URLink: Using Names As Sole Internet Addresses to Tackle Scanning Attacks in IoT

Quentin De Coninck*
quentin.deconinck@umons.ac.be
University of Mons
Mons, Belgium

Shirin Kalantari*
shirin.kalantari@kuleuven.be
DistriNet, KU Leuven
Leuven, Belgium

Laurens Sion
laurens.sion@kuleuven.be
DistriNet, KU Leuven
Leuven, Belgium

Danny Hughes
danny.hughes@kuleuven.be
DistriNet, KU Leuven
Leuven, Belgium

Abstract

The Internet adopts a layered architecture where IP addresses are used to identify endpoints and port numbers serves as application multiplexers over a single host. Nowadays, names are usually used to expose a service to public access. However, even with the current DNS architecture, nodes must still know what the running host's IP address and service's port number are to access the service. In fact, any node can directly contact a publicly available node, sometimes for other purposes than accessing its public services. This is specially a challenge in IoT as highlighted by numerous high-profile DDoS attacks which leverage Internet scanning to find vulnerable IoT nodes. Defending against this is often a challenge for service operators. This paper questions this current architecture and calls for an alternative called URLink, where names are used as the sole identifier and access door towards a service. Through a new network abstraction called URLSocket, clients are no longer aware of the public service's IP address and port number. We argue that such an approach is beneficial for IoT networks, as it can be used to address various security and privacy issues in these network. While such an architecture calls for changes in the client application stacks, existing applications (e.g. those running on an IoT node) can still leverage the proposed system in the current Internet.

CCS Concepts

• **Networks** → Network privacy and anonymity; *Network manageability*; **Denial-of-service attacks**; • **Security and privacy** → *Malware and its mitigation.*

Keywords

Scanning attack, DoS, QUIC

*Both authors equally contributed to this research.

ACM Reference Format:
Quentin De Coninck, Shirin Kalantari, Laurens Sion, and Danny Hughes. 2023. URLink: Using Names As Sole Internet Addresses to Tackle Scanning Attacks in IoT. In *The 21st ACM Conference on Embedded Networked Sensor Systems (SensorsS&P), November 12–17, 2023, Istanbul, Turkiye.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3628356.3630115

1 Introduction

Our cyberspace, historically dominated by personal computing devices, is now being shared with IoT devices of all shapes and sizes at a growing rate. It is estimated that the number of IoT connections exceeded 13 billion in 2022 [13, 38]. With each device having an autonomous network identity and a significant number of them lacking basic security capabilities [26, 34, 39], IoT devices have been linked to numerous security incidents [9, 15, 25, 27, 28, 31, 33, 35, 42]. Scanning attacks, where entities want to identify all the running services over the Internet [7, 11, 12], has been the stepping stone for many of these security incidents. In addition, based on the network packets' metadata, a monitoring third party may infer the running application and the types of IoT devices in a network [1, 36], raising privacy concerns.

In this paper, we question OSI model which rely on IP addresses and port numbers for identify nodes and services. We propose a new network abstraction called URLSocket, in which the clients of a service (e.g., an IoT node) are no longer aware of the public service's IP address and port number. Instead, URLSocket takes names as the sole identifiers of an application service over the Internet. Our key insight is that by eliminating the use of the tuple (IP address, port number) as the application identifier of a service, we make the network easier to manage, less complex and less exposed to scanning attacks. While this paper is not the first proposal for a name-based addressing for the Internet [16, 24, 29], our proposal harnesses the potentials of QUIC, a recent addition to the networking landscape. This paradigm shift in networking protocols enable us to build a solution that is compatible with the current Internet architecture and incorporates security, exceptional agility, and robust performance compared to previous works.

We argue that such an approach is beneficial for IoT networks, as it tackles various security and privacy issues. Solving IoT security problem is a formidable challenge because of the peculiar characteristics of these devices, including limited power and computational resources, and difficulties in modifying the software components running on these devices. While earlier works called for such named-centric network management [17, 24, 30] we argue that the recent introduction of encrypted protocols such as QUIC [21] makes this idea eventually practical. The modifications that URLSocket imposes are all within the user space (i.e. the client application stacks) which leaves services running on IoT nodes unmodified. We first discuss how applications became dependent on the tuple (IP address, port number) in Section 2.1. We then propose our system, URLink, in Section 3 and discuss its advantages for the IoT ecosystem in Section 4. We finally discuss deployment considerations and early evaluations in Section 5 and conclude the paper in Section 7.

2 Background

2.1 The Socket Architecture

Building a large-scale network such as the Internet, containing heterogeneous devices and technologies, was complex and took years. To achieve this, network designers adopted a layered approach known as the OSI model to abstract this heterogeneity and make distant communication with devices using different technologies (Wi-Fi, cellular, Ethernet, . . .) possible. Establishing such an abstraction required defining interfaces between the different network layers. Driven by its evolution, the Internet chose to rely on IP addresses to identify nodes. However, it is often unpractical for applications to directly rely on a numerical IP address to contact a specific device. Instead, users prefer to rely on human-readable names, making the DNS infrastructure be a key element of the Internet connectivity to resolve names into IP addresses. Still, several application flows can take place between the same devices. The multiplexing of such different data streams is handled by the transport protocols that add port numbers. Today, application developers are mostly stuck with two main options: either TCP if the service needs a reliable bytestream, or UDP otherwise.

Because networked applications are programs, they need some interface with the network. In line with the layered approach at that time, the established Socket API provides a application network channel, given the IP address and port number. Yet, many applications, such as Web browsing, rely on names to identify a service. To initiate communication to such a named service, client applications follow the process as depicted in Figure 1. To find out the device on which the service is running, the client application first needs to resolve the name into an IP address. Under the hood, the device will have a socket sending UDP packets to an IP address known to run a DNS resolver over port 53. The DNS response then provides the IP address that the application should contact to fetch the service. For this, it creates a socket over the resolved IP address along with the transport protocol used and the associated port number. Both the protocol and the port numbers are determined in an out-of-band way, the application often defining the transport protocol and port number on which the service runs. This ensures that any client can seamlessly contact a given public service. Once the socket is created, the application can then exchange data over the socket, making a flow with the remote service being contacted.

While this process drove the creation of exchanges on the Internet, we argue that it has flaws for both clients and service operators. For each of the contacted a service, besides its name, the client

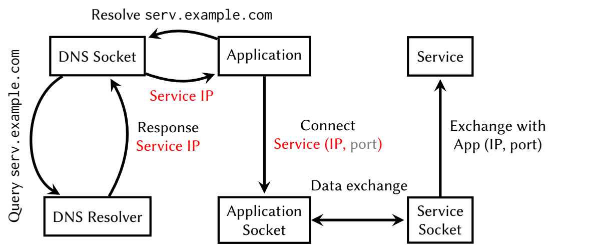

Figure 1: The current communication architecture using the Socket API. The application needs to know the IP of the service it wants to reach, the port number often being defined by the application itself.

first requires the IP address on which it runs through the DNS. Another downside relies on the arbitrary allocation of port numbers to specific services. Running multiple web services on a single server is common, and the handling of these services running on the same port needs to be performed at higher layer, e.g., by relying on the TLS Server Name Indication (SNI) [40] or the HTTP Host header [14]. For service operators, running services on a port number different from the IANA assigned one requires some out-of-band communication with clients to let them discover them (e.g., the SRV records in DNS [18]). All these elements (IP resolution, TLS SNI, HTTP Host, port number attribution, . . .) introduce both complexity and potential points of failure.

To provide the required service, we believe the Internet should adopt the principle of "the less said, the better". Currently, in addition to the service name, the clients know its running host's IP address and its running port. Such information does not only expose the application service, but also the running host itself. This may pose security issues by enabling attacks on other services running on that host. This also enables scanning attacks where entities want to identify all the running services over the Internet [7, 11, 12]. To tackle these issues, when a service needs to be available to a restricted user set on a potentially public network, a workaround consists in running a service on a different port number. Such an approach is often applied to, e.g., SSH [22]. However, in addition to the limitations described before, with the increasing network speed and the limited 32-bit wide addressing space of IPv4, recent works [23] show that those services can still be discovered, even though they might use non-conventional ports. IPv6 offers a much larger 128-bit addressing space. Still, researchers managed to build target lists of IPv6 addresses and there are now efficient IPv6 scanners [4].

2.2 The QUIC Protocol

QUIC [21] is a recently standardized transport protocol providing the services of TCP (reliability) and TLS (encryption) atop UDP. Although initially designed for HTTP/3 [5], QUIC allows explicit application protocol negotiation, enabling broader use cases [20]. Unlike TCP whose whole header is visible by the network entities, the built-in QUIC encryption leaves minimal clear-text metadata to the network — only a few flags and a Connection ID. The encrypted part of QUIC packets contains the core QUIC messages called *frames*. While application data can be reliably exchanged through STREAM

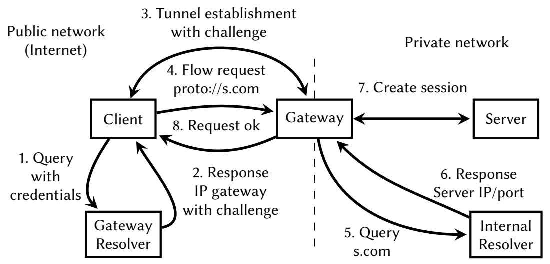
3. Tunnel establishment with challenge
Public network (Internet)
Private network
4. Flow request proto://s.com
7. Create session
Client
8. Request ok
Gateway
Server
1. Query with credentials
2. Response IP gateway with challenge
6. Response Server IP/port
Gateway Resolver
5. Query s.com
Internal Resolver

Figure 2: Establishing a URLink session and creating a reliable flow to `proto://s.com`.

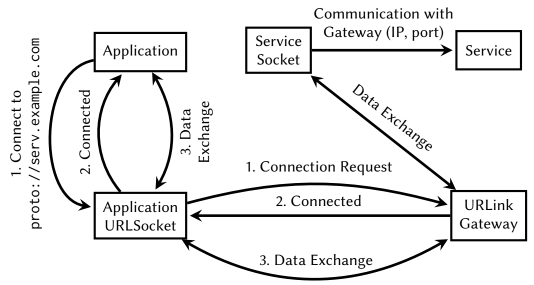

Figure 3: URLSockets Architecture. The client URLSocket only requires the IP address of the gateway resolver.

frames, there are many other control frames such as ACK (to acknowledge packets for loss recovery) and MAX_DATA (to advertise the receive buffer size) frames. In addition, QUIC negotiates extensions during the connection handshake. For instance, the Datagram extension [32] enables endpoints to send data in a unreliable way through DATAGRAM frames, providing a UDP-like service with a QUIC-provided encryption layer. There are more than 20 publicly known QUIC implementations and QUIC represents more than a third of the Google network traffic on the Internet [6].

3 The URLink Paradigm

This paper questions two core design decisions of the current Internet. The first relates to the arbitrary link between applications and (destination) port numbers. The second concerns the communication meta data that a client needs to know (such as the IP address and port number) to access some named service, such as a web page. Such public network information exposition introduces concerns about unwanted communications such as bot traffic [41].

To address these, this paper proposes a paradigm shift by abstracting applications from the IP addresses and port numbers used. The introduced *URLink* enables them to rely on names, and more specifically, URLs, as human-readable service identifiers across a network. Such services would run in a private network only reachable through a *gateway*. Figure 2 illustrates how the URLink connection process takes place. To prevent clients from directly accessing the servers, those are deployed in a private network, directly unreachable from the public one. Clients can still access the services running on these servers, but only by passing through gateways connected to both networks. There may be multiple gateway instances deployed in the network. To find out these gateways, *gateway resolvers* in URLink are deployed in the public network. In URLink, these gateways resolvers have a role similar to the DNS resolvers in the Internet, although the communication between clients and gateway resolvers are not done over plain DNS due to integrity and authentication concerns. There may be multiple gateway instances deployed in the network to serve both geographical and load balancing purposes. The gateway resolver then provides to the client the IP address of the gateway along with a challenge/authenticating value. This serves as a way to authenticate the tunnelling connection between the client and its gateway. A specific multiplexing protocol is used between the client and the gateway, such that a unique session between them can serve various same client to different services' flows. The gateway first queries the internal resolver in the private network to figure out the actual service address (e.g., `proto://s.com`) and forward traffic information through the host providing the requested service. For the client, such an approach abstracts the service name from its running hosts, enabling transparent moves of the services to different internal servers and port numbers.

Such a network paradigm calls for an adapted networking API. URLink introduces the *URLSocket* API. Figure 3 depicts the establishment of a connection-oriented transfer using a URLSocket. Instead of requiring the IP address and port, usually passed by a `struct sockaddr`, the application can directly communicate the full URL of the service to the URLSocket. This URL can contain the textual identifier of the application protocol used by the contacted service. Under the hood, the URLSocket will perform the steps as described in Figure 2 to establish a flow towards a service, and provides back network events to the application.

To prevent direct communication between a client and a gateway, the gateway resolver can provide some authentication data that should be provided by the client to contact the gateway. This authentication data can also be used to log the client's activities, notably when abnormal behaviors are detected.

Non-goals

The core idea of URLink lies in hiding low-level network information of running services from clients to ease the service management by operators. It is not meant to act as an anonymity network such as Tor [10]. However, the proposed architecture could be extended to add a second gateway, similar to MASQUE [3], such that none of the gateways know both the exact client IP and accessed service [37]. Such an extension would require coordinating mechanisms between hop gateways to retrieve a misbehaving client when a service's operator wants to further audit specific traffic. Note that URLink gateway does not have access to the payload of the communication if the original communication is already encrypted (e.g., HTTPS) as depicted in Fig. 4. Therefore, it is not intended to prevent attacks based on malicious payloads or act as a deep packet inspector.

4 Benefit for IoT Deployments

Consider an IP-based IoT video-surveillance camera. Such a device provides a video stream service accessible on a given port. Directly

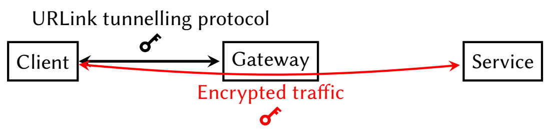

Figure 4: Handling encrypted traffic. The original traffic stays encrypted, but gains an additional encryption layer when carried over the QUIC proxy.

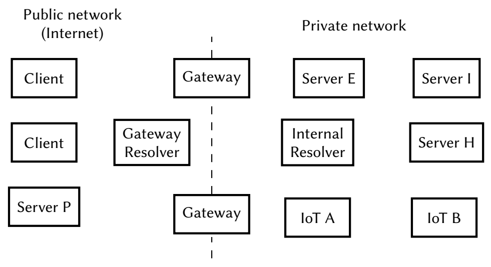

Figure 5: From a network viewpoint, the services' hosts are in a private network communicating with the service gateway. Clients need to pass through the gateway to exchange with servers.

connecting such a device on a public network would enable anyone on the Internet to discover its presence and potential running services using network scanning tools. Furthermore, the device itself often lacks the necessary security protections, to address the potential attacks that may rise from being publicly accessible. For example e.g., it might have unnecessary open ports and exposed services are vulnerable to network attackers such as brute force attacks and default credentials. In fact, Internet-wide scanning of IP-based surveillance camera and the prevalent use of insecure default passwords in these devices have been used in Mirai botnet for lunching DDoS attacks [2]. As shown by a recent study [39], vulnerable IoT devices still exists on the Internet with more than 1.8 million devices that could get exploited by adversaries.

Our URLink proposal especially help with IoT deployments where access to services running on IoT devices will be made through a unique service name, instead of devices being directly accessible. For example in Fig. 5, let us consider `IoT A` device to be a surveillance camera. Such device would run a video streaming service which enables users to remotely connect to camera. To access the video stream service on this device, a dedicated name is assigned to the services, http://video.iot-b.mynetwork.org. The legitimate clients of a service know the corresponding service name. Using the gateway resolver (Fig. 5), they can reach the gateway and possibly acquire the authentication data. The gateway verifies the authentication data confirming that the client is authorized to reach the service, and based on the network information provided by the internal resolver, it reaches the actual service running on device `IoT B`. Then, service-specific communication takes place as if URLink was not used, including any application-level authentication.

Despite its simple idea, URLink effectively addresses numerous security problems that could arise if the services on device `IoT B` were publicly accessible. Notably, the risk of scanning attacks is mitigated, as probing clients are unaware of the actual service names and the vast namespace makes brute-forcing impractical. This alone significantly reduces the risk of IoT malware targeting publicly accessible vulnerable devices. Even if a malicious client becomes aware of a specific service name, the potential harm is reduced. First, the client is unable to gain direct network knowledge of other services running on the device, whereas previously it could use different port numbers associated on the same IP to find them. Additionally, the authenticated connection makes harder to succeed any brute-force attack attempts on the service. Since the authentication data expires at a specific time or after a certain number of round trips, the client must repeatedly acquire new authentication data, further impeding its progress and making sustained attacks more difficult to perform. Finally, we note that the gateway's visibility into the traffic enables it to actively monitor and search for malicious behaviors during communication. If any suspicious activity is detected, the gateway is capable of taking necessary measures to halt the communication and prevent potential harm. This proactive anomaly detection approach adds an additional resilience-layer against malicious parties. Although these security measures do complicate the responsibilities of the gateway, they provide a valuable benefit in scenarios such as IoT, where devices lack sufficient built-in protection. By assuming these protective roles, the gateway plays a crucial role in safeguarding the network and enhancing the overall security posture, compensating for the inherent vulnerabilities often found in IoT devices.

5 Deployment Considerations

From a network viewpoint, the node's URLSocket and the service gateway need to define a protocol atop which they can bridge communications between the public network and the private one. Given the system and privacy requirements, the protocol behind URLSocket should support reliable stream multiplexing (for TCP transfers), unreliable data streams (for UDP communications) and encryption (for potentially cleartext traffic as discussed in Section 4). To provide these requirements, the QUIC transport protocol [21] is a relevant base protocol given that the aforementioned requirements are supported by design, if the DATAGRAM extension is negotiated [32]. An application protocol should however be built to make the link between the public flows and the private ones.

From an endpoint perspective, applications need to adopt the approach taken by URLink. We identify two kinds of served applications: (i) those with a native URLSocket support, and (ii) unmodified ones behind a client-side proxy.

When the benefits of URLink are known before the birth of an application, it can start using the URLSocket API from day one. To provide such native support for URLSocket, several approaches are possible. One consists in providing a library that applications can leverage to implement the application URLSocket as depicted in Figure 3. The provided library will perform all the required connectivity operations on behalf of the application. The library can provide a API very similar to the Socket API, the main changes being the URLSocket creation, its `connect()` and `bind()` functions (as they rely on URLs instead of `sockaddr`). Such a URLSocket API

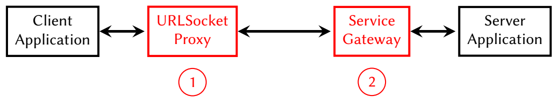

Figure 6: Moving an existing application to the URLink architecture.

can be provided by a user-space library that would leverage a user-space QUIC implementation. On the long run, if a large adoption is met, operating systems may eventually integrate a native support for the URLSocket API. Still, such native applications will only form a marginal fraction of the potential URLink traffic.

On the other hand, there are many existing applications where the proposed scheme could bring advantages to the service providers, but would require incentives and deployment time at client side. Assuming that many client applications will adopt the URLSocket API in a short time span is unrealistic, specially in IoT networks, as client applications and systems need to be updated. Still, it remains possible to keep the network advantages brought by the URLink architecture without modifying the existing applications. Such a solution requires a proxy-based architecture. The deployment process consists in a two-step process. First, the client application must be configured to forward its traffic to a client-side proxy. This proxy may run on the client host itself. For instance, web browsers can be configured to use some proxy (e.g., SOCKS proxy) and networking rules may redirect host-generated DNS traffic towards the proxy solution. The client-side proxy can provide a backward-compatible solution (i.e., bypass the proposed solution) while the service's server is still only accessible through its IP address and port number. Later, at an announced date, the server can migrate behind the service resolver and rely on the URLink system.

5.1 Implementation

To demonstrate the feasibility of our approach, we implement the proxy-based URLink (Fig. 6) using two pieces of software. The first is the URLink gateway bridging the private network with the public one. The second is a client-side daemon enabling client applications to take advantage of the URLink. Concretely, the client daemon listen on a dedicated, local port incoming traffic (either TCP or UDP). It then proxies the incoming traffic into URLink flows over a long-lived QUIC [21] connection. This connection notably exchanges control information about the state of the flow requests (client request connection to a specific service, gateway replies whether it is accepted or denied). When accepted, each TCP connection is mapped to a QUIC stream, while UDP packets are mapped to QUIC datagrams. The gateway then proxies each flow to the appropriate private server (the internal resolver is currently co-located with, but uniquely accessible by, the gateway). Clients try to establish flow communications by sending requests through the encrypted QUIC tunnel (Fig.4). Even if the base service's protocol is not encrypted, the QUIC tunnel ensures that the communication is protected from any external monitoring node, thus providing additional security and privacy for the communication. Our Proofs-of-Concept (PoC) are written in Rust, rely internally on the quiche [8] QUIC implementation and are made of about 5000 lines of code. We performed

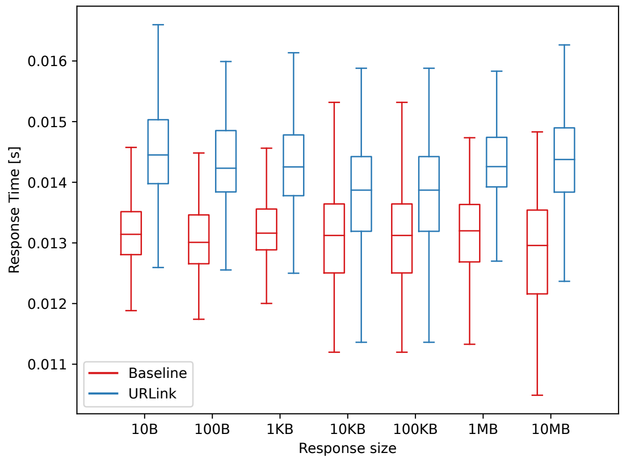

Figure 7: Comparing latency according to different response sizes with and without URLink when making requests to the webserver.

a simple evaluation of our PoC to access a simple web server. For this basic evaluation, all components (i.e., gateway, client, and webserver) run on localhost. We repeat our evaluations for varying server's response size (between 10B to 10MB) and repeat each request 100 times. Under the same test conditions, we compare the average response time when the server is reached through URLink gateway, and when its directly reached in Fig. 7. The difference between the response time with and without URLink is statistically significant in each case (Welch's t-test $p < 0.05$).

6 Discussion and Future Work

In the previous section, we presented early performance test results for a PoC implementation of URLink which indicate the feasibility of our approach. However, our current implementation lacks crucial security features. Specifically, we have not implemented authentication data that clients need to present to the gateway. We outline two different methods for obtaining this authentication data, which we plan to address in future work. In both approaches, we will rely on the gateway resolver to provide the authentication data to the client. This reliance emphasizes the importance of securing communication between the client and the gateway resolver. For instance, when using DNS, we will rely on mechanisms such as DNSSEC [19] or DNS-over-QUIC [20]. As for the authentication data itself, the first alternative involves using proof-of-work schemes in a challenge/response process between the gateway and the client. The second option is to use a cryptographic scheme based on digital signatures, in which the resolver signs attributes related to the client's connection.

Even with the implementation of the authentication data, gateways and gateway resolvers remain susceptible to DoS attacks. For instance, attackers can establish connections with the gateway while presenting fraudulent authentication data, with the aim of rendering the services inaccessible hoping that the gateway would waste CPU resources during the verification process. In a way, URLink transfers the responsibility of countering scanning attacks from the service nodes to the gateways and gateway resolvers. We argue that this shift in responsibility is advantageous especially in application domains such as IoT in which modifying the service nodes might be challenging. Introducing a gateway to the private network, which has sufficient computational power and can be readily extended, serves as an additional layer of protection for all nodes within the private network. We should also note that, in its current design, URLink does not provide protection against attackers within the private network. One way to tackle this is by configuring the services to exclusively handle requests originating from the gateway. We leave this for future works.

7 Conclusion

This paper proposed URLink, a paradigm shift in how network services can be made accessible over the Internet. It revealed that services can be exposed only with their names, i.e., without communicating the running host's IP address and port. This paper discussed the possible security issues that URLink may introduce and described how an implementation can handle them. Finally, it described how applications could leverage URLink, either through native URLSocket support or using a proxy-based solution. Our future works consist in concretely designing the tunnelling protocol proposed by URLink, implementing it and evaluating URLink using IoT testbeds.

References

[1] Prashant Anantharaman, Liwei Song, Ioannis Agadakos, Gabriela Ciocarlie, Bogdan Copos, Ulf Lindqvist, and Michael E. Locasto. 2020. IoTHound: Environment-Agnostic Device Identification and Monitoring. In *Proceedings of the 10th International Conference on the Internet of Things* (Malmö, Sweden) *(IoT '20)*. Association for Computing Machinery, New York, NY, USA, Article 7, 9 pages. https://doi.org/10.1145/3410992.3410993

[2] Manos Antonakakis, Tim April, Michael Bailey, Matthew Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. 2017. Understanding the Mirai Botnet. In *Proceedings of the 26th USENIX Conference on Security Symposium* (Vancouver, BC, Canada) *(SEC'17)*. USENIX Association, 1093–1110.

[3] Apple. 2021. iCloud Private Relay Overview. https://www.apple.com/privacy/docs/iCloud_Private_Relay_Overview_Dec2021.PDF.

[4] Robert Beverly, Ramakrishnan Durairajan, David Plonka, and Justin P Rohrer. 2018. In the IP of the beholder: Strategies for active IPv6 topology discovery. In *Proceedings of the Internet Measurement Conference 2018*. 308–321.

[5] Mike Bishop. 2022. HTTP/3. RFC 9114. https://doi.org/10.17487/RFC9114

[6] Chromium Blog. 2020. Chrome is deploying HTTP/3 and IETF QUIC. https://blog.chromium.org/2020/10/chrome-is-deploying-http3-and-ietf-quic.html.

[7] Carna Botnet. 2012. Internet Census 2012:Port scanning /0 using insecure embedded devices. https://census2012.sourceforge.net/paper.html.

[8] Cloudflare. 2023. quiche. https://github.com/cloudflare/quiche.

[9] Consumer Reports. 2017. Consumer Reports Launches Digital Standard, Begins Evaluating Products, Services for Privacy and Data Security. https://www.consumerreports.org/privacy/consumer-reports-to-begin-evaluating-products-services-for-privacy-and-data-security-a2018541943/.

[10] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. *Tor: The second-generation onion router*. Technical Report. Naval Research Lab Washington DC.

[11] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J. Alex Halderman. 2015. A Search Engine Backed by Internet-Wide Scanning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15)*. Association for Computing Machinery, New York, NY, USA, 542–553. https://doi.org/10.1145/2810103.2813703

[12] Zakir Durumeric, Eric Wustrow, and J Alex Halderman. 2013. {ZMap}: Fast Internet-wide Scanning and Its Security Applications. In *22nd USENIX Security Symposium (USENIX Security 13)*. USA, 605–620.

[13] Ericsson. 2023. Ericsson Mobility Report. https://www.ericsson.com/49dd9d/assets/local/reports-papers/mobility-report/documents/2023/ericsson-mobility-report-june-2023.pdf.

[14] Roy T. Fielding, Mark Nottingham, and Julian Reschke. 2022. HTTP Semantics. RFC 9110. https://doi.org/10.17487/RFC9110

[15] Andy Greenberg. 2015. Hackers Remotely Kill a Jeep on the Highway. https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/.

[16] Mark Gritter and David R. Cheriton. 2001. An Architecture for Content Routing Support in the Internet. In *Proceedings of the 3rd Conference on USENIX Symposium on Internet Technologies and Systems - Volume 3* (San Francisco, California) *(USITS'01)*. USENIX Association, USA, 4.

[17] Mark Gritter and David R Cheriton. 2001. An architecture for content routing support in the internet. In *3rd USENIX Symposium on Internet Technologies and Systems (USITS 01)*.

[18] Arnt Gulbrandsen and Dr. Levon Esibov. 2000. A DNS RR for specifying the location of services (DNS SRV). RFC 2782. https://doi.org/10.17487/RFC2782

[19] Paul E. Hoffman. 2023. DNS Security Extensions (DNSSEC). RFC 9364. https://doi.org/10.17487/RFC9364

[20] Christian Huitema, Sara Dickinson, and Allison Mankin. 2022. DNS over Dedicated QUIC Connections. RFC 9250. https://doi.org/10.17487/RFC9250

[21] Jana Iyengar and Martin Thomson. 2021. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000. https://doi.org/10.17487/RFC9000

[22] Liz Izhikevich, Renata Teixeira, and Zakir Durumeric. 2021. LZR: Identifying Unexpected Internet Services.. In *USENIX Security Symposium*. 3111–3128.

[23] Liz Izhikevich, Renata Teixeira, and Zakir Durumeric. 2022. Predicting IPv4 Services across All Ports. In *Proceedings of the ACM SIGCOMM 2022 Conference* (Amsterdam, Netherlands) *(SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 503–515. https://doi.org/10.1145/3544216.3544249

[24] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. 2009. Networking Named Content. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies* (Rome, Italy) *(CoNEXT '09)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/1658939.1658941

[25] Level 3 Threat Research Labs. 2016. Attack of Things! https://web.archive.org/web/20161003194500/http://blog.level3.com/security/attack-of-things/.

[26] Linda Markowsky and George Markowsky. 2015. Scanning for Vulnerable Devices in the Internet of Things. In *2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)* (Warsaw, Poland). IEEE Press, 463–467. https://doi.org/10.1109/IDAACS.2015.7340779

[27] Jomilė Nakutavičiūtė. 2018. Hacker terrorizes family by hijacking baby monitor. https://nordvpn.com/blog/baby-monitor-iot-hacking/.

[28] Lily Hay Newman. 2016. What We Know About Friday's Massive East Coast Internet Outage. DNS service Dyn faces DDoS attacks. https://www.wired.com/2016/10/internet-outage-ddos-dns-dyn/.

[29] Erik Nordström, David Shue, Prem Gopalan, Rob Kiefer, Matvey Arye, Steven Ko, Jennifer Rexford, and Michael J. Freedman. 2012. Serval: An End-Host Stack for Service-Centric Networking. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. USENIX Association, San Jose, CA, 85–98. https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/nordstrom

[30] Erik Nordström, David Shue, Prem Gopalan, Rob Kiefer, Matvey Arye, Steven Ko, Jennifer Rexford, and Michael J Freedman. 2012. Serval: An {End-Host} stack for {Service-Centric} networking. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. 85–98.

[31] Jose Pagliery. 2014. Internet of fails: What's wrong with connected devices. https://money.cnn.com/2014/10/08/technology/security/internet-of-things-security.

[32] Tommy Pauly, Eric Kinnear, and David Schinazi. 2022. An Unreliable Datagram Extension to QUIC. RFC 9221. https://doi.org/10.17487/RFC9221

[33] Check Point. 2018. Faxploit: New Check Point Research Reveals How Criminals Can Target Company & Private Fax Machines to Take Over Networks and Spread Malware. https://www.checkpoint.com/press-releases/faxploit-new-check-point-research-reveals-criminals-can-target-company-private-fax-machines-take-networks-spread-malware/.

[34] Elvira Rodríguez, Arman Noroozian, Michel van Eeten, and Carlos Hernandez Gañán. 2021. Superspreaders: Quantifying the Role of IoT Manufacturers in Device Infections. , 18 pages.

[35] Lucas Ropek. 2021. A Home Security Worker Hacked Into Surveillance Systems to Watch People Have Sex. https://gizmodo.com/a-home-security-worker-hacked-into-surveillance-systems-1846111569.

[36] Said Jawad Saidi, Anna Maria Mandalari, Hamed Haddadi, Daniel J. Dubois, David Choffnes, Georgios Smaragdakis, and Anja Feldmann. 2021. Detecting Consumer IoT Devices through the Lens of an ISP. In *Proceedings of the Applied Networking Research Workshop* (Virtual Event, USA) *(ANRW '21)*. Association for Computing Machinery, New York, NY, USA, 36–38. https://doi.org/10.1145/3472305.3472885

[37] Paul Schmitt, Jana Iyengar, Christopher Wood, and Barath Raghavan. 2022. The decoupling principle: a practical privacy framework. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*. 213–220.

[38] Satyajit Sinha. 2023. State of IoT 2023: Number of connected IoT devices growing 16% to 16.7 billion globally. https://iot-analytics.com/number-connected-iot-devices/.

[39] Shreyas Srinivasa, Jens Myrup Pedersen, and Emmanouil Vasilomanolakis. 2021. *Open for Hire: Attack Trends and Misconfiguration Pitfalls of IoT Devices*. Association for Computing Machinery, New York, NY, USA, 195–215. https://doi.org/10.1145/3487552.3487833

[40] Tim Wright, Simon Blake-Wilson, Jan Mikkelsen, Magnus Nystrom, and David Hopwood. 2006. Transport Layer Security (TLS) Extensions. RFC 4366. https://doi.org/10.17487/RFC4366

[41] Haitao Xu, Zhao Li, Chen Chu, Yuanmi Chen, Yifan Yang, Haifeng Lu, Haining Wang, and Angelos Stavrou. 2018. Detecting and characterizing web bot traffic in a large e-commerce marketplace. In *Computer Security: 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part II 23*. Springer, 143–163.

[42] Zeljka Zorz. 2016. Insecure pacemakers can be easily hacked. https://www.helpnetsecurity.com/2016/12/01/insecure-pacemakers-easily-hacked/.