerading & Domain Generation Algorithms**

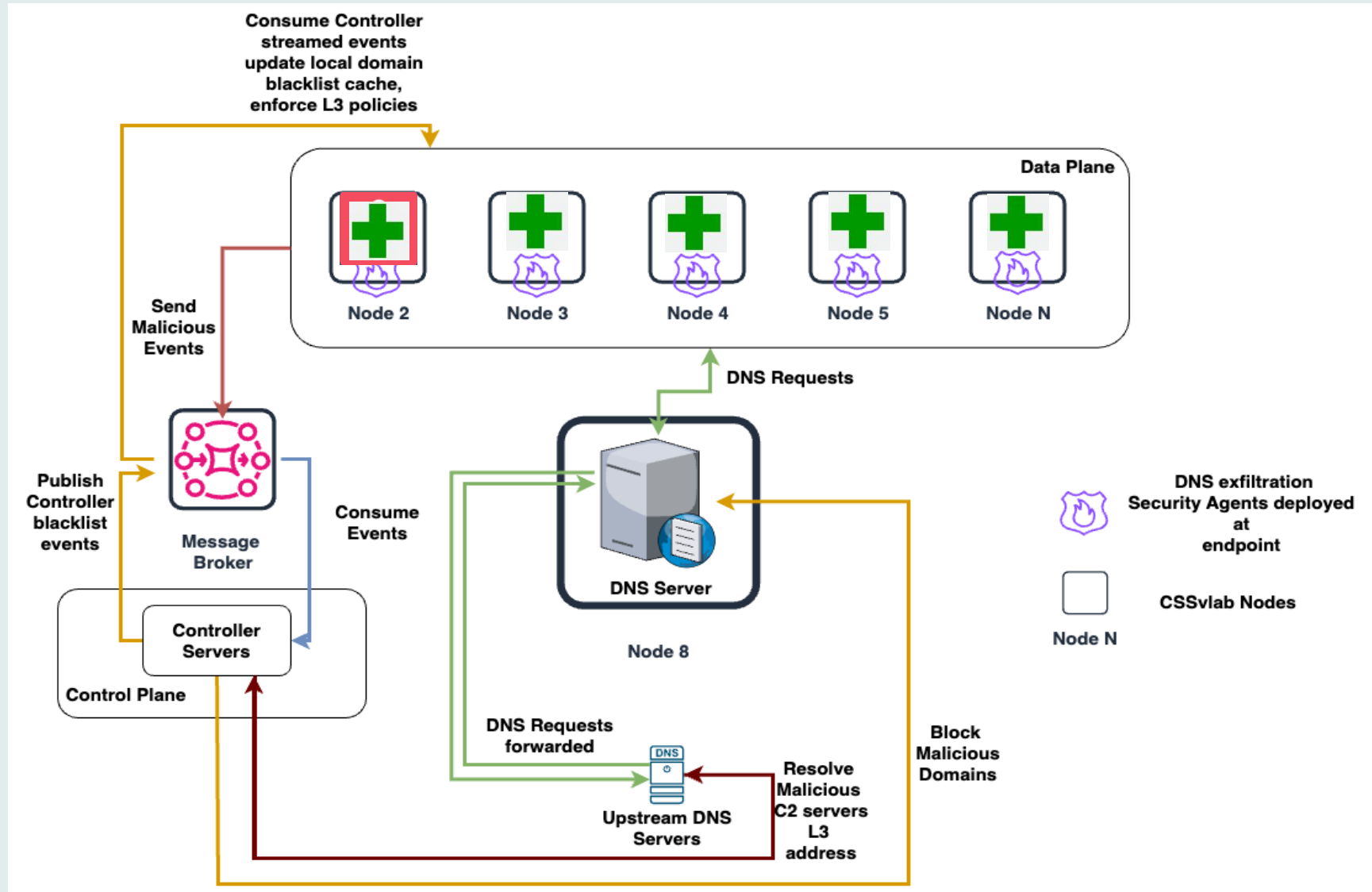
Solution:

Real-time, proactive enforcement at Ring 0 — inside the kernel, where no userland evasion can hide.

Security Framework Architecture

Architecture Components

- **Data Plane**
 - eBPF endpoint agents
- **Control Plane**
 - Controller Servers
- **Infrastructure**
 - DNS Server
 - Apache Kafka



Security Framework Goals

Real-Time DNS Exfiltration Prevention

Implement in-kernel deep packet inspection and enforcement to block all forms of DNS exfiltration channels.

AI-Assisted Threat Detection

Use deep learning in userspace to detect advanced obfuscated exfiltration payloads with high accuracy aiding kernel enforcements.

Dynamic Cross-Layer Policy Enforcement

Enforce in-kernel L3 network policies adaptively and domain blacklisting on DNS server to combat DGA.

Malicious Process Aware Active Response

Instantly detect and kill implants, preventing lateral movement and further damage.

Scalable Multi-Cloud Deployment

Ensure framework's horizontal scales for real-world production cloud environments.

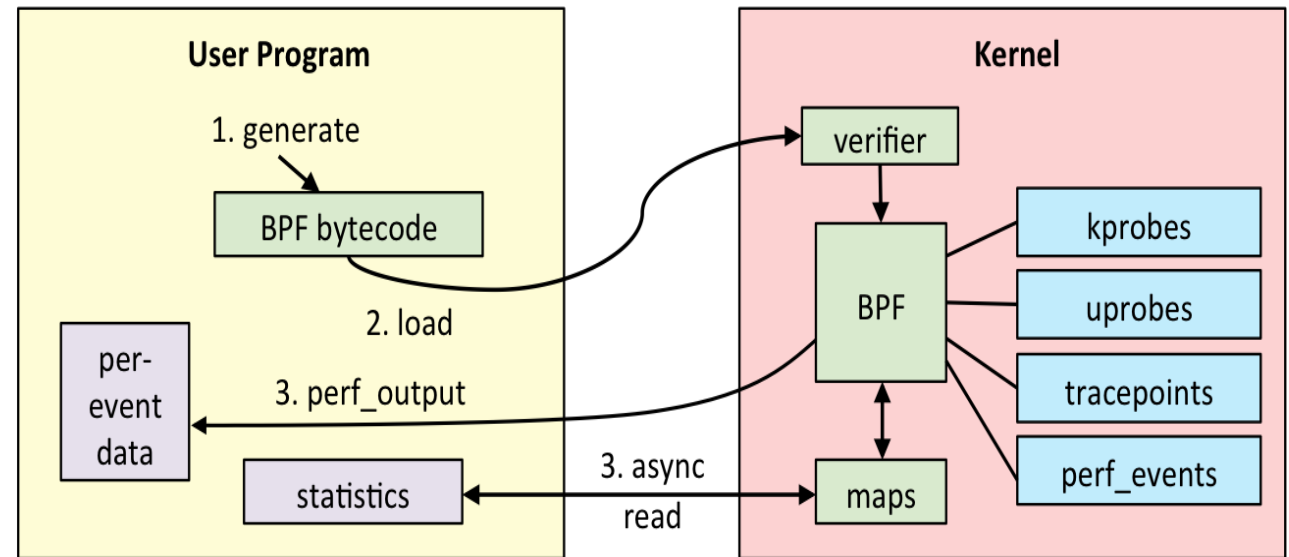
Broader Impact and Applicability

- **Cloud Providers & HyperScalers**
 - Strengthens DNS-layer security in managed services.
 - Examples: AWS Route 53, Google Cloud DNS, Azure DNS.
- **National Security & Defense**
 - Disrupts advanced malware APT groups alive using DNS-based C2 channels.
 - Examples: Turla Venom, Skitnet, Lazarus, OilRig, Hexane.
- **Regulated Enterprises (Finance, Healthcare)**
 - Augments DLP capabilities over DNS for private cloud and on-premise environments.
 - Examples: Financial institutions, healthcare networks.
- **Security Vendors (EDR/XDR/DNS Security)**
 - Integrates as a modular addon to extend EDR/XDR threat prevention at the DNS level.
 - Examples: CrowdStrike Falcon, Cisco HyperShield, Palo Alto Precision AI, Broadcom Carbon Black.

eBPF – Extended Berkley Packet Filter

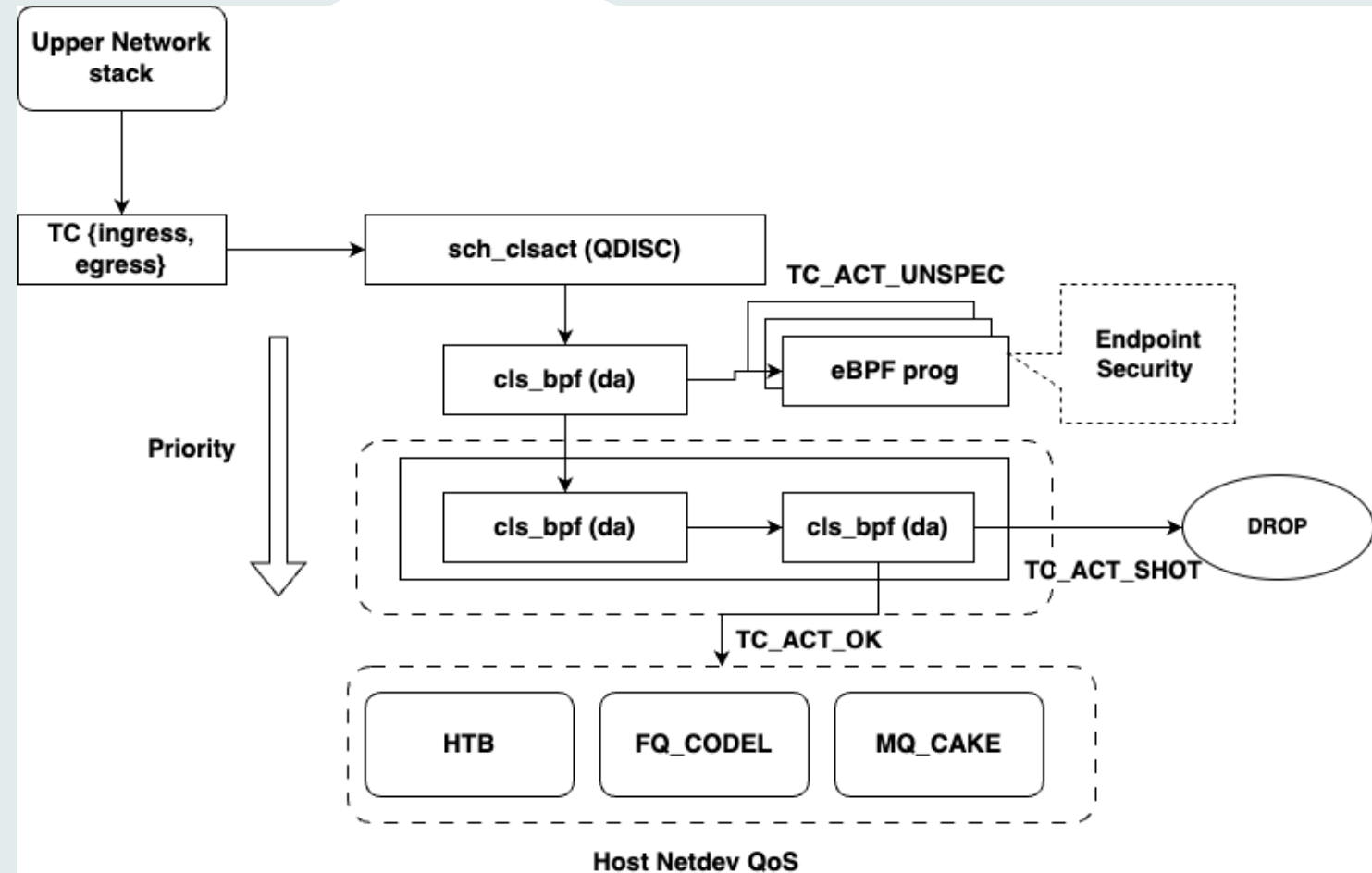
- Reprogram the Linux kernel in safe way
- Modern way to write kernel modules

1. Runs BPF virtual machine inside kernel
2. Custom BPF bytecode
3. Uses 512 bytes of stack
4. eBPF Maps as heap
5. CPU architecture agnostic



Linux Kernel Network Stack

- Sockets
- TCP/IP Stack
- Netfilter
- Traffic Control (QoS)
- Network Drivers



[[Jamal Salim et al.](#), [Daniel Borkmann et al.](#)]

Kernel Enforced Endpoint Security

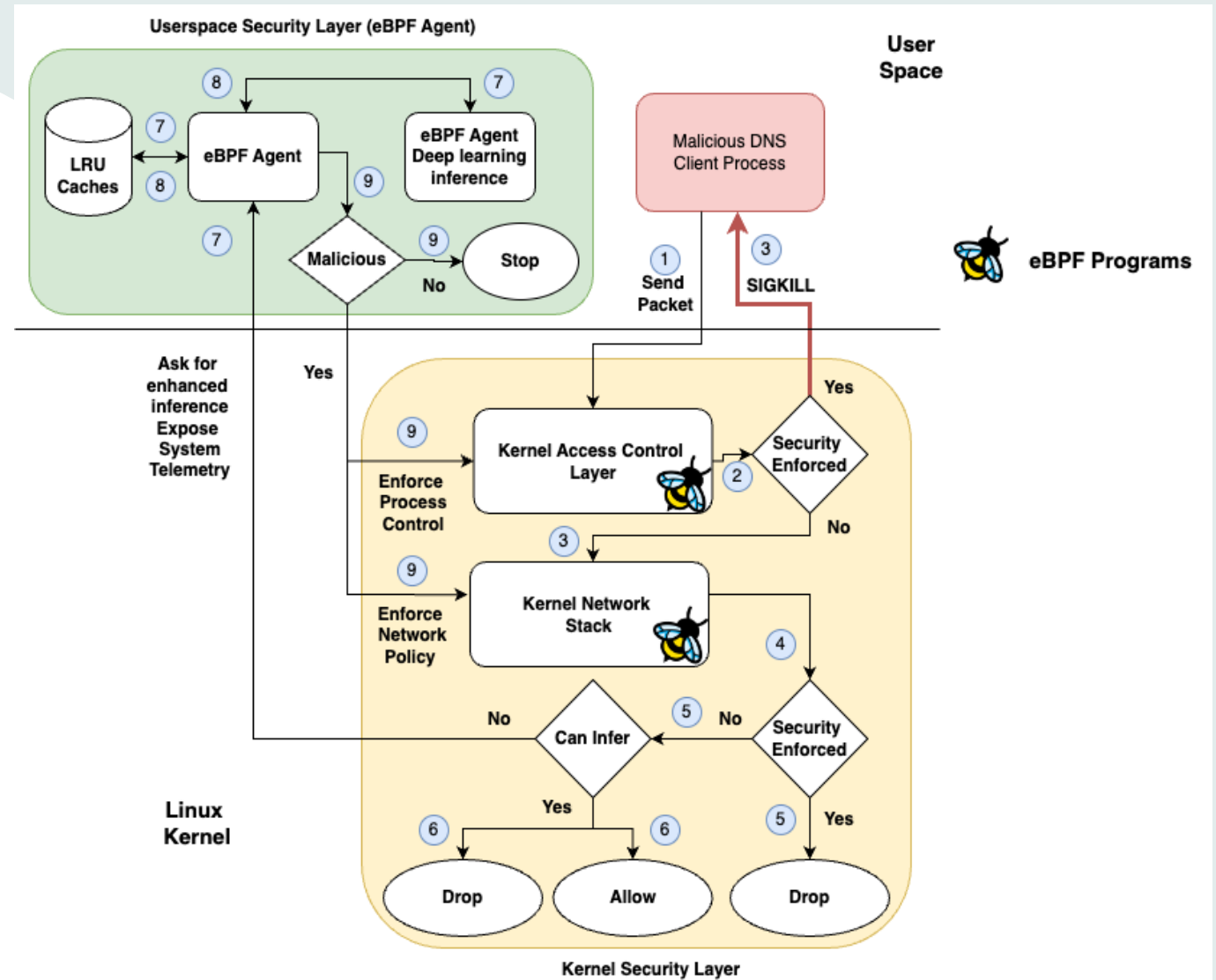
Agent based Endpoint Security

Userspace

- eBPF Userspace Bindings
- ONNX Quantized Deep Learning Model

Linux Kernel

- Inference Unix Domain Sockets
- Kernel Network Stack (eBPF)
 - Socket Layer
 - Traffic Control
- Kernel Security Layer (eBPF)
 - Kernel Security Modules
 - Kernel Syscall



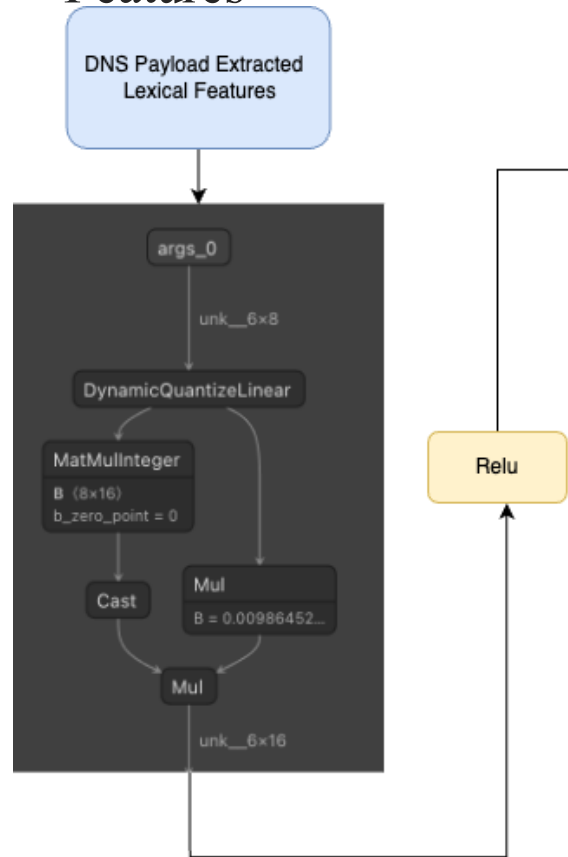
eBPF Agent Operations Modes

eBPF Agents in Data Plane handle DNS exfiltration over UDP

Mode	Goal	Requirement	Security Enforcement Process
Strict Enforcement Active Mode	Kill C2 Implants, ensure zero data loss and C2 command execution	DNS Traffic over UDP ports (53, 5353, 5355), for encapsulated and non-encapsulated traffic.	<ul style="list-style-type: none">• Kernel: Live Redirects suspicious DNS packets to Userspace• Userspace Trace malicious process exfiltration count and terminates it.
Process-Aware Adaptive Passive Mode	Kill C2 Implants, ensure negligible data loss and minimal C2 command execution	DNS Traffic over random UDP ports.	<ul style="list-style-type: none">• Kernel: Allow suspicious traffic passthrough. In Kernel start threat hunting process tied to malicious DNS packets• Userspace: Trace malicious process and terminates it

ONNX DNN Model

- Model Architecture ONNX Graph
- Sample Malicious Data
- Features



Feature	Description
---------	-------------

Malicious Exfiltrated data DNS queries	
381c018e3f5d05b78e3f6a026381e0f3476c066e8017be6ba9f5a9d758ef.d04bc3e0fc58e5a2401da590f3ee268a6af637eaafd210e58060a41082dc.92d594840bcb32a6500f39248db646e4e602f8547294692d83a4b4680223.b4d0ce0ec94abc9b6821cea90561aac558a6ba30b53e6b.bleed.io	
ae8c018e3f235392a20ca002649bd124bb6b506ba0771986720cbb1ad2e2.d59ca990aaa3eb1c580f5fb16d3b59d7eeb142458c8c54199c56e87b751c.69bbf57db184d263ed85a5ba5c9281ba327646f5638587016c9e0aa7b9b8.af182352de5de5b76a32242f04428b7d01b9a6d7999eb3.bleed.io7el4BGh376549344247687c217c3030393739363038373833303765353.bleed.io7el4BGh6a70677c217c52454749535445527c217c61343266363038366.bleed.io	
sebubx76xk4erpp3rwehoo3ubmbqeaqbaeq.a.e.e5.sk	
4az3kiecotwu3okbtvfm7pdpcabqeaqbaeq.a.e.e5.sk	
B (1)	unk__6x1
B (1)	unk__6x1
B (1)	unk__6x1

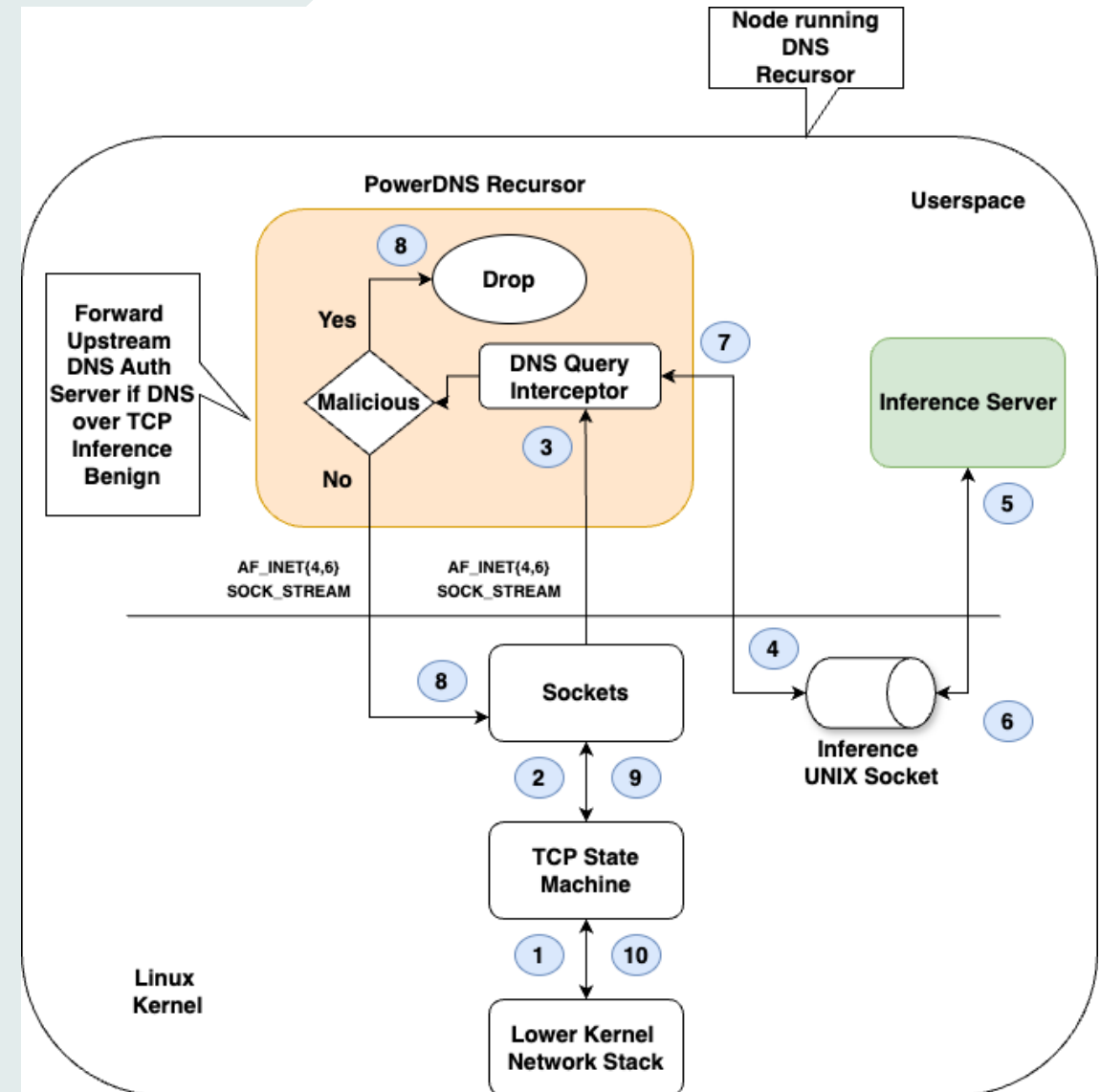
Datasets

Dataset Type	Source / Characteristics	Size	Primary Goal
Trusted Benign Cache	Top 1M Cisco Second-Level Domains (SLDs)	1 Million	Reduce inference on known-good traffic.
ISP-Captured DNS	Live-sniffed ISP DNS traffic [Ziza et al.]	50 Million	Provide real-world benign & malicious baseline.
Synthetic Exfiltration	Custom-generated (DET, DNSCat2, Sliver, Nuages, Custom Scripts, etc.);	2.4 Million	Malicious samples use varied obfuscation across file formats
Final Combined Dataset	Synthetically formed	3.8 millions	Balanced dataset w/ obfuscated payloads across file formats

Design Adjustment

Prevent DNS Exfiltration over TCP

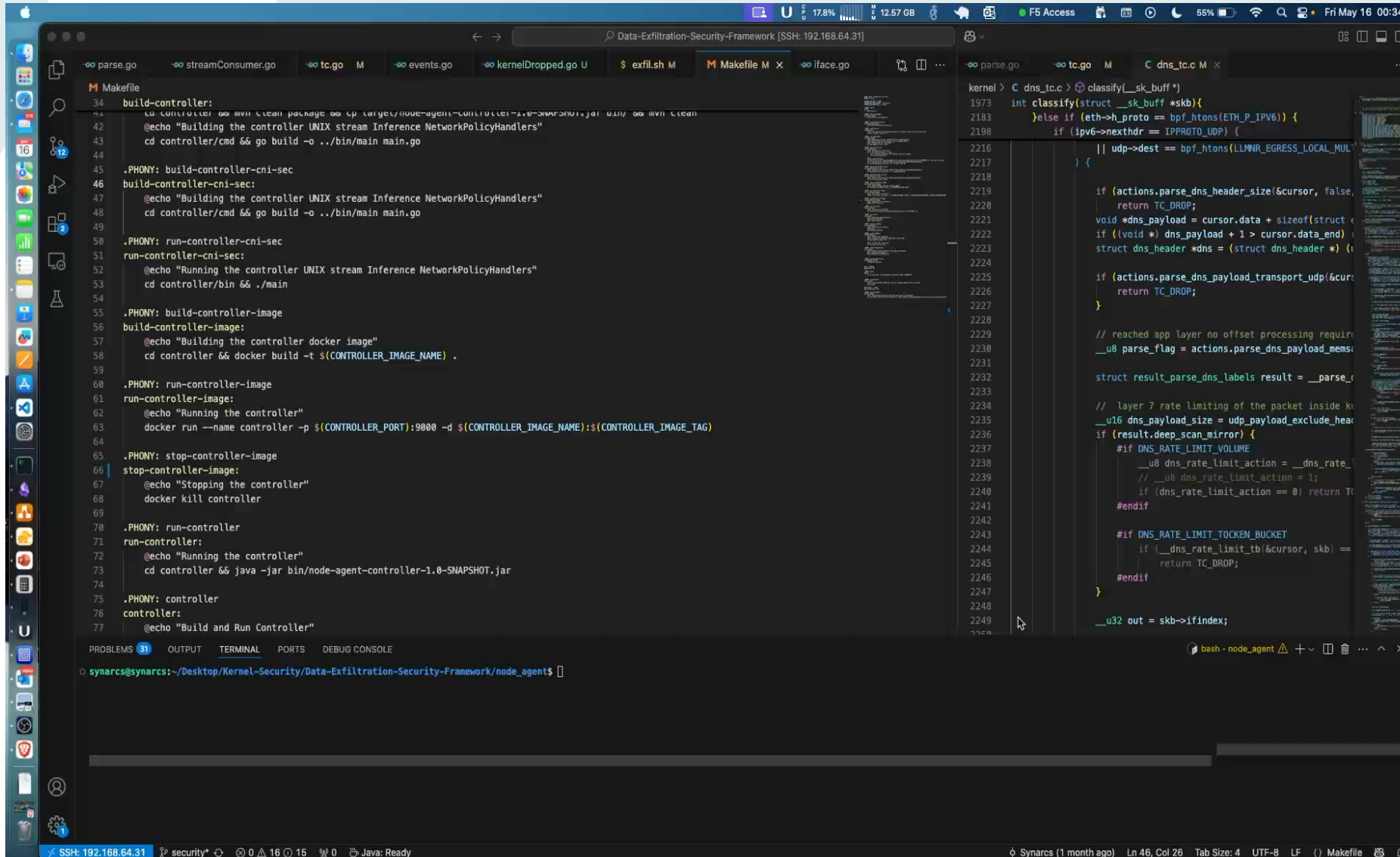
- Runs on
 - PowerDNS Recursor
- Relies on
 - PowerDNS recursor Query Interceptors
 - Inference UNIX domain sockets



Success Measure

- Response Speed
- Detection Accuracy
 - High Precision and Low False positives
- Volume of Data loss prior removal
- Scalability in distributed environments
- System Performance Impact
 - Kernel
 - Userspace

Framework Security Strength



The image shows a screenshot of a Visual Studio Code editor window. The top status bar indicates the connection is via SSH to 192.168.64.31. The editor has two tabs open: 'Makefile' and 'C dns_tc.c M'. The 'Makefile' tab is active, showing a series of build and run commands for a controller. The 'C dns_tc.c M' tab is also visible, showing C code for a kernel module. The bottom status bar shows the current file is 'security' and the cursor is at line 46, column 26.

```
34 build-controller:
35     @echo "Building the controller UNIX stream Inference NetworkPolicyHandlers"
36     @echo "Building the controller UNIX stream Inference NetworkPolicyHandlers"
37     cd controller/cmd && go build -o ../bin/main main.go
38
39 .PHONY: build-controller-cni-sec
40 build-controller-cni-sec:
41     @echo "Building the controller UNIX stream Inference NetworkPolicyHandlers"
42     @echo "Building the controller UNIX stream Inference NetworkPolicyHandlers"
43     cd controller/cmd && go build -o ../bin/main main.go
44
45 .PHONY: run-controller-cni-sec
46 run-controller-cni-sec:
47     @echo "Running the controller UNIX stream Inference NetworkPolicyHandlers"
48     @echo "Running the controller UNIX stream Inference NetworkPolicyHandlers"
49     cd controller/bin && ./main
50
51 .PHONY: build-controller-image
52 build-controller-image:
53     @echo "Building the controller docker image"
54     @echo "Building the controller docker image"
55     cd controller && docker build -t $(CONTROLLER_IMAGE_NAME) .
56
57 .PHONY: run-controller-image
58 run-controller-image:
59     @echo "Running the controller"
60     @echo "Running the controller"
61     docker run --name controller -p $(CONTROLLER_PORT):9000 -d $(CONTROLLER_IMAGE_NAME):$(CONTROLLER_IMAGE_TAG)
62
63 .PHONY: stop-controller-image
64 stop-controller-image:
65     @echo "Stopping the controller"
66     @echo "Stopping the controller"
67     docker kill controller
68
69 .PHONY: run-controller
70 run-controller:
71     @echo "Running the controller"
72     @echo "Running the controller"
73     cd controller && java -jar bin/node-agent-controller-1.0-SNAPSHOT.jar
74
75 .PHONY: controller
76 controller:
77     @echo "Build and Run Controller"
```

```
1973 int classify(struct __sk_buff *skb){
1974     int classify(struct __sk_buff *skb){
1975     } else if (eth->h_proto == bpf_htons(ETH_P_IPV6)) {
1976     } else if (eth->h_proto == bpf_htons(ETH_P_IPV6)) {
1977     if (ipv6->nexthdr == IPPROTO_UDP) {
1978     if (ipv6->nexthdr == IPPROTO_UDP) {
1979     || udp->dest == bpf_htons(LLMNR_EGRESS_LOCAL_MULTICAST)
1980     || udp->dest == bpf_htons(LLMNR_EGRESS_LOCAL_MULTICAST)
1981     } {
1982     } {
1983     if (actions.parse_dns_header_size(&cursor, false,
1984     if (actions.parse_dns_header_size(&cursor, false,
1985     return TC_DROP;
1986     return TC_DROP;
1987     void *dns_payload = cursor.data + sizeof(struct
1988     void *dns_payload = cursor.data + sizeof(struct
1989     if ((void *) dns_payload + 1 > cursor.data_end)
1990     if ((void *) dns_payload + 1 > cursor.data_end)
1991     struct dns_header *dns = (struct dns_header *) (
1992     struct dns_header *dns = (struct dns_header *) (
1993     if (actions.parse_dns_payload_transport_udp(&cursor,
1994     if (actions.parse_dns_payload_transport_udp(&cursor,
1995     return TC_DROP;
1996     return TC_DROP;
1997     }
1998     }
1999     // reached app layer no offset processing required
2000     // reached app layer no offset processing required
2001     __u8 parse_flag = actions.parse_dns_payload_memory
2002     __u8 parse_flag = actions.parse_dns_payload_memory
2003     struct result_parse_dns_labels result = __parse_dns_labels
2004     struct result_parse_dns_labels result = __parse_dns_labels
2005     // layer 7 rate limiting of the packet inside kernel
2006     // layer 7 rate limiting of the packet inside kernel
2007     __u16 dns_payload_size = udp_payload_exclude_header_size
2008     __u16 dns_payload_size = udp_payload_exclude_header_size
2009     if (result.deep_scan_mirror) {
2010     if (result.deep_scan_mirror) {
2011     #if DNS_RATE_LIMIT_VOLUME
2012     #if DNS_RATE_LIMIT_VOLUME
2013     __u8 dns_rate_limit_action = __dns_rate_limit_action
2014     __u8 dns_rate_limit_action = __dns_rate_limit_action
2015     // __u8 dns_rate_limit_action = 1;
2016     // __u8 dns_rate_limit_action = 1;
2017     if (dns_rate_limit_action == 0) return TC_DROP;
2018     if (dns_rate_limit_action == 0) return TC_DROP;
2019     #endif
2020     #endif
2021     #if DNS_RATE_LIMIT_TOKEN_BUCKET
2022     #if DNS_RATE_LIMIT_TOKEN_BUCKET
2023     if (__dns_rate_limit_tb(&cursor, skb) == 0)
2024     if (__dns_rate_limit_tb(&cursor, skb) == 0)
2025     return TC_DROP;
2026     return TC_DROP;
2027     #endif
2028     #endif
2029     }
2030     }
2031     __u32 out = skb->ifindex;
2032     __u32 out = skb->ifindex;
```

Results and Evaluation

- Model Metrics
- Throughput comparisons (Active mode)
- Resources
 - Memory Usage
 - Security Agent memory usage at endpoints in data plane
 - CPU Flame Graph
 - eBPF Agent CPU Flame Graph

Test Bench

CPU: Intel Xeon 6130

Memory: 8 GB

Linux Kernel: 6.12.4

Network Driver: netvsc

Bandwidth: 100 Gb/sec

Root QDISC: Fq_Codel

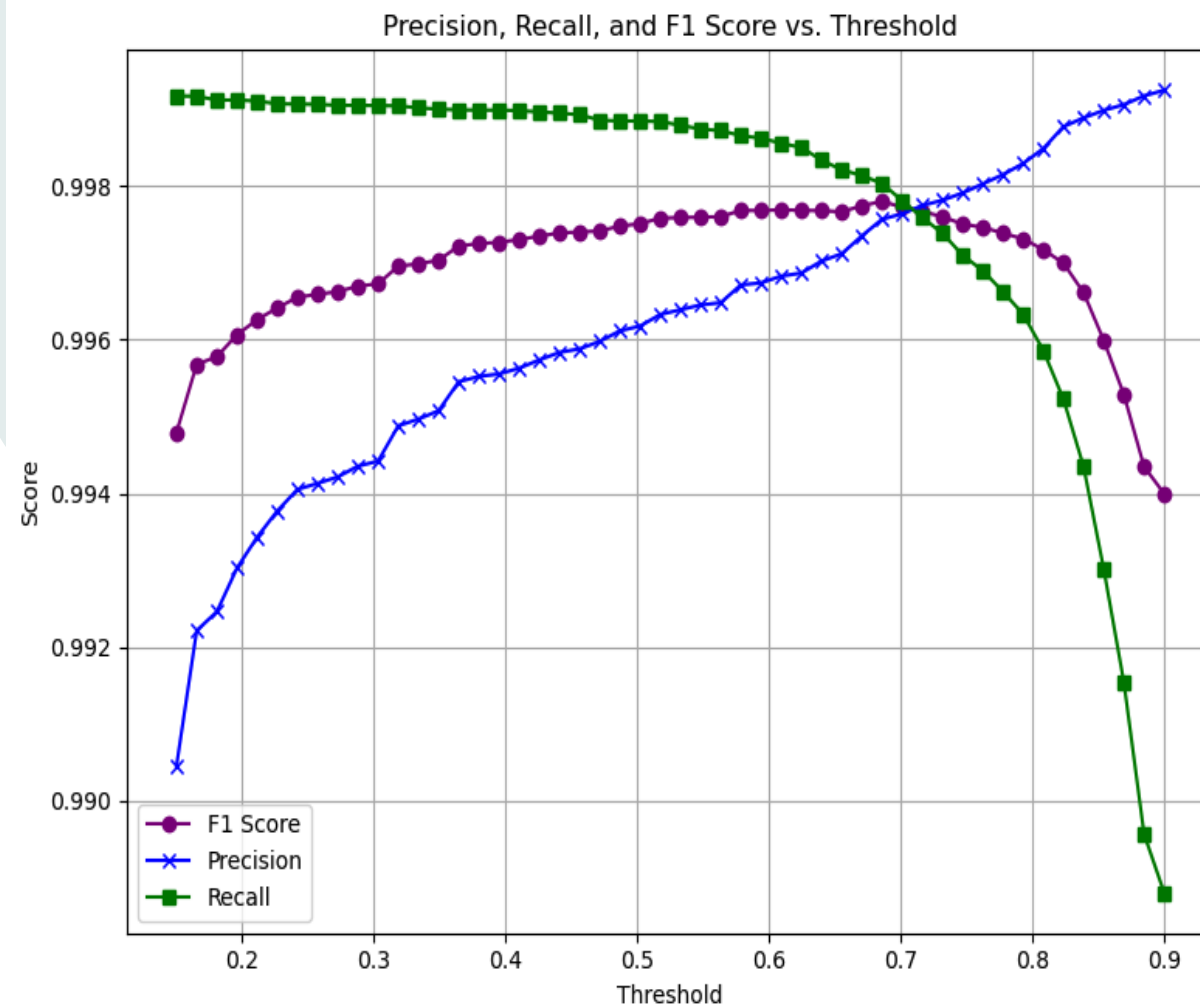
Queues: 8 RX / TX

DNN Model Metrics

Metric	Training	Validation
Accuracy	0.9973	0.9997
AUC	0.9997	0.9997
Loss	0.0099	0.0091
Precision	0.9959	0.9959
Recall	0.9987	0.9988

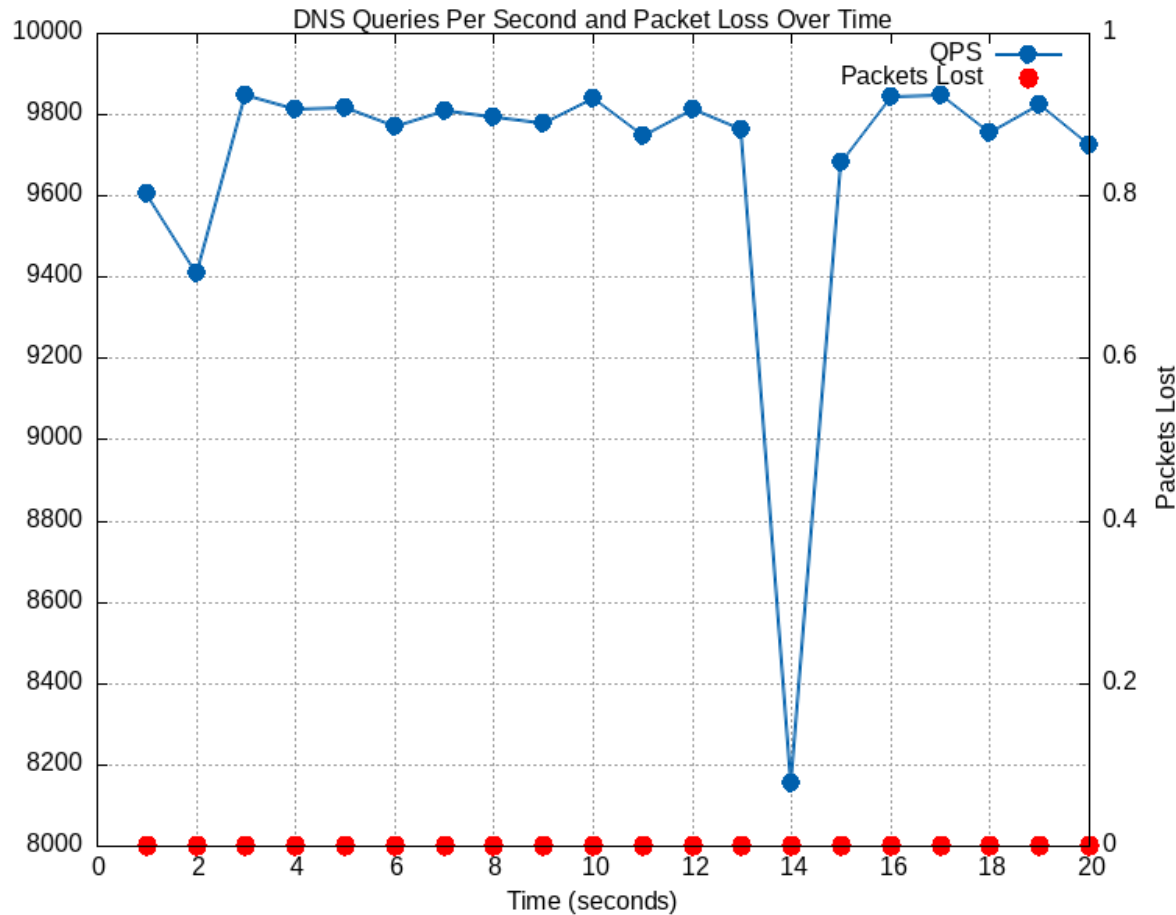
Table 5.1: Model Evaluation Metrics

Model Performance

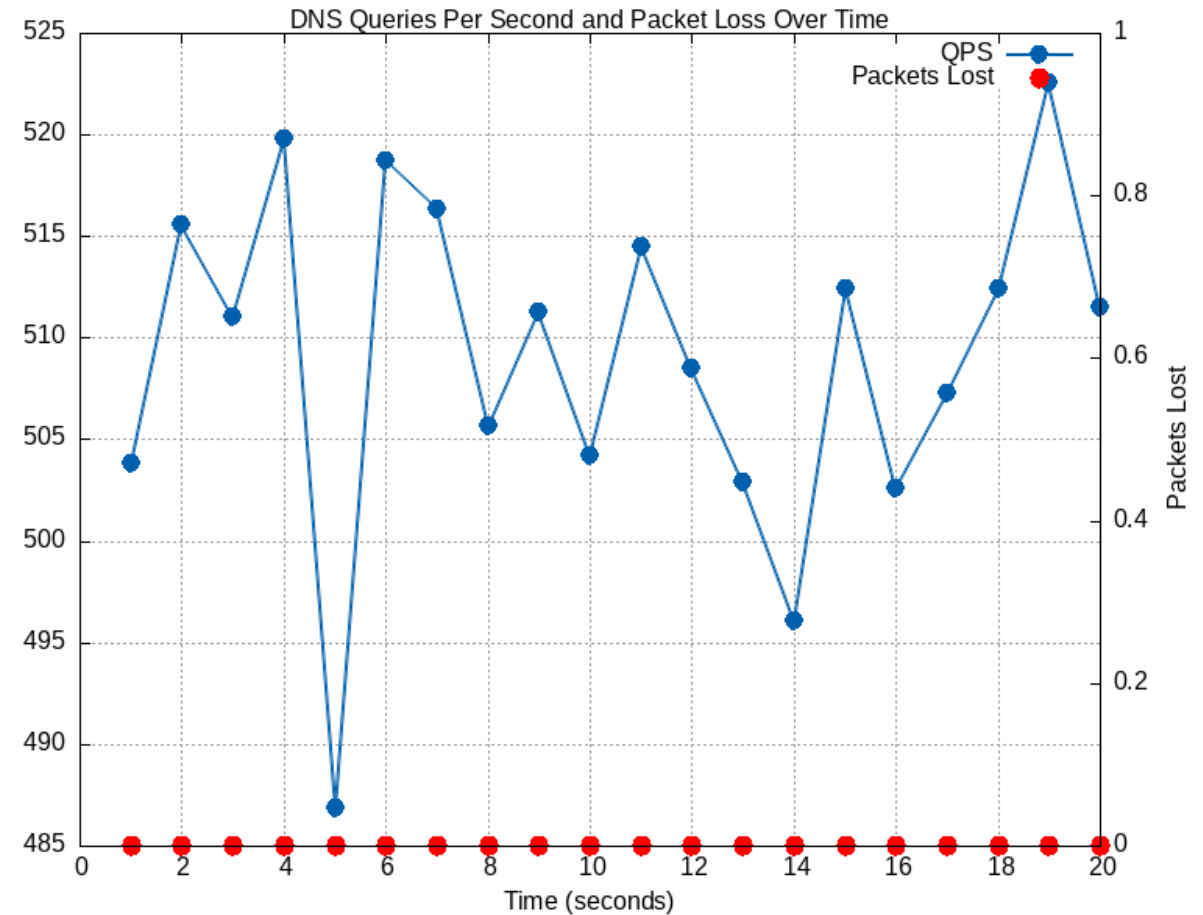


Model Scores

Throughput comparisons – Active Mode

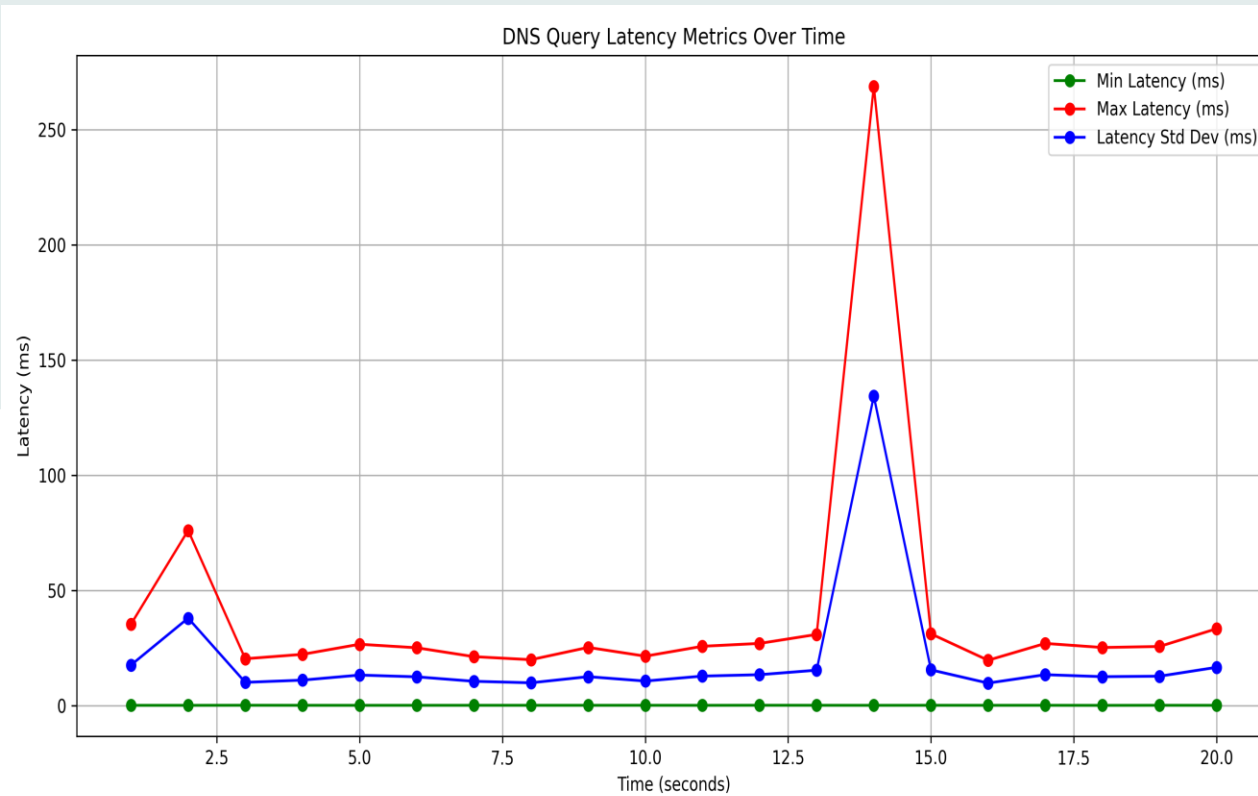


Agent LRU Cache Read-Through Hit

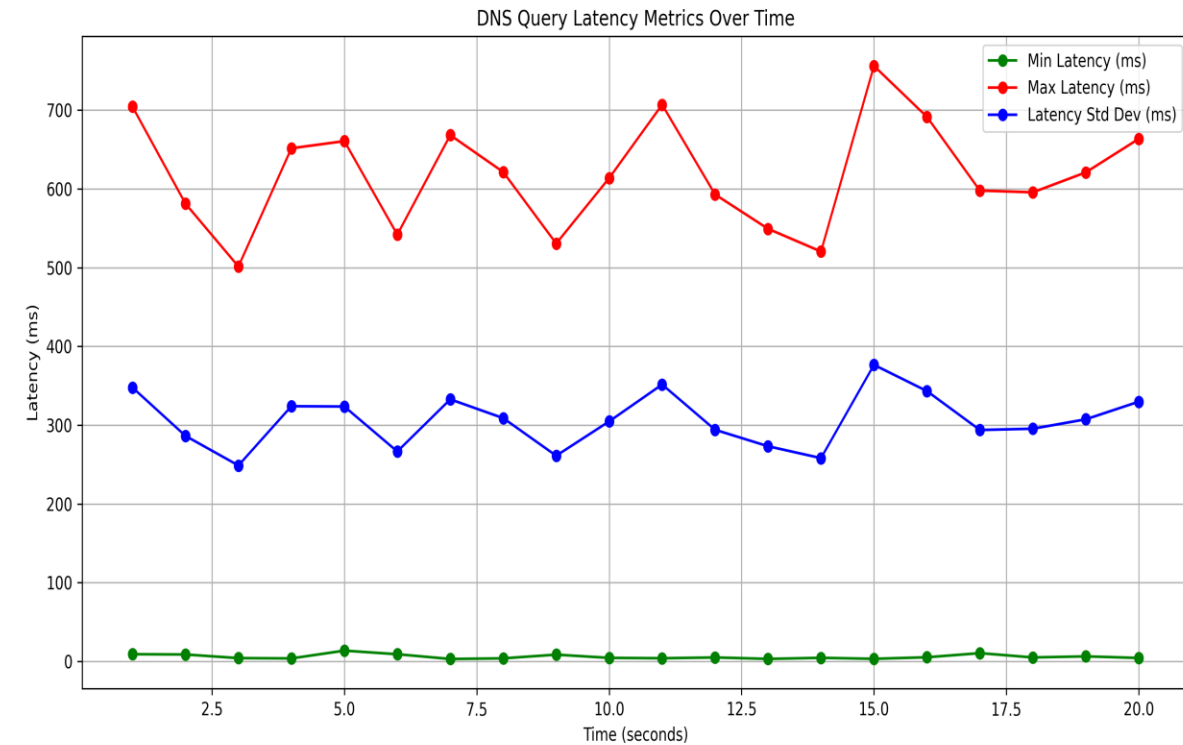


ONNX Live Inferencing

Throughput comparisons – Active Mode (continued)



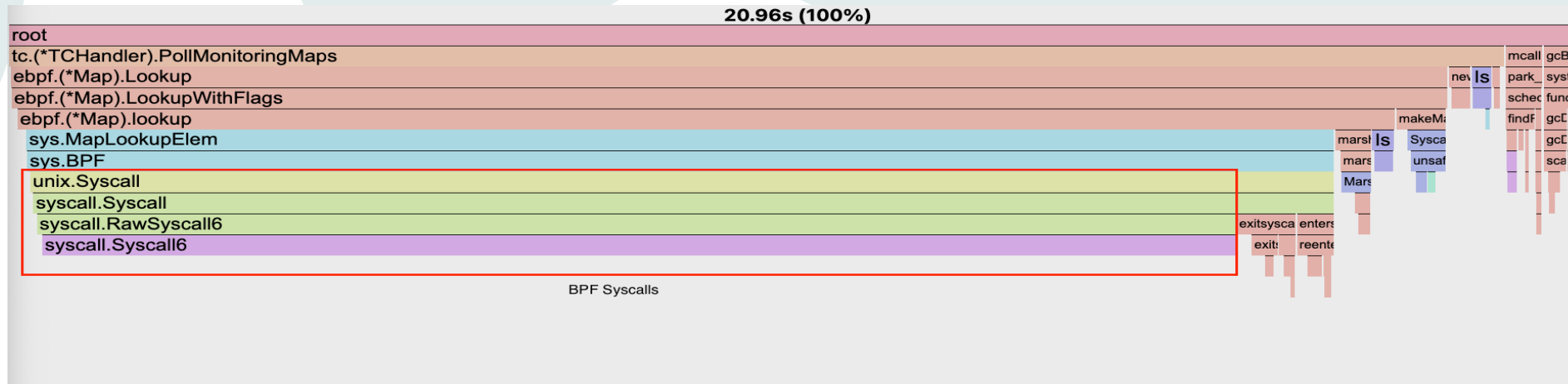
Agent LRU Cache Read-Through Hit



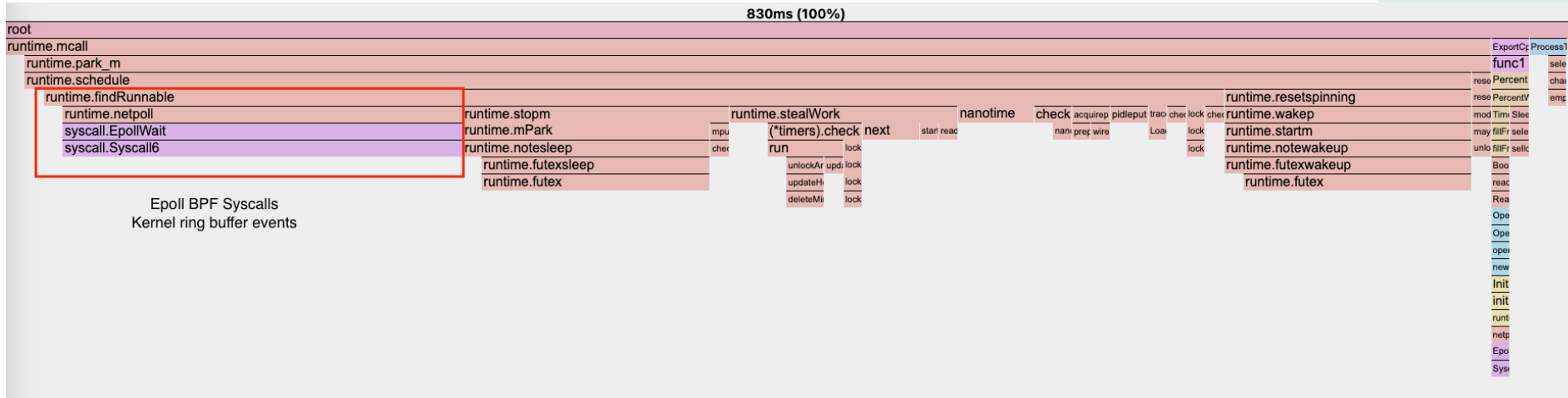
ONNX Live Inferencing

Resource Usage – eBPF Agent Flame Graph

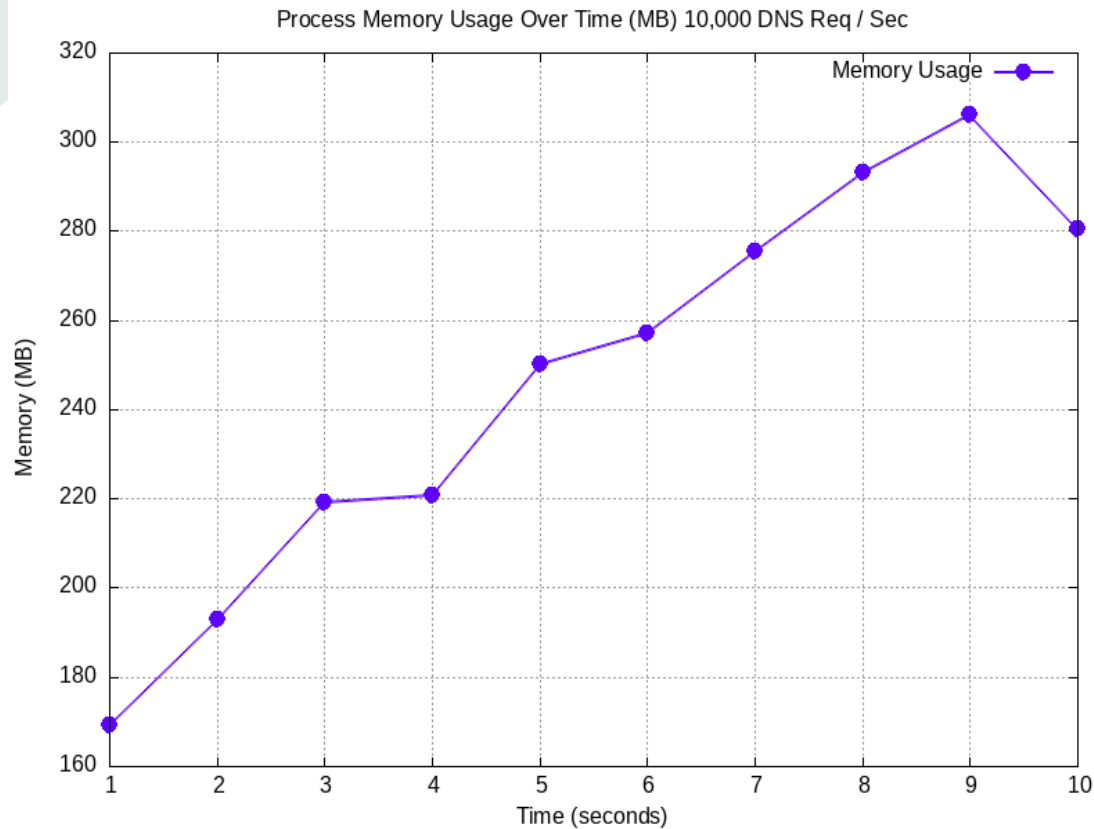
Userspace Busy-Polling overhead monitoring maps



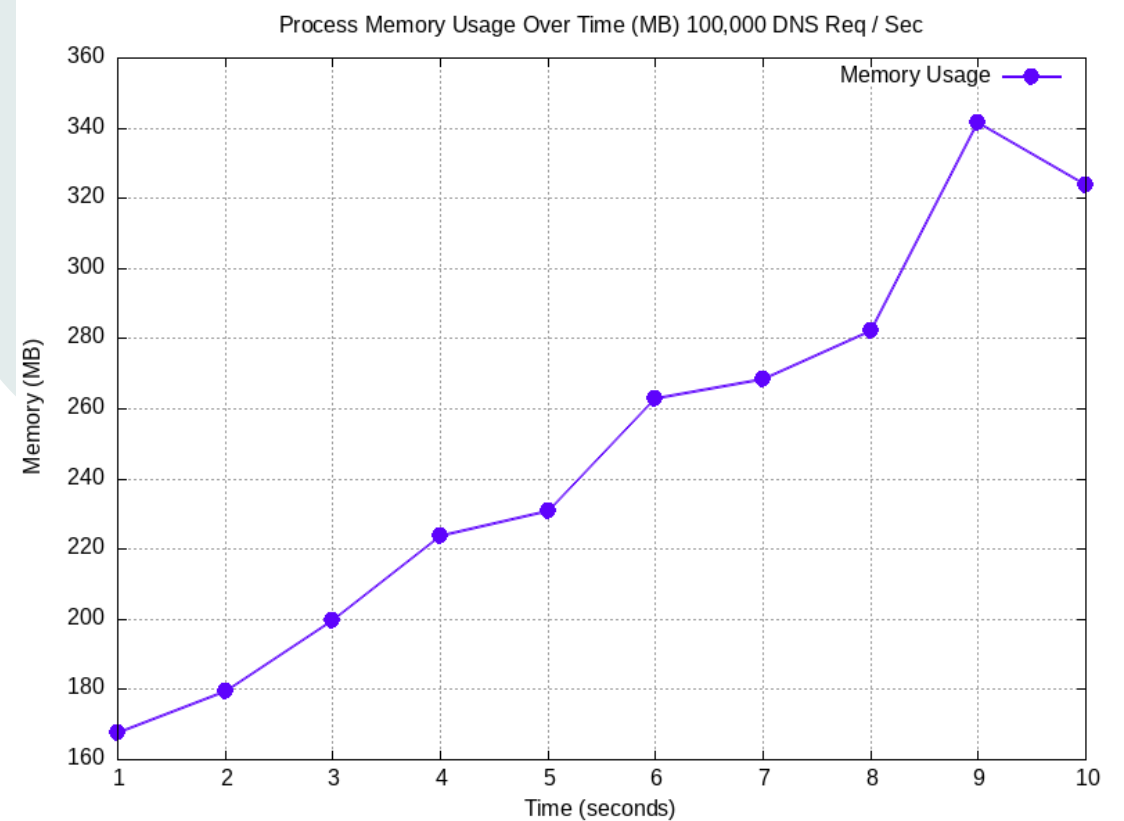
Kernel Epoll asynchronous I/O agent performance boost



Resource Usage - Memory



10,000 DNS Req / Sec



100,000 DNS Req / Sec

Knowledge Gained

- Kernel Traffic control (QoS) Qdiscs (clsact, fq_codel, codel, htb)
- Kernel TCP Congestion control (rene, cubic, BBR)
- Userspace-kernel synchronization (kernel spin locks, RCU, userspace mutex, atomic ref_counters), kernel asynchronous I/O
- Kernel Security Layer (LSM, seccomp, TEE)
- Distributed Systems concepts intersection with system performance
 - Caching Write / Read-through policies
 - Caching Eviction Policies
 - Data Streaming
 - NUMA cache coherence → NetFlow Steering

Future Work

- **Extend Support for DNS-over-TCP and Encrypted Tunnels:** Implement in-kernel eBPF-based detection for DNS-over-TCP replicating TCP state machine over kernel socket layer, paired with userspace DPI via Envoy proxy.
- **Add In-Kernel TLS Fingerprinting:** Use eBPF for TLS fingerprinting (e.g., JA3/JA4) to detect DNS exfiltration over TLS (DOH), DNS over mTLS, WireGuard.
- **Rate-Limiting Based on Volume and Throughput:** Integrate egress CSLACT-based dynamic rate limiting for DNS mass data breaches integrating EDT_BPF, FQ_CODEL and HTB QDISC's.
- **XDP-Based Flood Prevention:** Introduce XDP ingress filtering inside kernel to mitigate NXDOMAIN-based DNS water torture and DNS amplification attacks on the endpoint.

References

- Singamaneni Krishnapriya and Sukhvinder Singh. "A comprehensive survey on advanced persistent threat (apt) detection techniques." *Computers, Materials and Continua*, 80(2):2675-2719, 2024.
<https://www.sciencedirect.com/org/science/article/pii/S1546221824005952>
- Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. "Exposure: Finding malicious domains using passive dns analysis." In *NDSS*, pages 1-17, 2011. https://sites.cs.ucsb.edu/~chris/research/doc/ndss11_exposure.pdf
- Manos Antonakakis, Roberto Perdisci, David Dagon, Wenke Lee, and Nick Feamster. "Building a dynamic reputation system for {DNS}." In *19th USENIX Security Symposium (USENIX Security 10)*, 2010.
https://www.usenix.org/legacy/event/sec10/tech/full_papers/Antonakakis.pdf
- Samaneh MahdaviFar, et al. "Lightweight hybrid detection of data exfiltration using dns based on machine learning." In *Proceedings of the 2021 11th International Conference on Communication and Network Security, ICCNS '21*, pages 80-86, 2022.
<https://dl.acm.org/doi/10.1145/3507509.3507520>
- Jawad Ahmed, Hassan Habibi Gharakheili, Qasim Raza, Craig Russell, and Vijay Sivaraman. "Real-time detection of dns exfiltration and tunneling from enterprise networks." <https://ieeexplore.ieee.org/document/8717806>
- Jamal Hadi Salim. "Linux traffic control classifier-action subsystem architecture." *Proceedings of Netdev 0.1*, 2015.
<https://people.netfilter.org/pablo/netdev0.1/papers/Linux-Traffic-Control-Classifer-Action-Subsystem-Architecture.pdf>
- Daniel Borkmann. "On getting tc classifier fully programmable with cls bpf." *Proceedings of netdev*, 1, 2016.
<https://www.netdevconf.org/1.1/proceedings/papers/On-getting-tc-classifier-fully-programmable-with-cls-bpf.pdf>
- Kristijan Ziza, Pavle Vuletić, and Predrag Tadić. Dns exfiltration dataset, 2023. <https://data.mendeley.com/datasets/c4n7fckkz3/3>