EDNS0 como Vetor Adversarial: Lacunas de Visibilidade Empírica em Ferramentas Populares de DPI e Detecção de Intrusão

Bruno Carvalho Alvarenga[1], **André Ricardo Abed Grégio**[1]

[1]Programa de Pós-Graduação em Informática – Universidade Federal do Paraná (UFPR)
Curitiba – PR – Brasil

{brunoalvarenga,gregio}@ufpr.br

Abstract. *Options carried in the EDNS0 OPT pseudo-resource record constitute an adversarial vector for covert channels in DNS traffic that the defensive literature has largely neglected. Empirical evaluation of Suricata, Snort and Zeek in stock configuration yields zero detections against six adversarial techniques (T1 to T6). Source-code inspection identifies the cause as a structural parser gap in Zeek 8.1.x. We address it with a Spicy plugin exposing per-option and per-OPT-record events, and derive five heuristics with precision and recall of 1.00. Porting to Suricata reveals an empirical ceiling: pure rule-language matches precision via semantic anchoring but cannot reproduce recall on multi-option enumeration or cross-connection stateful heuristics.*

Resumo. *Opções carregadas no pseudo-registro OPT do EDNS0 constituem vetor adversarial para canais encobertos em tráfego DNS negligenciado pela literatura defensiva. A avaliação empírica de Suricata, Snort e Zeek em configuração padrão resulta em zero detecções contra seis técnicas adversariais (T1 a T6). A inspeção do código-fonte identifica a causa como lacuna estrutural do parser no Zeek 8.1.x. Endereçamos a lacuna com um plugin Spicy que expõe eventos por opção e por pseudo-RR OPT, e derivamos cinco heurísticas com precisão e revocação de 1,00. A portabilidade para Suricata revela um teto: a rule-language pura alcança precisão equivalente mediante ancoragem semântica, mas não reproduz revocação em enumeração multi-opção nem em heurísticas com estado por origem.*

1. Introdução

A extensão EDNS0 (*Extension Mechanisms for DNS*, RFC 6891 [Damas et al. 2013]) introduz no protocolo DNS um pseudo-registro denominado OPT, cujo RDATA pode carregar opções no formato TLV (*type-length-value*). A mesma flexibilidade que torna o EDNS0 útil como mecanismo de extensão o torna substrato adequado para canais encobertos, e esse risco não é conjectural: a RFC 7830 [Mayrhofer 2016] reconhece explicitamente, em sua seção de *security considerations*, que o conteúdo da opção Padding pode ser empregado como canal encoberto, e demonstrações públicas recentes confirmam a viabilidade prática do vetor, como o SiphonDNS [ttpreport 2025], que exfiltra dados pela opção ECS atravessando resolvedores públicos de produção. Da estrutura TLV decorrem três famílias de abuso: opções cujo código IANA não possui alocação ou é de uso

experimental; comprimentos atípicos em opções legítimas, dado que as RFCs correspondentes especificam faixas amplas [Eastlake and Andrews 2016]; e conteúdo arbitrário em campos cuja semântica normativa tolera valores livres [Mayrhofer 2016].

Em contraste com essa superfície, a literatura defensiva sobre túneis DNS é madura quanto ao abuso do QNAME [Born and Gustafson 2010, Aiello et al. 2013, Nadler et al. 2019] e escassa quanto a canais estabelecidos via opções EDNS0. Essa assimetria motiva as três perguntas que guiam o estudo: que cobertura efetiva as ferramentas de inspeção profunda de pacotes (DPI, *deep packet inspection*) oferecem, em configuração padrão, contra opções EDNS0 abusivas; que extensões do mecanismo de análise se fazem necessárias quando essa cobertura é estruturalmente limitada; e que heurísticas podem ser derivadas do vetor, com que grau de portabilidade entre plataformas de detecção.

As contribuições são cinco. Primeiro, o gerador adversarial EDNStego e uma taxonomia de seis técnicas (T1 a T6). Segundo, a avaliação de Suricata, Snort e Zeek em configuração padrão, com zero detecções. Terceiro, a caracterização da lacuna estrutural do *parser* EDNS do Zeek 8.1.x, com `TODO` no próprio código do projeto. Quarto, o plugin Spicy `edns0-arbitrary` v2, que emite eventos por opção TLV e por pseudo-RR OPT. Quinto, cinco heurísticas calibradas contra tráfego real capturado em modo *full-packet* (calibração que produziu correções mensuráveis, como incluir o código 15 na *whitelist* após observá-lo em $\sim$24% das respostas reais) e validadas em seis corpora, com portabilidade parcial para Suricata que evidencia um teto estrutural de revocação da sua linguagem de regras em cenários multi-opção e com estado por origem.

A contribuição excede a avaliação empírica: a ausência de alertas é atribuída a uma causa estrutural verificável no código-fonte (Seção 5), removida por um artefato reutilizável (Seção 6), e o teto de expressividade da *rule-language* é estabelecido por construção (Seção 9). Os dois últimos resultados são proposições sobre os mecanismos de detecção, não sobre o corpus observado.

2. Trabalhos relacionados

2.1. O vetor clássico: túnel DNS via QNAME

O abuso de DNS para canal encoberto data dos primeiros experimentos com túneis em `qname` e respostas TXT, materializados em ferramentas como `iodine`, `dnscat2` e `DNSCat`. A literatura defensiva concentra-se nesse vetor com técnicas razoavelmente maduras: Born e Gustafson [Born and Gustafson 2010] usam análise de unigramas, bigramas e trigramas para detectar QNAMEs com payload codificado contra a distribuição de Zipf de domínios legítimos; Aiello *et al.* [Aiello et al. 2013] aplicam classificadores supervisionados sobre *features* estatísticas como comprimento da consulta, razão de caracteres distintos e entropia do subdomínio; e Nadler *et al.* [Nadler et al. 2019] estendem a família para exfiltração de baixo *throughput* via análise comportamental por domínio em janelas longas.

Uma segunda geração substitui heurísticas explícitas por classificadores treinados [Buczak et al. 2016, Lambion et al. 2020], incluindo trabalhos sobre robustez a exfiltradores adaptativos [Žiža et al. 2023]. A comparação experimental direta não é possível sem redesenho: todos tomam o QNAME e agregados de fluxo como entrada, de modo que aplicá-los ao pseudo-RR OPT exigiria a construção do *dataset* rotulado que a Se-

ção 8 mostra inexistir para EDNS0. As duas famílias são complementares, e a instrumentação de *parsing* deste trabalho é pré-requisito para ambas: sem eventos por opção não há *features* de EDNS0 a extrair.

Em comum, a literatura sobre o vetor clássico opera sobre o próprio QNAME e não toca no pseudo-RR OPT, que é o objeto deste trabalho. Os *datasets* públicos refletem essa orientação: o CIC-Bell-DNS-EXF 2021 [Mahdavifar et al. 2021] contém exfiltração via subdomínios, mas não serve como *baseline* para canais via EDNS0, por duas razões independentes. A primeira é de captura: o *snaplen* de 96 bytes é insuficiente para preservar o pseudo-RR OPT, que tipicamente ocupa os últimos 30 a 50 bytes da resposta. A segunda dispensa a primeira e é conclusiva: o campo ARCOUNT reside em *offset* fixo do cabeçalho DNS, sempre dentro dos 96 bytes capturados, e nossa inspeção dos 18 PCAPs encontrou ARCOUNT igual a zero em 100% dos 849.674 pacotes DNS do conjunto, sem um único cabeçalho truncado. As mensagens do *dataset* declaram, portanto, ausência total de registros adicionais: não se trata de EDNS ocultado pela truncagem, e sim de tráfego sem EDNS.

2.2. O vetor negligenciado: opções EDNS0

Trabalhos sobre EDNS0 são escassos no âmbito defensivo. Rijs [Rijs 2014] discute DNS Cookies como mitigação de amplificação sem abordá-lo como vetor de exfiltração. As RFCs especificam códigos individuais (RFC 7830 [Mayrhofer 2016], RFC 7873 [Eastlake and Andrews 2016], RFC 7901 [Wouters 2016], RFC 8914 [Kumari et al. 2020]) sem tratar de seu uso adversarial, com a exceção já mencionada da RFC 7830. No âmbito ofensivo, o SiphonDNS [ttpreport 2025] demonstra publicamente exfiltração via ECS através de resolvedores públicos como prova-de-conceito, oferecendo implementação aberta da técnica. Medições complementares (Farrokhi [Farrokhi 2025]; BYU sobre cookies no PAM 2021 [Davis and Deccio 2021]; ISC sobre conformidade EDNS [Internet Systems Consortium 2015, Internet Systems Consortium 2026]) caracterizam o comportamento de opções EDNS em produção como heterogêneo entre resolvedores, achado que a Seção 10 discute. A escassez complementar de *rulesets* e *datasets* para o vetor EDNS0 é, em si, indicador de que a comunidade defensiva ainda não tratou o vetor como ameaça operacional.

Tabela 1. Posicionamento frente aos trabalhos correlatos.

Trabalho	**Campo DNS**	**Técnica**	**Cobre OPT**
Born [Born and Gustafson 2010]	QNAME	n-gramas	Não
Aiello [Aiello et al. 2013]	QNAME	class. superv.	Não
Buczak [Buczak et al. 2016]	QNAME, fluxo	*random forest*	Não
Nadler [Nadler et al. 2019]	QNAME	comport. domínio	Não
Lambion [Lambion et al. 2020]	QNAME	ML	Não
Žiža [Žiža et al. 2023]	QNAME	ML adversarial	Não
CIC-EXF [Mahdavifar et al. 2021]	subdomínio	—	Não
SiphonDNS [ttpreport 2025]	OPT	ofensiva	Parc.
Este	**OPT**	***parser*+heur.**	**Sim**

A Tabela 1 posiciona este trabalho frente aos correlatos. O padrão que emerge

é o de uma literatura concentrada no QNAME, na qual a única incursão sobre o OPT é ofensiva e restrita a um único código já visível aos *parsers* existentes.

3. Ambiente experimental e modelo adversarial

3.1. Topologia do laboratório

O laboratório consiste em cinco máquinas virtuais sobre KVM/QEMU em Ubuntu 24.04, interligadas por *bridges* Open vSwitch em duas redes lógicas. A primeira, `br-internal`, liga o cliente `victim-linux`, onde roda o agente EDNStego que emite as técnicas T1 a T6, ao resolvedor `dns-resolver`, que executa BIND9 em modo *forward only*. A segunda, `br-external`, liga esse resolvedor aos resolvedores públicos (Cloudflare, Google e Quad9), ao servidor de C2 `c2-server` que decodifica o payload, e ao `traffic-monitor`. Uma quinta máquina, `fw-test`, executa Suricata 6.0.10 e Snort 3.12.1.0 em escuta passiva sobre duas interfaces dedicadas, alimentadas pelo espelhamento de portas das duas *bridges*; o Zeek 8.1.2 executa no próprio `dns-resolver`. Esse espelhamento fornece cópias passivas do tráfego em cada *bridge*, capturadas via `tcpdump` em dois pontos: o *leg* interno, entre a vítima e o resolvedor, e o *leg* externo, entre o resolvedor e os destinos públicos. A topologia detalhada e os *playbooks* de provisionamento estão no repositório do trabalho[1].

A captura nos dois *legs* é essencial: o resolvedor descarta opções EDNS desconhecidas no encaminhamento [Damas et al. 2013], e capturar antes e depois do BIND9 permite quantificar esse *stripping* e demonstrar que a detecção precisa estar no *leg* interno para códigos não-RFC.

3.2. Modelo adversarial

O cenário primário é o de adversário com execução em *endpoint* corporativo que busca canal encoberto observável por observador adversarial posicionado no *leg* interno, entre vítima e resolvedor recursivo. O modelo cobre reconhecimento e sinalização entre *endpoints* comprometidos no mesmo segmento, *beaconing* para *sniffer* passivo, e testagem da superfície de detecção corporativa. O canal é funcional para todas as técnicas T1 a T6 porque a observação ocorre antes do encaminhamento. Já a exfiltração *end-to-end* via resolvedor até C2 externo não é coberta no caso geral, porque o resolvedor descarta opções EDNS desconhecidas ao encaminhar [Damas et al. 2013]. A Seção 6 quantifica esse fenômeno sob o BIND9 do laboratório, e dele decorre a localização interna da detecção.

Uma variante relevante, T3-variante (tratada nas Seções 3.3 e 8), descreve adversário que escapa de heurísticas de comprimento emitindo cookies com 8 bytes (dentro do conjunto $\{8, 16, 24\}$ de H2) mas conteúdo único por consulta derivado do payload, mantendo *source port* fixo (comportamento realista de socket UDP persistente). Cenários alternativos não modelados, como adversário que controla o resolvedor, autoritativo malicioso ou DNS direto sem resolvedor intermediário, ficam para investigação subsequente (Seção 10).

3.3. EDNStego e a taxonomia de seis técnicas adversariais

O gerador adversarial EDNStego foi implementado para produzir tráfego controlado e reproduzível, tanto para a avaliação das ferramentas DPI quanto para o desenvolvimento

[1] https://github.com/the-red-ace/edns0-arbitrary

das regras. A ferramenta consiste em um agente Python na vítima, que codifica payload arbitrário em opções EDNS0 e o emite através de consultas DNS legítimas, e um servidor de C2 que decodifica o payload no lado oposto. As consultas seguem todas as convenções do protocolo DNS, de modo que o tráfego adversarial é indistinguível de tráfego legítimo fora da inspeção das opções EDNS.

Seis técnicas distintas foram implementadas, designadas T1 a T6 e organizadas segundo dois eixos taxonômicos: o campo do pseudo-RR OPT abusado (opções TLV no RDATA, *header* do OPT, ou composição mista) e a categoria do código IANA explorado. A Tabela 2 consolida a taxonomia. T6 foi composta deliberadamente como caso adversarial pessimista para regras baseadas em *rule-language* pura, enfileirando código na *whitelist* (10) e fora dela (65001) no mesmo pseudo-RR OPT (Seção 9). O corpus adversarial principal consiste em doze PCAPs sob payload idêntico de 1279 bytes; os corpora complementares T3-variante (50 PCAPs) e T4-variante (200 PCAPs em quatro regimes) são descritos na Seção 8.

Tabela 2. Taxonomia das técnicas adversariais T1 a T6 implementadas em EDNStego. *bufsize* denota o campo `udp_payload_size` do *header* do pseudo-RR OPT.

ID	Família	Código IANA	RFC	Categoria
T1	TLV	65001	—	Experimental
T2	TLV	12 (Padding)	7830	*Std. Track*
T3	TLV	10 (Cookie)	7873	*Std. Track*
T4	*header*	*bufsize*	6891	Campo *header*
T5	TLV	13 (CHAIN)	7901	Experimental
T6	híbrida	65001+10	—	T1+T3+T4

A T3-variante é tratada como variante adversarial de T3, e não como sétima técnica principal da taxonomia. Sua finalidade é fornecer *ground truth* para a heurística H3, que aborda cenário complementar ao de H2, em que o adversário preserva o comprimento canônico do cookie (8 bytes) e varia o conteúdo entre consultas. A T3-variante usa código IANA 10 (Cookie), comprimento fixo em 8 bytes para evitar disparo de H2, conteúdo derivado do payload e único por consulta, e *source port* fixo por execução. O módulo correspondente foi adicionado ao EDNStego sem qualquer alteração das implementações de T1 a T6, de modo a preservar a comparabilidade com os resultados anteriores. A justificativa do *source port* fixo é discutida na Seção 8, junto com a semântica de `c$uid` no Zeek para tráfego DNS sobre UDP.

4. Avaliação de ferramentas DPI em configuração padrão

A seleção das três plataformas cobre os dois paradigmas arquiteturais das soluções DPI: Suricata e Snort representam *signature matching* sobre *rule-language* declarativa, dominante em NIDS de produção; Zeek representa o analisador de protocolo com *script-land* programável. Levantamentos de IDS de código aberto [Waleed et al. 2022] apontam esses três como referência operacional. Essa distinção importa porque o limite encontrado no Suricata (Seção 9) é propriedade do paradigma declarativo, não da ferramenta, estendendo-se a qualquer motor que o compartilhe.

As três plataformas foram avaliadas contra T1 a T6 em configuração padrão: Suricata 6.0.10 com o *ruleset* Emerging Threats Open (49.604 assinaturas), Snort 3.12.1.0 com o *ruleset community* (4.664 regras), e Zeek 8.1.2 com o pacote `local`. A avaliação em configuração padrão fixou o Suricata na série 6.0.10, a imagem estável do laboratório; o exercício de portabilidade da Seção 9 usa a série 7.0.x, e a ausência de cobertura para opções EDNS relatada a seguir não se altera entre as duas séries, por decorrer de ausência de assinaturas e não de versão do *engine*. O payload exfiltrado foi um arquivo neutro de 1279 bytes. No Suricata, após validação da saúde do *pipeline* via controle positivo (assinatura ET DNS Query for vpnoverdns.com), as seis técnicas geraram 194 consultas DNS e **zero alertas**. O Snort processou os doze PCAPs sem produzir alertas de assinatura. O Zeek *stock* produziu `dns.log` com 26 colunas dedicadas ao protocolo, mas **nenhuma referente a opções EDNS**; sob configuração customizada (`redef dns_skip_all_addl = F` mais script para registrar opcionais), apenas o código 10 (Cookie) tornou-se visível, permanecendo os códigos 12, 13 e 65001 invisíveis ao *script-land* ainda que presentes nos pacotes.

A ausência uniforme de detecção, com *rulesets* da ordem de dezenas de milhares de regras, não decorre de configuração incorreta. A causa é ausência de cobertura: nenhuma das três plataformas em configuração padrão possui regras ou *parsers* que enxerguem o abuso de opções EDNS como canal encoberto. O caso do Zeek customizado merece investigação à parte, pois nele a barreira não está na ausência de regras, e sim na ausência de eventos sobre os quais escrevê-las. É essa investigação que a seção seguinte relata.

5. Lacuna estrutural do parser EDNS do Zeek 8.1.x

A invisibilidade dos códigos 12, 13 e 65001 ao *script-land* do Zeek motivou a inspeção do código-fonte. O arquivo `Zeek_DNS.events.bif.zeek` expõe apenas cinco eventos BIF relacionados a EDNS, dos quais só três enxergam opções individuais por código: `dns_EDNS_ecs` (código 8, RFC 7871 [Contavalli et al. 2016]), `dns_EDNS_cookie` (código 10, RFC 7873 [Eastlake and Andrews 2016]) e `dns_EDNS_tcp_keepalive` (código 11, RFC 7828 [Wouters et al. 2016]); os demais (`dns_EDNS`, depreciado, e `dns_EDNS_addl`) são genéricos, sem dimensão por código.

A inspeção do *parser* nativo confirma a hipótese: opções com códigos fora de $\{8, 10, 11\}$ são descartadas antes que qualquer evento seja emitido. Trata-se de lacuna reconhecida pelo próprio projeto, que a registra por meio do comentário `# TODO: figure out how to handle these` em `base/protocols/dns/main.zeek`. A consequência operacional é direta: como três das seis técnicas (T1, T2 e T5) usam códigos fora desse conjunto, a ausência de detecção observada na Seção 4 decorre de *gap* de capacidade, e não de má configuração. Corrigi-la exige, portanto, intervenção no mecanismo de *parsing*, e não no conjunto de regras.

6. Plugin Spicy para enumeração arbitrária de opções

A solução adotada para preencher a lacuna foi um plugin Spicy paralelo ao *analyzer* DNS nativo do Zeek (originalmente publicado como Bro [Paxson 1999]). Spicy [Sommer et al. 2016] é um *framework* de geração de *parsers* integrado ao Zeek, que

compila descrições declarativas em código nativo carregado em tempo de execução via arquivo `.hlto`. A escolha por Spicy em vez da modificação do *parser* DNS em C++ se justifica pela menor superfície de manutenção, pelo *packaging* via `.hlto` e pela facilidade de reprodução por terceiros, já que o código do plugin cabe em três arquivos. A omissão da cláusula `replaces` no arquivo `.evt` faz com que o plugin coexista com o *analyzer* nativo, já que ambos recebem o mesmo tráfego UDP/53, preservando o `dns.log` *stock* sem regressão. Não identificamos plugins públicos voltados para enumeração de opções EDNS0.

6.1. Decisões de arquitetura

Duas decisões de arquitetura merecem registro. Primeiro, a **granularidade dupla**: além do evento por opção EDNS (idiomático em relação aos eventos BIF), a v2 acrescenta um evento por pseudo-RR OPT, necessário para T4, que abusa do campo `udp_payload_size` do *header*. Segundo, o **registro por *hash***: como T1 a T6 carregam payload nas opções enumeradas, o plugin grava `opt_data_sha256` em vez do conteúdo bruto, preservando a identidade entre transações de que H3 depende sem expor o payload em disco.

6.2. Implementação e validação

O plugin consiste em três arquivos: o *parser* declarativo em Spicy (cerca de 120 linhas na v2, em modo *skip-only* que desce apenas para registros OPT na seção Additional), o mapeamento de eventos `.evt` (que gera `edns0_arbitrary_opt` por opção e `edns0_opt_record` por pseudo-RR, expondo `udp_payload_size`), e o *script* Zeek que grava os *logs* com *hashes* SHA-256.

A Tabela 3 compara a capacidade de enumeração do modo customizado (BIF nativo) com a do plugin sobre os doze PCAPs canônicos e, nas duas últimas colunas, apresenta observação preliminar do efeito do BIND9 sobre as opções no encaminhamento entre os dois *legs* de captura.

Tabela 3. Enumeração no *leg* interno (BIF, plugin v1/v2) e observação preliminar de remoção de opções pelo BIND9 entre os *legs*.

T	BIF	Plugin (eventos)	Exclusivos	Removidos pelo BIND9	Eventos (ext.)
T1	0, 10	10, 65001 (9)	65001	65001	4
T2	0, 10	10, 12 (9)	12	12	4
T3	0, 10	10 (165)	—	(nenhum)	55
T4	0, 10	10 (11)	—	(nenhum)	9
T5	0, 10	10, 13 (43)	13	13	21
T6	0, 10	10, 65001 (15)	65001	65001	4

O valor 0 na coluna BIF é sentinela indicando disparo do evento genérico `dns_EDNS_addl` sem código identificável. O plugin enumera os códigos 12, 13 e 65001, antes invisíveis ao *script-land*. O evento por pseudo-RR OPT, exclusivo da v2, produz contagem maior ou igual à contagem por opção (16, 11, 165, 12, 43 e 16 *records* para T1 a T6, respectivamente), porque pseudo-RRs OPT sem opções TLV geram evento por *record* mas não por opção.

6.3. Observação preliminar de *stripping* pelo BIND9

As duas últimas colunas da Tabela 3 evidenciam que códigos não-RFC (65001 em T1 e T6; 12 em T2; 13 em T5) são removidos pelo BIND9 do laboratório, comportamento que varia entre implementações [Internet Systems Consortium 2015, Internet Systems Consortium 2026]. Para T3 e T4, por outro lado, o conjunto de códigos é idêntico em ambos os *legs* e ainda assim há redução na contagem de eventos, provavelmente por *cache* interno e agregação de cookies pelo resolvedor. A consequência operacional é direta: a detecção de técnicas que abusam de códigos não-RFC precisa estar posicionada no *leg* interno, em coerência com o modelo adversarial (Seção 3.2). A caracterização sistemática desse comportamento entre resolvedores diversos, sob os múltiplos eixos de variabilidade documentados na literatura, consta como trabalho futuro prioritário (Seção 10).

7. Heurísticas de detecção

A inspeção dos *logs* do plugin sobre os PCAPs T1 a T6 revelou padrões consistentes em duas dimensões primárias do RDATA (código e comprimento da opção), uma dimensão auxiliar (recorrência de *hash* do conteúdo) e uma dimensão do *header* OPT (*UDP payload size*). Delas derivamos cinco heurísticas.

H1a (código fora da *whitelist*). Opção EDNS com código fora do conjunto $\{0, 3, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16\}$, que reúne o código 0 (sentinela do *parser*) e os códigos com alocação IANA [Internet Assigned Numbers Authority 2026] e uso observado em produção. A opção por *whitelist* explícita, em vez de limite numérico, decorre de códigos experimentais e de alocação alta coexistirem no espaço de valores; o código 13 (CHAIN, RFC 7901 [Wouters 2016]) é excluído por ser vetor de T5 e de uso raro. O conjunto é datado, e a Seção 10.2 trata da manutenção da *whitelist* frente a novas alocações. Cobre T1, T5 e o componente experimental de T6.

H1b (Padding com bytes não-zero). Opção de código 12 (Padding, RFC 7830 [Mayrhofer 2016]) com bytes diferentes de zero. A RFC 7830 recomenda preenchimento com 0x00 mas tolera valores arbitrários, e sua Seção 6 reconhece o conteúdo do Padding como canal encoberto potencial. Bytes não-zero não violam a especificação, mas configuram desvio do uso recomendado. Cobre T2.

H2 (comprimento anômalo de Cookie). Opção de código 10 (Cookie, RFC 7873 [Eastlake and Andrews 2016]) com comprimento fora de $\{8, 16, 24\}$. A RFC autoriza 8 e a faixa de 16 a 40, mas o conjunto adotado reflete a calibração empírica: 100% dos 58 cookies legítimos do *baseline* weberblog têm comprimento 8 ou 24. O PAM 2021 [Davis and Deccio 2021] corrobora de fonte independente (mais de 99% dos cookies de servidor observados têm 16 bytes, gerando opção de 24 somada ao cookie cliente), mas documenta implementações minoritárias de 32 bytes que produziriam falso-positivo, tratadas na Seção 10.2. Cobre T3 e o componente cookie de T6.

H3 (alta unicidade de *hash* em código 10 por sessão). Em janela de eventos de uma mesma conexão DNS (agregada por `c$uid`), a razão entre *hashes* únicos de `opt_data` e o total de eventos de código 10 supera 0,5, com mínimo de 5 eventos. Cookies legítimos recorrem dentro da sessão e produzem razão baixa; cookies usados para exfiltração têm conteúdo único por *chunk* e produzem razão alta. Cobre o cenário com-

plementar a H2 em que o adversário mantém o comprimento nominal e varia o conteúdo, materializado pela T3-variante.

H4 (variabilidade anômala de *UDP payload size*). Diferente das anteriores, H4 opera sobre o campo `udp_payload_size` do *header* do pseudo-RR OPT, exposto pelo evento `edns0_opt_record` da v2. A regra agrega por `id$orig_h` (estado por origem entre conexões, necessário porque *stubs* modernos não reusam *source port*) e dispara quando a razão entre valores únicos de `udp_payload_size` e total de eventos supera 0,5, com mínimo de 5 eventos. Clientes legítimos anunciam valor fixo ou em conjunto pequeno (tipicamente $\{512, 1232, 1452, 4096\}$), com razão $< 0{,}05$; um adversário que codifica dados no campo eleva a razão. A avaliação é incremental, disparando no primeiro evento em que a razão cruza o limiar, o que a Seção 8 quantifica em quatro regimes. Cobre T4. É Zeek-*only* por desenho, pois estado por origem entre conexões está fora do alcance da *rule-language* pura (Seção 9).

No conjunto, as cinco heurísticas cobrem as seis técnicas mais a T3-variante: T1, T5 e o componente experimental de T6 por H1a; T2 por H1b; T3 e o componente cookie de T6 por H2; T3-variante por H3; T4 por H4.

8. Validação experimental

8.1. Implementação das regras Zeek

As cinco heurísticas foram materializadas como *scripts* Zeek associados aos eventos do plugin (`edns0_arbitrary_opt` para H1a–H3; `edns0_opt_record` para H4), cada um emitindo um `Notice::Type` dedicado. A deduplicação usa `c$uid` para H1a–H3 e `id$orig_h` para H4, pela razão da Seção 7. O código completo está no repositório.

8.2. Corpora de validação

Não identificamos *dataset* público que combine volume operacional com preservação *full-packet* do pseudo-RR OPT: weberblog [Weber 2019] preserva o OPT mas amostra poucas dezenas de opções, enquanto o CIC-Bell-DNS-EXF 2021 [Mahdavifar et al. 2021] oferece 2,5 milhões de pacotes com *snaplen* de 96 bytes, inadequado ao propósito. A validação usou portanto quatro corpora primários complementares: (i) o corpus adversarial principal, com doze PCAPs cobrindo T1 a T6 nos *legs* interno e externo; (ii) o *baseline* EDNS legítimo weberblog (três PCAPs; 27 eventos de opção e 38 pseudo-RRs OPT enumerados pelo plugin); (iii) o *baseline* DNS amplo do subconjunto benigno do CIC-Bell-DNS-EXF 2021 (18 PCAPs, 2,5 milhões de pacotes), do qual o plugin não emite eventos porque o conjunto não carrega EDNS (Seção 2), medindo apenas que o *pipeline* não produz alertas espúrios em volume; e (iv) um *baseline* sintético de 50.000 consultas DNS contra os resolvedores públicos Cloudflare, Google e Quad9 em rotação aleatória, usando `dig +edns +cookie` contra domínios da Tranco [Le Pochat et al. 2019] top-1000 e captura com `snaplen 0`, resultando em 55.343 opções EDNS. Um quinto corpus, dedicado à T3-variante, consiste em 50 PCAPs com payload aleatório entre 256 e 4096 bytes e *source port* fixo, caso adversarial realista de socket UDP persistente, necessário porque H3 agrega por `c$uid`. Um sexto corpus, dedicado à T4-variante, reúne 200 PCAPs em quatro regimes adversariais de variação de `udp_payload_size` e fornece *ground truth* para H4.

O corpus sintético contém 50.000 cookies cliente (código 10, comprimento 8) e 5.343 opções de código 15 (EDE). Essa composição produziu correção empírica relevante: $\sim$24% das respostas carregavam código 15, não incluído na *whitelist* inicial $\{0, 8, 10, 11, 12\}$, que foi então expandida antes da bateria principal sem alterar a revocação em T1, T5 e T6. Isso exemplifica o valor de calibrar contra tráfego real, e não apenas contra o corpus adversarial.

8.3. Resultados

A Tabela 4 apresenta as contagens de *notices* por regra contra cada PCAP do corpus adversarial.

Tabela 4. *Notices* emitidos por regra, *leg* interno dos corpora adversariais. Detalhamento de T4-variante por regime na Tabela 5.

PCAP	H1a	H1b	H2	H3	H4	Eventos	Cobertura
T1-internal	3 (65001)	0	0	0	0	9	H1a
T2-internal	0	3	0	0	0	9	H1b
T3-internal	0	0	54	0	0	165	H2
T4-internal	0	0	0	0	1	12 (*records*)	H4
T5-internal	20 (13)	0	0	0	0	43	H1a
T6-internal	3 (65001)	0	3	0	0	15	H1a + H2
T3-variante (50)	0	0	0	104	0	*cookie events*	H3
T4-variante (200)	0	0	0	0	150	vide Tab. 5	H4

A H3, validada contra o corpus T3-variante, registrou 104 *notices* agregados nos 50 PCAPs, com revocação de 50/50. A média de $\sim$2 *notices* por PCAP decorre da granularidade de `c$uid`: o Zeek aplica *timeouts* de inatividade ao tráfego DNS sobre UDP e abre novo *uid* após o *flow* ser fechado, mesmo com *five-tuple* fixo. A H4 foi validada contra um corpus T4-variante com 200 PCAPs em quatro regimes de variação de `udp_payload_size` (50 PCAPs por regime; payload aleatório entre 256 e 4096 bytes; *source port* fixo por execução), sumarizados na Tabela 5. R1 (sorteio uniforme em $[512, 4096]$ a cada consulta) e R3 (valores canônicos com perturbação de ± 10 bytes) acionam H4 em todos os 50 PCAPs de cada regime, com razões médias de 0,966 e 0,335. R2 (alternância entre 1232 e 4096) produz razão assintótica $2/N$ e não excede o limiar por construção; o regime confirma que clientes que oscilam entre dois valores canônicos durante o *warmup* de sessão não geram falso-positivo, e fica declarado fora do escopo de H4 (Seção 10). O regime R4 é o mais informativo: projetado como ataque consciente do detector, repete cada valor duas vezes consecutivas antes de trocar, mirando razão final $\approx 0{,}5$. A razão mediana observada é 0,492, abaixo do limiar; ainda assim H4 dispara em todos os 50 PCAPs, porque a avaliação incremental alcança o padrão A, A, B, B, C na janela dos cinco primeiros eventos, onde a razão vale $0{,}6$, antes da convergência. Para evadir H4 sob o limiar 0,5, o adversário precisaria repetir cada valor ao menos três vezes, reduzindo proporcionalmente a banda de codificação disponível no campo. Os três corpora benignos preexistentes produzem zero *notices* de H4, confirmando precisão 1,00 após a ampliação.

Tabela 5. Validação de H4 em quatro regimes adversariais de variação de `udp_payload_size` (50 PCAPs por regime). As razões de unicidade agregam por `id$orig_h`, considerando apenas consultas. Veredito de *cobertura completa* quando H4 dispara em ao menos 95% dos PCAPs do regime.

Regime	H4 disp.	Notices	Razão méd.	Razão med.	Veredito
R1 contínuo	50/50	50	0,966	0,967	cobertura completa
R2 saltos canônicos	0/50	0	0,009	0,007	não dispara (esperado)
R3 perturbado	50/50	50	0,335	0,248	cobertura completa
R4 *detector-aware*	50/50	50	0,493	0,492	limiar 0,5 sobrevive

Consolidando por heurística, tomando o PCAP como unidade de decisão e contando falsos-positivos sobre a união dos três corpora benignos (55.370 opções EDNS, 55.343 em modo *full-packet*): cada heurística atinge precisão e revocação de 1,00 nas suas técnicas (H1a sobre T1/T5/T6, H1b sobre T2, H2 sobre T3/T6, H3 sobre as 50 PCAPs de T3-variante, H4 sobre as 151 PCAPs de T4 e T4-variante R1/R3/R4), com agregado de 207 verdadeiros-positivos, zero falsos-negativos e zero falsos-positivos, com precisão 1,00 e revocação 1,00, superando o critério de no máximo dois alertas benignos em volume fixado a priori. Contabilizar o regime R2, declarado fora de escopo, reduziria a revocação de H4 a 0,75; nenhuma outra métrica se altera.

9. Portabilidade da regra para Suricata

Para verificar a portabilidade conceitual das heurísticas, implementamos H1a também em Suricata 7.0.x, limitando o exercício a essa heurística por três razões. H1b exige inspeção de *buffer* de comprimento variável, factível em *rule-language* mas com perda de precisão. H2 é portável com `byte_test` análogo, sem novidade demonstrativa. H3 e H4, por fim, exigem estado por entidade (`c$uid` e `id$orig_h`, respectivamente) acumulado ao longo de múltiplos eventos antes de produzir decisão, o que está fora do alcance da *rule-language* pura, que avalia cada pacote em isolamento. As duas formam, portanto, exemplos complementares de um mesmo limite estrutural, retomado ao final desta seção.

9.1. Versão inicial e seu fracasso

A primeira versão da regra (identificada como v2 na Tabela 6, por suceder um protótipo inicial descartado) usou *content match* sobre o pseudo-RR OPT seguido de `byte_test` para verificar que o código da opção não pertencia à *whitelist*. Sintaticamente válida e conceitualmente análoga à H1a do Zeek, ela produziu 532 falsos-positivos contra os 55.343 eventos do *baseline* sintético. A investigação revelou *matching* espúrio: a sequência `|00 29|`, que codifica o tipo OPT no DNS, aparecia em payloads de respostas portadoras de EDE com texto explicativo curto (comprimento 37), e o `byte_test` relativo lia em *offset* onde valores fortuitos passavam em todos os 13 testes `!=N`.

9.2. Versão refinada com âncora semântica

A segunda versão (v3) adicionou validação do campo CLASS do pseudo-RR OPT. A RFC 6891 [Damas et al. 2013] estabelece que o CLASS de um OPT carrega o `udp_payload_size`, valor entre 512 (mínimo histórico) e 4096 (máximo típico em produção). Adicionamos, por isso, dois `byte_test` antes dos testes de código, verificando que o valor do CLASS é maior ou igual a 512 e menor ou igual a 4096. Os

dois testes eliminaram completamente os 532 falsos-positivos, alcançando paridade de precisão com o Zeek em todos os corpora, conforme a Tabela 6.

Tabela 6. Comparação entre H1a em Zeek, Suricata v2 e Suricata v3.

Implementação	VP adv.	FP weberblog	FP CIC	FP sintético
Zeek (plugin Spicy)	26	0	0	0
Suricata v2 (*content only*)	23	0	0	532
Suricata v3 (*content* + CLASS)	23	0	0	0

9.3. Discussão do *gap* de revocação

Ainda que a v3 atinja precisão equivalente à do Zeek, resta um *gap* de três alertas em revocação, correspondentes inteiramente à técnica T6, que enfileira código 10 (na *whitelist*) e código 65001 (fora dela) no mesmo pseudo-RR OPT. O `byte_test` relativo do Suricata inspeciona apenas a primeira opção após o *content match*: identifica o código 10, encontra-o na *whitelist* e não dispara, de modo que o código 65001 seguinte permanece invisível. O Zeek com plugin Spicy, ao emitir um evento por opção enumerada, alcança ambas.

Portar H3 e H4 via *Lua scripting* (Turing-completo) seria tecnicamente factível, mas converteria a regra em par regra-mais-*script*, com superfície de manutenção próxima da do Zeek; o exercício metodologicamente relevante é caracterizar o que a regra como artefato declarativo único alcança e o que não alcança. A conclusão operacional é que precisão é portável mediante refinamento, mas revocação em cenários de múltiplas opções por OPT e em heurísticas com estado por entidade tem teto estrutural na *rule-language* pura. Defensivamente, recomenda-se Zeek com plugin como detector primário, por cobertura completa, e Suricata v3 como camada secundária com cobertura parcial.

10. Discussão

10.1. Limitações reconhecidas

Cobertura restrita a DNS sobre UDP. O escopo `port 53/udp` decorre do modelo de ameaça: o observador da Seção 3.2 está no *leg* interno, onde o tráfego permanece em Do53 sempre que a política de rede bloqueia resolvedores externos, precondição comum em ambientes gerenciados. DoT e DoH não alteram o vetor, e sim a sua observabilidade: as opções EDNS0 continuam presentes na mensagem, e as heurísticas se aplicam sem modificação assim que o *pipeline* dispuser do texto claro. DNS sobre TCP é lacuna de implementação, não de modelo, exigindo apenas o tratamento do prefixo de comprimento de dois bytes, previsto como trabalho futuro.

***Baselines* limitados.** Os três *baselines* benignos utilizados são complementares, mas nenhum é por si só suficiente: weberblog é pequeno em volume, o CIC não carrega EDNS algum e por isso mede apenas ausência de alertas espúrios, e o sintético é filtrado por *proxy* de rede que suprime ECS e cookies de servidor. A inexistência de *dataset* público em volume com EDNS preservado é, em si, lacuna metodológica da área.

Custo computacional não avaliado. Não medimos o sobrecusto do plugin sobre a vazão do Zeek. Duas propriedades de projeto sugerem que ele é contido, ainda que sem

substituir medição: o *parser* opera em modo *skip-only*, descendo apenas para registros OPT, e o estado de H3 e H4 é limitado às entidades ativas na janela, sujeito aos *timeouts* do Zeek. A omissão é de escopo: o objeto do trabalho é caracterizar a lacuna e demonstrar que ela é removível, não otimizar o artefato que a remove.

Validação em laboratório e escopo do Suricata. A avaliação não inclui tráfego corporativo de produção, e o exercício de portabilidade avalia o Suricata pela sua *rule-language* pura, não pelo máximo de construções que o *engine* aceita; é escolha metodológica, pois o valor distintivo do modelo está na regra como artefato declarativo único.

10.2. Generalização, DoH, ECS e heterogeneidade de resolvedores

Generalização. As heurísticas dividem-se em dois grupos. H1a e H1b derivam de invariantes normativos (registro IANA e recomendação da RFC 7830) e independem do perfil de tráfego; sua transposição exige apenas que a *whitelist* de H1a acompanhe o registro IANA, que segue em expansão (os códigos 17 a 21 foram alocados após a calibração, dois com status *Standard*). H2, H3 e H4 dependem de limiares calibrados e demandam recalibração: para H2, o PAM 2021 [Davis and Deccio 2021] indica suporte pleno a cookies minoritário (menos de 20% dos resolvedores e IPs de TLDs), de modo que implementações idiossincráticas podem gerar comprimentos fora de $\{8, 16, 24\}$; para H3 e H4, ambientes com *forwarders* ou NAT em larga escala podem elevar a razão legítima. A recalibração, que coleta tráfego benigno e reestima percentis, é pré-requisito de *deployment*.

DoH. Sob DNS-over-HTTPS o tráfego fica opaco à inspeção passiva, exigindo *pipeline* com *interception* TLS. O DoH ainda altera o tratamento de opções EDNS em pelo menos um resolvedor de grande porte: o Google Public DNS rejeita via Do53 consultas com ECS de prefixo mais específico que /24, mas as aceita via DoH truncando para /24 [Farrokhi 2025].

ECS. A opção ECS (código 8, RFC 7871 [Contavalli et al. 2016]) é sétima técnica plausível, não implementada porque seu perfil difere das T1 a T6: o código já é visível ao *parser* nativo do Zeek e sobrevive ao encaminhamento em parte dos cenários. Levantamento de dez resolvedores públicos [Farrokhi 2025] mostra comportamento heterogêneo (metade encaminha ECS para autoritativos, apenas 20% o ecoa nas respostas). ECS é problema de heurística aplicada, não de *parsing*, e pertence a trabalho irmão.

10.3. Trabalho futuro e implicações para defensores

Trabalho futuro. Identificamos cinco prioridades. A primeira é a caracterização formal do limite expressivo da *rule-language* pura para detecção com estado por entidade, transformando o achado empírico da Seção 9 em proposição formal. A segunda é a implementação do vetor ECS como T7, com medição de seu regime de preservação em resolvedores públicos. A terceira é a caracterização sistemática da remoção de opções EDNS entre múltiplos resolvedores (BIND9, Unbound, Knot, Cloudflare, Google, Quad9) e múltiplos códigos IANA, estendendo a observação preliminar da Tabela 3 para identificar o regime sob o qual o canal sobrevive até o autoritativo. A quarta é a extensão do escopo para DNS sobre TCP e para DoH via *pipeline* com *interception*, acompanhada de heurística complementar a H4 para adversários que repetem cada valor de `udp_payload_size` três ou mais vezes, regime que evade o limiar atual e demandaria medida baseada em

entropia da sequência ou *jitter* temporal intra-sessão. A quinta é a construção de *dataset* público em volume com EDNS preservado em modo *full-packet*, cuja ausência foi obstáculo recorrente neste trabalho.

Implicações operacionais. A combinação do plugin Spicy v2 com as cinco regras Zeek fornece detecção em três fronteiras complementares: estrutural sobre opções TLV, estatística sobre conteúdo de cookies por sessão, e estrutural sobre o *header* do pseudo-RR OPT. Defensores que adotem o conjunto obtêm cobertura para as técnicas conhecidas no modelo adversarial da Seção 3.2, com as ressalvas de calibração, DoH e ECS acima. A portabilidade para Suricata, ainda que parcial, é operacionalmente útil: a regra v3 oferece detecção sem dependência adicional para organizações que já operam Suricata. Código, regras, módulo da T3-variante e inventário com *hashes* SHA-256 dos corpora estão disponíveis em repositório público sob licença compatível com a do Zeek; os mantenedores do projeto Zeek foram notificados sobre a lacuna documentada na Seção 5.

11. Conclusão

Este trabalho caracterizou canais encobertos via opções EDNS0 como vetor adversarial sub-explorado e produziu artefatos para fechar a lacuna. A avaliação empírica de Suricata, Snort e Zeek registrou zero detecções contra as seis técnicas do gerador EDNStego, e a inspeção do código-fonte atribuiu esse resultado a uma lacuna estrutural do *parser* EDNS do Zeek 8.1.x, e não a configuração inadequada. O plugin Spicy `edns0-arbitrary` v2 remove essa lacuna ao expor eventos por opção e por pseudo-RR OPT, sobre os quais derivamos cinco heurísticas validadas em seis corpora, com precisão e revocação de 1,00 nas técnicas cobertas e zero falsos-positivos em 55.343 opções *full-packet* capturadas contra resolvedores públicos de produção. O exercício de portabilidade para Suricata 7, por fim, identificou um teto estrutural de revocação na *rule-language* pura para cenários de múltiplas opções por OPT e para heurísticas com estado por entidade.

Dois desses resultados são proposições sobre os mecanismos de detecção, e não sobre a amostra observada: a lacuna de *parsing* e o teto da *rule-language*. As direções prioritárias de continuação, listadas na Seção 10, destacam a caracterização formal desse teto e a implementação do vetor ECS como sétima técnica.

Referências

Aiello, M., Mongelli, M., and Papaleo, G. (2013). Basic Classifiers for DNS Tunneling Detection. In *Proceedings of the 18th IEEE Symposium on Computers and Communications (ISCC)*, pages 880–885, Split, Croatia.

Born, K. and Gustafson, D. (2010). Detecting DNS Tunnels Using Character Frequency Analysis. arXiv:1004.4358 [cs.CR]. Apresentado em 9th Annual Security Conference, Las Vegas, NV. Disponível em: https://arxiv.org/abs/1004.4358. Acesso em: maio de 2026.

Buczak, A. L., Hanke, P. A., Cancro, G. J., Toma, M. K., Watkins, L. A., and Chavis, J. S. (2016). Detection of Tunnels in PCAP Data by Random Forests. In *Proceedings of the 11th Annual Cyber and Information Security Research Conference (CISRC)*. ACM.

Contavalli, C., van der Gaast, W., Lawrence, D., and Kumari, W. (2016). Client Subnet in DNS Queries. RFC 7871, Internet Engineering Task Force.

Damas, J., Graff, M., and Vixie, P. (2013). Extension Mechanisms for DNS (EDNS(0)). RFC 6891, Internet Engineering Task Force.

Davis, J. and Deccio, C. (2021). A Peek into the DNS Cookie Jar: An Analysis of DNS Cookie Use. In *Proceedings of the Passive and Active Measurement Conference (PAM)*, volume 12671 of *Lecture Notes in Computer Science*, pages 302–316. Springer.

Eastlake, D. and Andrews, M. (2016). Domain Name System (DNS) Cookies. RFC 7873, Internet Engineering Task Force.

Farrokhi, B. (2025). EDNS Client Subnet in Practice: Evaluating Public Resolver Behaviors. Disponível em: https://farrokhi.net/posts/2025/10/edns-client-subnet-in-practice-evaluating-public-resolver-behaviors/. Acesso em: maio de 2026.

Internet Assigned Numbers Authority (2026). DNS EDNS0 Option Codes (OPT). Registro DNS Parameters. Disponível em: https://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml#dns-parameters-11. Acesso em: julho de 2026.

Internet Systems Consortium (2015). Partial EDNS Compliance Hampers Deployment of New DNS Features. ISC Blog. Disponível em: https://www.isc.org/blogs/partial-edns-compliance-hampers-deployment-of-new-dns-features/. Acesso em: maio de 2026.

Internet Systems Consortium (2026). EDNS Compliance Reports. Disponível em: https://ednscomp.isc.org/compliance/. Acesso em: maio de 2026. Relatórios automatizados periódicos sobre conformidade EDNS de servidores autoritativos e resolvedores recursivos, publicados desde 2016.

Kumari, W., Hunt, E., Arends, R., Hardaker, W., and Lawrence, D. (2020). Extended DNS Errors. RFC 8914, Internet Engineering Task Force.

Lambion, D., Josten, M., Olumofin, F., and De Cock, M. (2020). Malicious DNS Tunneling Detection in Real-Traffic DNS Data. In *Proceedings of the 2020 IEEE International Conference on Big Data (Big Data)*, pages 5736–5738. IEEE.

Le Pochat, V., Van Goethem, T., Tajalizadehkhoob, S., Korczyński, M., and Joosen, W. (2019). Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium (NDSS)*.

Mahdavifar, S., Salem, A. H., Victor, P., Razavi, A. H., Garzon, M., Hellberg, N., and Lashkari, A. H. (2021). Lightweight Hybrid Detection of Data Exfiltration using DNS based on Machine Learning. In *Proceedings of the 11th International Conference on Communication and Network Security (ICCNS)*, pages 80–86, Weihai, China. ACM. Dataset CIC-Bell-DNS-EXF 2021 disponível em https://www.unb.ca/cic/datasets/dns-exf-2021.html.

Mayrhofer, A. (2016). The EDNS(0) Padding Option. RFC 7830, Internet Engineering Task Force.

Nadler, A., Aminov, A., and Shabtai, A. (2019). Detection of Malicious and Low Throughput Data Exfiltration over the DNS Protocol. *Computers & Security*, 80:36–53.

Paxson, V. (1999). Bro: A System for Detecting Network Intruders in Real-Time. *Computer Networks*, 31(23–24):2435–2463.

Rijs, S. (2014). Combating DNS Amplification Using Cookies. Master's thesis, University of Amsterdam, OS3 Master's Programme. Supervisor: Roland van Rijswijk-Deij. Disponível em: https://rp.os3.nl/2013-2014/p64/report.pdf.

Sommer, R., Amann, J., and Hall, S. (2016). Spicy: A Unified Deep Packet Inspection Framework for Safely Dissecting All Your Data. In *Proceedings of the 32nd Annual Computer Security Applications Conference (ACSAC)*, pages 558–569. ACM.

ttpreport (2025). SiphonDNS: Covert Data Exfiltration via DNS. Blog post e ferramenta. Disponível em: https://ttp.report/evasion/2025/02/03/siphondns-covert-dns-exfiltration.html e https://github.com/ttpreport/siphondns. Acesso em: maio de 2026.

Waleed, A., Jamali, A. F., and Masood, A. (2022). Which Open-Source IDS? Snort, Suricata or Zeek. *Computer Networks*, 213:109116.

Weber, J. (2019). DNS Capture: UDP, TCP, IP-Fragmentation, EDNS, ECS, Cookie. weberblog.net. Disponível em: https://weberblog.net/dns-capture-udp-tcp-ip-fragmentation-edns-ecs-cookie/. Acesso em: maio de 2026.

Wouters, P. (2016). CHAIN Query Requests in DNS. RFC 7901, Internet Engineering Task Force.

Wouters, P., Abley, J., Dickinson, S., and Bellis, R. (2016). The edns-tcp-keepalive EDNS0 Option. RFC 7828, Internet Engineering Task Force.

Žiža, K., Tadić, P., and Vuletić, P. (2023). DNS Exfiltration Detection in the Presence of Adversarial Attacks and Modified Exfiltrator Behaviour. *International Journal of Information Security*, 22(6):1865–1880.